

Doctoral thesis

Doctoral theses at NTNU, 2023:64

Abdulmajid Murad

Uncertainty-Aware Autonomous Sensing with Deep Reinforcement Learning

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical
Engineering
Dept. of Information Security and
Communication Technology



Norwegian University of
Science and Technology

Abdulmajid Murad

Uncertainty-Aware Autonomous Sensing with Deep Reinforcement Learning

Thesis for the Degree of Philosophiae Doctor

Trondheim, March 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU

Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

© Abdulmajid Murad

ISBN 978-82-326-6625-6 (printed ver.)
ISBN 978-82-326-6974-5 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:64

Printed by NTNU Grafisk senter

Abstract

The goal of many Internet of Things (IoT) sensing applications, such as environmental monitoring, is to support decision-making by providing valuable information about various phenomena. One approach to achieve this goal is to deploy a network of wireless sensors that collect and transmit measurements of the phenomena. However, typical wireless sensors have limited energy, computation, and communication resources, making it infeasible to measure continuously at a high rate. Fortunately, we can use efficient sensing techniques to address these resource constraints by selecting only the most informative measurements and avoiding wasting resources on redundant or least informative measurements. Yet, designing such sensing techniques often involves manually fine-tuned heuristics or algorithms that are task-specific and non-adaptive. This approach is not scalable to the IoT setting with a typically large number of devices deployed in dynamic and non-stationary environments. There is a need for more autonomy to achieve scalable sensing solutions, where individual sensors autonomously learn strategies suitable for their specific environments and capabilities. Therefore, we investigate using deep reinforcement learning to develop autonomous sensing solutions. We first address the challenge of computational complexity by building a general-purpose simulator to train the behavior of sensing agents. Through three use cases, we demonstrate the feasibility of using deep reinforcement learning to achieve autonomous sensing. We then explore using various reward functions that align with the application goals and guide learning toward desired behaviors, including utility-based, information-based, and value-based rewards. Finally, we show that explicitly representing our understanding of a phenomenon can guide the autonomous sensing agent in selecting informative measurements that maximally inform our prior knowledge of the phenomenon. Overall, the results of this work show that deep reinforcement learning can lead to increased autonomy and, thus, scalability in IoT sensing applications while achieving comparable performance to conventional methods.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* (Ph.D.) in Information Security and Communication Technology, at the Faculty of Information Technology and Electrical Engineering (IE), Norwegian University of Science and Technology (NTNU). The research presented here was conducted under the supervision of Associate Professor Frank Alexander Kraemer, and co-supervision of Professor Kerstin Bach and Associate Professor Gavin Taylor.

The thesis takes the form of a paper collection. The included papers have been published or submitted for publication in scientific conferences or journals. The papers have been reformatted to have consistent formatting within the thesis and may deviate visually from the published versions.

Acknowledgements

First and foremost, I am grateful to my supervisor Frank Alexander Kraemer for his guidance, support, and mentorship throughout my PhD journey. Your expertise, patience, and encouragement have been instrumental in my academic and personal growth, and I cannot thank you enough for your dedication. I am also grateful to my co-supervisors: Kerstin Back and Gavin Taylor. You were the perfect complement to Frank's supervision, and I am forever indebted to your invaluable contributions. Our weekly meetings, although sometimes stressful to prepare for, were always insightful, and I left them feeling motivated and inspired. I always admire Frank's organizational skills, Kerstin's ability to get things done, and Gavin's ability to see the big picture. I am lucky to have had the opportunity to work with each of you.

I would also like to thank my professors and colleagues who contributed to my learning and development. First, thanks to my teaching supervisor, David Palma, for his guidance and patience throughout my teaching journey. Thanks to my fellow PhD students: Befekadu Gebraselase, Mayank Raikwar, Mattia Veroni, Ali Esmaily, Sverre Herland, and Even Klemsdal, for the insightful discussions. Thanks to the IIK Networking Group members: Stanislav Lange, Yuming Jiang, Bjarne Helvik, and Poul Heegaard, for the invaluable seminars and feedback. Special thanks to Mona Nordaune; you were incredibly helpful during my four years at NTNU.

Last but not least, thanks to my family and friends back home. You have been my source of strength and inspiration. I am forever grateful for your love and support.

Abdulmajid Murad

Trondheim, February 2023

Contents

Preface	iii
Contents	v

Part 1: Research Overview

1	Introduction	3
1.1	Research questions	5
1.2	Overview of research contributions	6
1.3	Research methodology	8
1.4	Thesis structure	11
2	Background	13
2.1	(Deep) reinforcement learning	13
2.2	Prediction models and uncertainty quantification	26
2.3	IoT sensing	38
3	Related work	43
3.1	Sensor power management	43
3.2	Adaptive sensing	45
3.3	Prediction-based sensing	46
3.4	Reward function design	47
4	Research results	49
4.1	RQ1: Feasibility of autonomous sensing	49
4.2	RQ2: Reward function design	53
4.3	RQ3: Guided autonomous sensing	61
5	Concluding remarks	67
5.1	Summary of research contributions	67
5.2	Limitations and future directions	68
	References	71

Part 2: Publications

Paper I: Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning	85
I.1 Introduction	87
I.2 Deep Reinforcement Learning	88

I.3	Related Work	89
I.4	System Setup	91
I.5	Reward Function Based on Energy Neutrality	92
I.6	Reward Function Based on Application Goals	95
I.7	Results and Discussion	96
I.8	Conclusion	100
	References	101
Paper II: IoT Sensor Gym: Training Autonomous IoT Devices with Deep Reinforcement Learning		103
II.1	Introduction	105
II.2	The Sensor-Gym Design	105
II.3	Use Case: Duty-Cycle Optimization Under Uncertain Energy-Harvesting	107
II.4	Conclusion	107
	References	108
Paper III: Information-Driven Adaptive Sensing Based on Deep Reinforcement Learning		111
III.1	Introduction	113
III.2	Related Work	114
III.3	System Setup	115
III.4	Gaussian Processes as Predictors	116
III.5	Frugal Operation Mode	119
III.6	Evaluation Results	123
III.7	Discussion	123
III.8	Conclusion	125
	References	125
Paper IV: Probabilistic Deep Learning to Quantify Uncertainty in Air Quality Forecasting		129
IV.1	Introduction	131
IV.2	Related Work	133
IV.3	Air Quality Prediction, Base Models and Metrics	135
IV.4	Deep Probabilistic Forecast	141
IV.5	Discussion	152
IV.6	Conclusion	159
	References	160
IV.7	Supplementary Materials	167
Paper V: Uncertainty-Aware Autonomous Sensing with Deep Reinforcement Learning		175
V.1	Introduction	177
V.2	Related Work	179
V.3	Uncertainty-Aware Autonomous Sensing	180
V.4	Probabilistic Prediction Model	183
V.5	Autonomous Sensing Policy	186
V.6	Evaluation Results	191

V.7	Conclusion	199
	References	199
Paper VI: Learning Task Agnostic Skills with Data-driven Guidance 205		
VI.1	Introduction	207
VI.2	Related Work	208
VI.3	Preliminaries	208
VI.4	Proposed Method	210
VI.5	Experiments	212
VI.6	Discussion	216
VI.7	Conclusion	218
	References	218
VI.8	Supplementary Materials	220

Part 1

Research Overview

Chapter 1

Introduction

The ultimate goal of many IoT sensing applications, such as environmental monitoring [1], is to support decision-making. By providing valuable information about phenomena, these applications help stakeholders make informed decisions about various issues, such as managing resources more efficiently or responding to potential hazards. For instance, policymakers of a municipality can use data from an air quality monitoring system to decide when to close a road or start a street cleaning initiative to reduce air pollution. The main challenge of IoT sensing applications is providing access to information about the phenomena with minimal cost and effort.

One approach to support decision-making is to deploy a network of wireless sensors. These sensors provide information by collecting and transmitting measurements of the phenomena. If the network of sensors has sufficient coverage of the environment and the sensors are precise enough and measure at a high rate, this approach would provide timely and accurate information. The stakeholders would have “real-time” access to the actual value of the phenomena and can then use this information to make informed decisions.

However, typical wireless sensors have limited energy, computation, and communication resources. These resource constraints make it infeasible to measure continuously at a high rate. With recent technological advances, computation time and communication bandwidth are becoming cheaper and are, in practice, only limited by energy resources. Thus, the energy constraint is the main bottleneck in wireless sensor networks. We need to address the energy constraints to realize the full potential of IoT sensing applications.

The act of sensing (i.e., taking a measurement) is the most energy-consuming task in many applications. For instance, in an air quality monitoring system, the energy consumed for measuring the air particles is often an order of magnitude higher than the energy consumed for computation and communication. Additionally, the energy consumed for computation and communication is often proportional to the amount of data measured by the sensing module. Therefore, to address the energy constraints in wireless sensor networks, we need to reduce the energy consumption of the sensing module.

The most straightforward approach is to reduce the number of measurements. But does this reduction lead to information loss and, therefore, to less accurate decisions? Fortunately, not all data are equally informative regarding the application goal. We have an opportunity to use efficient sensing techniques to only select the most informative measurements [2]. In these techniques, sensors identify the most informative data points and only measure at those points. Thus, we avoid wasting resources on acquiring, processing, and communicating those unnecessary, redundant, or least informative measurements. In wireless sensor networks, these techniques

have been studied extensively in the literature under the term “data-reduction” [3, 4, 5, 6].

Despite the extensive research on efficient sensing, the design of such sensing techniques often involves manually fine-tuned heuristics or algorithms. These heuristics and algorithms are often task-specific and non-adaptive. However, IoT presents a setting with a typically large number of devices deployed in dynamic and non-stationary environments. It would be unfeasible to manually design sensing policies for different phenomena, settings, sensors, and batteries. Even dynamic optimization approaches do not scale well, as they require significant domain knowledge and are not robust to major changes in the sensors’ environment, configurations, or constraints. We, therefore, see a need for more autonomous approaches to achieve scalable sensing solutions. The sensors themselves should be able to learn efficient sensing strategies suitable for their specific environment and capabilities.

Reinforcement learning provides an appealing approach to autonomy. It provides a framework to learn good policies from data with less design effort. The sensors can learn efficient sensing strategies by interacting with the environment and receiving feedback. This way, we can achieve scalable sensing solutions without requiring significant domain knowledge or manual intervention. Indeed, reinforcement learning has already been successfully applied to many IoT sensing applications, such as sensor power management [7, 8, 9, 10, 11, 12, 13, 14]. However, conventional reinforcement learning approaches are limited in their ability to learn good policies for IoT sensing. They require the state and action spaces to be finite and small enough to be easily tabulated. This not only adds manual design effort but also limits the scalability of the approach. To live up to the potential of reinforcement learning, we need to develop new methods to learn good policies in an end-to-end fashion without explicitly designing features or discretizing the state or action spaces. Reinforcement learning methods should be able to sort out the complicated and domain-specific technicalities on their own and abstract away most of the manual design effort.

Luckily, we can use the recent advances in deep learning to achieve this goal through deep reinforcement learning. In deep reinforcement learning, we leverage the powerful representational learning of deep neural networks [15] to learn expressive policies. The neural networks can represent complex non-linear functions that are difficult to approximate using conventional function approximations methods. This allows us to learn policies in an end-to-end fashion and to define reward functions more aligned with the application goals. Indeed, deep reinforcement learning has been successfully applied to various real-world autonomous applications [16]. This success is mainly due to the scalability and efficiency of the recent deep reinforcement algorithms [17, 18, 19, 20]. However, we see a gap between the full potential of deep reinforcement learning and its application to IoT sensing. The existing studies are limited to a few specific applications and do not address the unique challenges of automating IoT sensing.

Therefore, we explore the potential of using deep reinforcement learning to enable autonomous IoT sensing. We first formulate the sensing problem as a deep reinforcement learning task and discuss the unique challenges of automating IoT sensing. We then address these challenges by developing novel methods and

frameworks for applying deep reinforcement learning in sensing problems to increase autonomy and, thus, scalability in IoT. We evaluate our approaches on a real-world IoT sensing application and show that they can reduce manual design effort while achieving comparable performance to conventional methods. We also discuss our approaches' advantages and limitations and propose future research directions.

1.1 Research questions

This section describes the research questions we aim to answer in this thesis. We start with the overall research goal of creating autonomous IoT sensing solutions through deep reinforcement learning. We then break down the research goal into more specific research questions. These research questions address the challenges of applying deep reinforcement learning to the autonomous sensing problem.

The first challenge is formulating the sensing problem as a deep reinforcement learning task. We need to define the input state of a sensing policy, the actions that the policy can take, and the goal that the policy should optimize. We also need to define what is being learned and how to learn it. One particular issue we need to address is computational complexity. Deep reinforcement learning requires many interactions with the environment to learn a good policy. This is unfeasible in IoT sensing applications, as the sensors are typically resource-constrained and have limited energy and computation capabilities to afford such a large number of interactions. We need to build a framework or infrastructure to learn sensing policies before deploying them to the sensors. We should demonstrate the feasibility of using this framework in real use cases and evaluate its performance. We, therefore, formulate the first research question as follows:

Research Question 1 (Feasibility of autonomous sensing):

How can we utilize deep reinforcement learning to autonomously learn policies that control sensing of IoT devices?

The second challenge is designing a suitable reward function. This function should be learnable and goal-oriented. It should be learnable since it provides the feedback signal to the learning algorithms, for example, through gradient descent. Most importantly, it should be goal-oriented by reflecting the application goals and guiding the learning toward the desired behaviors. Therefore, we first need to identify the correct application goals before framing them into a scalar reward signal. In our case, we need to answer the question: what are the goals of IoT sensing? One possible answer is to maximize the value of collected data. Specifically, we want the sensing solution to maximize the *value* of collected data while maintaining *stable* and *perpetual* operation. The aim is to ensure the stability and perpetual operation of the IoT device to continue collecting valuable data over the long term. The main challenge is how to quantify the value of collected data and encode the stability and perpetual operation of the IoT device in a reward function. We formulate the second research question as follows:

Research Question 2 (Reward function design):

In the setting of RQ1, how can we define reward functions that support application goals and lead to learning?

Generally, a data point or measurement’s value is the amount of information it adds to or corrects our belief about a phenomenon. We usually have an implicit understanding or prior belief about a phenomenon. This belief is uncertain and imprecise. That is why we take measurements to get facts about the phenomenon and update our understanding. The challenge is to make this understanding explicit and use it to guide the sensing agent in selecting more informative measurements that optimally update our understanding. To address this challenge, we need to define a suitable representation model of the phenomena and an update mechanism consistent with the prior belief and the new measurements. Then, we need to build a framework for using the representation model to guide the sensing agent. We formulate the third research question as follows:

Research Question 3 (Guided autonomous sensing):

How can we use our understanding about a phenomenon to guide an autonomous sensing agent?

1.2 Overview of research contributions

This section provides a brief overview of the research contributions of this thesis. The contributions are organized in six publications, presented in the second part of this thesis. Table 1.1 shows the main contributions of each paper toward the research questions. In Chapter 4, we discuss the contributions toward each research question in more detail.

Table 1.1: Overview of the research contributions of each paper.

RQ	Paper I	Paper II	Paper III	Paper IV	Paper V	Paper VI
<i>RQ1: Feasibility of autonomous sensing</i>	*	*	*		*	
<i>RQ2: Reward function design</i>	*		*		*	*
<i>RQ3: Guided autonomous sensing</i>			*	*	*	

The first paper motivates the need for scalable sensing solutions and how automation is a key enabler for scalability. The paper formulates the sensing problem as a deep reinforcement learning task and shows its feasibility through a use case of power management for energy harvesting IoT nodes. The paper also proposes using a reward function that reflects conflicting objectives of maximizing the *amount* of collected data while maintaining *stable* and *perpetual* operation. Finally, the paper shows how to tune the reward function to achieve different trade-offs between the conflicting objectives.

Paper I (Murad A, Kraemer FA, Bach K, Taylor G):

Title:

Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning

Published at:

13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), IEEE, 2019

The second paper addresses the computational complexity challenge by building a general-purpose simulator to learn sensing policies with deep reinforcement learning. The simulator is based on the OpenAI Gym interface, which can be extended to new applications and environments. The simulator comprises various modules that can be combined and configured to match a specific use case. The main modules of the sensor gym are energy harvesting prediction, energy storage, and energy consumption. We use real data from a solar panel to train a sensor power management agent.

Paper II (Murad A, Kraemer FA, Bach K, Taylor G):

Title:

IoT Sensor Gym: Training Autonomous IoT Devices with Deep Reinforcement Learning

Published at:

9th International Conference on the Internet of Things, ACM, 2019

The third paper addresses how to maximize the *value* of collected data while minimizing energy consumption. It proposes a framework for learning adaptive sensing by integrating a representation model of a phenomenon with a deep reinforcement learning agent. The model quantifies the information gain, thus guiding the agent to select the most informative measurements. Finally, the paper demonstrates the approach's feasibility in a case study of workplace noise monitoring and discusses the proposed framework's implementation and constraints.

Paper III (Murad A, Kraemer FA, Bach K, Taylor G):

Title:

Information-Driven Adaptive Sensing Based on Deep Reinforcement Learning.

Published at:

10th International Conference on the Internet of Things, ACM, 2020

The fourth paper focuses on building representation models of environmental phenomena with uncertainty quantification. It uses various probabilistic deep learning models to quantify uncertainty in the representation models and investigate the reliability of the uncertainty estimates. The paper uses an air quality forecasting problem as a case study to demonstrate the practical impact of uncertainty quantification for making informed decisions. Finally, the paper proposes improving uncertainty estimation with adversarial training and defines the tools and metrics for evaluating uncertainty quantification in data-driven models. The results from this paper are also used in the fifth, more comprehensive paper.

1. Introduction

Paper IV (Murad A, Kraemer FA, Bach K, Taylor G):

Title:

Probabilistic Deep Learning to Quantify Uncertainty in Air Quality Forecasting.

Published in:

Sensors, 2021

The fifth paper combines the results of the previous papers (Paper III, Paper IV) and presents the approach more comprehensively. It investigates the choice of reward functions and proposes a simple reward that maximizes the value of collected data while minimizing energy consumption. It defines value as capturing events of interest while representing the phenomenon with a more credible model. The paper applies the proposed framework to two use cases of indoor noise and air quality monitoring. Finally, it demonstrates the adaptability and transferability of the learned policies to new sensing stations.

Paper V (Murad A, Kraemer FA, Bach K, Taylor G):

Title:

Uncertainty-Aware Autonomous Sensing with Deep Reinforcement Learning

Submitted in:

Internet of Things Journal, IEEE, 2022

The sixth paper focuses on unsupervised reinforcement learning and how to use data from expert-designed policies to guide learning without a reward function. The paper discusses the challenges of designing a reward function and motivates reward-free exploration for learning state representation and preparing the agent for future tasks. Then, it proposes a new framework for using data from expert-designed policies to learn state projection that guides the agent towards desired behaviors.

Paper VI (Klemsdal E, Herland S, Murad A):

Title:

Learning Task Agnostic Skills with Data-driven Guidance.

Published at:

ICML Workshop on Unsupervised Reinforcement Learning, 2021

1.3 Research methodology

This section details the research methodology we used during the research project. Given the research questions in Section 1.1, we observe that they are design tasks because they imply concrete and action-oriented objectives. Such design tasks typically have more than one solution because they can be answered in different ways and with varying degrees of success. Specifically, we can refine the research questions into design tasks as follows:

- Design a framework for utilizing deep reinforcement learning to autonomously learn policies that control sensing of IoT devices

- Design reward functions that support applications goals and lead to learning?
- Design a framework for using our understanding about a phenomenon to guide an autonomous sensing agent.

The design science approach to research [21] can provide a structure for answering the above design tasks. The design science methodology fits well with our research topic because it defines a design cycle approach appropriate for answering or solving our design tasks. In addition to answering design tasks, come knowledge questions that ask for knowledge about the solutions, the problem context, and the interaction between the two.

We use the design cycle to iterate over designing and investigating. We design artifacts to change or improve the design problem and empirically investigate the artifacts to answer some knowledge questions. We use the answers to the knowledge questions to reiterate the design and improve the artifacts. There are many choices of artifacts we can design, and these artifacts do not necessarily solve the design problems, but they try to change or improve the problems. In more detail, we use the following four steps to design and investigate the artifacts.

Step 1: problem investigation

We first identify the problem and knowledge context during the problem investigation step. For instance, we conduct a literature review of autonomous sensing and (deep) reinforcement learning. Then, we identify the challenges regarding the design problems above, such as:

- Computational complexity of deep reinforcement learning makes it infeasible to train agents on resource-constrained sensors.
- Energy-based reward functions lead to undesired emergent behavior, such as duty cycles with high variance.
- Information-based reward functions require quantifying measurements' information gain, which is infeasible to do precisely in practice.
- Explicitly representing our prior belief about a phenomenon requires complex, uncertainty-aware models. Additionally, it is not straightforward to choose the correct input features and external information when building the model.
- It is challenging to update the representation model in a way that is consistent with the prior belief and the new measurement.

Step 2: solution design

We design artifacts to address the design problems during the solution ¹ design step. For example, we design the following artifacts to address the above challenges:

¹The design science literature uses the term “treatment” to refer to a “solution” to indicate that it may partially or not solve the problem. We use the more colloquial term “solution.”

1. Introduction

- We design the IoT Sensor Gym as a simulator to learn sensing policies with deep reinforcement learning.
- We design new goal-oriented reward functions, such as utility-based, information-based, and value-based rewards. These rewards are intended to support the application goals and exclude other side concerns, such as energy.
- We build several probabilistic models to represent our prior belief about the phenomenon explicitly.
- We define several metrics to evaluate the performance of the representation model and the reliability of their uncertainty estimates.
- Using the representation models, we define proxy measures to quantify the information content of measurements, such as Fisher Information and relative entropy.
- We define new methods to update the representation model with new measurements. For instance, we add the new measurements to the input features of the model. Thus the model itself learns how to use new measurements to refine its predictions and take account of the prior belief.
- We design new frameworks to incorporate the representation model into the IoT Sensor Gym and guide the autonomous sensing agent.

Step 3: solution validation

In this step, we empirically investigate the designed artifacts to answer knowledge questions about these artifacts. We start by identifying the knowledge context (prior knowledge) and the knowledge questions. Then, we use the artifacts to answer the knowledge questions. We use the answers to the knowledge questions to reiterate the design cycle and improve the artifacts. These knowledge questions ask for knowledge about the artifacts, the problem context, and the interaction between the two. We answer the knowledge questions using propositions or knowledge claims, then justify these claims by empirical evaluation of the artifacts in real case studies. Example of empirical investigations in our research include:

- We use the IoT Sensor Gym in three use cases to empirically investigate the different artifacts, such as frameworks, reward functions, representation models, metrics, loss functions, and input features. We use the simulator to quickly iterate over many design choices.
- We evaluate the built representation models using various established metrics.
- We evaluate the reward functions and the sensing frameworks based on their utility to the application goals.
- We conduct various ablation studies to investigate the impact of different design choices on the performance of the artifacts.

Step 4: solution implementation

We do not implement the artifacts in real-world settings during the solution implementation step. However, we use real data in the IoT Sensor Gym to investigate the artifacts. For example, we use real solar and weather data to evaluate the sensing policies of energy-harvesting sensors. Additionally, we use data from four air quality measurement stations in Trondheim in the air quality use case. We also use noise data collected using a commercial system in the noise use case. Although we do not implement the artifacts in real-world settings, we ask conceptual questions about the implementation. For example, we ask questions about the following aspects of the implementation:

- Computational complexity of the sensing policy and the representation model.
- Transferability and “out-of-the-box” deployment.
- How to use data observed by the sensor after deployment to update the simulator modules. Thus, addressing how to bridge the gap between simulation and reality (“sim-to-real” gap).

1.4 Thesis structure

The format of this thesis is a compendium of publications. It consists of two main parts: the first is an overview of the research project, and the second is a compendium of six articles.

The first part includes five chapters. The first chapter introduces the research project and discusses the research questions, a brief overview of research contributions, research methods, and thesis structure. Then, the second chapter presents the relevant background on deep reinforcement learning, prediction models and uncertainty quantification, and IoT sensing. The third chapter is a literature review of the related work in IoT sensing and the application of reinforcement in IoT sensing. The fourth chapter presents the research contributions of the project and how these contributions answer the research questions in the first chapter. The fifth chapter concludes the thesis and discusses future work. Finally, the second part of the thesis is the compendium of six publications.

“General methods that leverage computation are ultimately the most effective, and by a large margin”

– Richard S. Sutton, 2019

Chapter 2

Background

2.1 (Deep) reinforcement learning

Our goal in this section is to provide a brief introduction to reinforcement learning (RL) and deep reinforcement learning (DRL). We will first define the problem of RL and then discuss the main approaches to solving it. We will not go into the algorithms’ details but focus on the concepts and the intuition behind them.

2.1.1 Reinforcement Learning Problem

Reinforcement learning is a subfield of machine learning suitable for sequential decision-making under uncertainty. The main idea is to learn by trial and error and have a mechanism to increase the likelihood of “good” trajectories and decrease the likelihood of “bad” trajectories. Concretely, an agent learns to take sequential actions by interacting with an environment. The agent is given a reward signal indicating how good or bad its choice of actions was. The agent’s goal is to maximize an objective expressed in the total reward it receives over time. Figure 2.1 shows a simple diagram of an agent-environment interaction in reinforcement learning.

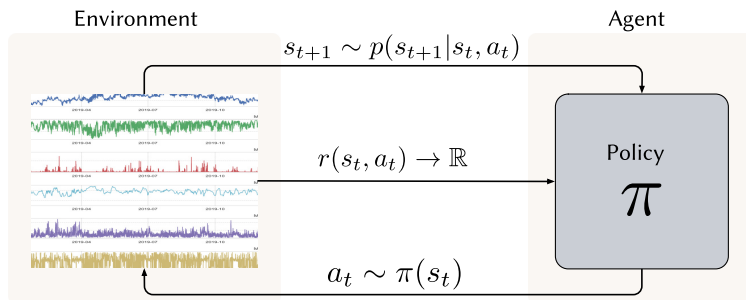


Figure 2.1: An agent interacting with an environment to make sequential decisions, modeled as a Markov Decision Process (MDP).

Mathematically, we specify a reinforcement learning problem as a Markov Decision Process (MDP), which is an extension of a Markov chain in the setting of sequential decision-making. The MDP consists of a set of states \mathcal{S} , a set of actions \mathcal{A} , a transition function P , a reward function R , and a discount factor γ . In more detail, an MDP is represented by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ with the following elements:

- **State** $s \in \mathcal{S}$ is the set of all states the problem might have.

2. Background

- **Action** $a \sim \pi(s)$ is the set of all possible actions the agent can take in a given state s .
- **Transition probability** $p(s'|s, a) \forall s', s \in \mathcal{S}, a \in \mathcal{A}$ is the probability of transitioning from state s to next state s' when taking action a .
- **Reward** $r : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function that maps the current state to a real number. For example, this reward can be a positive number in states where the agent has achieved its goal and a negative number in states where the agent has harmed itself. In this thesis, we use a reward function that depends only on the current state $r(s)$, but in general, the reward function can also depend on action $r(s, a)$ and the next state $r(s, a, s')$.
- **Discount factor** $\gamma \in (0, 1)$ is the factor to discount future rewards. This factor describes how much the agent prefers short-term rewards over long-term ones. A discount factor of 1 means that the agent cares equally about the current and future rewards. A discount factor of 0 means that the agent cares only about the current reward.

The agent learns a behavior expressed as a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. This policy maps the state space \mathcal{S} to the action space \mathcal{A} , specifying the action to take in each state. The agent interacts with the environment by taking action and observing the resulting state and reward. The agent uses this information to iteratively update its policy until it converges to a policy that maximizes the expected cumulative reward over a specified horizon T . Thus, we can express the objective of any reinforcement learning algorithm as follows:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi, p} \left[\sum_{t \in T} \gamma^t r(s_t) \right] \quad (2.1)$$

This objective implies finding a policy π^* that maximizes the expected cumulative reward over the horizon T where the expectation is under the policy distribution π and the transition probability p .

Although the reinforcement learning objective (2.1) is a starting point for thinking about a reinforcement learning solution, it can be intractable to solve directly. We need to derive a metric that evaluates the performance or quality of a policy π . One possible metric is the *value function* $V^\pi(s)$, which is the expected cumulative reward starting from a state s and following policy π :

$$V^\pi(s) = r(s) + \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) \gamma V^\pi(s') \quad (2.2)$$

Then, we can reformulate the reinforcement learning objective (2.1) as the value function starting from the initial state s_1 :

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi, p} \left[\sum_{t \in T} \gamma^t r(s_t) \right] = \arg \max_{\pi} \mathbb{E}_{s_1 \sim p(s_1)} [V^\pi(s_1)] \quad (2.3)$$

Another way to measure the quality of a policy is to use the *Q-function* $Q^\pi(s, a)$ (*action-value function*), which is the expected cumulative reward starting from state

s , taking action a and following policy π :

$$Q^\pi(s, a) = r(s) + \sum_{s' \in \mathcal{S}} p(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') \gamma Q^\pi(s', a') \quad (2.4)$$

We can express the value function in terms of the Q-function or vice versa:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a) \quad (2.5)$$

$$Q^\pi(s, a) = r(s) + \sum_{s' \in \mathcal{S}} p(s'|s, a) \gamma V^\pi(s') \quad (2.6)$$

The value function and Q-function are also connected through the advantage function $A^\pi(s, a)$:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (2.7)$$

The advantage essentially means how much better the action a in state s is compared to the average performance of the policy π in state s .

The value function of a state is the expected cumulative reward that the agent can expect to receive if it starts in that state and follows the policy π . Thus, the goal of any solution is to learn a policy that produces high value for all states.

2.1.2 Designing a reinforcement learning solution

At a high level, any reinforcement learning solution should optimize the application's objective expressed as maximizing the expected cumulative reward (2.1). But when designing a reinforcement learning solution, there are several design choices to consider regarding *what* is being learned and *how* it is being learned. These design choices are easier to think about if we decompose a reinforcement learning solution into three fundamental building blocks as follows:

1. **Policy:** Should our solution have an explicit representation of the policy? Should our solution learn the parameters of a policy function that maps the current state to an action?
2. **Value function:** Should our solution have an explicit representation of the value function? Should our solution learn the parameters of a value function that maps the current state to the predicted or expected cumulative reward?
3. **Model:** Should our solution explicitly represent the dynamics in the environment? Should our solution learn the parameters of a model that maps the current state and action to the next state and reward?

The combination of these three building blocks gives rise to various reinforcement learning solutions and concrete algorithms. These blocks are not mutually exclusive, and many solutions use a combination of these blocks. In the following sections, we discuss how to build a reinforcement learning solution using these three building blocks and how to choose the right combination of these blocks for a given application. We start with conventional reinforcement learning methods in each section and then progress to deep reinforcement learning methods. We have used many of these methods in our research and settled on the more suitable for autonomous sensing applications.

2.1.3 Value-based methods

One way to approach a reinforcement learning problem is to start thinking from the metric. Instead of directly learning a policy that picks actions, we can build a mechanism to evaluate any policy, then use this mechanism to choose and improve upon the policy.

Suppose we have a simple reinforcement learning problem with small and discrete states and actions. We can represent the Q-function $Q^\pi(s, a)$ as a table, where each entry in the table is the Q-value of a state-action pair. We can then use this table to pick actions by choosing the action with the highest Q-value for a given state:

$$a = \arg \max_{a \in \mathcal{A}} Q^\pi(s, a) \quad (2.8)$$

The policy π is not explicitly represented in this approach. Instead, it is implicitly represented by the Q-function. The question now is how to learn the Q-function. We can learn the Q-function using on-policy methods, such as SARSA [22], or off-policy methods, such as Q-learning [23].

SARSA

One approach to learn the Q-function is by trial and error: start with a random policy π , then collect trajectory samples using this policy and use these samples to update the Q-function. The trajectory samples are of the form $\tau = (s_t, a_t, r_t, s_{t+1})$, where t denotes the timestep, and $s_t, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}, r_t = R(s_t)$. We then use these trajectory samples to update the Q-function as follows:

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \delta_t \quad (2.9)$$

where α is the learning rate, and δ_t is the temporal difference error:

$$\delta_t = r(s_t) + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t) \quad (2.10)$$

Essentially, we are updating the current estimate of the Q-value $Q^\pi(s_t, a_t)$ using the observed reward $r(s_t)$ and the estimated Q-value of the next state-action pair $Q^\pi(s_{t+1}, a_{t+1})$. The learning rate α controls how much we update the current estimate of the Q-value. The discount factor γ controls how much we care about future rewards. We can also find a better estimate of δ_t using multi-step bootstrapping:

$$\delta_t = r(s_t) + \gamma r(s_{t+1}) + \dots + \gamma^H r(s_{t+H}) + \gamma^{H+1} Q^\pi(s_{t+H+1}, a_{t+H+1}) - Q^\pi(s_t, a_t) \quad (2.11)$$

where H is the number of steps to look ahead.

The update rule (2.10) is called the SARSA (State-Action-Reward-State-Action) algorithm [22]. One notable observation is that the trajectory samples are collected using the policy π ; thus, SARSA is an *on-policy* algorithm. Consequently, using on-policy can result in less efficient exploration of the state space since the agent may not explore states that are not visited by the policy. To address this issue, we can use various exploration strategies, such as ϵ -greedy, annealing ϵ , or Boltzmann exploration [24]

Q-learning

We can also use a different policy ($\hat{\pi}$) to collect trajectory samples and use these samples to update the Q-function of the policy π :

$$\delta_t = r(s_t) + \gamma \max_{a \in \mathcal{A}} Q^{\hat{\pi}}(s_{t+1}, a) - Q^{\pi}(s_t, a_t) \quad (2.12)$$

This type of learning is called *off-policy* learning, such as the Q-learning algorithm [23]. The main advantage of off-policy learning is sample efficiency. Since it is not necessary to use the policy π to collect trajectory samples, we can use previously collected data. This data can be collected using a previous version of the policy π , or even a different policy $\hat{\pi}$. Another advantage is that we can use a more exploratory policy to collect trajectory samples and use these samples to update the Q-function of the policy π without worrying about the exploration-exploitation trade-off.

In both SARSA and Q-learning, given Q-function approximations, a new and likely-improved policy π^* is implied by $\pi^*(s) = \arg \max_{a \in \mathcal{A}} Q^{\pi}(s, a)$. With a sufficient sampling of the whole state space and alteration between sampling, policy evaluation, and policy improvement steps, these approaches are proven to converge to excellent policies. This is a convenient guarantee but requires the state and action spaces to be finite and small enough to be easily tabulated.

In many real-world applications, the state space is usually large and continuous. In these cases, it is not feasible to represent the Q-function as a table. Instead, we can use a function approximation to approximate the Q-function. One possible function approximator is a neural network. The only difference between conventional and deep reinforcement learning methods is using a neural network. In deep reinforcement learning, we use neural networks to represent the policy, value function, or model. This way, we can leverage the powerful representation learning capabilities of neural networks [15] in reinforcement learning. The main advantage of using neural networks is that they can learn complex non-linear functions that are difficult to learn using conventional methods. This is especially useful in continuous control, where the state and action spaces are often continuous, very large, and complex. Thus, we can perform end-to-end training without explicitly designing features or discretizing the state and action spaces. The downside of using neural networks is that they require a lot of samples to learn a good policy.

Deep Q-networks

In Deep Q-Networks, we use a neural network to approximate the Q-function. The neural network takes the state as input and outputs the Q-values for all actions in the state. This neural network can be trained with gradient descent using the same update rules (2.12) as in the tabular case. Figure 2.2 shows a simple example of a neural network that approximates the Q-function and policy deduction from the Q-function.

We can represent the neural network as a function $Q^{\pi}(s, a; \phi)$, where ϕ is the set of neural network parameters. The update rule (2.12) can be written as:

$$\delta_t = r(s_t) + \gamma \max_{a \in \mathcal{A}} Q^{\hat{\pi}}(s_{t+1}, a; \phi) - Q^{\pi}(s_t, a_t; \phi) \quad (2.13)$$

2. Background

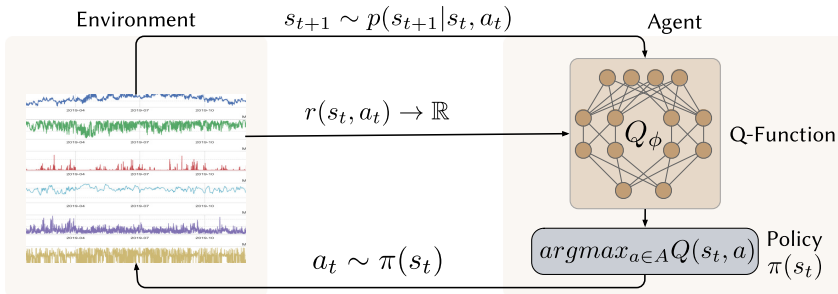


Figure 2.2: Deep Q-Network

Then we can construct a loss function $L(\phi)$ as follows:

$$L(\phi) = \mathbb{E}_{\tau \sim \hat{\pi}} [\delta_t^2] \quad (2.14)$$

where $\tau = (s_t, a_t, r_t, s_{t+1})$ is a trajectory sample. This loss function minimizes the mean squared error between the observed (target) Q-values and the estimated Q-values. We can then use gradient descent to update the parameters ϕ of the neural network:

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} L(\phi) \quad (2.15)$$

Since we use the same neural network to approximate the target Q-values and the estimated Q-values, circular dependency can occur between the target and the estimated Q-values. This can result in unstable learning. We can decouple this circular dependency by using a separate neural network to approximate the target Q-values [17]. The second network is called the target network. The target network is updated periodically with the parameters of the main network. This way, the target Q-values are updated sparingly, and the learning is more stable. The update can occur at a regular interval, for example, every C step [17]. Alternatively, we can do a soft update using a weighted average (polyak averaging) of the main and target network.[25].

Another improvement we can make is to use a replay buffer to store the trajectory samples. Then, we sample a batch of trajectories from this buffer to update the neural network instead of using trajectories from the current policy. This way, we can improve the stability of training and sample efficiency. The sampling from the buffer can be done uniformly [17] or using a prioritized replay buffer [26], in which the more important trajectories are sampled more frequently.

Double Q-learning & Dueling DQN

In (2.13), we are using the maximum Q-value of the next state to update the current Q-value, which can result in an overestimation of the Q-values. To address the overestimation issue, we can use two networks to estimate the target Q-values: one network to select the action and another to evaluate the Q-value of the selected action [27]. This way, we can decouple the action selection from the Q-value estimation. A different approach is to use a single network that has two streams at the end: one stream to estimate the value function $V(s)$, and the other stream to

estimate the advantage function $A(s, a)$ [28]. Then, we estimate the Q-value using the advantage function (2.7). This approach can lead to more stable training by having a better policy evaluation because, in some states, it can be enough to have a better estimate of the value function instead of a better estimate of the Q-value of every action separately.

Limitations of deep Q-networks

These complex approaches of using neural networks to approximate the Q-function allow state space to be continuous but result in a loss of the convergence guarantee. In practice, we get good enough policies by training the neural network with more and more data. However, applying value-based methods successfully to continuous and complex domains remains fundamentally difficult.

Since the neural network needs to have one output node for each action, it is infeasible to only use value-based methods in a continuous action space. We need to either discretize the action space or use a different approach to handle continuous actions, such as policy gradient or actor-critic methods.

2.1.4 Policy-based methods

Value-based methods (2.1.3) have inherent limitations, especially regarding continuous action spaces, exploration, sample-efficiency, and convergence (training) speed. We need to approach the problem from a different perspective. Instead of first learning a value function, and then using the value function to derive policy, we can directly learn the policy. This might seem infeasible since the reinforcement learning objective 2.1 is intractable to solve directly. However, we can start thinking of a different (or approximate) objective that is easier to solve.

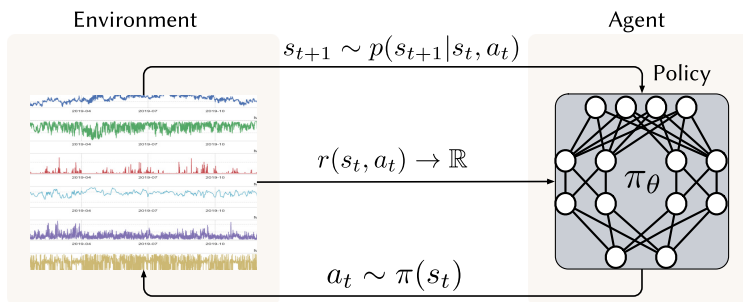


Figure 2.3: Agent-environment interaction using a policy represented by a neural network.

Policy gradient

Suppose we have a policy π that is parameterized by a set of parameters θ , then we can start by rewriting the reinforcement learning objective in terms of the policy

2. Background

parameters:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\pi_{\theta,p}} \left[\sum_{t \in T} \gamma^t r(s_t) \right] = \arg \max_{\theta} J(\theta) \quad (2.16)$$

The objective means: to search for a parameterized policy π_{θ} that maximizes the expected return $J(\theta)$. Then, we can use gradient ascent to find the optimal policy parameters θ^* :

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \quad (2.17)$$

The main challenge is estimating the gradient of an intractable objective $\nabla_{\theta} J(\theta)$. One way is first to construct an unbiased estimate (Monte Carlo sampling) of the objective using several collected trajectory samples:

$$J(\theta) \approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \gamma^t r(s_{i,t}) \quad (2.18)$$

where N is the number of collected trajectory samples using the policy π_{θ} . Then, we can estimate the gradient of the objective function using the trick proposed in the REINFORCE algorithm [29]:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i \in N} \left(\sum_{t \in T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) \left(\sum_{t \in T} \gamma^t r(s_{i,t}) \right) \quad (2.19)$$

where $\nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t})$ is the gradient of the log probability of the action $a_{i,t}$ given the state $s_{i,t}$ using samples collect by the policy π_{θ} . This trick is useful since it enables us to estimate the gradient of an expected value in terms of an expectation of a gradient. In turn, the expectation of a gradient can be estimated using the collected trajectory samples (Monte Carlo sampling).

The REINFORCE algorithm described above is sometimes called *likelihood ratio policy gradient* method. It is similar to maximum likelihood estimation in supervised learning (or imitation learning), which tries to maximize the likelihood of the collected samples. The main difference is that the likelihood is multiplied by the reward function, i.e., we increase the likelihood of a trajectory proportional to the total reward along that trajectory. This insight allows us to implement the policy gradient method using the same infrastructure used in supervised learning with only a slight modification: multiply the gradient by the total reward along a trajectory.

Although the REINFORCE algorithm is simple, it has several issues that make it difficult to use in practice. First, the variance of the gradient estimate is high, i.e., the gradient estimate is very sensitive to the trajectory samples. This high variance makes it difficult to converge to the optimal policy. One improvement we can make is to use the reward-to-go instead of the total reward along a trajectory, which reduces the variance of the gradient estimate as first proposed and proofed by [30]:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) G_{i,t} \quad (2.20)$$

where the reward-to-go $G_{i,t}$ is the sum of the discounted rewards starting from the current time step t until the end of the trajectory i [31]:

$$G_{i,t} = \sum_{t'=t}^T \gamma^{t'-t} r(s_{i,t'}) \quad (2.21)$$

Another improvement we can make to reduce the variance of the gradient estimate is to use a baseline [32]. This baseline is a constant (b) that is subtracted from the reward-to-go $G_{i,t}$:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) (G_{i,t} - b) \quad (2.22)$$

For example, we can use the average reward over many trajectories as the baseline:

$$b = \frac{1}{N} \sum_{i \in N} r(s_{i,t}) \quad (2.23)$$

This way, we only increase the likelihood of trajectories that are better than the average and decrease the likelihood of trajectories that are worse than the average. Before, we increased the likelihood of all trajectories with positive rewards (proportional to the total reward along that trajectory). It is straightforward to prove that subtracting a baseline from the reward-to-go does not change the objective (unbiased estimate of the objective in expectation) [32]. We can further reduce the variance by using a baseline that depends on the state, such as the value function $V(s)$. However, we need to explicitly learn the value function and the policy simultaneously, which is what actor-critic methods do [33].

Actor-critic

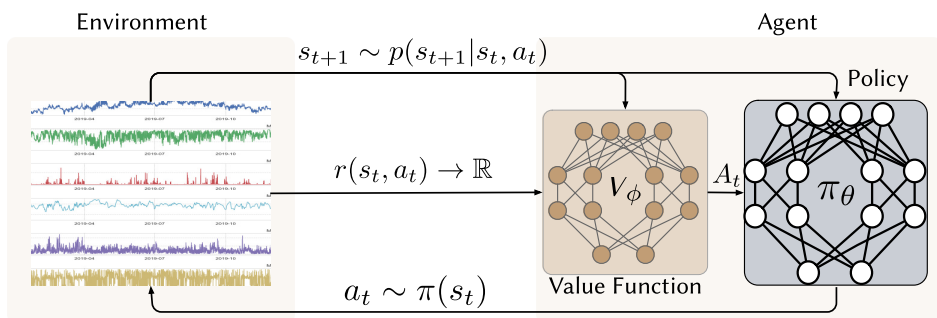


Figure 2.4: Value function approximation using a neural network in an Actor-Critic algorithm.

An actor-critic algorithm combines two building blocks: explicitly representing the policy and the value function. It builds on the policy gradient method (2.22) by using the value function as the baseline. Additionally, it uses the q-function

2. Background

instead of reward-to-go because the reward-to-go is only a single-sample estimate of the return, while the q-function is an expectation of the return. This way, we can reduce the variance of the gradient estimate and make the policy gradient method more stable and improve the convergence rate.

$$\begin{aligned}\nabla_{\theta} J(\theta) &\approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) (Q^{\pi_{\theta}}(s_{i,t}, a_{i,t}) - V^{\pi_{\theta}}(s_{i,t})) \\ &\approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) A^{\pi_{\theta}}(s_{i,t}, a_{i,t})\end{aligned}\tag{2.24}$$

This policy gradient with the advantage function means: we increase the likelihood of actions that have a high advantage (i.e., the action is better than the average action in that state) and decrease the likelihood of actions that have a low advantage (i.e., the action is worse than the average action in that state). We can approximate the advantage using a single sample of the next state:

$$\begin{aligned}A^{\pi_{\theta}}(s_t, a_t) &= Q^{\pi_{\theta}}(s_t, a_t) - V^{\pi_{\theta}}(s_t) \\ &= \left(r(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} | s_t, a_t) V^{\pi_{\theta}}(s_{t+1}) \right) - V^{\pi_{\theta}}(s_t) \\ &\approx r(s_t, a_t) + \gamma V^{\pi_{\theta}}(s_{t+1}) - V^{\pi_{\theta}}(s_t)\end{aligned}\tag{2.25}$$

The advantage approximation now is the expected return starting from state s_t and taking action a_t , but using only a single sample of the next state s_{t+1} . This is the idea behind the advantage actor-critic (A2C) or asynchronous advantage actor-critic (A3C) algorithm [34].

Regarding implementation, we can use a neural network to represent the policy $\pi_{\theta}(a|s)$, where θ are the parameters of the network. The policy is trained using the policy gradient objective (2.24). We also use a second neural network to represent the value function $V^{\pi_{\theta}}(s)$, where ϕ are the parameters of the network. Similar to DQN (2.14), we use the mean squared error loss to train the value function network. It is also possible to estimate the advantage (2.25) using a q-function network instead of a value-function network. However, it is easier to train a value function network than a q-function network because the value function network only needs to predict the value of the current state, while the q-function network needs to predict the value of all possible actions in the current state.

TRPO

The second issue with the policy gradient method (2.24) is that it is an on-policy method: we are collecting trajectory samples using the same policy we are trying to optimize. Consequently, we need to collect new trajectory samples after each policy update and discard the old ones. This is because the objective (2.16) is an expected value under π_{θ} , which means that the gradient estimate is only valid for the current policy. Therefore, policy gradient methods are inefficient and slow to train, especially when approximating the policy using neural networks, since the network's parameters have only a small change with each gradient step. One way to

mitigate this issue is to use importance sampling to estimate the gradient $\nabla_{\theta} J(\theta)$ using samples from different policy $\pi_{\theta'}$ [35]:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i \in N} \sum_{t \in T} \frac{\pi_{\theta}(a_{i,t} | s_{i,t})}{\pi_{\theta'}(a_{i,t} | s_{i,t})} \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) A^{\pi'_{\theta}}(s_t, a_t) \quad (2.26)$$

This means: we are weighting the gradient estimate by the ratio of the probability of the action under the current policy π_{θ} and the probability of the action under the sampling policy $\pi_{\theta'}$. The sampling policy $\pi_{\theta'}$ can be any policy, but it is usually a policy close to the current policy π_{θ} , for example, the older version of the current policy. This way, we can use the old trajectory samples to estimate the gradient and update the network's parameters.

The gradient estimate (2.26) is biased, but the bias is small (bounded) if $\pi_{\theta'}$ is close to π_{θ} . The question is how close $\pi_{\theta'}$ should be to π_{θ} ? The answer to this question is the main idea behind the trust region policy optimization (TRPO) algorithm [35]. The TRPO algorithm uses the KL divergence between the current policy π_{θ} and the sampling policy $\pi_{\theta'}$ to constrain the change in the policy parameters θ between consecutive policy updates.

PPO

A new variant of TRPO called proximal policy optimization (PPO) [36] uses a clipped (surrogate) objective instead of the KL divergence to constrain the change in the policy parameters θ between iterations. Thus, PPO is much simpler to implement and is more efficient to train.

In this thesis, we mostly use PPO because it is very efficient to train and can be used to solve complex, continuous tasks. It enables us to learn state abstractions and effective policies directly from continuous input data. Additionally, PPO enables inherent exploration by sampling actions from the policy's probability distribution. Therefore, using PPO eliminates the need for manual state-space discretization and hand-tuning of exploration, which leads to a lower manual design effort and hence scalable autonomy for IoT systems.

Off-policy actor-critic

TRPO and PPO algorithms mitigate the sample-inefficiency issue of on-policy gradient methods by enabling us to use old trajectory samples to estimate the gradient. However, these algorithms enforce a constraint on how distant the current policy π_{θ} can be from the old sampling policy $\pi_{\theta'}$. Thus, it is impossible to use trajectories sampled from different policies, such as exploratory ones. This limitation can be an issue when it is expensive to get new trajectory samples, and we already have some data collected from a different policy. We would like to have a method that combines the strengths of policy gradient (i.e., supporting continuous action spaces) and value-based methods (i.e., supporting off-policy learning). We can make tradeoffs when combining policy gradient and value-based methods. This tradeoff is essentially the idea behind off-policy actor-critic [37].

DDPG

One approach that extends value-based methods to continuous action spaces is the deterministic policy gradient (DPG) algorithm [38]. This algorithm is developed to the deep deterministic policy gradient (DDPG) algorithm [25]. In DDPG, a deterministic policy $\mu_\theta(s)$ is used to generate actions:

$$a_t = \mu_\theta(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \quad (2.27)$$

where ϵ is a noise term to encourage exploration. In contrast to policy gradient, the policy $\mu_\theta(s)$ is trained to maximize the q-value predicted by a Q-function network $Q_\phi(s, a)$:

$$\theta^* = \arg \max_{\theta} [Q_\phi(s, \mu_\theta(s))] \quad (2.28)$$

where the Q-function parameters ϕ are fixed. Similar to DQN (2.13), the q-function in DDPG is trained using the mean squared error loss between the target q-value and the estimated q-value. However, the action is selected by the policy $\mu_\theta(s)$ instead of argmax over all actions:

$$\delta_t = r(s_t) + \gamma Q_\phi(s_{t+1}, \mu_\theta(s_{t+1})) - Q_\phi(s_t, a_t) \quad (2.29)$$

This is why DDPG can handle continuous action spaces while DQN can only handle discrete action spaces: it is impossible to argmax over all continuous actions. DDPG also uses other tricks to improve training stability, such as using separate target networks for the policy and the q-function, using a replay buffer to store trajectory samples, and using a soft update rule (polyak averaging) to update the target networks.

TD3

Although DDPG is useful for enabling value-based methods to handle continuous action spaces, it inherits some of the issues of DQN, such as over-estimating the Q-values. This can lead to instability during training because the policy can exploit the over-estimated Q-values and produces less optimal actions. To mitigate this issue, we can use two Q-functions instead of one and use the minimum of the two Q-values in the loss (2.29). Additionally, we can decouple the dependency between the policy and the Q-function by delaying the policy update for a few iterations. This delay allows the Q-value estimate to converge to more accurate values, thus reducing per-update error. These are the main ideas behind the twin delayed deep deterministic policy gradient (TD3) algorithm [39]. TD3 is an effective state-of-the-art algorithm for solving complex, continuous tasks with sample efficiency in mind.

Soft actor-critic

A fundamental problem in reinforcement learning is the exploration-exploitation tradeoff. Most of the reinforcement learning algorithms suffers from premature convergence to sub-optimal policies. Although existing exploration strategies can help, they are not effective in complex tasks with sparse rewards or hierarchical

behaviors. To address this issue, we can add an intrinsic reward or regularization to encourage exploration. For example, we can add an entropy term to the reward [40]:

$$r_{\text{new}}(s_t) = r(s_t) + \beta H(\pi_\theta(\cdot|s_t)) \quad (2.30)$$

where $H(\pi_\theta(\cdot|s_t))$ is the entropy of the policy at state s_t :

$$H(\pi_\theta(\cdot|s_t)) = - \sum_{a \in \mathcal{A}} \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \quad (2.31)$$

while β is a hyperparameter that controls the tradeoff between the original reward and the entropy term. The addition of the entropy term is reflected in the reinforcement learning objective and the value function. This is the idea behind the soft actor-critic (SAC) algorithm [19]. SAC similar to TD3 except that it uses a stochastic policy instead of a deterministic policy and the addition of the entropy term.

Reward-free exploration

We can take the exploration one step further in 2.30 by removing the original reward term altogether:

$$r_{\text{new}}(s_t) = \beta H(\pi_\theta(\cdot|s_t)) \quad (2.32)$$

This new reward will encourage the agent to explore the state space by acting as randomly as possible. The objective of this exploration is for the agent to learn the state space representation in such a way that it will be maximally prepared to perform any downstream task when introducing a reward function.

Additionally, we can introduce the idea of skills [41] to encourage the agent to explore the state space in a more structured way. A skill (or option [42]) is sequence of actions that the agent can perform to achieve a specific task. For example, a skill can be a sequence of actions that the agent can perform to reach a specific location. This intuition is the basis of the last paper in this thesis, where we introduce the idea of data-driven guidance to encourage the agent to explore the state space in a more structured way [43].

2.1.5 Model-based methods

In all the previous methods, the agent does not have access to the environment model and does not explicitly try to learn it. This environment model is represented in the MDP by the transition probability $p(s'|s, a) \forall s', s' \in \mathcal{S}, a \in \mathcal{A}$. Instead, the agent learns through interaction with the environment and observing the rewards and states. This type of learning is called model-free learning. However, in some cases, the agent can benefit from having access to the environment model. For example, if the environment model is known or can be learned, the agent can use it to plan and make better decisions. This is the idea behind model-based reinforcement learning [44, 45, 46]. Model-based methods are useful for sample-efficient learning because they can use the model to simulate the environment and plan ahead. However, they are computationally expensive and introduce another source of error that affects the planning algorithms' performance.

2. Background

This thesis uses model-free reinforcement learning, where the agent learns through interaction with the environment without explicitly learning the transition probability. However, we use prediction models that predict the evolution of a phenomenon in the environment. These prediction models are used to guide the agent’s actions. Thus, we gain some benefits from model-based methods without the computational overhead. At the same time, we inherit some of the challenges of model-based methods, such as the objective mismatch [47, 48] between the agent and the prediction model. This is discussed in more detail in Paper V. In the following section, we describe the relevant prediction models we use in this thesis.

2.2 Prediction models and uncertainty quantification

A key part of our work involves representing a phenomenon with an estimation or prediction model informed by sparse measurements. We use these models to predict the phenomenon’s future evolution and guide the agent’s actions. Since any human being considers the prediction’s uncertainty when making decisions, we also envision that autonomous sensing systems need to consider the prediction’s uncertainty when making sensing decisions. Thus, these models also need to quantify their predictive uncertainty. In this section, we describe the different types of prediction models and uncertainty quantification methods used in our work. We also show examples using these models to forecast Trondheim’s air quality.

2.2.1 Persistence model

We start with the simplest baseline predictor: a persistence model that predicts the future value of a time series using the last observed value. We can also leverage the diurnal patterns of a phenomenon to improve the persistence model. For example, indoor noise level tends to have a diurnal pattern, where the noise level is higher during the day and lower at night. Additionally, the noise level is higher on weekdays than on weekends. Thus, we can predict the future value of a time series using the observed value 24 hours ago (daily persistence model) or the last observed value of the same day of the week (day-of-the-week persistence model). Figure 2.5 shows an example of a daily persistence model for forecasting the air quality in Trondheim over 20 days. The model achieves a root mean squared error (RMSE) of 12.04. This is a reasonable baseline for many time series that exhibit diurnal patterns.

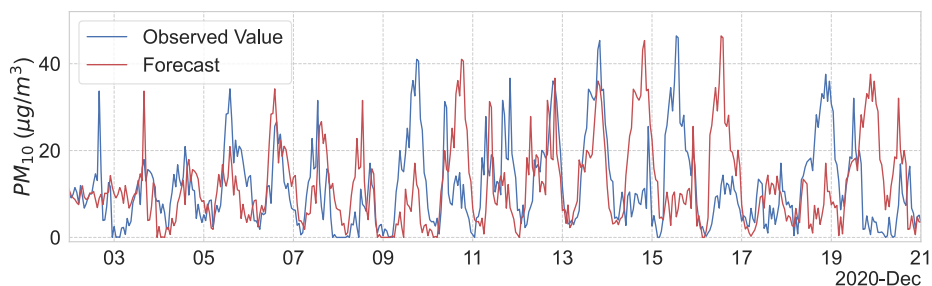


Figure 2.5: Daily persistence model for forecasting air quality.

2.2.2 XGBoost

Another simple yet powerful prediction model is XGBoost (*eXtreme Gradient Boosting*) [49]. XGBoost is a gradient-boosting framework that uses decision trees as base learners. XGBoost uses the same model representation and inference as Random forests (i.e., gradient-boosted decision trees) but has a different training mechanism since it uses the second-order approximation of the training objective. This allows XGBoost to be much faster and more accurate than Random forests. Figure 2.6 shows an example of XGBoost for forecasting the air quality in Trondheim over 20 days. The model is trained on one year of data and achieves a RMSE of 8.81, which is a significant improvement over the daily persistence model.

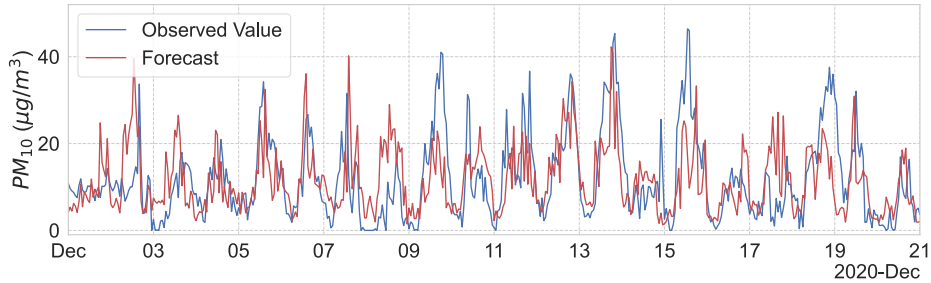


Figure 2.6: XGBoost model for forecasting air quality.

One advantage of using an XGBoost model is the feasibility of retrieving the feature importance by assigning a score to each input feature, which indicates how useful that feature is when making a prediction, thus contributing to prediction interpretation. Figure 2.7 shows the feature importance of the XGBoost model for forecasting the air quality in Trondheim. The model assigns higher importance to the observed air quality of the previous day. This means that the model can capture the diurnal patterns of the air quality, i.e., XGBoost learned by itself the persistence model.

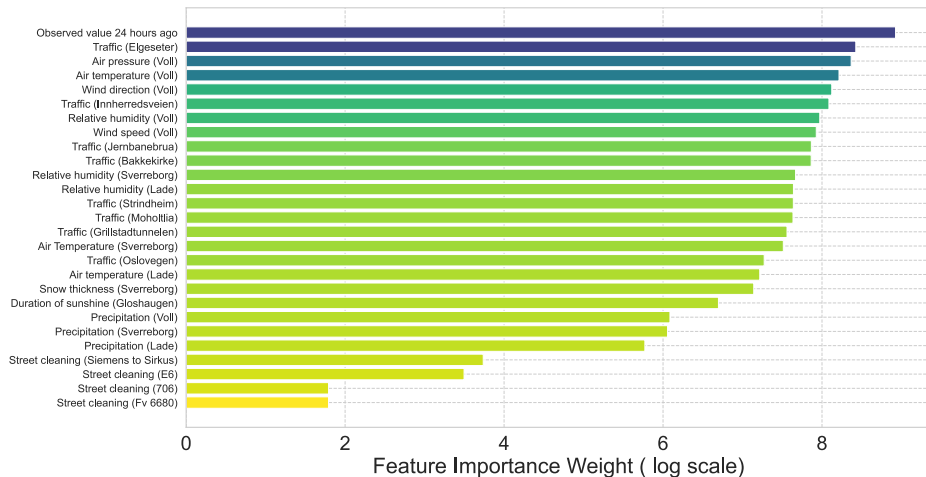


Figure 2.7: Feature importance of the XGBoost model for forecasting air quality.

2.2.3 Gaussian processes

The third prediction model we use (Paper III) is Gaussian processes (GP), a non-parametric Bayesian method [50]. Essentially, GP defines a distribution over functions, where each function is a possible trajectory of a phenomenon. This means that we model the future evolution of the phenomenon as a random function f drawn from a distribution $p(f)$. We update the distribution to reflect the new information as we observe new measurement data. This is done by computing a posterior distribution $p(f|\mathcal{D})$ where \mathcal{D} is the set of observed measurements. We then use the posterior distribution to predict the phenomenon's future evolution.

More specifically, suppose the new measurements are $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where \mathbf{x}_i is a feature vector consisting of explanatory variables and y_i is the corresponding measured value. We can write the posterior distribution $p(f|\mathcal{D})$ as a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$p(f|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (2.33)$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are given by:

$$\boldsymbol{\mu} = \mathbf{K}^T \mathbf{K}_{\mathcal{D}}^{-1} \mathbf{y} \quad (2.34)$$

$$\boldsymbol{\Sigma} = \hat{\mathbf{K}} - \mathbf{K}^T \mathbf{K}_{\mathcal{D}}^{-1} \mathbf{K} \quad (2.35)$$

where $\mathbf{K}_{\mathcal{D}}$ is the covariance matrix of the measurements data, $\hat{\mathbf{K}}$ is the covariance matrix of the prediction points, \mathbf{K} is the covariance matrix between the measurements data and the prediction points, and \mathbf{y} is the vector of observed measurements. Essentially, GP produces a prediction distribution $p(f|\mathcal{D})$ based on similarity with the observed measurements data \mathcal{D} (training examples).

GP measures the similarity using the kernel function, a positive definite function that computes the covariance matrix between two feature vectors. The kernel function is a crucial component for any successful GP application. It is often the most challenging part since it requires domain knowledge about the phenomenon. In this thesis, we use the Matern kernel function, a generalization of the radial basis function (RBF kernel):

$$\mathbf{K}_{mat}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_{mat}^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\mathbf{d}}{\rho} \right)^{\nu} \mathbf{K}_{\nu} \left(\sqrt{2\nu} \frac{\mathbf{d}}{\rho} \right) \quad (2.36)$$

where \mathbf{d} is the Euclidean distance between \mathbf{x}_i and \mathbf{x}_j , ρ is the length scale, ν is the smoothness parameter, σ_{mat}^2 is the variance, Γ is the gamma function, and \mathbf{K}_{ν} is the modified Bessel function of order ν . The kernel parameters σ_{mat}^2 , ρ , and ν are hyperparameters that we need to tune for each application, either manually or automatically, using the maximum likelihood method. Additionally, we add a periodic kernel function to model the periodicity in the environmental variable.

$$\mathbf{K}_{per}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_{per}^2 \exp \left(-\frac{2 \sin^2 \left(\frac{\pi \mathbf{d}}{p} \right)}{\ell^2} \right) \quad (2.37)$$

where ℓ is the length scale, σ_{per}^2 is the variance, and p is the period of the kernel. Then, the final kernel function is given by:

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{K}_{mat}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{K}_{per}(\mathbf{x}_i, \mathbf{x}_j) \quad (2.38)$$

One limitation of using GP is the computational cost. Learning in GP involves kernel matrix inversion, which has an asymptotic complexity of $O(N^3)$, where N is the number of training examples. This is a major drawback, especially when working with large datasets, such as air quality data. Luckily, there are several approaches we can use to reduce computational costs. For example, we can use sparse online GP to train incrementally without retaining the entire model when new data are observed [51]. Another approach is to use model approximations or approximate GP inference with cheaper complexity, such as Nystrom approximation [52], FITC approximation [53], or variational sparse GP [54].

To show an example of how GP works, we use one year of air quality data from Trondheim to train the GP model. This is equivalent to computing the posterior distribution $p(f|\mathcal{D})$ where \mathcal{D} is the training data. Then, we sample a function from the posterior distribution $f \sim p(f|\mathcal{D})$ to make a prediction. Figure 2.8 shows an example of two function samples from the posterior distribution

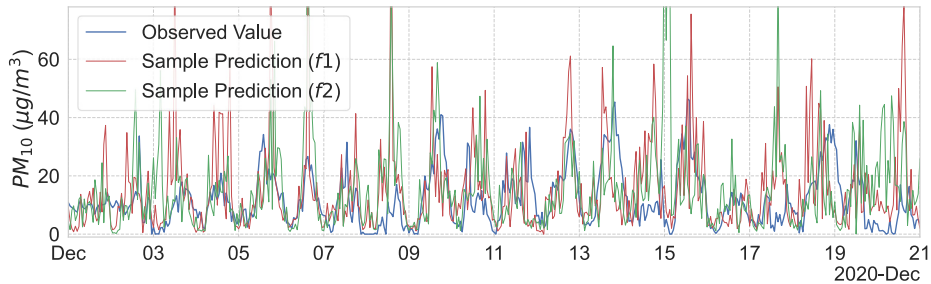


Figure 2.8: A sample prediction from the GP model for forecasting air quality.

A better way to make a prediction is to compute the mean and standard deviation of the posterior distribution $p(f|\mathcal{D})$. Then, we can plot the prediction's mean function and confidence interval. Figure 2.9 shows an example of the mean function and the confidence interval using one standard deviation above and below the mean. The mean function is a smoother curve than the sample function in figure 2.8 because it represents the average prediction of the GP model. The confidence interval is a band around the mean that expresses the uncertainty of the prediction: the wider the interval, the more uncertain the prediction is.

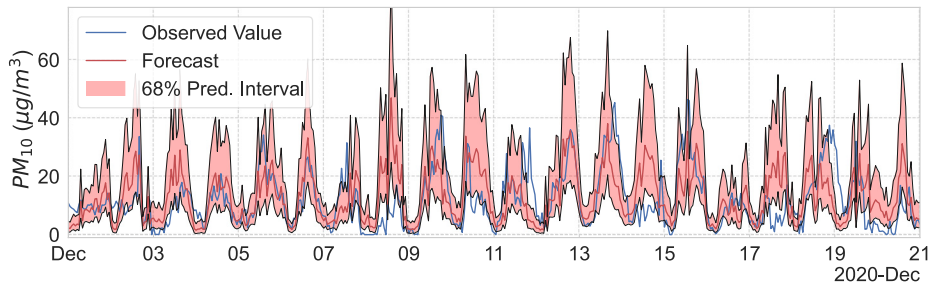


Figure 2.9: The mean function and the confidence interval of the prediction from the GP model for forecasting air quality.

2. Background

Evaluation metrics

The mean prediction of the GP model achieves an RMSE of 9.4, which is comparable to the XGBoost model. However, the RMSE metric only considers the mean value of the prediction. We should therefore consider other metrics that also measure the uncertainty of the prediction. One such metric is the negative log-likelihood (NLL). Since we have a prediction mean $\hat{\mu}_t$, a standard deviation $\hat{\sigma}_t$ and the true value y_t for each prediction point t , the NLL is given by:

$$NLL = - \sum_{t \in T} \log \mathcal{N}(y_t | \hat{\mu}_t, \hat{\sigma}_t^2) = - \sum_{t \in T} \log \frac{1}{\sqrt{2\pi\hat{\sigma}_t^2}} \exp\left(-\frac{(y_t - \hat{\mu}_t)^2}{2\hat{\sigma}_t^2}\right) \quad (2.39)$$

The NLL measures how well the prediction distribution fits the true value, where a lower NLL indicates a better fit. The NLL exponentially punishes overconfident, incorrect predictions by emphasizing tail probabilities. Thus, it is sensitive to the predicted probabilities of the infrequent periods with spikes or anomalously high or low values.

For a given prediction interval, we would like to know the probability that the true value falls within the interval. This is called the prediction interval coverage probability (PICP). We can calculate the PICP using the upper \hat{U}_t and lower \hat{L}_t bounds of the prediction interval:

$$PICP = \frac{1}{T} \sum_{t \in T} \mathbb{1}(\hat{L}_t \leq y_t \leq \hat{U}_t) \quad (2.40)$$

where $\mathbb{1}(\hat{L}_t \leq y_t \leq \hat{U}_t)$ is an indicator function that is equal to 1 if $\hat{L}_t \leq y_t \leq \hat{U}_t$ and 0 otherwise. The higher the PICP, the better the prediction. In contrast, we would like the prediction interval to be as narrow as possible. Thus, we can also consider the average (mean) width of the prediction interval (MPIW) as a metric. The MPIW is given by:

$$MPIW = \frac{1}{T} \sum_{t \in T} (\hat{U}_t - \hat{L}_t) \quad (2.41)$$

The lower the MPIW, the better the prediction.

Another metric that measures the prediction's uncertainty is the continuous ranked probability score (CRPS) [55]. The CRPS generalizes the mean absolute error (MAE) to a probabilistic setting. It measures the accuracy of the prediction distribution by comparing the cumulative distribution function (CDF) of the prediction distribution with the CDF of the true value. Thus, the lower the CRPS, the better the prediction. The CRPS is given by:

$$CRPS = \sum_{t \in T} \int_{\mathbb{R}} (F_{\hat{\mu}_t, \hat{\sigma}_t^2}(z) - F_{y_t}(z))^2 dz \quad (2.42)$$

where $F_{\hat{\mu}_t, \hat{\sigma}_t^2}(z)$ is the CDF of the prediction distribution, and $F_{y_t}(z) = \mathbb{1}(z \geq y_t)$ is the CDF of the true value. We can derive the CRPS analytically for parametric distributions such as the normal distribution, but we can only approximate it for non-parametric distributions using Monte Carlo sampling.

In figure 2.9, we use a prediction interval of 1 standard deviation above and below the mean, corresponding to a 68% prediction interval. We can get other prediction intervals by changing the number of standard deviations. For example, a 98% prediction interval corresponds to 2 standard deviations above and below the mean. Table 2.1 shows an example of a performance summary of the GP prediction when using a 95% prediction interval. We will use these metrics throughout the thesis to evaluate the performance of most prediction models.

Table 2.1: The performance of the GP model for forecasting air quality with 95% prediction interval.

Metric	RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓
GP Prediction	9.46	0.85	42.30	0.55	1.51

2.2.4 Neural networks

We can also use neural networks to build a prediction model. We start with the simplest neural network model, the multi-layer perceptron (MLP). The MLP is a feed-forward neural network with multiple hidden layers. The essence of the MLP is simply matrix multiplication and non-linear activation functions. Each layer has a set of weights that multiply the input and add a bias. The output of each layer is then passed through a non-linear activation function. We input the features \mathbf{x} into the first layer, and the output of the last layer is the prediction \hat{y} .

We need to define a loss function to train the MLP (i.e., find the optimal weights and biases). We can start with the simplest loss function: the mean squared error (MSE) between the prediction \hat{y} and the actual value y :

$$\mathcal{L} = \frac{1}{T} \sum_{t \in T} (\hat{y}_t - y_t)^2 \quad (2.43)$$

Then, we use gradient descent with respect to the weights and biases to minimize the loss function. Figure 2.10 shows an example of predicting air quality with an MLP model. We train the model using one-year data and test it for 20 days. The model achieves an RMSE of 8.58, slightly better than the GP and XGBoost models.

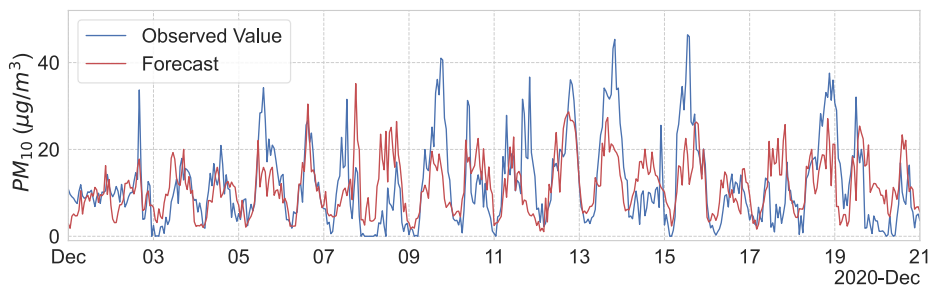


Figure 2.10: The prediction of the MLP model for forecasting air quality.

Aleatoric uncertainty

We can also set the MLP to output the mean $\hat{\mu}_t$, and standard deviation $\hat{\sigma}_t$ of the prediction distribution [56]. We then use the NLL as the loss function:

$$\mathcal{L} = -\frac{1}{T} \sum_{t \in T} \log \mathcal{N}(y_t | \hat{\mu}_t, \hat{\sigma}_t^2) \quad (2.44)$$

We then use the estimated mean and standard deviation to construct the prediction distribution. Figure 2.11 shows an example of predicting air quality. The uncertainty of the prediction is known as aleatoric uncertainty, which is the uncertainty inherent in the data. Aleatoric uncertainty is not due to the model but due to the intrinsic randomness of the data.

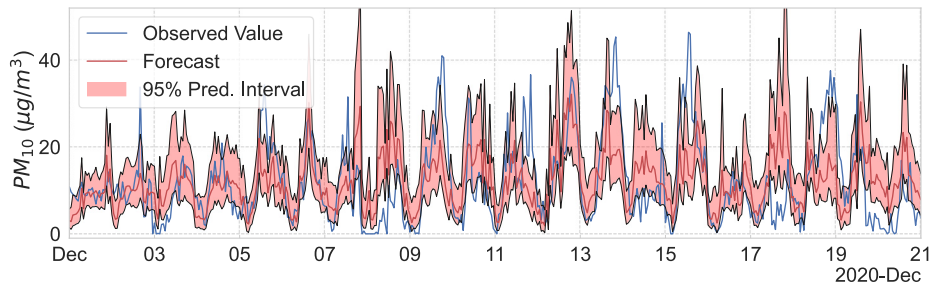


Figure 2.11: Estimated aleatoric uncertainty of the MLP model for forecasting air quality. The prediction interval is estimated using the predicted standard deviation, not the model uncertainty.

Epistemic uncertainty

We would also like to estimate the model or epistemic uncertainty. This uncertainty is due to the model parameters. The epistemic uncertainty manifests in the model's inability to capture the true relationship between the input and the target, especially in regions of the input space with low training data density. We can reduce this type of uncertainty by increasing the amount of training data.

There exist several approaches to quantify the epistemic uncertainty of deep neural networks. Most of these approaches use Bayesian methods to represent the prediction uncertainty with probabilistic modeling. The main idea is to assume a prior distribution over the model parameters and then use the training data to update the prior distribution to a posterior distribution. However, the posterior distribution is intractable for deep neural networks, so we must approximate it. There exists a variety of approximation methods that use either variational inference [57, 58, 59], Markov chain Monte Carlo (MCMC) [60, 61, 62], or Laplace approximation [63, 64]. In this thesis, we will use only those methods that use variational inference. The following sections discuss these methods in more detail.

Bayesian Neural Networks (BNNs)

The most obvious choice for estimating the uncertainty in the model's parameters is to represent the parameters with a probability distribution instead of a point

estimate. This is the idea behind Bayesian neural networks (BNNs) [63]. In BNNs, we represent the weights of the neural network with a probability distribution $p(\mathbf{w})$. We then sample the weights from the distribution to make predictions. We sample multiple times to get a distribution of predictions, which we use to estimate the epistemic uncertainty.

To train the BNN, we need to find a way to approximate the posterior distribution given the training data $p(\mathbf{w}|\mathcal{D})$. We can use variational inference [65, 66] to approximate the posterior with a variational distribution $q(\mathbf{w}|\theta)$, where θ are trainable parameters. Thus, the training objective is to minimize the KL divergence between the posterior and the variational distribution. We can achieve this by maximizing the evidence lower bound (ELBO) [57]:

$$\mathcal{L}(\theta) = KL[q(\mathbf{w}|\theta)||p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D}|\mathbf{w})] \quad (2.45)$$

where $\mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D}|\mathbf{w})]$ is the log-likelihood of the training data. We can calculate the ELBO analytically if we choose to use a Gaussian distribution for prior and variational distribution. Then, we update the variational parameters θ using the gradient descent of the ELBO. Additionally, the distribution over activations will also be a Gaussian distribution. This allows us to use the reparameterization trick [67] to sample the weights from the activation distribution. This way, we reduce the variance of the gradient estimator, which improves the convergence of the training.

To allow for more flexibility and adaptation to a broader range of problems, we can use non-Gaussian distributions, such as a mixture of Gaussians or a mixture of Laplace, to represent the prior and variational distribution. However, this will require us to use Monte Carlo methods to approximate the ELBO since we cannot analytically calculate the ELBO for non-Gaussian distributions [58]:

$$\mathcal{L}(\theta) \approx \sum_{i=1}^M \log q(\mathbf{w}^i|\theta) - \log p(\mathbf{w}^i) - \log p(\mathcal{D}|\mathbf{w}^i) \quad (2.46)$$

where \mathbf{w}^i are samples from the variational distribution $q(\mathbf{w}|\theta)$. The number of samples M is a hyperparameter that needs to be tuned.

Figure 2.12 shows an example of the BNN air quality prediction. The uncertainty of the prediction is the epistemic uncertainty, which is the uncertainty due to the model. This uncertainty is different from the aleatoric uncertainty, which is the uncertainty inherent in the data.

To also estimate the aleatoric uncertainty, we can use the BNN to predict the mean $\hat{\mu}_t$ and standard deviation $\hat{\sigma}_t$ of the target distribution at time t . Then, we sample the weights $\mathbf{w} \sim \mathcal{N}(\mu, \sigma^2)$ multiple time (e.g., M samples). Then, we measure the dispersion in predictions resulting from multiple forward passes using the sampled neural networks. These multiple forward passes result in a (uniformly weighted) mixture of normal distributions with the mean $\hat{\mu}_{t,mix}$ and variance $\hat{\sigma}_{t,mix}^2$, calculated as follows:

$$\hat{\mu}_{t,mix} = \frac{1}{M} \sum_{i=1}^M \hat{\mu}_{t,i} \quad (2.47)$$

$$\hat{\sigma}_{t,mix}^2 = \frac{1}{M} \sum_{i=1}^M (\hat{\sigma}_{t,i}^2 + \hat{\mu}_{t,i}^2) - \hat{\mu}_{t,mix}^2 \quad (2.48)$$

2. Background

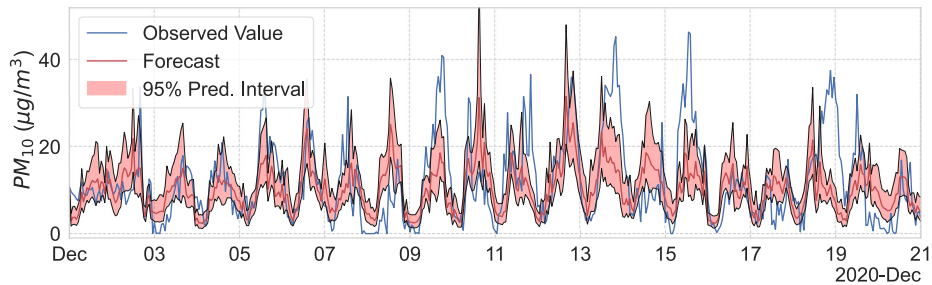


Figure 2.12: Estimated epistemic uncertainty of the BNN model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction distribution resulting from multiple samples of the weights.

In Paper IV and Paper V, we use BNN to predict air quality and indoor noise levels. Figure 2.13 shows an example of the BNN prediction of air quality with estimated aleatoric and epistemic uncertainty.

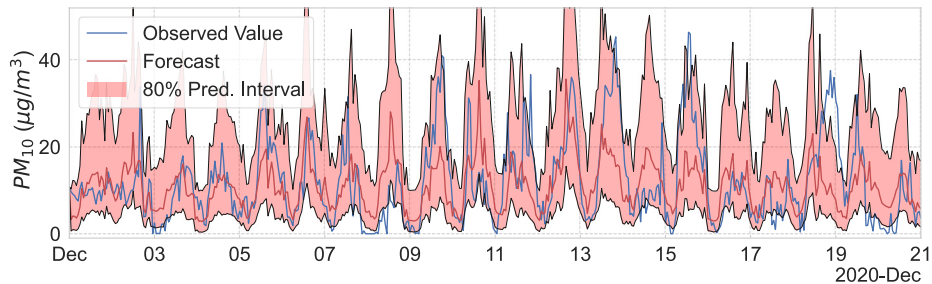


Figure 2.13: Estimated epistemic and aleatoric uncertainty of the BNN model for forecasting air quality. The prediction interval is estimated using the standard deviation of the mixture distribution resulting from the predicted standard deviation of the target distribution and multiple samples of the weights.

One major drawback of BNNs is that they are computationally expensive to train and take longer to converge than standard neural networks. This is especially true for non-Gaussian distributions since we need to use Monte Carlo methods to approximate the ELBO. Additionally, they require twice as many parameters as standard neural networks since we need to estimate the mean and standard deviation of the weights. A possible solution is to gracefully prune the weights with the lowest signal-to-noise ratio [58], which leads to a loss in uncertainty information. Still, BNNs is a popular choice for quantifying uncertainty in deep learning models because they provide a principled way to reason about uncertainty.

Monte Carlo dropout

Due to the computational cost of BNNs, it would be convenient to have a straightforward or approximate method to estimate uncertainty directly from standard neural networks. One such method is Monte Carlo Dropout [68]. This method uses the idea that we can use dropout [69] to estimate the uncertainty of the prediction. Dropout is a regularization technique that prevents neural networks from

overfitting by randomly dropping out (setting to zero) a set of nodes along with their connection during training. We can use this method during inference to estimate uncertainty by running multiple forward passes with different dropout masks. This corresponds to sampling the nodes, equivalent to sampling from ensembles of neural networks [69]. We then estimate the uncertainty of the prediction using the standard deviation of the predictions resulting from multiple forward passes. We can interpret the Monte Carlo Dropout method as performing variational inference. More specifically, it is mathematically equivalent to an approximation of a probabilistic deep Gaussian process [68]. In Paper IV and Paper V, we use Monte Carlo Dropout to estimate the uncertainty of the prediction of air quality, as shown in figure 2.14.

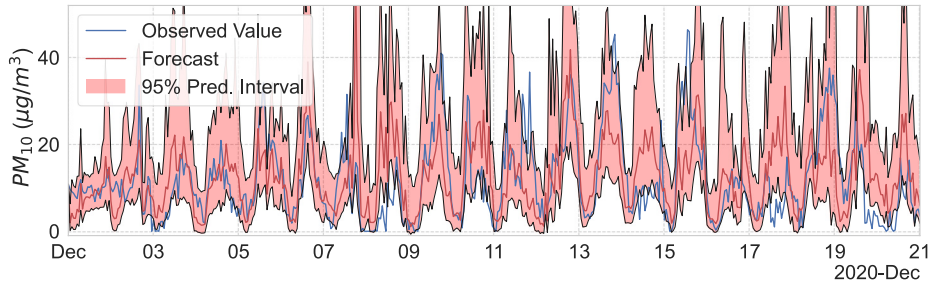


Figure 2.14: Estimated epistemic and aleatoric uncertainty of the MC Dropout model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction resulting from multiple forward passes with different dropout masks.

Deep ensembles

One perspective to sampling BNNs is that we get a different neural network for each weights sample. Since BNNs have distribution over weights, sampling the weights is equivalent to sampling from an infinite ensemble of networks. We can use this insight to train multiple neural networks instead and use deviations in predictions to estimate the epistemic uncertainty. This is the idea behind deep ensembles [70] for estimating uncertainty. Ensembles are already an established method for improving prediction performance through averaging predictions from multiple models [71].

We train multiple neural networks with different random initializations using the same training data during training. The stochasticity in the training process and the random initialization ensure the results are distinct neural networks. Then, we use the average prediction of the neural networks during inference. We estimate the epistemic uncertainty using the standard deviation of the predictions resulting from multiple neural networks. In Paper IV and Paper V, we use the deep ensemble method to estimate uncertainty in the air quality prediction, as shown in figure 2.15.

One drawback of the deep ensemble method is the computational cost of training multiple neural networks. Additionally, it requires more parameters (order of magnitude) compared to standard neural networks. A possible solution is to use

2. Background

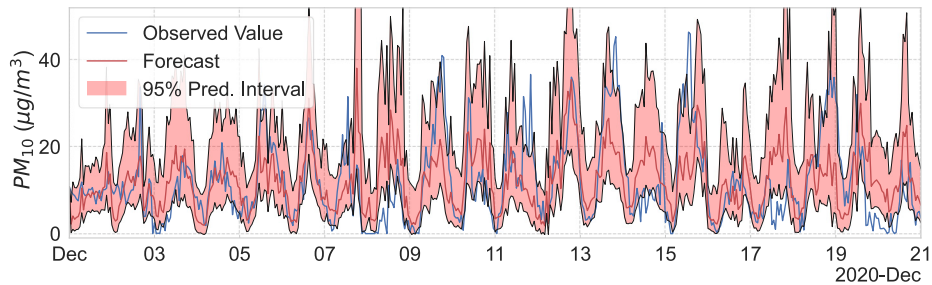


Figure 2.15: Estimated epistemic and aleatoric uncertainty of the deep ensemble model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction resulting from multiple neural networks.

knowledge distillation [72] to compress the knowledge of the ensemble into a single neural network. However, this method is not guaranteed to preserve the information about uncertainty.

Stochastic weight averaging–Gaussian (SWAG)

In BNNs, we approximate the posterior distribution of the weights using variational inference. However, it would be more convenient to approximate the posterior distribution using an easier way. Can we get information about the distribution of the weights from the training process of a standard neural network? This is the idea behind Stochastic Weight Averaging–Gaussian (SWAG) [73].

SWAG is a method for estimating the posterior distribution of the weights of a neural network using the geometric information in the trajectory of a stochastic optimizer. More specifically, SAWG approximates the posterior distribution of the weights by a Gaussian distribution with mean and covariance matrix. We estimate the mean by the running average of the weights along the trajectory of the stochastic optimizer with a modified learning rate schedule (stochastic weight averaging [74]). Additionally, we estimate the covariance matrix by a diagonal matrix plus a low-rank deviation matrix using the information of the second moment of the weights along the trajectory.

We train the neural network during training with stochastic gradient descent (SGD). We sample the Gaussian distribution weights during inference to make predictions. The standard deviation of the predictions resulting from multiple samples of the weights estimate the uncertainty. In Paper IV and Paper V, we use SWAG to estimate uncertainty in air quality prediction, as shown in figure 2.16.

2.2.5 Recurrent neural networks

Although standard neural networks are powerful tools for representation learning, they are not well suited for sequential data. They do not have a memory mechanism to store information from previous time steps. Thus they cannot capture the temporal dependencies in the data. We can use recurrent neural networks (RNNs) to exploit the inherent temporal correlation in time series data. RNNs are neural networks with cyclic (recurrent) connections to store information from previous

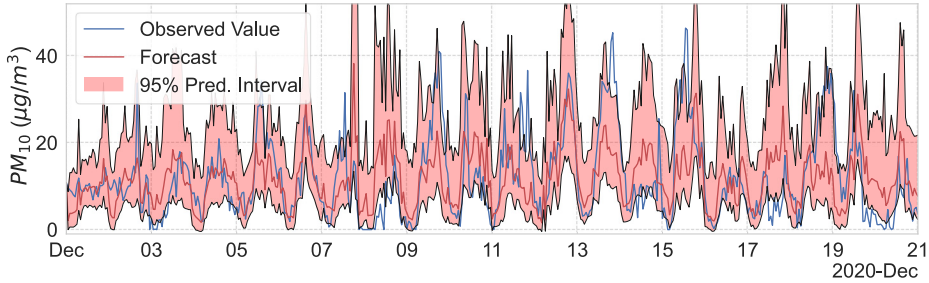


Figure 2.16: Estimated epistemic and aleatoric uncertainty of the SWAG model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction resulting from multiple samples of the weights.

time steps. Another way to look at RNNs: they are neural networks with shared layers across time steps where the output of the last time step is also used as the input of the current time step.

In Paper IV, we use a specific type of RNNs called long short-term memory (LSTM) [75]. LSTMs have gated memory cells that control information flow between time steps. [76]. They can capture long-term dependencies in the data while avoiding the vanishing and exploding gradients problems in standard RNNs [75, 77].

Figure 2.17 shows an example of the LSTM air quality prediction. We estimate the epistemic uncertainty by the standard deviation of the predictions resulting from multiple forward passes with different dropout masks. Additionally, we estimate the aleatoric uncertainty by the predicted standard deviation of the target distribution.

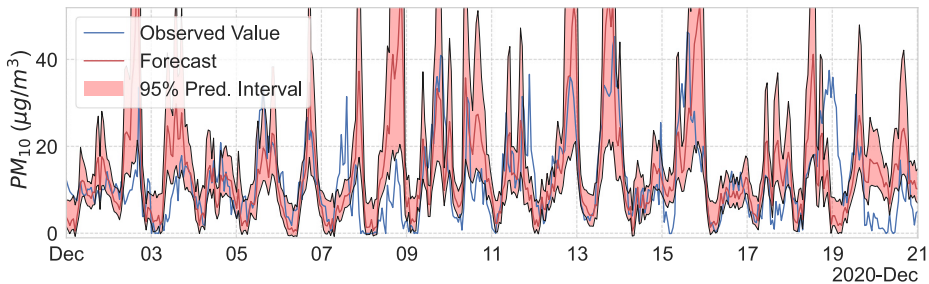


Figure 2.17: Estimated epistemic and aleatoric uncertainty of the LSTM model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction resulting from multiple forward passes with different dropout masks.

2.2.6 Graph neural networks

RNNs are powerful tools for modeling temporal correlations in time series data. However, they are not suited for modeling spatial correlations in spatio-temporal data. To address this shortcoming, we can use graph neural networks (GNNs) to exploit the inherent spatial correlation in spatio-temporal data. GNNs are neural networks that have connections between nodes in a graph. The connections

2. Background

propagate information between nodes. For example, in the air quality data, each node represents a sensor in a specific location, while the connections represent the spatial correlation between the sensors. We can learn the graph structure from the data or use a predefined graph structure.

In Paper IV, we use a specific type of GNNs called Spectral Temporal Graph Neural Networks (StemGNN) [78]. StemGNN is a GNN that uses spectral graph theory to learn the graph structure from the data. StemGNN learns the structural and temporal correlations in the frequency domain. StemGNN learns the structural correlation using graph convolutional network (GCN) [79] while it learns the temporal correlation using gated linear units (GLUs) [80]. Figure 2.18 shows an example of the StemGNN air quality prediction. We estimate the epistemic uncertainty by the standard deviation of the predictions resulting from multiple forward passes with different dropout masks. Additionally, we estimate the aleatoric uncertainty using the predicted standard deviation of the target distribution.

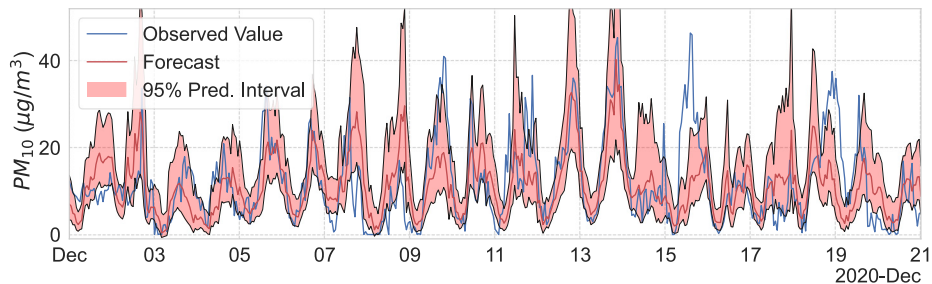


Figure 2.18: Estimated epistemic and aleatoric uncertainty of the StemGNN model for forecasting air quality. The prediction interval is estimated using the standard deviation of the prediction resulting from multiple forward passes with different dropout masks.

2.3 IoT sensing

Internet of Things (IoT) is a broad term that essentially refers to embedding physical objects (things) with sensors and connecting them to the internet. The term is often used interchangeably with other terms, such as cyber-physical systems and ubiquitous computing. However, the essence of all these terms is the same, which is the integration of sensing, computing, and communication technologies to enable the collection, processing, and analysis of data from the physical world for decision-making.

Some IoT nodes can also perform control or actuation tasks. This thesis focuses on IoT nodes that only collect data from the physical world in applications such as environmental monitoring. These applications aim to get information about the environment to help stakeholders make decisions. For example, policymakers can use data from an air quality monitoring system to decide when to close a road or start a street cleaning initiative. The challenge of environmental monitoring is ensuring seamless access to the phenomenon's value with minimal cost and effort.

The main constraints of deploying IoT nodes for environmental monitoring are the limited energy, computation time, and communication bandwidth. With

recent technological advances, computation time and communication bandwidth are becoming cheaper and are in practice only limited by energy resources. For many applications, the act of sensing (i.e., taking a measurement) is the most energy-consuming task. Additionally, the energy consumed for computation and communication is often proportional to the amount of data collected by the sensing module. Therefore, this thesis focuses on the scheduling of sensing actions for IoT nodes.

2.3.1 Design choices for sensing solutions

Sensing actions

At a high level, a sensing policy is a set of rules that determine *when* and *how* the sensor should collect data. We can formulate the sensing problem as a sequential decision problem, where we have a sequence of time steps. At each time step, the policy decides how to schedule the sensing actions. We have several design choices for the sensing actions. For example, the policy can control the duty cycle, sensing rate, or dynamic interval between measurements:

- **Duty cycle:** The duty cycle is the fraction of time that the sensor is active. For example, a sensor with a duty cycle of 50% is active half the time and inactive for the other half. At each time step, the policy decides the duty cycle of the sensor for the next time step. The action space of the policy is the set of all possible duty cycles: $a_t \in \mathcal{A} = [D_{min}, D_{max}]$ where D_{min} and D_{max} are the minimum and maximum duty cycles, respectively. The action space can be continuous or discrete.
- **Sensing rate:** The sensing rate is the number of measurements per unit of time. For example, a sensor with a sensing rate of 1 Hz makes one measurement per second. At each time step, the policy decides the sensing rate of the sensor for the next time step. The action space of the policy is the set of all possible sensing rates: $a_t \in \mathcal{A} = [R_{min}, R_{max}]$ where R_{min} and R_{max} are the minimum and maximum sensing rates, respectively.
- **Dynamic interval:** The measurement interval is the time between two consecutive measurements. For example, at each time step, the policy decides the time interval the sensor should wait or sleep before making the next measurement. There are two possible choices to get a dynamic interval:
 - *One-step look-ahead:* At each time step, the policy decides whether to take a measurement or not: $a_t \in \mathcal{A} = \{0, 1\}$ where 0 means no measurement and 1 means take a measurement.
 - *Multi-step look-ahead:* At each time step, the policy decides when to measure in the future: $a_t \in \mathcal{A} = [0, T]$ where T is the maximum time horizon. For example, if $a_t = 5$, the sensor will measure after 5-time steps.

This thesis uses the duty cycle as the sensing action in Paper I. The duty cycle is a general action that reflects the amount of collected data, which is suitable for

2. Background

general-purpose applications. We use the dynamic interval as the sensing action in Paper III and Paper V. More specifically, we use the multi-step look-ahead dynamic interval. This choice is motivated by the need for fine-grained control of measurement timing in value-based sensing.

Energy resource

We can categorize any sensing solution to either have battery-powered or energy-harvesting sensors. Battery-powered sensors are powered only by fixed-size batteries that have a limited lifetime. In contrast, energy-harvesting sensors are powered by ambient energy sources such as solar, wind, or vibration. Additionally, energy-harvesting sensors can have storage capacitors or rechargeable batteries to store the harvested energy.

For battery-powered sensors, the following are the main design choices we need to consider when designing a sensing solution:

- **Energy dimensioning:** We need to determine the battery size and, thus, the energy budget available for the sensing solution.
- **Energy consumption:** We need to determine the energy consumption of the sensing module when taking a measurement.
- **Recharging or replacement:** We need to determine whether the battery needs to be recharged or replaced periodically.

For energy-harvesting sensors, in contrast, we have the following main design choices:

- **Energy harvesting module:** We need to determine the size and efficiency of the energy harvesting module.
- **Energy storage module:** We need to determine the size and type of the energy storage module. Additionally, we need to determine charge-discharge efficiency and the rate of self-discharge.
- **Energy consumption module:** We need to determine the energy consumption of the sensing module in terms of energy required to take a measurement.
- **Energy prediction module:** We need to determine whether our sensing solutions have access to an energy prediction model and how accurate it is. The energy prediction model predicts the energy available for harvesting in the future.

In our Sensor Gym simulator (Paper II), we simulate energy-harvesting sensors with four modules: energy harvesting, energy storage, energy consumption, and energy prediction module. We use real data from a solar panel for the energy harvesting module and the weather forecast to predict the energy available for harvesting in the future. Additionally, we use simple and ideal models for the energy storage and consumption modules. For example, we do not consider the charge-discharge efficiency of the energy storage module, and we assume that the energy consumption of the sensing module is proportional to the amount of collected

data. This choice is reasonable for our simulator because we are interested in the design of the sensing policy and not the specific details of energy storage and energy consumption. We focus on value-based sensing in Paper III and Paper V. Thus, we simulate battery-powered sensors with fixed energy budgets without recharging or replacement.

2.3.2 Sensing objectives

Overall, the objective of any sensing solution is to maximize the system's utility, given a resource constraint. The utility of a system depends on the value or amount of collected data. The resource constraint depends on the energy budget available for the sensing solution. We can roughly sort the sensing objective into three categories:

- Maximize the *amount* of collected data while maintaining *stable* and *perpetual* operation. This objective is suitable for general-purpose solutions with energy-harvesting sensors. For example, we want to collect as much data as possible from an energy-harvesting sensor while avoiding running out of energy. We can measure the amount of collected data by the sensing rate, duty cycle, or the number of measurements per unit of time. Running out of energy is undesirable since it causes data loss or missing important events.
- Maximize the *value* of collected data while minimizing the energy consumption. This objective is suitable for battery-powered sensors and applications where we can measure the information content of a measurement. For example, we want only to collect data that provide information for decision-making and avoid wasting energy on unnecessary, redundant, or least informative measurements.
- Maximize the *value* of collected data while maintaining perpetual operation. This objective is suitable for energy-harvesting sensors and applications where we can measure the information content of a measurement.

Our work in Paper I focuses on the first objective of maximizing the amount of collected data while maintaining stable and perpetual operation. Our work in Paper III and Paper V focuses on the second objective of maximizing the value of collected data while minimizing energy consumption.

“Internet of Things will become a global system of systems interconnecting people and things in a way which helps everybody in their lives and hopefully makes the plant a little bit smarter”

– Andy Stanford-Clark, 2012

Chapter 3

Related work

This chapter provides a brief overview of the related work in IoT sensing. IoT sensing is a broad area that includes various research topics, such as power management, adaptive sensing, sparse sensing, and data reduction. These topics are closely related to each other and are often used interchangeably. We divide the related work into three main categories: sensor power management, adaptive sensing, and prediction-based sensing. We start each section with the conventional approaches and then discuss the recent advances in using reinforcement learning in these areas.

3.1 Sensor power management

Sensor power management describes the techniques that control the power consumption of sensors. The goal of sensor power management is to reduce the energy consumption of sensors while maintaining the required performance or quality of service. There has been extensive research on sensor power management in the past decades, especially in the context of wireless sensor networks (WSNs) [81, 82, 83].

For battery-powered sensors, the main objective is to minimize energy consumption by reducing the number of measurements while maintaining an acceptable level of service. The main challenge in sensor power management is finding the optimal tradeoff between energy consumption and quality of service [84, 85, 86]. There are many different approaches to sensor power management of battery-powered sensors, such as event-based sensing, adaptive sensing, and prediction-based sensing. We discuss these approaches in later sections.

For energy harvesting sensors, the main objective is to maximize the system’s utility while maintaining perpetual operation. One possible approach to power management of energy harvesting sensors is to control the energy expenditure through duty cycling, which turns the sensor on and off periodically. Energy management of energy-harvesting sensors presents a use case of a dynamic environment. The node needs to decide on its energy allocation in a stochastic environment. Usually, dynamic optimization methods, such as dynamic programming or adaptive control theory, are used to manage the harvested energy [87, 88]. These methods usually have some assumptions or require domain knowledge about the environment, such as the energy profile.

Classic management of energy-harvesting nodes focuses on energy-neutral operation (ENO) [89, 90]. Energy neutrality is concerned with the balance between energy harvesting and energy consumption. It desires to keep the energy harvested by the node equal to the energy consumed over some time. Many energy-management

3. Related work

solutions use energy neutrality as an objective function. The goal is to find a sensing policy that satisfies energy neutrality. This can be challenging because the harvested energy is highly stochastic and difficult to predict.

Reinforcement learning provides an approach to solving energy management problems in a dynamic environment. It does not require any assumptions about the environment, thus reducing manual design effort. Indeed, many related works use reinforcement learning for energy management [8, 9, 10, 13, 91, 92, 93]. The main objective of these works is to achieve energy neutrality by encoding the reward function to contain some energy-related terms. For example, the reward can be a function of the distance from the energy-neutral point [13]. The closer the node is to the energy-neutral point, the higher the reward. Other works also include quality of service as part of the reward function. For example, the reward can be high if the device meets the user demand while retaining long-term energy neutrality [94, 95, 96, 97].

Encoding the reward function to contain energy-related terms might seem like a suitable reward function, but it can lead to undesired behaviors. One such undesired behavior is the high variance in the node's operation. The agent is trying to achieve an energy-neutral operation at all costs. It will increase the energy consumption to the maximum or shut down the node if there is a slight deviation from the energy-neutral point. This is undesirable in many applications. For example, in event-detection applications, the duty cycle should be stable over time to ensure that the node can detect events. If a node has a 100% duty cycle for one hour and then 0% duty cycle for the next hour, the node will not be able to detect events compared to a node with a 50% duty cycle for two hours, even though the total energy consumption is the same. Some works address this issue by modifying the reward to have a smoother slope around the energy-neutral point [13]. However, this approach involves domain-specific assumptions and is not general enough to be applied to other applications. It also requires a lot of manual design effort, and finding the optimal slope is difficult. Another issue with energy-based reward functions is that they optimize the wrong objective. The energy is a resource to be managed, not a goal to be optimized [98]. Energy management aims to maximize the system's utility; thus, the agent should optimize the utility function.

In this thesis, we address the above issues using a utility-based reward function in Paper I. We propose a reward function that optimizes three conflicting objectives: energy utilization, variance reduction, and avoiding node shutdown. We can control the tradeoff between the three conflicting objectives using tunable parameters in the reward function. Our work in Paper I is the first to use deep reinforcement learning for autonomous management of energy harvesting sensors. Specifically, we use a neural network to represent the management policy and train it using Proximal Policy Optimization (PPO) [36]. Before our work, most of the related works used tabular methods, such as SARSA or Q-learning, that require manual state-space discretization. After we proved the feasibility of using deep reinforcement learning for sensor power management, many recent works follow our approach and use deep reinforcement learning for energy management [99, 100, 101]

3.2 Adaptive sensing

Similar to sensor power management, adaptive sensing is concerned with reducing the energy consumption of sensors while maintaining an acceptable level of service. However, adaptive sensing is more concerned with the quality or value of information in the measurements. The goal of adaptive sensing is to reduce the amount of unnecessary, redundant, or least informative measurements. Thus, power management answers the question of how much power we can (or should) use, while adaptive sensing answers the question of how valuable the measurements are. Still, sensor power management and adaptive sensing are closely related terms and are often used interchangeably.

We can define adaptive sensing as the process of selecting *when*, *where*, or *what* to sense, subject to some resource constraints. This corresponds to monitoring temporally-varying, spatially-varying, or multiple data types-varying phenomena. The resource constraints can be energy consumption, bandwidth, or storage capacity. This thesis focuses only on temporally-varying phenomena with energy consumption as the resource constraint.

A straightforward approach to adaptive sensing is to use variance-based methods [102, 103, 104, 105]. The basic idea of variance-based adaptive sensing is to adjust the sensing rate according to changes observed in the environment. For example, the sensing rate can increase when the environment changes rapidly and decrease when the environment is stable. Another naïve approach is to use threshold-based adaptive sensing. The sensing rate increases when the measured value is above a certain threshold and decreases when the measured value is below a certain threshold [106, 107]. Variance-based and threshold-based adaptive sensing methods are simple, easy to implement, and suitable for some applications such as event detection. However, they require domain-specific knowledge and thus imply design effort. Other approaches use more complex techniques to estimate the state of the environment and adjust the sensing rate accordingly. These techniques include, but are not limited to, Kalman Filter-based methods [108, 109], Bayesian-based methods [110, 111], Fisher information-based methods [112, 113] and Fuzzy logic-based methods [114]. These methods are more complex and require more design effort.

In contrast, our work aims to avoid or reduce the need for domain-specific knowledge and design effort. We use reinforcement learning to learn the adaptive sensing policy from interaction with the environment. This learning-based approach is generic and applicable to a wide range of applications. Recent works have used reinforcement learning for adaptive sensing [115, 116]. However, these works use tabular methods, such as Q-learning, that require manual state-space discretization. They also use an energy-based reward function that does not optimize the system's goal. Our work in Paper III uses an information-based reward function that encourages the agent to select the most informative measurements. We also use deep reinforcement learning to represent the policy and train it using the PPO algorithm. After we published our work, other recent works followed our approach and used deep reinforcement learning for adaptive sensing [117].

3.2.1 Compressive sensing

Our goal in taking measurements is to get information about a phenomenon. Ideally, we would like to have a complete description of the phenomenon. We can achieve this by measuring at Nyquist rate [118]. The Nyquist rate implies sampling at twice the highest frequency in the signal. For example, in indoor noise monitoring, the Nyquist rate is 20 kHz. This means we must take at least 40,000 measurements per second to reconstruct the signal without aliasing. This approach is impractical. We cannot afford to take 40,000 measurements per second. Even in other applications, real-world phenomena are often changing fast. We need to find a way to get information about the phenomenon with a limited number of measurements. Our goal is to find a way to reduce the number of measurements while preserving the information content of the signal.

One way is to use the theory of compressive sensing [119]. Compressive sensing allows us to reconstruct a signal from a small number of measurements, far fewer than the Nyquist rate. The key idea is to exploit the sparsity of the signal [120], which is the notion that the signal is composed of only a small number of non-zero components in a certain projection space.

Although there is a large body of work on compressive sensing, we will not use it in this thesis. The reason is that compressive sensing requires domain knowledge of the signal. Additionally, it requires a high computational cost to reconstruct the signal. Thus, compressive sensing is not the focus of this thesis. We want to automate the process of deciding when to make measurements and remove most of the human efforts.

Furthermore, the goal is to provide enough information to make decisions rather than reconstruct the signal. In most IoT applications, we do not need to reproduce the exact value of the phenomenon. This is why Nyquist rate and compressive sensing are correct but distractions for designing an adaptive sensing system.

3.3 Prediction-based sensing

In many applications, we represent a phenomenon with an estimation or prediction model. The prediction model is constructed and potentially adjusted over time by measurements done by IoT sensing devices. This opens the possibility for prediction-based sensing, where the prediction model guides the sensing devices to make measurements most informative to the representation model. The goal of prediction-based sensing is to select the most informative measurements that improve the quality of the representation model while minimizing energy consumption.

In the literature, many works in wireless sensor networks use prediction-based sensing. They usually call it prediction-based data reduction [5], where they incorporate predictive models for inferring measurement values while infrequently correcting the model using sparse measurements. The prediction models include, but are not limited to, Gaussian Processes [121, 113, 122, 123], Holt's Method [124], Grey Models [125], and LSTMs [126]. Others use dual prediction schemes [127] using naive models such as persistence forecasting or statistical properties of the phenomenon such as fixed-weight moving average, least-mean-square, or

auto-regressive integrated moving average [128, 129, 130, 131]. Most of these works manually design heuristics or algorithms to select the sparse measurements.

In contrast, our work in Paper III and Paper V uses deep reinforcement learning to learn the sensing policy guided by the prediction model. We build an uncertainty-aware model of the phenomenon using probabilistic deep learning (Paper IV) and use the model to guide the agent in selecting the most informative measurements. After our work, recent works have also used deep reinforcement learning for prediction-based sensing [132]. However, they do not quantify the prediction uncertainty, which is essential for decision-making.

3.4 Reward function design

A successful application of reinforcement learning relies heavily on the design of a suitable reward function. This function needs to frame the application's goal and guide the learning towards a desirable behavior. Designing a suitable and learning-friendly reward function is a challenging task. It requires domain-specific knowledge and design effort. In this section, we discuss the challenges of reward function design, some common reward functions, and how we can use data to guide the learning without a reward function.

3.4.1 Reward shaping

In many practical applications, it can be difficult or intractable to state the application's goal in a single reward function. In this case, we can use reward shaping [133] to decompose the application's goal into multiple sub-goals or approximate goals and then reward the agent for achieving each individual sub-goal. Unless the reward shaping is carefully designed, the agent may not learn the desired behavior [133].

Almost all previous works on adaptive sensing with reinforcement learning use reward shaping. They manually design a shaped reward function that produces arguably acceptable results without justifying how well the reward frames the application goal [8, 9, 10, 13, 91, 92, 93]. Some other works investigate the effect of different reward functions on the learning performance of sensor power management [134]. However, they only consider a few reward functions and do not provide a general guideline for designing a suitable reward function.

3.4.2 Information-based rewards

We can summarize our sensing goal as *selecting the most informative measurements while minimizing energy consumption*. The main challenge is how to quantify the information content of a measurement. It is impossible to know the true information content before taking the measurement. Otherwise, we would not need to measure in the first place. However, we can quantify the information content of a measurement based on how much our belief about the phenomenon changes after taking the measurement. For example, suppose we represent our prior belief about the phenomenon with a probabilistic model. In that case, we can quantify the information content by measuring the model's distribution change after taking the

measurement. This corresponds to the reduction in the uncertainty or prediction error of the model. We can use this reduction as a proxy for the information content of the measurement.

One approach to measuring uncertainty reduction is using the Fisher Information [135]. Some related works manually designed adaptive sensing heuristics using the Fisher Information with assumptions about the smoothness of the phenomenon [113]. Other works use the Kullback-Leibler divergence [136] between prior and posterior distributions to quantify the information content [137, 138]. However, these works do not use reinforcement learning to learn the sensing policy. They use heuristic algorithms to select the most informative measurements.

In contrast, our work in Paper III uses the Fisher Information as a reward function for the autonomous sensing agent. Additionally, our work in Paper V discusses the challenges of using the Kullback-Leibler divergence as a reward function and proposes a new reward function based on measured peaks-over-threshold.

3.4.3 Data-driven guidance

Although reinforcement learning enables autonomous learning of the sensing policy, it requires an external reward function to guide the learning. Manually designing such a reward function involves domain-specific knowledge that can hinder autonomy and the adoption of reinforcement learning. On the other hand, we already have a large amount of data from expert-designed sensing policies. Can we use this data to reverse-engineer the reward function? This is the idea behind inverse reinforcement learning [139, 140, 141, 142]. To our knowledge, no work in the literature uses inverse reinforcement learning to reverse engineer the reward function for adaptive sensing. Reverse engineering the reward function from data can be a challenging, if not impossible, task.

Luckily, we do not need to reverse engineer the reward function. We can use the data to guide learning without a reward function. For example, we can use imitation learning to learn a policy from demonstration data [143]. Some prior works use imitation learning to learn a sensor power management policy from oracle policies [144]. They use dynamic optimization methods to generate the oracle policies offline and then train an online policy using imitation learning.

Another way to use data is to pre-train the agent without a reward function. This way, the agent can learn a representation of the environment to prepare for future tasks maximally. For example, we can use empowerment as an intrinsic motivation to explore and acquire abilities [145, 146, 41, 147, 148]. However, most emergent behaviors are not useful because of *under-constrained* skill discovery in complex and high dimensional state space. One possible solution is to bias skill discovery towards the desired behavior using data from expert-designed policies. This approach is the idea behind our work in Paper VI. However, more work is needed to understand the effect of data-driven guidance on the resulting behavior.

“Software 2.0 will become increasingly prevalent in any domain where repeated evaluation is possible and cheap, and where the algorithm itself is difficult to design explicitly”

– Andrej Karpathy, 2017

Chapter 4

Research results

This chapter describes the main research results done throughout this Ph.D. project. We organize the results into three sections, where each section presents the contributions towards the answer to one of the research questions set down in section 1.1.

4.1 RQ1: Feasibility of autonomous sensing

The first research question asks about the feasibility of using deep reinforcement learning to learn sensing policies autonomously. We address this question in Paper I, Paper II, Paper III and Paper V. To answer this question, we first formulate the sensing problem as a deep reinforcement learning task. In this formulation, there is an agent with a policy that controls the sensing of an IoT node. The agent interacts with the environment through actions and receives observations and a reward signal. We model the agent-environment interaction as a Markov Decision Process and define the state and action spaces depending on the specific use case.

4.1.1 Sensor Gym: a simulator for learning sensing policies

The first challenge we address is computational complexity. Deep reinforcement learning requires many interactions with the environment to learn a good policy. This complexity is unfeasible in IoT applications, as the sensors are typically resource-constrained and have limited energy and computation capabilities to afford such a large number of interactions. Therefore, we build a framework to learn sensing policies before deploying them to the sensors. We call this framework IoT sensor Gym. Paper II details the design of the sensor gym as a general-purpose simulator to learn sensing policies with deep reinforcement learning. The simulator uses the OpenAI Gym interface, which can be extended to new applications and environments. It comprises various modules that can be combined and configured to match a specific use case.

Figure 4.1 illustrates the architecture of the IoT Sensor Gym framework and the process of training and deploying sensing policies. In this specific figure, we use energy management of energy-harvesting nodes as a use case. The main modules in this use case are energy harvesting, energy storage, energy consumption, and energy prediction. We use actual data from a solar panel for the energy harvesting module and the weather forecast to predict future energy available for harvesting. Additionally, we use simple and ideal models for the energy storage and consumption modules. For example, we do not consider the charge-discharge efficiency of the

4. Research results

energy storage module, and we assume that the energy consumption of the sensing module is proportional to the amount of collected data. This choice is reasonable for our simulator because we are interested in the design of the sensing policy and not the specific details of energy storage and energy consumption.

The architecture in figure 4.1 also illustrates the interaction between the Sensor Gym and the agent. This interaction starts with creating a simulated environment and initializing it with the default values. Additionally, the architecture defines the environment’s boundaries, such as allowed actions (action space), the environment’s possible states (state space), sampling, and returning the first state. Then, the interaction proceeds by running one epoch of the environment’s dynamics after receiving an agent’s action. The *training* of agents occurs off-board on a server, for instance as a part of the IoT device management [149]. The trained agents are then deployed to devices in the field and updated regularly.

We use neural networks to approximate the agent’s policies. In particular, we use neural networks that require acceptable computational effort and memory footprint so they can also be deployed in modern, energy-efficient sensor devices. After deployment, we can use data observed by the sensor to update the simulator modules. This update process requires corresponding instrumentation, for instance, by measuring its energy intake and consumption, indicated by the feedback in figure 4.1.

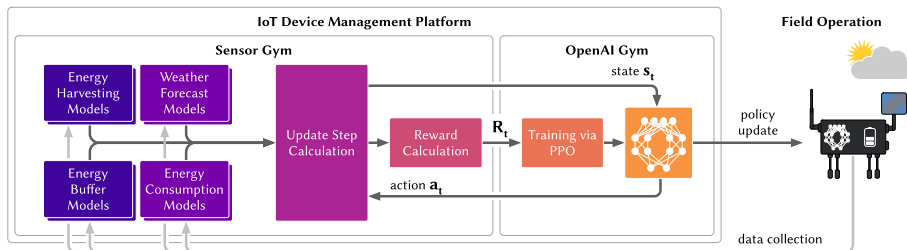


Figure 4.1: Architecture of the IoT Sensor Gym as a framework for training deep reinforcement learning policies to control the sensing of sensor devices autonomously. (Adapted from figure II.8 in Paper II)

4.1.2 Use cases: feasibility of autonomous sensing

In the following, we present the results of the use cases we explored in Paper I, Paper III, and Paper V. We adapt the IoT Sensor Gym to each use case. Then, we use it to train agents that autonomously control the sensing of sensor devices. We will discuss the choice of the reward function in the next section. For now, we want to show, as a proof of concept, the feasibility of using deep reinforcement learning to autonomously learn sensing policies for different use cases.

Energy-harvesting sensors

In Paper I, we explore the feasibility of using deep reinforcement learning to autonomously manage energy-harvesting sensors. This is a general-purpose

application where we want the agent to control the duty cycle of the sensor to maximize the amount of collected data while maintaining stable and perpetual operation. We encode the above three conflicting objectives in a single reward function with tunable hyperparameters to control the trade-off between the objectives. Then, we train several agents using different hyperparameters to explore the trade-offs between the objectives. An agent controls the duty cycle of the node. We use a neural network to represent the policy, which allows the agent to set continuous duty cycles. This enables more accurate control of the consumed energy and, therefore, the utility of a node. We define the agent’s state space as the current energy level, harvested energy, future energy prediction, and the previous duty cycle. As an example, figure 4.2 shows the performance of two agents over six days.

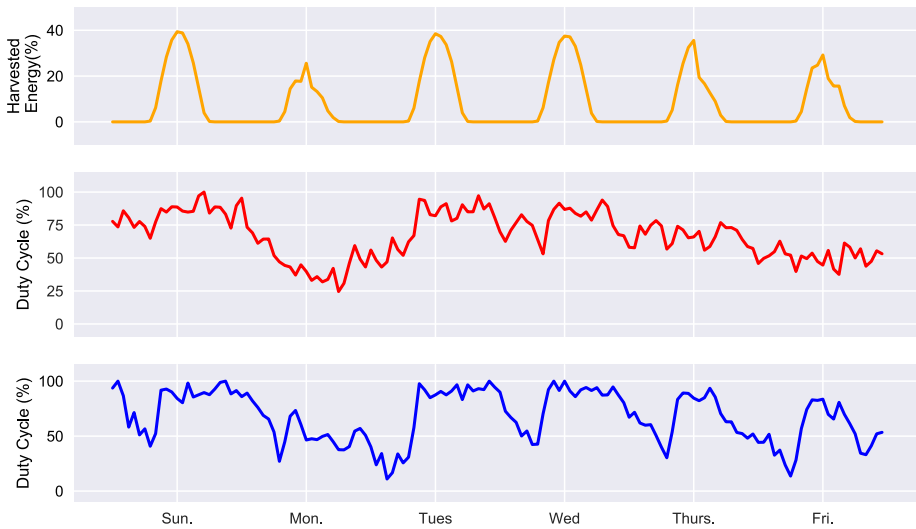


Figure 4.2: Example performance of two agents trained with different hyperparameters that weight the variance penalty in the reward function. The top row shows the solar energy intake. The middle row shows the duty cycle of an agent trained with a reward function that penalizes variance more than the agent in the bottom row. The middle agent has a smoother operation but utilizes slightly less energy. (Adapted from figure II.10 in Paper II)

Indoor noise monitoring

In Paper III, we explore the feasibility of using deep reinforcement learning to autonomously learn sensing policies for indoor noise monitoring. This use case represents an application with battery-powered sensors that aims to maximize the *value* of collected data while minimizing energy consumption. In many applications, we want to collect valuable data for the application goal while reducing energy consumption to maintain operation over the long term. We assume that we represent the phenomenon with an estimation or prediction model informed by measurements

4. Research results

done by the sensing device. Then, we define the value of a measurement by how much it improves the representation model.

Figure 4.3 shows an example of a noise level prediction model fitted with measurements selected using a deep reinforcement learning agent. The agent’s goal is to schedule measurements contributing to the overall model quality. The sensing actions in this use case set a dynamic interval with multi-step look-ahead to schedule measurements. The state space consists of the future prediction, the current energy level, and other external information.

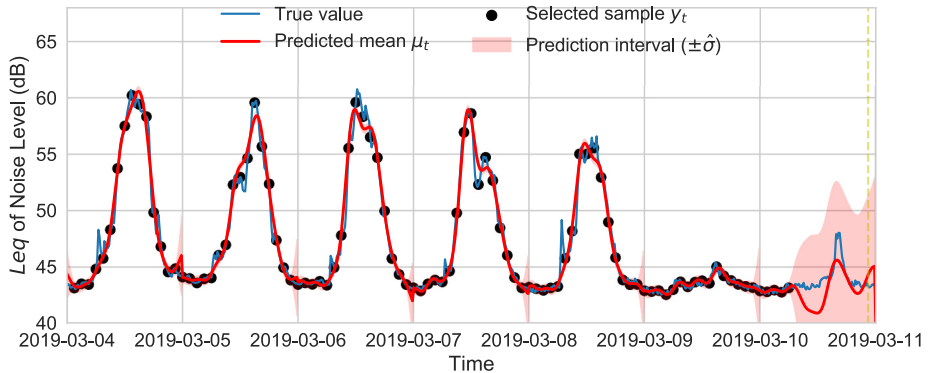


Figure 4.3: An example of a Gaussian Process prediction model of the noise level over one week. The model is fitted with measurements selected using a deep reinforcement learning agent. In contrast to uniform sensing, the learned sensing policy achieves a better information gain and a lower RMSE. (Adapted from figure III.17 in Paper III)

Figure 4.4 shows another example of a prediction model with measurements selected using a deep reinforcement learning agent. The agent’s goal, in this case, is to schedule measurements that detect and dense-sense periods of interest when the noise level is high. The learned policy also produces a better representation model than the uniform policy. The reason is that a well-trained model is good at modeling the phenomenon at regular (usual) periods but makes more mistakes when modeling critical (unusual) periods with spikes or anomalously high or low values. Therefore, taking more measurements during these periods of interest leads to a more refined representation model.

Air quality monitoring

In Paper V, we also explore the feasibility of using deep reinforcement learning to autonomously learn sensing policies for air quality monitoring. The goal is also to capture periods of interest and represent the phenomenon with a credible representation model. Figure 4.5 shows an example of an air quality prediction model fitted with measurements selected using a deep reinforcement learning agent.

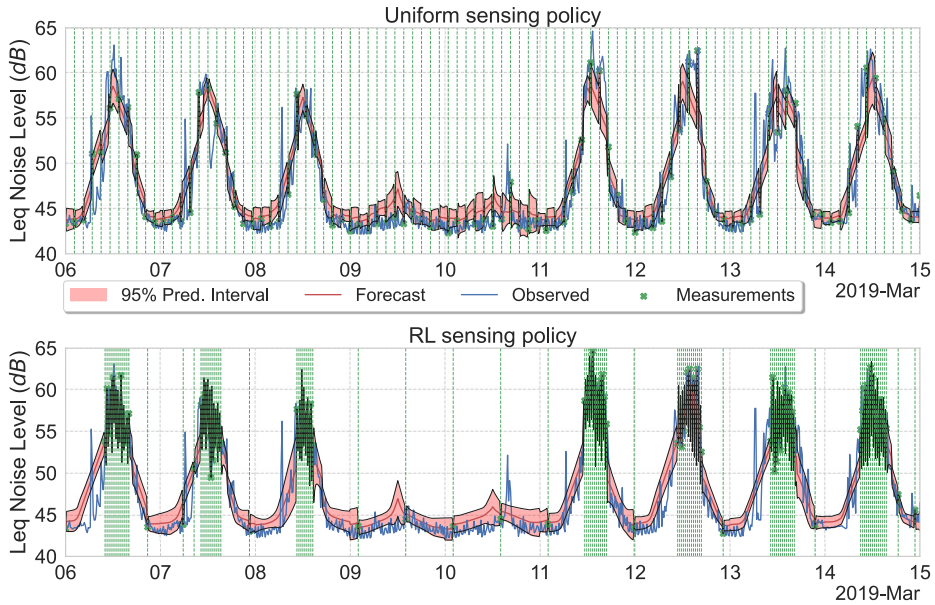


Figure 4.4: An example of a Bayesian neural network prediction model of the noise level over nine days. The top model is fitted with measurements selected uniformly, while the bottom model is fitted with measurements selected using a deep reinforcement learning agent. The dotted, green vertical lines show the measurements’ time index. The learned policy detects and densely senses the noise during periods when the level is high, while the uniform policy spreads its measurements uniformly. (Adapted from figure V.62 in Paper V)

4.1.3 Summary

In summary, the above use cases and examples show that we can use deep reinforcement learning to learn policies that autonomously control the sensing of sensor devices. The sensing actions vary between the use cases. In energy-harvesting use cases, the sensing actions set the operational duty cycle of the node. The duty cycle is a general action that reflects the amount of collected data, which is suitable for general-purpose applications. In contrast, we use a dynamic interval with multi-step look-ahead to schedule measurements in indoor noise and air quality monitoring cases. This design choice enables more fine-grained control of measurement timing. Thus, it is suitable for value-based sensing applications. Finally, the results show that we achieve the desired behavior by choosing a proper reward function. The results also show that we can make trade-offs between conflicting objectives and set the trade-off parameters more suitable for the application.

4.2 RQ2: Reward function design

The second research question asks how to define reward functions that support application goals and lead to learning in autonomous sensing. We address this

4. Research results



Figure 4.5: An example of a Bayesian neural network prediction model of the air quality over one month. The top model is fitted with measurements selected uniformly, while the bottom model is fitted with measurements selected using a deep reinforcement learning agent. The dotted, green vertical lines show the measurements’ time index. The learned policy detects and densely senses the air quality during periods when it is high, while the uniform policy spreads its measurements uniformly. (Adapted from figure V.61 in Paper V)

question in Paper I, Paper III, and Paper V. In this section, we summarize the results of these papers regarding reward function design.

Before designing a goal-oriented reward function, we need to identify the goals of the application. Overall, any sensing solution aims to maximize the system’s utility. The systems’ utility is supporting decision-making by providing valuable information to stakeholders. Thus, we can set the objective of any solution to maximize the value of collected data. However, the definition of value and, more generally, utility depends on the context of the application. In the following subsections, we discuss the different reward functions that we derive from the utility of sensing systems.

4.2.1 Utility-based reward functions

In Paper I, we define the utility of sensing systems as maximizing the *amount* of collected data while maintaining *stable* and *perpetual* operation of sensor devices. This definition is reasonable for applications where the value is proportional to the amount of collected data. In many applications, the more data we collect, the more accurate information we can provide to stakeholders. This definition of utility is especially true in applications with energy-harvesting sensors:

- Maximizing the *amount* of collected data: we want to collect as much data as possible from an energy-harvesting sensor. We can measure the amount of collected data by the sensing rate, duty cycle, or the number of measurements per unit of time.
- Maintaining *stable* operation: the amount of data is not the only factor that matters, but also how the measurements are spread over time. Sometimes, much data with high variance can be worse than fewer evenly spread measurements. Thus, stable operation means there should be less variance in the duty cycle over time. For example, in event-detection applications, the duty cycle should be smooth over time to ensure that the node can detect events. If a node has a 100% duty cycle for one hour and then 0% duty cycle for the next hour, the node will not be able to detect events compared to a node with a 50% duty cycle for two hours, even though the total energy consumption is the same.
- Maintaining *perpetual* operation: the node should run perpetually to ensure that it does not run out of energy. Running out of energy is undesirable since it causes data loss or missing important events.

We summarize the above desirable properties into a single reward function:

$$R_A(s_t) = \begin{cases} D_t - \zeta[\text{Var}_t]^2 & \text{if } B_t > 0 \\ -F & \text{if } B_t = 0 \end{cases} \quad (4.1)$$

where D_t is the duty cycle, ζ is a hyperparameter, Var_t is the variance of the duty cycle, and F is a penalty for running out of energy. We use the duty cycle as a proxy for the amount of collected data, which is a reasonable choice for general-purpose applications. We can tune ζ and F to control the trade-off between the amount of collected data, stability, and perpetual operation. For example, we can set ζ to a low value to collect as much data as possible or to a high value to ensure that measurements are equally spread over time. We can also set F to a high value to avoid running out of energy. Finally, we can set ζ and F to zero to disable the corresponding objectives of *stable* and *perpetual* operations.

Figure 4.6 shows the performance summary of more than 300 different agents over a whole year. These agents are trained using the reward function in I.14 but with different hyperparameters. For deployment, we can choose the agent with the trade-off performance that best suits the context of the application. For example, we can choose an agent with a low ζ to collect as much data as possible, or we can choose an agent with a high F to be more conservative with energy consumption and avoid running out of energy.

4.2.2 Information-based reward functions

Another way to define the utility of sensing systems is to maximize the *information* provided to stakeholders. We want our sensors to provide more information about the phenomenon of interest while using less energy. Thus, we can set the objective of the sensing solution as selecting the most *informative* measurements given a limited energy budget. The challenge is how to quantify the information content.

4. Research results

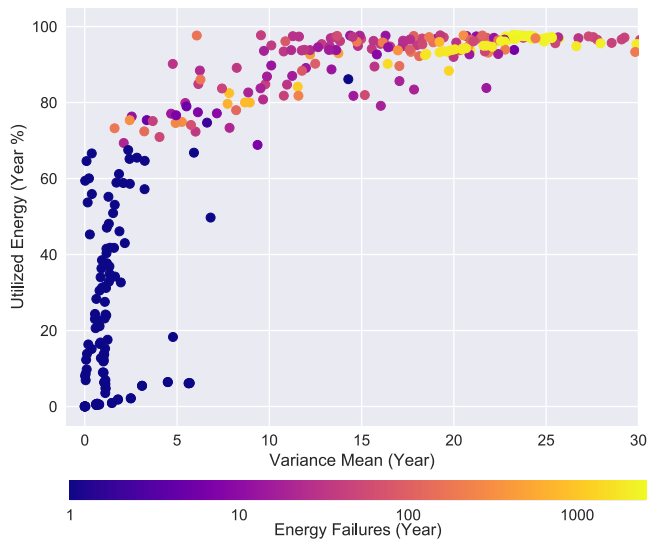


Figure 4.6: Performance results of over 300 agents. The x-axis corresponds to the mean variance of the duty cycle, and the y-axis to the yearly utilized energy. Each dot represents an agent, and the color indicates the number of times an agent has emptied its energy buffer, i.e., failed. (Adapted from figure II.9 in Paper II)

A simple approach is to measure the change in our belief or understanding about the phenomenon after taking the measurement. This change can be the amount of reduction in the uncertainty or prediction error of a representation model about the phenomenon.

Fisher Information

In Paper III, we use Fisher Information as a proxy for the information content of collected data and use it as a reward function. Fisher Information measures the reduction in the uncertainty of a representation model about the phenomenon. For example, figure 4.7 shows two different Gaussian Process predictive distributions of noise levels over one day. The upper model has less confident predictions and, thus, less Fisher Information than the lower model.

Using the Fisher Information as a reward function is reasonable since we are basing the reward function on the quality of the resulting model. In many applications, the goal is to represent a phenomenon with a credible representation model informed by the selected measurements. In that case, there is a good alignment between the application goal and the reward function. That means the reward function is optimizing the agent to select measurements that will contribute optimally to the quality of the resulting model. Practically, we can use the Fisher Information as a reward function by using the following equation:

$$R(\mathbf{s}_T) = \frac{1}{T} \sum_T \frac{1}{\sigma_t^2} \quad (4.2)$$

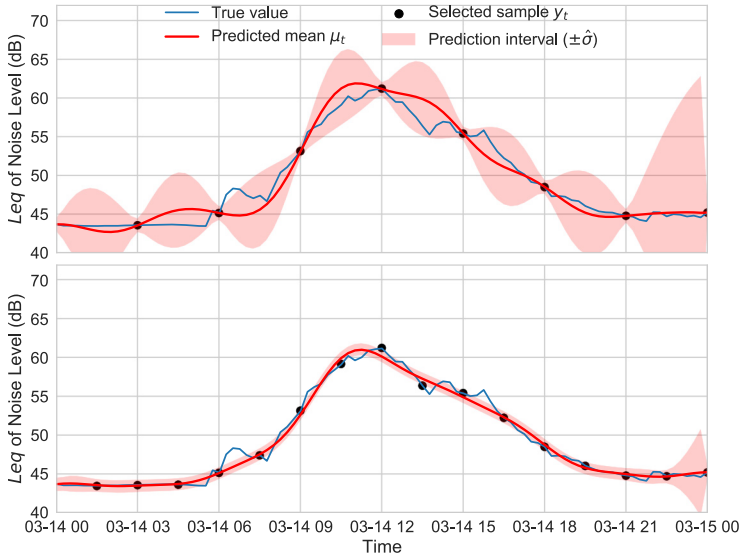


Figure 4.7: Two different Gaussian Process predictive distributions of noise levels over one day. (Adapted from figure III.13 in Paper III)

where $\frac{1}{\sigma_t^2}$ is the prediction confidence at point t . This function means we reward the agent after a period of T , a day in our case, based on the quality of the representation model. Observations selected by the agent fit this model, and the quality of the model is abstracted by the mean Fisher information over T .

The Fisher information does not depend on unobserved samples compared to the RMSE and decodes the application goal into a scalar value that is easy to learn. Hence, it eliminates the need for reward shaping and applies to large, scalable applications without requiring system-specific knowledge. Under some assumption about the smoothness of the monitored phenomenon, we show in Paper III that this technique performs favorably compared to uniform sensing methods.

The downside of the Fisher information reward is the assumption that the model is correct when its confidence is high. This assumption is reasonable if the phenomenon is smooth but does not hold in general. For example, in a phenomenon with high spikes or outliers, the model may be confident about the mean but wrong about the outliers. In that case, the Fisher information will be high even though the model is inaccurate. To use the Fisher Information as a reward function, we should ensure that the model is confident only when it is accurate. This *reliability* of confidence estimate is an active research topic in probabilistic machine learning. We discuss this topic in Paper IV, and we briefly summarize it here.

Generally, we want to avoid over-confident, incorrect predictions. Therefore, we evaluate the reliability of a confidence estimate when selecting a probabilistic prediction model. One approach to evaluate reliability is to measure the amount of loss (or the number of incorrect predictions) a model makes when its confidence is above a certain threshold. In detail, we plot the number of inaccurate predictions

4. Research results

the model makes when its confidence is above τ . We expect the curve to decrease monotonically for a reliable confidence estimate since a rational model has fewer incorrect predictions at high confidence. Additionally, we plot the total number of predictions as a function of τ . This curve is monotonically decreasing, but the decreasing rate indicates the amount of confidence in a model. The decreasing rate would be high in a model with lower confidence since it makes fewer predictions with high confidence. Figure 4.8 show plots of loss vs. confidence and count vs. confidence in air quality prediction using different probabilistic models. We observe that a Bayesian neural network (BNN) model is more reliable. Still, most of the selected models produce rational behaviors.

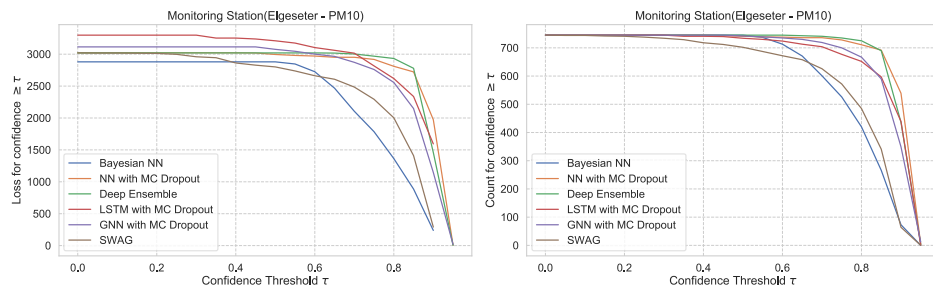


Figure 4.8: An example for evaluating the reliability of confidence estimates for different probabilistic models in air quality prediction task. **Left:** loss versus confidence. **Right:** count versus confidence. (Adapted from figure IV.45 in Paper IV)

Kullback-Leibler Divergence

Ideally, we want to use an information-theoretic metric to quantify the information content. We take what we know before measuring (e.g., entropy before taking a measurement) and subtract what we know after measuring. The difference is the information content. In the same vein, we can use the relative entropy between a representation model and the target (true) distribution as a measure of the information content. The relative entropy is known as the Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions.

It is intuitive to use the KL divergence between a representation model and the target distribution as the reward function. This way, the agent gets more rewards if it makes measurements during points when the divergence is high. Thus, the agent will seek out states where the predictor makes mistakes. For example, suppose the agent decides to measure at a time t . In that case, the resulting target distribution will be $\mathcal{N}(y_t, \epsilon^2)$, assuming a measurement error of ϵ and a normal distribution. The reward will be the KL divergence between the prediction and the target distribution:

$$R(s_t) = D_{KL}(\mathcal{N}(y_t, \epsilon^2) | \mathcal{N}(\hat{\mu}_{t,mix}, \hat{\sigma}_{t,mix}^2)) \quad (4.3)$$

Figure 4.9 shows an example visualization of the reward value as the KL divergence.

Although the KL divergence is a theoretically appealing reward, it does not work in practice. We observe an undesired behavior where the agents exploit the

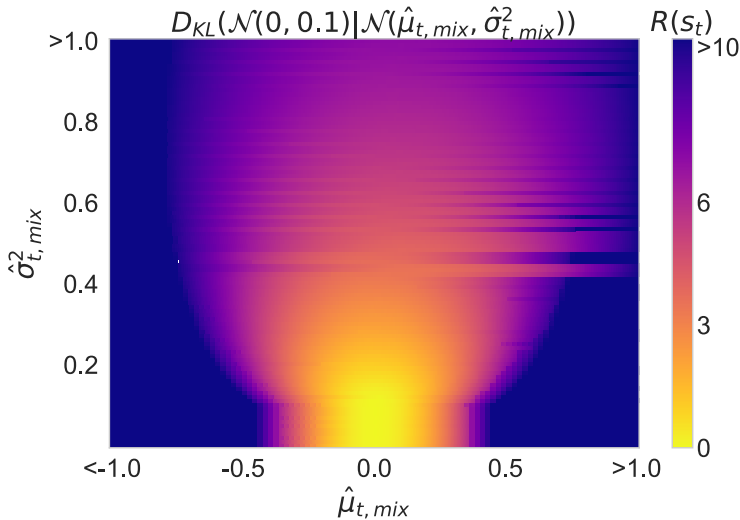


Figure 4.9: Reward value as the KL divergence between a target distribution $\mathcal{N}(y_t = 0.0, \epsilon^2 = 0.1)$, and a predicted distribution $\mathcal{N}(\hat{\mu}_{t,mix}, \hat{\sigma}_{t,mix}^2)$, where the predicted variance is not less than the measurement error.

predictor’s inaccuracies. The agent encourages inaccuracies in the prediction model so that it gets more rewards when it corrects the more severe inaccuracies later on. This challenge is closely related to the objective mismatch problem [47, 48] in model-based reinforcement learning. The problem arises when there is a discrepancy between the predictor and policy objectives. We are using the KL divergences as the agent’s objective in the hope that it will also optimize the predictor. But the predictor’s objective is to minimize the negative log-likelihood, which is uncorrelated with the agent’s objective of maximizing the KL divergence.

4.2.3 Value-based reward functions

In Paper V, we discuss the challenge of using the KL divergence as a reward function and propose a new reward function based on measured peaks-over-threshold (MPOT). This reward function is motivated by many applications’ goal of detecting and dense-sensing the phenomenon during periods when it exceeds a certain threshold. For example, in indoor noise monitoring, we want to detect and dense-sense events when the noise levels are high to get more granular details. Therefore, we simply reward the agent if it selects a measurement during a period of interest when the phenomenon exceeds the threshold. We define the reward function as follows:

$$R(s_t) = \begin{cases} \mathbb{1}(y_t - Q) & \text{if } E_t > 0 \\ -F & \text{if } E_t = 0 \end{cases} \quad (4.4)$$

where $\mathbb{1}(\cdot)$ is the indicator function, y_t is the measurement value, Q is the threshold, E_t is the energy budget, and F is a hyperparameter for the penalty term. The reward is positive if the measurement exceeds the threshold and the energy budget is positive. Otherwise, the reward is negative. The reward function encourages

4. Research results

the agent to select measurements when the phenomenon exceeds the threshold. The reward function also discourages the agent from wasting its energy budget on non-valuable measurements. Otherwise, the agent will get a negative reward if it exhausts its energy budget too early.

By using the MPOT as a reward function, we disentangle the agent’s objective from the predictor and prevent the issue of exploiting the predictor’s inaccuracies, as in the KL divergence reward. This way, we base our reward on facts, not predictions. And to get these facts, the agent needs to make measurements. We also show in Paper V that the MPOT reward function leads to a more accurate representation model. This behavior is because the predictor is usually good at modeling the phenomenon at regular periods and makes more mistakes during critical periods when there are spikes or high values. Simultaneously, the agent will measure when the phenomenon exceeds the threshold and when the prediction model is more likely to be inaccurate. Thus, the agent will provide more informative inputs for the model to refine its future prediction.

In summary, we define the *value* of collected data as capturing periods of interest while representing the phenomenon with a credible representation model. Then, we design a reward function that encourages the agent to exhibit these behaviors. We show that the MPOT reward function is a good candidate for this purpose. It is goal-oriented since it maximizes the value of collected data. Additionally, it is learnable since it is differentiable and provides an instant feedback signal to the learning algorithms. Finally, it is stable since it does not depend on the predictor’s accuracy. This way, the agent will not exploit the predictor’s inaccuracies to get more rewards.

4.2.4 Data-driven guidance

We see from the above discussions that designing a reward function is challenging. It requires a deep understanding of the application goals and the intricate interplay between the different components of that autonomous sensing framework. Manually designing reward functions adds design efforts that can hinder autonomy and the adoption of reinforcement learning.

Meanwhile, we have a large amount of data from expert-designed sensing policies in wireless sensor networks. It can be worth exploring how to use this data to guide the learning process without a reward function. This idea is explored in recent works [144], where they use imitation learning to learn a sensor power management policy from oracle policies. They use dynamic optimization methods to generate the oracle policies offline and then train an online policy using imitation learning.

In Paper V, we explore using data to guide learning without a reward function. We propose a new framework for using data from expert-designed policies to learn state projection that guides the agent towards desired behaviors. Although we do not apply the framework to IoT sensing problems, we evaluate it on continuous control problems that involve substantially different locomotion behaviors. We show that the framework can learn a projection that guides the agent toward the desired behaviors.

4.3 RQ3: Guided autonomous sensing

Normally, we have an implicit understanding or prior belief about a phenomenon. This belief is uncertain and imprecise. That is why we take measurements to get facts about the phenomenon and update our belief. The third research question asks how to make this belief explicit and how to use it to guide an autonomous sensing agent. We address this question in Paper III, Paper IV, and Paper V. This section summarizes the main contributions of these papers related to the third research question.

4.3.1 Representation model

In many applications, we want to provide *anytime* access to information about the phenomenon with minimal cost. To achieve this objective, we represent a phenomenon with an estimation or prediction model informed by sparse measurements. This model corresponds to an explicit representation of our belief or understanding about the phenomenon. The challenge is how to design a suitable representation model and how to update it given new measurements.

Probabilistic models

“All models are wrong, but some are useful” [150]. This quote expresses the context behind the challenge of designing a suitable representation model. The goal is to design a model that is useful for decision-making while taking into account its uncertainty. Thus, a useful model provides accurate enough estimates of the phenomenon while providing credible uncertainty quantification.

In Paper IV, we build various probabilistic deep learning models that also quantify their predictive uncertainty. We define the tools and metrics for evaluating performance and uncertainty quantification in these models. We also investigate the reliability of the uncertainty estimates and propose improving these estimations using adversarial training. Finally, we demonstrate the practical impact of uncertainty quantification for making informed decisions. This ties back to the ultimate goal of IoT sensing applications: to support decision-making. For example, we show that probabilistic models can quantify both aleatoric and epistemic uncertainty. This way, probabilistic models provide stakeholders with a wider area of control over the decision’s risk profile. The stakeholders can choose to what extent their model should be confident about its predictions to make a decision. This trade-off can be based on the costs of false positives versus false negatives. Figure 4.10 shows an illustrative example of this trade-off in the air quality application.

Refinement with new measurements

When taking a new measurement, we get an actual numerical value of the phenomenon. This measurement establishes a new fact about the phenomenon at a specific time. Thus, we can use this fact to update our prior belief about the phenomenon. This update is done by refining the representation model. The challenge is updating the model in a way consistent with the prior belief and the new measurement.

4. Research results

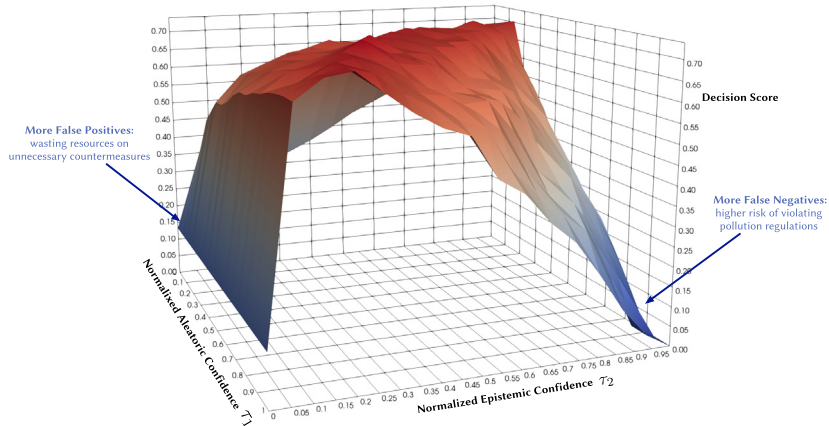


Figure 4.10: Illustrative example of how probabilistic models can provide more information to support decision-making. The figure shows the trade-off between false positives and negatives using the decision F1 score as a function of normalized aleatoric and epistemic confidence thresholds. (Adapted from figure IV.19 in Paper IV)

One approach is to add the new measurements to the input features of the model. This way, the model itself learns how to use new measurements to refine its predictions and take account of the prior belief. For example, in Paper V, we use a Bayesian neural network (BNN) model to predict air quality. The model takes as input the recent measurements, their elapsed time, and other explanatory variables. The recent measurements are stored in a queue (FIFO). Thus, the oldest measurements are removed from the queue when new measurements arrive. Figure 4.11 shows an example prediction for air quality using BNNs, with a prediction horizon of three days and an hourly prediction granularity. The figure also shows the prediction progress over a month, where the model refines its future prediction once a new measurement becomes available. In Paper III, we use a Gaussian process to build a representation model. Conventional Gaussian processes require retraining the entire model when new data arrives. However, other approaches exist to train the model incrementally, such as sparse online Gaussian processes [51].

4.3.2 Guiding autonomous sensing agents

After building a representation model, we want to use it to guide the deep reinforcement learning agent in taking more informative measurements. For instance, we want to guide the agent to take measurements that are more informative for the prediction model or the application goal (e.g., detecting periods of interest). The challenge is designing a suitable framework for integrating the representation model with the deep reinforcement learning agent.

In Paper III and Paper V, we design a framework for autonomous sensing with deep reinforcement learning guided by an uncertainty-aware prediction model. The framework is an extension of the IoT Sensor Gym we introduced in Paper II. Figure 4.12 depicts the key components of the designed framework. The high-level idea is

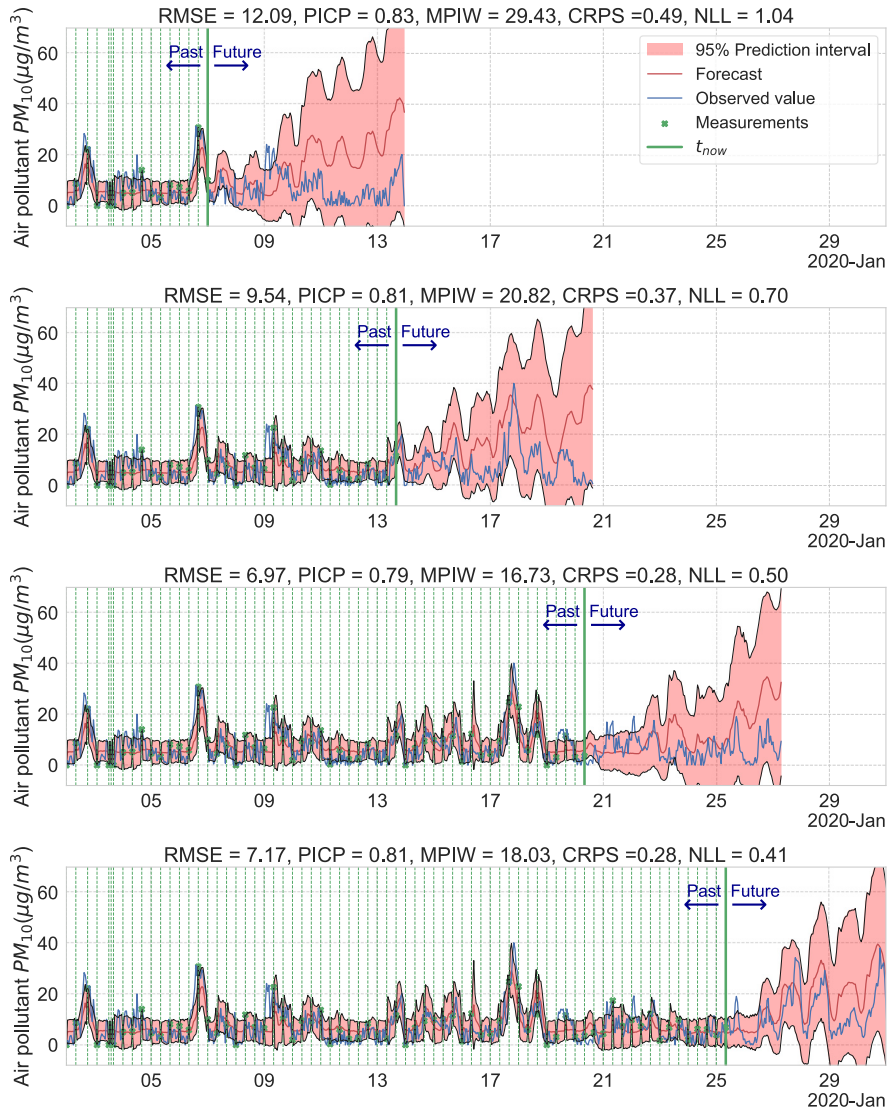


Figure 4.11: An example prediction for air quality using BNNs. t_{now} represents the current time step when the model performs future predictions using explainable variables, the latest measurements (green stars), and the difference in time (elapsed time) between the prediction points and these latest measurements. The Selected measurements provide input features to refine future predictions. The prediction is more confident when the prediction point is close to the recent measurements (shorter elapsed time). (Adapted from figure V.59 in Paper V)

4. Research results

to build a prediction model that forecasts the future evolution of the monitored phenomenon. Then, a deep reinforcement learning agent selects measurement data points that optimize the application goal subject to resource constraints. The agent leverages the prediction to focus its constrained resources on selecting the most informative measurements. These new measurements provide input features to refine future predictions, similar to a feedback loop.

The framework is generic to apply to various application domains. For example, in Paper III, we use a Gaussian process to build a representation model and apply the framework for autonomous sensing in indoor noise monitoring with Fisher information as the reward function. In Paper V, we use a BNNs model to build a representation model and apply the framework to air quality and indoor noise monitoring with MPOT as the reward function.

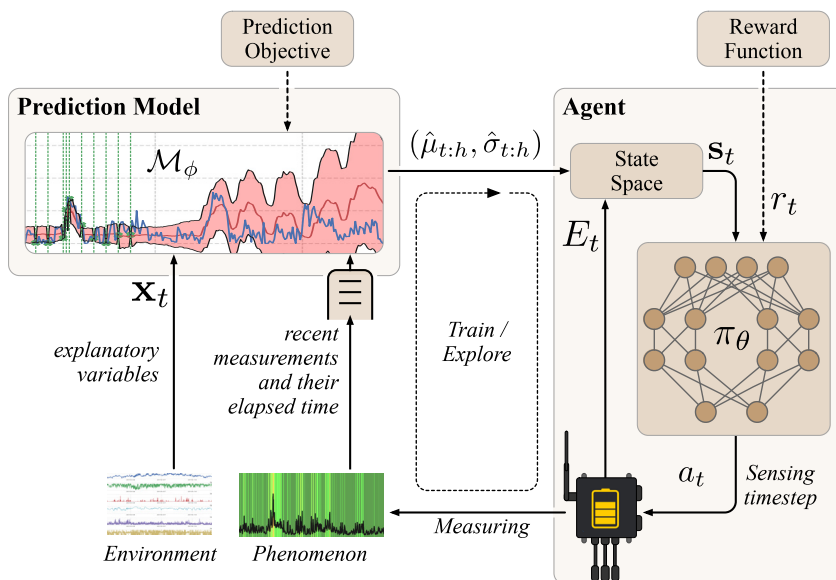


Figure 4.12: A framework for autonomous sensing with deep reinforcement learning guided by an uncertainty-aware prediction model. (Adapted from figure V.57 in Paper V)

4.3.3 Discussion

One could argue that it is possible to use deep reinforcement learning for autonomous sensing without a representation model. Indeed, we can use deep reinforcement learning in an end-to-end fashion where we feed the input features to the agent directly without the need for the representation model. Thus, it seems redundant to incorporate the model, even contradicts our goal of reducing manual design efforts. To address this concern, we offer the following arguments.

First, one of the application goals is to construct a representation model of the phenomena informed by sparse measurements. We need this model to infer missing

data that we don't measure. We can use the model to provide the stakeholders with *anytime* access to information about the phenomenon with minimal cost. The sensing policies aim to schedule fewer measurements that contribute optimally to the quality of the resulting model. Therefore, it is a good use to construct a model in the first place.

Second, training end-to-end would overfit the sensing policy to the training location. The policy would not generalize to new sensing locations. In contrast, incorporating a representation model enables the policy to transfer to new sensing locations with different dynamic environments or air quality patterns. For example, we show in Paper V that a policy trained in one station can transfer to a new station that is spatially distant and has a different air quality pattern. Figure 4.13 shows that the two stations “Elgeseter”, and “Tiller” are spatially distant (approximately 7 km apart), and figure 4.14 shows that the air quality pattern is quite different.

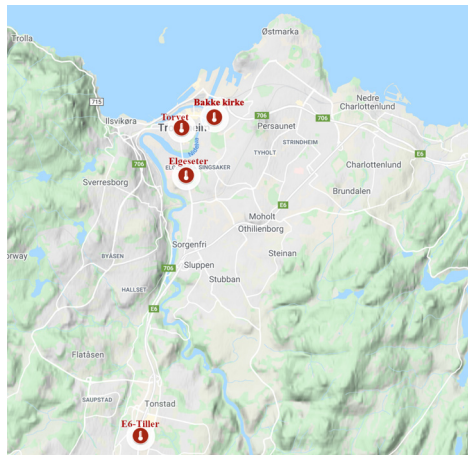


Figure 4.13: Locations of air quality sensing stations in Trondheim, Norway. The two stations “Elgeseter”, and “Tiller” are spatially distant.

The policies are transferable because they depend mainly on the output of the representation model, thus, relying on fewer bits of information and avoiding overfitting the training task. This is similar to the information bottleneck technique used in RL to improve generalization [151, 152]. Another explanation is that the policies reduce the modeling gap between a source station and a new station by relying only on the predictor. This transferability can lower the cost of fine-tuning policies required in RL when deployed to different locations, enabling “out-of-the-box” deployment.

Third, end-to-end training requires the agent to learn a good state-space representation from scratch. This task is challenging because the state space is high-dimensional and the agent has limited feedback information through the reward signal. In contrast, the prediction model has more feedback information through the loss function. It can learn a good state-space representation faster and more efficiently. Thus, we can consider the agent as a fine-tuning that selects new measurements to refine the predictors best while the predictor guides the agent

4. Research results

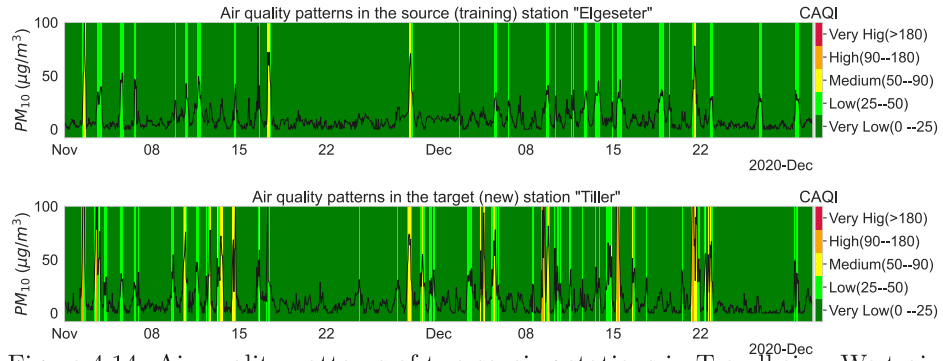


Figure 4.14: Air quality patterns of two sensing stations in Trondheim. We train the RL and the prediction model using Elgeseter, the source (training) station. Then, we test the ability of the policy to transfer to a new station (Tiller) with different air quality patterns.

in the high-dimensional state space. Finally, we can see the representation model as our *prior* belief or understanding of the phenomena. The agent uses this belief to make better decisions. This is similar to the idea of using a prior in Bayesian optimization.

“Cut until there is nothing left to cut and all there is left is principled and fundamental”
– Bjarne Stroustrup, 2019

Chapter 5

Concluding remarks

5.1 Summary of research contributions

This section summarizes the main research contributions described in Chapter 4. We organize the research contributions into three main categories that correspond to the three research questions of this thesis.

RQ1: Feasibility of autonomous sensing

How can we utilize deep reinforcement learning to autonomously learn policies that control sensing of IoT devices?

- Our work in Paper I is the first demonstration of the feasibility of using deep reinforcement learning for autonomous sensing in (energy-harvesting) sensor devices.
- We address the challenge of computational complexity by building a general-purpose simulator to train deep reinforcement learning agents.
- Through three different use cases, we show that using deep reinforcement learning for autonomous sensing leads to increased autonomy and, thus, scalability by reducing manual engineering efforts involved in designing sparse sensing policies.
- We demonstrate that we can achieve the desired behavior by choosing a proper reward function. Additionally, we show how to make trade-offs between conflicting objectives and set the trade-off parameters more suitable for the application.

RQ2: Reward function design

In the setting of RQ1, how can we define reward functions that support application goals and lead to learning?

- We demonstrate that deep reinforcement learning enables using learnable reward functions that are more aligned with the application goals.
- We show that we can design utility-based reward functions that balance the trade-offs between conflicting objectives and avoid including side concerns in the reward function, such as energy.

5. Concluding remarks

- Information-based reward functions are more suitable for applications where we can quantify the information gain of a measurement, such as applications that incorporate representation models. We show that we can use information-based rewards to learn sensing policies for selecting informative measurements with regard to the representation model.
- We demonstrate the benefits of using value-based rewards, such as measured peaks-over-threshold, in capturing events of interest and leading to more accurate representation models while reducing energy expenditure.

RQ3: Guided autonomous sensing

How can we use our understanding about a phenomenon to guide an autonomous sensing agent?

- We introduce a framework for explicitly representing our understanding about a phenomenon and use this representation to guide the autonomous sensing agent in selecting informative measurements with regard to the representation model.
- We use probabilistic deep learning to build data-driven representation models that also quantify their predictive uncertainty. These models learn how to use new measurements to refine their predictions and take account of the prior knowledge about the phenomenon.
- We demonstrate that incorporating a representation model enables learning sensing policies that adapt and transfer to new sensing stations.

5.2 Limitations and future directions

This section discusses the limitations of the proposed approaches and future directions for improving the results presented in this thesis. We also discuss promising applications that we foresee for the proposed approaches in various domains.

Stability

We observed throughout the research project that the performance of trained agents varies significantly due to the algorithm’s sensitivity to hyperparameters and the intricate interplay between the policy and the prediction model. Similar to model-based reinforcement learning, our approach has more “moving parts” than model-free reinforcement learning or a manually-engineered algorithm. This performance fluctuation, however, does not necessarily prohibit the proposed approach since the training takes place on a server, and it is sufficient to choose the best-performing policy before deploying it on a sensor. This also suggests that a promising direction for future work is improving stability and reducing complexity while maintaining autonomy.

Objective mismatch

A promising direction for future work is to address the objective mismatch problem that arises when there is a discrepancy between the prediction model and policy objectives. We encountered this problem when we used KL divergence as a reward function. We observe an undesired behavior where the agents exploit the predictor's inaccuracies. The agent encourages inaccuracies in the prediction model so that it gets more rewards when it corrects the more severe inaccuracies later on. One possible approach to address this problem is to use a single objective that jointly optimizes the policy and the predictor [48] or use Nash equilibrium between two players: predictor and sensing agent.

Hybrid prediction models

In our work, we only explored data-driven probabilistic models. However, a promising future direction would include hybrid models that combine physics-based and data-driven models. There is a need for further investigation into how to use existing knowledge to constrain deep learning models. For example, by leveraging structure in the data, we can impute our prior knowledge, such as invariance, equivariance, or scale separation into the models. These inductive priors will make our models more sample-efficient and generalize better by ignoring harmful or irrelevant information. Accordingly, this will reduce the required training data and result in stable and predictable performance.

Robust prediction models

Future work should address uncertainty estimation with out-of-domain or seasonal variations in prediction models. This is commonly referred to as robustness to covariate shift, where input data distribution changes over time. Thus, further studies are required in fine-tuning predictors to adapt to new downstream tasks. One possibility is using reinforcement learning to decide when to fine-tune the predictor. In our experiments, we test the transferability of the learned policies to new sensing stations. We compare the performance with and without fine-tuning the predictor. We observe that fine-tuning the predictor improves the performance of the resulting representation model. However, there is a need for further investigation into how to fine-tune the predictor incrementally and more efficiently.

Sparse sensing for data management

Sparse sensing is particularly useful in addressing the problem of data swamps in many industrial IoT applications. These applications often have abundant energy resources and can afford to collect more data. However, collecting more data from many sensors at a high rate results in data lakes that are difficult to manage. Sparse sensing techniques can reduce the amount of data collected, thus reducing the cost of data storage and management. Therefore, a promising direction for future work is to investigate the benefits of our proposed frameworks for sparse sensing in data management. Further studies are required on the trade-off between the amount of data collected and the cost of data management. For example, we can use reinforcement learning to decide when to collect data, how much to collect, and how to collect data.

Autonomous sensing for active learning

The sensing formulation has a compelling similarity to the active learning problem. In active learning, we want to select the most informative data to train a machine learning model. We select a data point from a pool of unlabeled data at each time step. The selected point is labeled and added to the training set. The goal is to reduce the number of labeled data points while maintaining the model's performance. This is usually done using manually-designed heuristics, for example, by selecting the data points far from the decision boundary. A promising future direction is formulating the active learning problem as MDP and using reinforcement learning to learn a data selection policy. We can use our proposed framework in such a formulation.

Data-driven guidance for autonomous sensing

In Paper V, we explore using data to guide learning without a reward function. However, we do not apply the framework to IoT sensing problems. Thus, a promising direction for future work is to explore how to use data to guide learning without a reward function in IoT sensing problems. In particular, we can leverage the large amount of data we already have from expert-designed sensing policies in wireless sensor networks.

In conclusion, this thesis investigates the use of deep reinforcement learning to develop autonomous sensing solutions and shows that it is a viable approach for addressing the scalability challenges of IoT sensing applications. By enabling individual sensors to learn sensing strategies suitable for their specific environments and capabilities, deep reinforcement learning can increase autonomy and reduce the need for manual design efforts. Through our investigation of three different use cases and various reward functions, we have demonstrated the feasibility of using deep reinforcement learning to achieve comparable performance to conventional methods while increasing autonomy and scalability. Additionally, we have shown that explicitly representing our understanding about a phenomenon can guide the autonomous sensing agent in selecting informative measurements that maximally inform our prior knowledge of the phenomenon. Overall, this work highlights the potential for deep reinforcement learning to lead to more scalable sensing solutions that align with supporting application goals of decision-making.

References

- [1] Lovett, G. M., Burns, D. A., Driscoll, C. T., Jenkins, J. C., Mitchell, M. J., Rustad, L., Shanley, J. B., Likens, G. E., and Haeuber, R. “Who needs environmental monitoring?” In: *Frontiers in Ecology and the Environment* vol. 5, no. 5 (2007), pp. 253–260.
- [2] McIntyre, G. A. and Hintz, K. J. “Information theoretic approach to sensor scheduling”. In: *Signal processing, sensor fusion, and target recognition V*. Vol. 2755. SPIE. 1996, pp. 304–312.
- [3] Santini, S. and Romer, K. “An adaptive strategy for quality-based data reduction in wireless sensor networks”. In: *Proceedings of the 3rd international conference on networked sensing systems (INSS 2006)*. TRF Chicago, IL. 2006, pp. 29–36.
- [4] Tan, L. and Wu, M. “Data reduction in wireless sensor networks: A hierarchical LMS prediction approach”. In: *IEEE Sensors Journal* vol. 16, no. 6 (2015), pp. 1708–1715.
- [5] Dias, G. M., Bellalta, B., and Oechsner, S. “A survey about prediction-based data reduction in wireless sensor networks”. In: *ACM Computing Surveys (CSUR)* vol. 49, no. 3 (2016), pp. 1–35.
- [6] Botero-Valencia, J., Castano-Londono, L., Marquez-Viloria, D., and Rico-Garcia, M. “Data reduction in a low-cost environmental monitoring system based on LoRa for WSN”. In: *IEEE Internet of Things Journal* vol. 6, no. 2 (2018), pp. 3024–3030.
- [7] Chaoming Hsu, R., Liu, C. T., and Lee, W. M. “Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Vol. 5579 LNAI. 2009.
- [8] Blasco, P., Gunduz, D., and Dohler, M. “A learning theoretic approach to energy harvesting communication system optimization”. In: *IEEE Transactions on Wireless Communications* vol. 12, no. 4 (2013), pp. 1872–1882.
- [9] Hsu, R. C., Liu, C.-T., and Wang, H.-L. “A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node”. In: *IEEE Transactions on Emerging Topics in Computing* vol. 2, no. 2 (2014), pp. 181–191.
- [10] Ortiz, A., Al-Shatri, H., Li, X., Weber, T., and Klein, A. “Reinforcement learning for energy harvesting point-to-point communications”. In: *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, pp. 1–6.
- [11] Lei, Z., Tang, H., Li, H., and Jiang, Q. “Dynamic power management strategies for a sensor node optimised by reinforcement learning”. In: *Int. J. Computational Science and Engineering* vol. 13, no. 1 (2016).

5. Concluding remarks

- [12] Aoudia, F. A., Gautier, M., and Berder, O. “RLMan: an Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks”. In: *IEEE Transactions on Green Communications and Networking* vol. 2, no. 2 (2018), pp. 1–1.
- [13] Shresthamali, S., Kondo, M., and Nakamura, H. “Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning”. In: vol. 16, no. 5s (2017), pp. 1–21.
- [14] Fraternali, F., Balaji, B., and Gupta, R. “Scaling configuration of energy harvesting sensors with reinforcement learning”. In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ACM. 2018, pp. 7–13.
- [15] LeCun, Y., Bengio, Y., and Hinton, G. “Deep learning”. In: *nature* vol. 521, no. 7553 (2015), pp. 436–444.
- [16] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. “Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 651–673.
- [17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. “Human-level control through deep reinforcement learning”. In: *Nature* vol. 518, no. 7540 (2015), p. 529.
- [18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [19] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.
- [20] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. “Continuous control with deep reinforcement learning”. In: *arXiv:1509.02971 [cs, stat]* (July 5, 2019). arXiv: 1509.02971.
- [21] Wieringa, R. J. *Design science methodology for information systems and software engineering*. 2014.
- [22] Rummery, G. A. and Niranjan, M. *On-line Q-learning using connectionist systems*. Vol. 37. 1994.
- [23] Watkins, C. J. and Dayan, P. “Q-learning”. In: *Machine learning* vol. 8, no. 3 (1992), pp. 279–292.
- [24] Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. “Boltzmann exploration done right”. In: *Advances in neural information processing systems* vol. 30 (2017).
- [25] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. “Continuous control with deep reinforcement learning”. In: *International Conference on Learning Representations, ICLR 2016*. International Conference on Representation Learning. 2016.

-
- [26] Schaul, T., Quan, J., Antonoglou, I., Silver David Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. “Prioritized experience replay”. In: *International Conference on Learning Representations, ICLR 2016*. International Conference on Representation Learning. 2016.
- [27] Van Hasselt, H., Guez, A., and Silver, D. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.
- [28] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. “Dueling network architectures for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.
- [29] Williams, R. J. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* vol. 8, no. 3 (1992), pp. 229–256.
- [30] Baxter, J. and Bartlett, P. L. “Infinite-horizon policy-gradient estimation”. In: *Journal of Artificial Intelligence Research* vol. 15 (2001), pp. 319–350.
- [31] Thomas, P. “Bias in natural actor-critic algorithms”. In: *International conference on machine learning*. PMLR. 2014, pp. 441–448.
- [32] Peters, J. and Schaal, S. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* vol. 21, no. 4 (2008), pp. 682–697.
- [33] Konda, V. and Tsitsiklis, J. “Actor-critic algorithms”. In: *Advances in neural information processing systems* vol. 12 (1999).
- [34] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.
- [35] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.
- [36] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017), pp. 1–12. arXiv: [arXiv:1707.06347v2](https://arxiv.org/abs/1707.06347v2).
- [37] Degris, T., White, M., and Sutton, R. “Off-Policy Actor-Critic”. In: *International Conference on Machine Learning*. 2012.
- [38] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. “Deterministic policy gradient algorithms”. In: *International conference on machine learning*. PMLR. 2014, pp. 387–395.
- [39] Fujimoto, S., Hoof, H., and Meger, D. “Addressing function approximation error in actor-critic methods”. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.
- [40] Williams, R. J. and Peng, J. “Function optimization using connectionist reinforcement learning algorithms”. In: *Connection Science* vol. 3, no. 3 (1991), pp. 241–268.

5. Concluding remarks

- [41] Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. “Diversity is All You Need: Learning Skills without a Reward Function”. In: *International Conference on Learning Representations*. 2018.
- [42] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.
- [43] Klemsdal, E., Herland, S., and Murad, A. “Learning Task Agnostic Skills with Data-driven Guidance”. In: *ICML 2021 Workshop on Unsupervised Reinforcement Learning*. 2021.
- [44] Deisenroth, M. and Rasmussen, C. E. “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.
- [45] Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566.
- [46] Chua, K., Calandra, R., McAllister, R., and Levine, S. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in neural information processing systems* vol. 31 (2018).
- [47] Lambert, N., Amos, B., Yadan, O., and Calandra, R. “Objective mismatch in model-based reinforcement learning”. In: *arXiv preprint arXiv:2002.04523* (2020).
- [48] Eysenbach, B., Khazatsky, A., Levine, S., and Salakhutdinov, R. “Mismatched No More: Joint Model-Policy Optimization for Model-Based RL”. In: *Deep RL Workshop NeurIPS 2021*. 2021.
- [49] Chen, T. and Guestrin, C. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.
- [50] Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*. Vol. 2. 2006.
- [51] Csató, L. and Opper, M. “Sparse on-line Gaussian processes”. In: *Neural computation* vol. 14, no. 3 (2002), pp. 641–668.
- [52] Williams, C. K. and Seeger, M. “Using the Nyström method to speed up kernel machines”. In: *Advances in neural information processing systems*. 2001, pp. 682–688.
- [53] Bui, T. D., Yan, J., and Turner, R. E. “A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation”. In: *The Journal of Machine Learning Research* vol. 18, no. 1 (2017), pp. 3649–3720.
- [54] Titsias, M. “Variational learning of inducing variables in sparse Gaussian processes”. In: *Artificial Intelligence and Statistics*. 2009, pp. 567–574.
- [55] Gneiting, T. and Raftery, A. E. “Strictly proper scoring rules, prediction, and estimation”. In: *Journal of the American statistical Association* vol. 102, no. 477 (2007), pp. 359–378.

-
- [56] Nix, D. A. and Weigend, A. S. “Estimating the mean and variance of the target probability distribution”. In: *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*. Vol. 1. IEEE. 1994, pp. 55–60.
- [57] Graves, A. “Practical variational inference for neural networks”. In: *Advances in neural information processing systems* vol. 24 (2011).
- [58] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. “Weight uncertainty in neural network”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1613–1622.
- [59] Louizos, C. and Welling, M. “Multiplicative normalizing flows for variational bayesian neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2218–2227.
- [60] Neal, R. M. *Bayesian learning for neural networks*. Vol. 118. 2012.
- [61] Welling, M. and Teh, Y. W. “Bayesian learning via stochastic gradient Langevin dynamics”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 681–688.
- [62] Chen, T., Fox, E., and Guestrin, C. “Stochastic gradient hamiltonian monte carlo”. In: *International conference on machine learning*. PMLR. 2014, pp. 1683–1691.
- [63] MacKay, D. J. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* vol. 4, no. 3 (1992), pp. 448–472.
- [64] Ritter, H., Botev, A., and Barber, D. “A scalable laplace approximation for neural networks”. In: *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*. Vol. 6. International Conference on Representation Learning. 2018.
- [65] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. “Stochastic variational inference.” In: *Journal of Machine Learning Research* vol. 14, no. 5 (2013).
- [66] Kingma, D. P. and Welling, M. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).
- [67] Kingma, D. P., Salimans, T., and Welling, M. “Variational dropout and the local reparameterization trick”. In: *Advances in neural information processing systems* vol. 28 (2015), pp. 2575–2583.
- [68] Gal, Y. and Ghahramani, Z. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.
- [69] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* vol. 15, no. 1 (2014), pp. 1929–1958.
- [70] Lakshminarayanan, B., Pritzel, A., and Blundell, C. “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles”. In: *Advances in Neural Information Processing Systems* vol. 30 (2017).
- [71] Dietterich, T. G. “Ensemble methods in machine learning”. In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

5. Concluding remarks

- [72] Hinton, G., Vinyals, O., and Dean, J. “Distilling the knowledge in a neural network”. In: *NIPS 2014 Deep Learning and Representation Learning Workshop*. 2014.
- [73] Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. “A simple baseline for bayesian uncertainty in deep learning”. In: *Advances in Neural Information Processing Systems* vol. 32 (2019), pp. 13153–13164.
- [74] Izmailov, P., Podoprikin, D., Garipov, T., Vetrov, D., and Wilson, A. G. “Averaging weights leads to wider optima and better generalization”. In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*. Association For Uncertainty in Artificial Intelligence (AUAI). 2018, pp. 876–885.
- [75] Hochreiter, S. and Schmidhuber, J. “Long short-term memory”. In: *Neural computation* vol. 9, no. 8 (1997), pp. 1735–1780.
- [76] Sak, H., Senior, A. W., and Beaufays, F. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”. In: *Google Research* (2014).
- [77] Bengio, Y., Simard, P., and Frasconi, P. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* vol. 5, no. 2 (1994), pp. 157–166.
- [78] Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., and Zhang, Q. “Spectral Temporal Graph Neural Network for Multivariate Time-series Forecasting”. In: *Proceedings of Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. Vol. 33. 2020, pp. 17766–17778.
- [79] Kipf, T. N. and Welling, M. “Semi-supervised classification with graph convolutional networks”. In: *Proceedings of the 5th International Conference on Learning Representations, ICLR '17*, 2017.
- [80] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. “Language modeling with gated convolutional networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 933–941.
- [81] Alippi, C., Anastasi, G., Di Francesco, M., and Roveri, M. “Energy management in wireless sensor networks with energy-hungry sensors”. In: *IEEE Instrumentation & Measurement Magazine* vol. 12, no. 2 (2009), pp. 16–23.
- [82] Alippi, C., Anastasi, G., Di Francesco, M., and Roveri, M. “An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors”. In: *IEEE Transactions on Instrumentation and Measurement* vol. 59, no. 2 (2009), pp. 335–344.
- [83] Khan, J. A., Qureshi, H. K., and Iqbal, A. “Energy management in wireless sensor networks: A survey”. In: *Computers & Electrical Engineering* vol. 41 (2015), pp. 159–176.

-
- [84] Beretta, I., Rincon, F., Khaled, N., Grassi, P. R., Rana, V., and Atienza, D. “Design exploration of energy-performance trade-offs for wireless sensor networks”. In: *Proceedings of the 49th Annual Design Automation Conference*. 2012, pp. 1043–1048.
- [85] Rault, T., Bouabdallah, A., and Challal, Y. “Energy efficiency in wireless sensor networks: A top-down survey”. In: *Computer networks* vol. 67 (2014), pp. 104–122.
- [86] Kraemer, F. A., Alawad, F., and Bosch, I. M. V. “Energy-Accuracy Tradeoff for Efficient Noise Monitoring and Prediction in Working Environments”. In: *Proceedings of the 9th International Conference on the Internet of Things*. 2019, pp. 1–8.
- [87] Mao, S., Cheung, M. H., and Wong, V. W. “An optimal energy allocation algorithm for energy harvesting wireless sensor networks”. In: *2012 IEEE International Conference on Communications (ICC)*. IEEE. 2012, pp. 265–270.
- [88] Rao, V. S., Prasad, R. V., and Niemegeers, I. G. “Optimal task scheduling policy in energy harvesting wireless sensor networks”. In: *2015 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2015, pp. 1030–1035.
- [89] Kansal, A., Hsu, J., Zahedi, S., and Srivastava, M. B. “Power management in energy harvesting sensor networks”. In: *ACM Transactions on Embedded Computing Systems* vol. 6, no. 4 (2007), 32–es.
- [90] Vigorito, C. M., Ganesan, D., and Barto, A. G. “Adaptive control of duty cycling in energy-harvesting wireless sensor networks”. In: *2007 4th Annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks*. IEEE. 2007, pp. 21–30.
- [91] Shresthamali, S., Kondo, M., and Nakamura, H. “Power management of wireless sensor nodes with coordinated distributed reinforcement learning”. In: *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE. 2019, pp. 638–647.
- [92] Fraternali, F., Balaji, B., Agarwal, Y., and Gupta, R. K. “Aces: Automatic configuration of energy harvesting sensors with reinforcement learning”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 16, no. 4 (2020), pp. 1–31.
- [93] Sawaguchi, S., Christmann, J.-F., and Lesecq, S. “Highly adaptive linear actor-critic for lightweight energy-harvesting IoT applications”. In: *Journal of Low Power Electronics and Applications* vol. 11, no. 2 (2021), p. 17.
- [94] Hsu, R. C., Liu, C.-T., Wang, K.-C., and Lee, W.-M. “QoS-aware power management for energy harvesting wireless sensor network utilizing reinforcement learning”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 2. IEEE. 2009, pp. 537–542.

5. Concluding remarks

- [95] Liu, C. T. and Hsu, R. C. “Dynamic power management utilizing reinforcement learning with fuzzy reward for energy harvesting wireless sensor nodes”. In: *IECON Proceedings (Industrial Electronics Conference)* (2011), pp. 2365–2369.
- [96] Hsu, R. C., Liu, C. T., and Wang, H. L. “A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node”. In: *IEEE Transactions on Emerging Topics in Computing* vol. 2, no. 2 (2014), pp. 181–191.
- [97] Hsu, R. C., Lin, T. H., Chen, S. M., and Liu, C. T. “Dynamic energy management of energy harvesting wireless sensor nodes using fuzzy inference system with reinforcement learning”. In: *Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015* (2015), pp. 116–120.
- [98] Murad, A., Kraemer, F. A., Bach, K., and Taylor, G. “Autonomous management of energy-harvesting IoT nodes using deep reinforcement learning”. In: *proceedings of IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Umeå, Sweden*. 2019.
- [99] Fraternali, F., Balaji, B., Sengupta, D., Hong, D., and Gupta, R. K. “Ember: energy management of batteryless event detection sensors with deep reinforcement learning”. In: *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 2020, pp. 503–516.
- [100] Basaklar, T., Tuncel, Y., and Ogras, U. Y. “tinyMAN: Lightweight Energy Manager using Reinforcement Learning for Energy Harvesting Wearable IoT Devices”. In: *Proceedings of the tinyML Research Symposium 2022, Burlingame, California, United States*. 2022.
- [101] Shresthamali, S., Kondo, M., and Nakamura, H. “Multi-Objective Resource Scheduling for IoT Systems Using Reinforcement Learning”. In: *Journal of Low Power Electronics and Applications* vol. 12, no. 4 (2022), p. 53.
- [102] Laiymani, D. and Makhoul, A. “Adaptive data collection approach for periodic sensor networks”. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2013, pp. 1448–1453.
- [103] Silva, J. M. C., Bispo, K. A., Carvalho, P., and Lima, S. R. “LiteSense: An adaptive sensing scheme for WSNs”. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2017, pp. 1209–1212.
- [104] Harb, H. and Makhoul, A. “Energy-efficient sensor data collection approach for industrial process monitoring”. In: *IEEE Transactions on Industrial Informatics* vol. 14, no. 2 (2017), pp. 661–672.
- [105] Salim, C., Makhoul, A., Darazi, R., and Couturier, R. “Adaptive sampling algorithms with local emergency detection for energy saving in wireless body sensor networks”. In: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2016, pp. 745–749.
- [106] Rieger, R. and Taylor, J. T. “An adaptive sampling system for sensor nodes in body area networks”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* vol. 17, no. 2 (2009), pp. 183–189.

-
- [107] Kurp, T., Gao, R. X., and Sah, S. “An adaptive sampling scheme for improved energy utilization in wireless sensor networks”. In: *2010 IEEE Instrumentation & Measurement Technology Conference Proceedings*. IEEE. 2010, pp. 93–98.
- [108] Jain, A. and Chang, E. Y. “Adaptive sampling for sensor networks”. In: *Proceedings of the 1st international workshop on Data management for sensor networks: in conjunction with VLDB 2004*. 2004, pp. 10–16.
- [109] Deshpande, A., Guestrin, C., Madden, S. R., Hellerstein, J. M., and Hong, W. “Model-based approximate querying in sensor networks”. In: *The VLDB journal* vol. 14, no. 4 (2005), pp. 417–443.
- [110] Xu, Y., Choi, J., Dass, S., and Maiti, T. “Sequential Bayesian prediction and adaptive sampling algorithms for mobile sensor networks”. In: *IEEE Transactions on Automatic Control* vol. 57, no. 8 (2011), pp. 2078–2084.
- [111] Trimpe, S. and Baumann, D. “Resource-aware IoT control: Saving communication through predictive triggering”. In: *IEEE Internet of Things Journal* vol. 6, no. 3 (2019), pp. 5013–5028.
- [112] Kho, J., Rogers, A., and Jennings, N. R. “Decentralised adaptive sampling of wireless sensor networks”. In: (2007).
- [113] Kho, J., Rogers, A., and Jennings, N. R. “Decentralized control of adaptive sampling in wireless sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 5, no. 3 (2009), pp. 1–35.
- [114] Sun, Y., Yuan, Y., Li, X., Xu, Q., and Guan, X. “An adaptive sampling algorithm for target tracking in underwater wireless sensor networks”. In: *IEEE Access* vol. 6 (2018), pp. 68324–68336.
- [115] Dias, G. M., Nurchis, M., and Bellalta, B. “Adapting sampling interval of sensor networks using on-line reinforcement learning”. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE. 2016, pp. 460–465.
- [116] Cobb, A. and Markham, A. “Adaptive sampling of lion accelerometer data”. In: *Oxford* (2016).
- [117] Su, R., Gong, Z., Zhang, D., Li, C., Chen, Y., and Venkatesan, R. “An adaptive asynchronous wake-up scheme for underwater acoustic sensor networks using deep reinforcement learning”. In: *IEEE Transactions on Vehicular Technology* vol. 70, no. 2 (2021), pp. 1851–1865.
- [118] Jerri, A. J. “The Shannon sampling theorem—Its various extensions and applications: A tutorial review”. In: *Proceedings of the IEEE* vol. 65, no. 11 (1977), pp. 1565–1596.
- [119] Donoho, D. L. “Compressed sensing”. In: *IEEE Transactions on information theory* vol. 52, no. 4 (2006), pp. 1289–1306.
- [120] Karakus, C., Gurbuz, A. C., and Tavli, B. “Analysis of energy efficiency of compressive sensing in wireless sensor networks”. In: *IEEE Sensors Journal* vol. 13, no. 5 (2013), pp. 1999–2008.
- [121] Muttreja, A., Raghunathan, A., Ravi, S., and Jha, N. K. “Active learning driven data acquisition for sensor networks”. In: *11th IEEE Symposium on Computers and Communications (ISCC’06)*. IEEE. 2006, pp. 929–934.

5. Concluding remarks

- [122] Osborne, M. A., Roberts, S. J., Rogers, A., and Jennings, N. R. “Real-time information processing of environmental sensor network data using bayesian gaussian processes”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 9, no. 1 (2012), pp. 1–32.
- [123] Ling, C. K., Low, K. H., and Jaillet, P. “Gaussian process planning with Lipschitz continuous reward functions: Towards unifying Bayesian optimization, active learning, and beyond”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.
- [124] Monteiro, L. C., Delicato, F. C., Pirmez, L., Pires, P. F., and Miceli, C. “Dpcas: Data prediction with cubic adaptive sampling for wireless sensor networks”. In: *International Conference on Green, Pervasive, and Cloud Computing*. Springer. 2017, pp. 353–368.
- [125] Lemlouma, T. “Adaptive Sensing Algorithm for IoT Applications with Data and Temporal Accuracy Requirements”. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–7.
- [126] Shu, T., Chen, J., Bhargava, V. K., and Silva, C. W. de. “An energy-efficient dual prediction scheme using LMS filter and LSTM in wireless sensor networks for environment monitoring”. In: *IEEE Internet of Things Journal* vol. 6, no. 4 (2019), pp. 6736–6747.
- [127] Dias, G. M., Bellalta, B., and Oechsner, S. “The impact of dual prediction schemes on the reduction of the number of transmissions in sensor networks”. In: *Computer communications* vol. 112 (2017), pp. 58–72.
- [128] Askari Moghadam, R. and Keshmirpour, M. “Hybrid ARIMA and neural network model for measurement estimation in energy-efficient wireless sensor networks”. In: *International Conference on Informatics Engineering and Information Science*. Springer. 2011, pp. 35–48.
- [129] Aderohunmu, F. A., Paci, G., Brunelli, D., Deng, J. D., Benini, L., and Purvis, M. “An application-specific forecasting algorithm for extending wsn lifetime”. In: *2013 IEEE international conference on distributed computing in sensor systems*. IEEE. 2013, pp. 374–381.
- [130] Fathy, Y., Barnaghi, P., and Tafazolli, R. “An adaptive method for data reduction in the internet of things”. In: *2018 IEEE 4th World Forum on Internet of things (WF-IoT)*. IEEE. 2018, pp. 729–735.
- [131] Fathy, Y. and Barnaghi, P. “Quality-based and energy-efficient data communication for the internet of things networks”. In: *IEEE Internet of Things Journal* vol. 6, no. 6 (2019), pp. 10318–10331.
- [132] Laidi, R., Djenouri, D., and Balasingham, I. “On Predicting Sensor Readings With Sequence Modeling and Reinforcement Learning for Energy-Efficient IoT Applications”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2021).
- [133] Ng, A., Harada, D., and Russell, S. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287.

-
- [134] Rioual, Y., Moullec, Y. L., Laurent, J., Khan, M. I., and Diguët, J.-p. “Reward Function Evaluation in a Reinforcement Learning Approach for Energy Management”. In: *2018 16th Biennial Baltic Electronics Conference (BEC)*. 2018, pp. 1–4.
- [135] Fisher, R. A. “On the mathematical foundations of theoretical statistics”. In: *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* vol. 222, no. 594–604 (1922), pp. 309–368.
- [136] Kullback, S. and Leibler, R. A. “On information and sufficiency”. In: *The annals of mathematical statistics* vol. 22, no. 1 (1951), pp. 79–86.
- [137] Padhy, P., Dash, R. K., Martinez, K., and Jennings, N. R. “A utility-based adaptive sensing and multihop communication protocol for wireless sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 6, no. 3 (2010), pp. 1–39.
- [138] Zhang, J., Li, Z., and Tang, S. “Value of information aware opportunistic duty cycling in solar harvesting sensor networks”. In: *IEEE Transactions on Industrial Informatics* vol. 12, no. 1 (2015), pp. 348–360.
- [139] Abbeel, P. and Ng, A. Y. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.
- [140] Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.
- [141] Wulfmeier, M., Ondruska, P., and Posner, I. “Maximum entropy deep inverse reinforcement learning”. In: *arXiv preprint arXiv:1507.04888* (2015).
- [142] Finn, C., Levine, S., and Abbeel, P. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International conference on machine learning*. PMLR. 2016, pp. 49–58.
- [143] Ross, S., Gordon, G., and Bagnell, D. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings*. 2011, pp. 627–635.
- [144] Yamin, N. and Bhat, G. “Near-Optimal Energy Management for Energy Harvesting IoT Devices using Imitation Learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).
- [145] Salge, C., Glackin, C., and Polani, D. “Empowerment—an introduction”. In: *Guided Self-Organization: Inception*. 2014, pp. 67–114.
- [146] Gregor, K., Rezende, D. J., and Wierstra, D. “Variational intrinsic control”. In: *arXiv preprint arXiv:1611.07507* (2016).
- [147] Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. “Dynamics-Aware Unsupervised Discovery of Skills”. In: *International Conference on Learning Representations*. 2019.

5. Concluding remarks

- [148] Campos, V., Trott, A., Xiong, C., Socher, R., Giro-i-Nieto, X., and Torres, J. “Explore, discover and learn: Unsupervised discovery of state-covering skills”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1317–1327.
- [149] Braten, A. E. and Kraemer, F. A. “Towards Cognitive IoT: Autonomous Prediction Model Selection for Solar-Powered Nodes”. In: *2018 IEEE International Congress on Internet of Things (ICIOT)*. Seattle, USA, 2018, pp. 118–125.
- [150] Box, G. E. “Science and statistics”. In: *Journal of the American Statistical Association* vol. 71, no. 356 (1976), pp. 791–799.
- [151] Igl, M., Ciosek, K., Li, Y., Tschitschek, S., Zhang, C., Devlin, S., and Hofmann, K. “Generalization in reinforcement learning with selective noise injection and information bottleneck”. In: *Advances in neural information processing systems* vol. 32 (2019).
- [152] Lu, X., Lee, K., Abbeel, P., and Tiomkin, S. “Dynamics generalization via information bottleneck in deep reinforcement learning”. In: *arXiv preprint arXiv:2008.00614* (2020).

Part 2

Publications

Paper I: Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning

Abdulmajid Murad¹, Frank Alexander Kraemer¹, Kerstin Bach¹, Gavin Taylor²

¹Norwegian University of Science and Technology, ²United States Naval Academy

In proceedings of *IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Umeå, Sweden, 6-20 June 2019, pp. 43-51, DOI: 10.1109/SASO.2019.00015.

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

Abstract

Reinforcement learning (RL) is capable of managing wireless, energy-harvesting IoT nodes by solving the problem of autonomous management in non-stationary, resource-constrained settings. We show that the state-of-the-art policy-gradient approaches to RL are appropriate for the IoT domain and that they outperform previous approaches. Due to the ability to model continuous observation and action spaces, as well as improved function approximation capability, the new approaches are able to solve harder problems, permitting reward functions that are better aligned with the actual application goals. We show such a reward function and use policy-gradient approaches to learn capable policies, leading to behavior more appropriate for IoT nodes with less manual design effort, increasing the level of autonomy in IoT.

1.1 Introduction

Digitizing domains like smart cities, precision agriculture, manufacturing, and healthcare requires the instrumentation of the physical world with wireless IoT nodes to sense environmental variables. The actual logic of an IoT device can be rather simple, as they usually make measurements, prepare the data and send it into a central fusion center for aggregation. However, challenges arise from the limited energy resources of the wireless and miniaturized nodes. Therefore, the choice of parameters like duty cycle (percentage of time a node is active) and sampling frequency is crucial since they have a dominant effect on the energy consumption and performance of the node. Currently, such parameters are often assigned manually to an entire family of IoT devices, and stay constant during deployment, or are determined by manually fine-tuned algorithms. However, IoT in its full scale presents a setting where nodes are placed in dynamic and non-stationary environments, so that a one-fits-all approach and manual design efforts are impossible. Future solutions should instead be based on autonomous nodes that optimize their behavior by learning within their specific environment.

Our vision for a more autonomous and hence scalable IoT is inspired by recent developments in deep reinforcement learning (RL), and its success in building capable autonomous agents with little manual design effort in complex domains. Instead of manually adapting IoT nodes, we want engineers to only formulate the goal in the form of a reward function for the IoT nodes as close to the actual application goal as possible.

Reinforcement learning has indeed been used to control IoT nodes by various works, especially for power management [14, 13, 96]. However, these approaches use older RL techniques like SARSA and are constrained by coarsely discretized action and observation spaces, and tackle only simpler problems. Many approaches [95, 97, 12] thus formulate reward functions in terms of indirect indicators like energy budget. Those are often easier to learn, but are imperfectly correlated with actual goals. This not only adds manual design effort, but also doesn't live up to the potential of RL. Instead, RL should be used to sort out the complicated and node-specific technicalities on its own through learning.

In this paper, we explore deep RL as an end-to-end approach to enable autonomy within IoT nodes. For the training, we use a policy gradient method, namely Proximal Policy Optimization (PPO, [36]). We use neural networks as function approximators to learn state abstractions and effective policies directly from continuous input data. We also develop a reward function that closely reflects application goals. We automated the search for the best reward signal by defining it as hyperparameters from the application perspective. Additionally, we conduct a series of simulations to explore the influence of reward function design on a node's behavior.

To manage the computational complexity, we propose training the agent on a server as a part of device management [149] and update the node regularly based on a static or dynamic update interval [14]. To this end, we built a sensor simulator, called Sensor Gym (based on OpenAI Gym [153]), as a toolkit for training and comparing various RL algorithms in an IoT context.

The results in this paper show that modern and more successful deep RL techniques outperform the older ones, and that they make it possible to use more sensible reward functions that are based on IoT application goals. This leads to tractable learning, less manual design efforts, and hence scalable autonomy for IoT systems.

The rest of this paper is organized as follows: Sect. II provides a brief overview of deep reinforcement learning, and Sect. III presents related work that adopted RL to optimize energy-harvesting sensor nodes. In Sect. IV, we describe the proposed system setup, and in Sect. V we present a baseline for performance and suitability of PPO in an IoT by comparing it to an older RL technique and study the performance with a reward function based on energy neutrality. We then introduce a reward function closer to the application goals in Sect. VI, and present the results in Sect. VII.

1.2 Deep Reinforcement Learning

In reinforcement learning an agent learns to better perform a task by learning from its experiences in interacting with that task. Mathematically, the task is expressed with a Markov Decision Process (MDP). An MDP is a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where \mathcal{S} is the set of all states the problem might be in; \mathcal{A} is the set of all actions the agent might take in response; \mathcal{P} is a transition kernel describing the probabilities of transitioning between two states when performing an action ($p(s'|s, a) \forall s, s' \in \mathcal{S}, a \in \mathcal{A}$); $R : \mathcal{S} \rightarrow \mathbb{R}$ is a reward function (for example, positive numbers in states where the agent has achieved its goal, and negative numbers in states where the agent has harmed itself); and $\gamma \in (0, 1)$ is a discount factor, which is used to describe how much the agent prefers rewards in the short term to rewards in the long term. The behavior of the agent is expressed as a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

The quality of a policy can be measured using the value function of a state $V^\pi(s)$, or, the Q-value of a state-action pair $Q^\pi(s, a)$:

$$V^\pi(s) = R(s) + \int_{\mathcal{S}} p(s'|s, \pi(s)) V^\pi(s') ds' \quad (\text{I.1})$$

$$Q^\pi(s, a) = R(s) + \int_{\mathcal{S}} p(s'|s, a) Q^\pi(s', \pi(s')) ds' \quad (\text{I.2})$$

The goal of the agent is to learn a policy which produces high value for all states by collecting trajectory samples of the form (s_t, a_t, r_t, s_{t+1}) , where t denotes the timestep, and $s_t, s_{t+1} \in \mathcal{S}, a_t \in \mathcal{A}, r_t = R(s_t)$.

A variety of learning algorithms for RL exist. One traditional approach is to begin by learning Q-functions as accurately as possible using function approximation techniques; some are used in previous work applying RL to IoT, and will be introduced here.

Samples are themselves collected using a policy π_{samp} , where $a_t = \pi_{\text{samp}}(s_t)$. The Q-functions of this sampling policy can be learned using an algorithm called SARSA; this is known as *on-policy* learning. Alternatively, the Q-functions of an improved policy may be learned using an algorithm known as Q-learning; this is known as *off-policy* learning. In either case, given Q-function approximations, a new and likely-improved policy π^* is implied by $\pi^*(s) = \arg \max_{a \in \mathcal{A}} Q^\pi(s, a)$. With sufficient sampling, alterations to allow for exploration of the full state space, and (in the on-policy case) alternation between sampling, Q-function approximation and policy improvement steps, these approaches are proven to converge to excellent policies. This is a convenient guarantee, but requires the state and action spaces to be finite and small enough to be easily tabulated. More complex approaches to value learning exist which allow state and action spaces to be continuous, but this results in a loss of these guarantees, and potentially much less capable policies. In recent years, these approaches have been further improved upon by approximating functions with deep neural networks [17], but they remain fundamentally difficult to apply successfully to continuous and complex domains.

In contrast with value-based methods, policy-search methods directly search for a parameterized policy π_θ . The result is a stochastic policy that learns a direct mapping from states to a probability distribution over actions, where preferred actions have a higher probability of being sampled. This output of a distribution makes it suitable for continuous domain tasks.

Policy gradient methods are the most prominent of policy-search methods. They optimize a policy by maximizing the value of sampled states $V^{\pi_\theta}(s_t)$ by performing gradient ascent over the parameters θ . Various approximations to this basic approach exist, with advantages in training time and sample efficiency. In this paper, we use Proximal Policy Optimization (PPO), one of the most successful of these approaches[36]. PPO is a trust region method, which seeks to iteratively maximize performance of π_θ compared to its previous iteration, without changing it too much. This allows for continuous, stable improvement, even in continuous and complex domains such as robotic running and difficult Atari games. This success in demanding domains make RL ready for application to real-world problems such as IoT.

I.3 Related Work

Several works have already adopted RL within various contexts of IoT. An example is a dynamic power manager for energy-harvesting wireless sensor networks [7]. Here Hsu et al. used the Q-learning algorithm to train an agent to choose an action from

four levels of duty cycles. Their environment state space is based on the distance from energy neutrality (i.e., the difference between harvested and consumed energy), the harvested energy, and the current battery level, while the reward function is based on the distance from energy neutrality and the current energy storage level. They extended this work in [94] to include quality of service (QoS), both in the state space and the reward function. Additionally, in [96], they included meeting a requested throughput along with the energy level in the reward function and added penalizing terms for overcharging, deep-discharge, and depletion of the energy storage. Furthermore, in [95] and [97], they used fuzzy decision processes to model an energy harvesting node as a fuzzy environment and used a modified Q-learning with fuzzy reward to train an RL agent.

Since the reward function formalizes the goal of an RL setup, Rioual et al. [134] discussed the choice of the reward function in the management of energy-harvesting IoT nodes but covered a limited range of design choices. To avoid intractable learning in old RL approaches when using an application-level reward, most of the previous work used reward shaping, which is an alternative method to guide the learning process by rewarding the agent for achieving subgoals or developing an approximation to the desired behaviors. Unless the reward shaping function is based on a state-dependent potential, it may lead to learning suboptimal or locally-optimal policies [133]. Almost all previous work of using RL in IoT uses manually-designed shaped functions that produce arguably acceptable results without justifying how well the reward frames the application goal.

RL has also been used for power allocation in energy-harvesting communication systems. Ortiz et al. [154] used the SARSA algorithm with linear function approximation to learn a power allocation policy in two-hop communication. The objective of their policy is to maximize throughput of a communication system, but they designed a reward function based on the total power assigned to transmissions, which they claimed was correlated. They simulated a communication environment and approximated the state space with discrete features that indicate battery level and its constraints, harvested energy over an hour, characteristics of the communication link, data arrival process, and the data buffers at the communicating nodes. Similarly, Aoudia et al. [12], used an actor-critic method with linear function approximation to learn approximation for both policy and value function. They used a Gaussian policy to generate continuous values of bounded packet rates and summarized the state space by continuous values of the current residual energy. The objective is to maximize transmitted packet rate while sustaining perpetual operation; hence they designed a reward function to be a multiplication of normalized residual energy and the packet rate.

Shresthamali et al. [13] used a SARSA(λ) RL algorithm to develop adaptive power management for a solar-energy harvesting sensor node. To simulate a sensor node, they used a scaled-up version of a real sensor powered by a battery and a solar panel, and used solar radiation data to calculate hourly harvested energy. They designed a reward function based the distance from energy neutrality, defined as the difference between the current level of energy and the optimum battery level, which they calculated to be 60% of the battery's maximum level. They trained an agent in episodes of 24 hours and rewarded it at the end of a training episode. The state space consisted of discretized information about the battery level, the

distance from energy neutrality, the harvested energy, and the weather forecast, which they approximate by calculating the total amount of energy harvested in a particular day.

Fraternali et al. [14] focused on the configuration of the sampling rate of indoor energy harvesting sensors. The objective of their agent is to maximize the number of samples while avoiding power failure, by designing a reward function that depends on the sampling rate and a penalty for power failure. They used Q-learning with a state space that included light intensity, the voltage level of the energy storage, and the current sampling rate. Another example of using RL for adaptive sampling can be found in [115]. Here Dias et al. proposed using Q-learning for adaptive sampling rate adaptation to avoid oversampling, while not missing environmental changes. They introduced a variable that describes the quality of measurement as the difference between two consecutive measurements and, depending on a specific application, this difference should be less than a threshold value. The action space is a range of possible sampling intervals, while the reward function is based on the transmission avoided and the quality of measurements.

I.4 System Setup

In this paper, we propose and apply deep RL solutions for training RL agents to control IoT nodes autonomously. We use PPO as a policy gradient method for optimizing the agent’s policy, using neural networks as function approximators. Executing the agent’s policy corresponds to inference in a neural network which has become feasible even for computationally- and energy-constrained IoT nodes [155]. We propose to integrate the much more computation-intensive training phase on a server, as a part of the IoT device-management [149], illustrated in Fig. I.1. The deployed agents can be updated regularly based on a static or dynamic update interval [14].

To train the agent, we built a simulator of a general sensor node with an energy-harvesting solar panel, an ideal energy buffer, and a load with variable duty-cycles. We base the simulator on the OpenAI Gym, which is a toolkit for developing and comparing RL algorithms [153]. We base our sensor specification on a realistic energy-harvesting IoT node. The simulated sensor has an energy buffer with a maximum capacity of $B_{max} = 40$ W and has a load that consumes energy during a time step according to its duty cycle. The maximum consumed energy in an hour is $Ec_{max} = 0.5$ W h, which corresponds to a duty cycle of $D_{max} = 100\%$ during that hour. We simulate the harvested energy by calculating the energy generated by a 6 W solar panel using solar radiation data. At time step t , we represent the level of the energy buffer as B_t , the harvested energy as Eh_t , the consumed energy as Ec_t , and the duty cycle, which is determined by the node’s policy, as D_t . The consumed energy is calculated according to:

$$Ec_t = \begin{cases} 5D_t & \text{if } B_t > 5D_t \\ 0 & \text{if } B_t \leq 5D_t \end{cases} \quad (\text{I.3})$$

The next level of the energy buffer is calculated according to:

$$B_{t+1} = \min(B_t + Eh_t - Ec_t, B_{max}). \quad (\text{I.4})$$

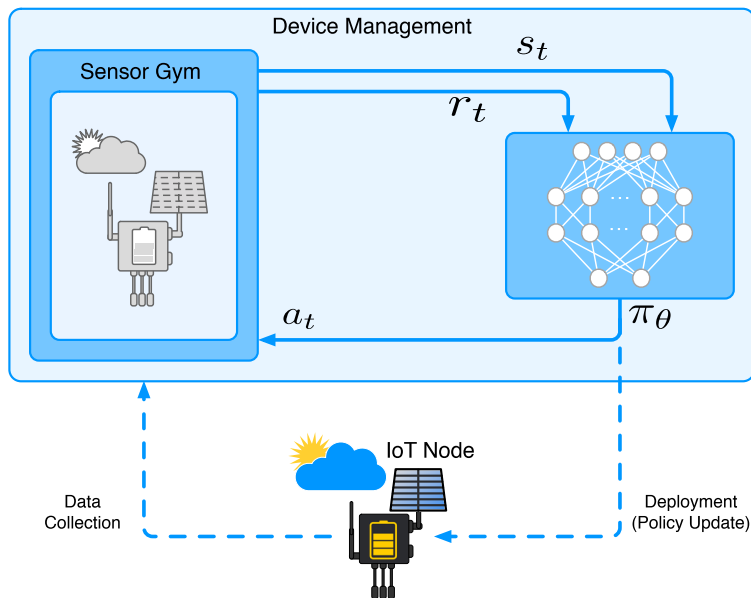


Figure I.1: System setup of agent-environment interaction

To handle a continuous action space $\mathcal{A} = [0, D_{max}]$, the policy resulting from PPO is defined as a Gaussian distribution with mean $\mu(s_t)$ and standard deviation $\sigma(s_t)$, both of which are approximated by a neural network with parameters θ :

$$\pi_\theta(a_t|s_t) = \frac{1}{\sigma(s_t)\sqrt{2\pi}} \exp\left(-\frac{(a_t - \mu(s_t))^2}{2\sigma(s_t)^2}\right) \quad (I.5)$$

Thus, the actions are real numbers, chosen from the normal distribution in I.5. Our neural network architecture has an output layer with two neurons which are the approximation of the mean and standard deviation of the Gaussian policy.

In Section I.5, we demonstrate that PPO outperforms older RL approaches on previously-proposed reward functions. Then, in Section I.6, we demonstrate that PPO can develop good policies for reward functions that make the domain more difficult, but better capture desired behavior. We also illustrate that older RL approaches cannot produce similarly competent policies.

I.5 Reward Function Based on Energy Neutrality

To get a baseline for the performance and suitability, we first apply PPO in the same setting as Shresthamali et al. [13], who used the SARSA(λ) algorithm and designed a reward function based on the distance from energy neutrality. The concept of energy-neutrality was introduced by Kansal et al. [89], which states that a node is in energy-neutral operation if the consumed energy is less than or equal to the harvested energy. Accordingly, the distance from energy neutrality is redefined as the difference between the current level of the energy buffer B_t and the optimum

buffer level B_0 to account for the variance in the harvested energy over a period:

$$Edist_t = |B_t - B_0|. \quad (I.6)$$

In [13], the reward function is formulated in terms of this distance to energy neutrality:

$$R_E(s_t) = \begin{cases} 500 & \text{if } Edist_t = 0 \text{ W h} \\ 500 - \frac{Edist_t}{10} & \text{if } 0 \text{ W h} < Edist_t \leq 1 \text{ W h} \\ 10 - \frac{Edist_t}{100} & \text{if } 1 \text{ W h} < Edist_t \leq 5 \text{ W h} \\ -500 & \text{if } 5 \text{ W h} < Edist_t \end{cases} \quad (I.7)$$

The action space is defined as discrete values of five duty cycles, which are

$$a_t \in \mathcal{A} = \{20\%, 40\%, 60\%, 80\%, 100\%\} \quad (I.8)$$

The observation space contains vectors with four observations:

$$s_t = [B_t, Edist_t, Eh_t, Wf_{day}], \quad (I.9)$$

where Wf_{day} represents the weather forecast of the day. In [13], actual harvested energy data in a day are calculated before the training and used to give information about the expected weather. The agents can leverage this information to plan their energy expenditure accordingly.

While the state space requires discretization for SARSA with a hand-designed mapping, policy approximation has the advantage of handling continuous state spaces by generalization, hence eliminating another step of manual mapping an IoT problem to RL.

To compare the capabilities of SARSA and PPO, we trained 20 agents with PPO and 20 with SARSA with data of Tokyo 2010 and evaluated their policies on data of Tokyo 2011, using the same settings and data as [13]. Figure I.2 represents a histogram for performance results of both policies. Here, the root mean square (RMS) of $Edist_{day}$ values are used to measure the deviation from energy-neutrality after a one-day window and the mean of daily deviation over the whole year to compare policies. We observe that the average PPO agent is considerably better than the best SARSA policy and that almost all PPO agents are better than the average SARSA agent. These excellent results support our motivation to use PPO for IoT agents.

We also take a more detailed look at the behavior of the agents by comparing their results over a one-week window. Figure I.3 shows the performance results of PPO, SARSA policies, and linear programming when running them over one week of Tokyo weather data in February of 2011. The linear programming policies are presented in [89], and use linear programming and acausal data (i.e., the exact future energy intake, hence not practical) to determine the optimal duty cycle given the energy neutrality constraint; this represents the upper limit of performance. The three policies started the week off with an energy level of $B_t = 60\%$ of B_{max} and try to maintain them at this level. RMS $Edist_{day}$ values are used to measure the deviation from energy-neutrality over the whole week, and we observe that the SARSA-trained policies have the highest deviation of (5.59%), PPO-trained policies have (5.43%) and the optimal policy based on linear programming has (4.17%).

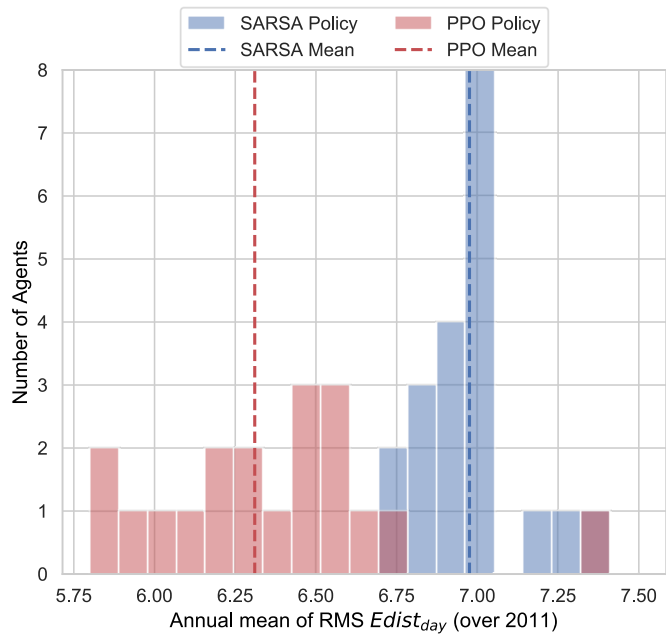


Figure I.2: Performance comparison of 20 agents trained with PPO and SARSA(λ)

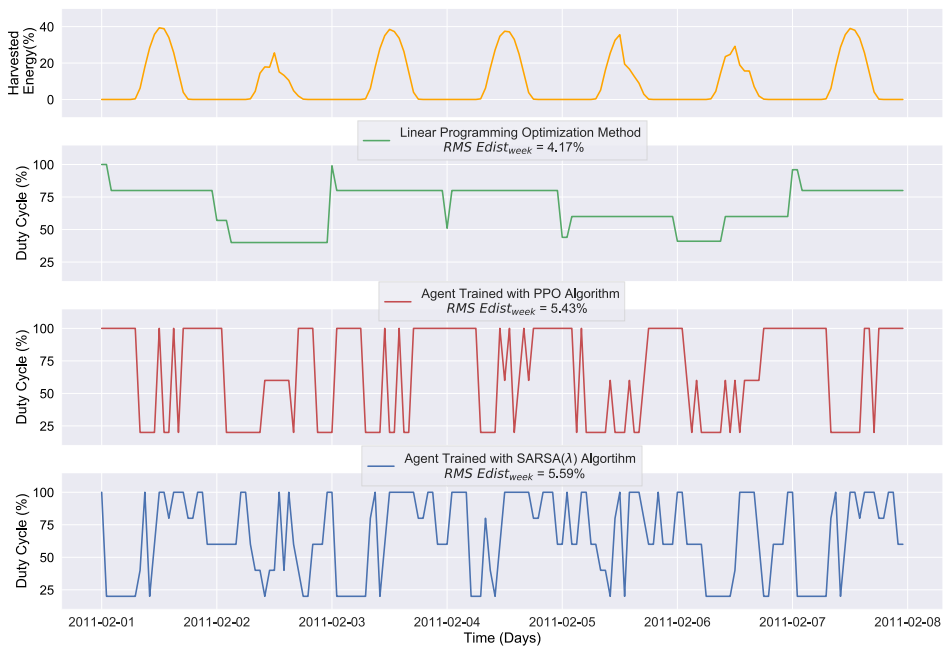


Figure I.3: Comparison of PPO policy to SARSA(λ) and offline policy for a week of Tokyo 2011

Figure I.3 illustrates a problem with the reward function based on distance from energy neutrality (I.7): The duty cycles of both RL policies are subject to high variance, often oscillating between highest and lowest duty cycle, instead of choosing a smoother course with more mid-range values. From an RL point-of-view, this maximizes the reward, but it is not a behavior appropriate for IoT nodes. Typically, we want to cover a phenomenon as continuously as possible, and spread out measurements over time, which corresponds to a smoother duty cycle.

Because PPO is so capable on this easier problem, in the next section we design a new, more difficult reward function, which no longer rewards the intermediate goal of energy neutrality, but more clearly expresses the goals for the agent.

I.6 Reward Function Based on Application Goals

Our objective in this section is to create a reward function that is closer to the actual application goals of an IoT system in order to encourage more desirable behavior. In particular, we want to depart from the inclusion of energy terms in the reward function, since energy is only a resource to be managed, but not a goal to be optimized. Instead, we want the agent to maximize the sum of the duty cycle over time Γ :

$$G_D = \sum_{t=0}^{\Gamma} D_t. \quad (\text{I.10})$$

At the same time, IoT nodes must not empty their energy buffer completely. Not only would this prevent the node from taking any measurements until enough energy is harvested again, but it could also lead to the loss of data or leave the node in an undefined state. Since the sensor gym simulates a failure when the idealized buffer is emptied, we want to minimize the occurrence of failures over time Γ

$$G_F = \sum_{t=0}^{\Gamma} \begin{cases} 1 & \text{if } B_t = 0 \\ 0 & \text{if } B_t > 0 \end{cases} \quad (\text{I.11})$$

To ensure a smooth course of the duty cycle, corresponding to a continuous stream of measurements, we want to minimize the variance of the selected duty cycles, and so minimize

$$G_{Var} = \sum_{t=0}^{\Gamma} Var_t, \quad (\text{I.12})$$

where the variance in the duty cycle is defined in an epoch t as the absolute difference to the previous duty cycle

$$Var_t = |D_t - D_{t-1}|. \quad (\text{I.13})$$

We combine the conflicting objectives (I.10), (I.11), (I.12) into one reward function R_A :

$$R_A(s_t) = \begin{cases} D_t - \zeta [Var_t]^2 & \text{if } B_t > 0 \\ -F & \text{if } B_t = 0 \end{cases} \quad (\text{I.14})$$

The agent receives its reward by maximizing the duty cycles D_t . To punish variance, we reduce the reward with the squared variance, scaled by a damping factor ζ . In the case of a failure, the reward is negative, using the punishment term F . The

actual values of F and ζ are hyperparameters of the reward function, and we will have a detailed look at them in the next section.

We generalized the reward function in (I.14) to accommodate other RL techniques of sparse reward assignments. In these techniques, the agent gets a reward at the end of a training episode E . An episode consists of a set of epochs (time steps) that range from $t = 1$ to $t = T$, thus $E = \{1, 2, \dots, t, \dots, T\}$

$$R_A(s_T) = \sum_{t=1}^T \begin{cases} D_t - \zeta[\text{Var}_t]^2 & \text{if } B_t > 0 \\ -F & \text{if } B_t = 0 \end{cases} \quad (\text{I.15})$$

If the reward is assigned at every epoch, $T = 1$ and (I.15) reduces to (I.14).

We considered the relevant attributes of the environment in the definition of state space. These attributes are observations of the current level in the energy buffer, current harvested energy, and the weather forecast of the whole episode. To simulate the harvested energy, we use solar radiation data to calculate the harvested energy in every hour. The weather forecast information is included to enhance performance by giving an estimate of the expected solar energy for that particular episode. This weather information can be acquired from external sources or real prediction algorithms with sufficient accuracy as the one presented in [156]. For our case study, we simulate weather information by calculating the total harvested energy in a particular day and introduce a 20% error to mimic inaccuracies in the weather forecast. We also include the previous duty cycle D_{t-1} in the state space, so the agent makes an informed decision to avoid high variability in the duty cycle. The agent takes the observation as input summarized in a vector of four continuous values

$$s_t = [B_t, Eh_t, Wf_t, D_{t-1}]. \quad (\text{I.16})$$

PPO allows us to use a continuous action space corresponding to setting the operational duty cycle of a node. This enables more accurate control of the consumed energy, and therefore the utility of a node. Therefore, the action space is

$$a_t \in \mathcal{A} = [D_{min}, D_{max}]. \quad (\text{I.17})$$

Depending on IoT application requirements, the minimum value of the duty cycle can be set accordingly. For our case, we set it to $D_{min} = 0$.

1.7 Results and Discussion

We conducted a series of simulations to systematically explore the influence of the designed reward function R_A and training hyperparameters on the agent behavior. In total, we trained more than 300 agents on data of the year 2010 and tested their performance on data of the year 2011, using different values for the damping factor (ζ) for the reward function. For the PPO algorithm, we also trained with different values for the hyperparameters, namely the learning rate (α), batch size (number of state transitions used to calculate policy gradient),

discount factor (γ) and trace decay parameter (λ) of the advantage function [36]. In the following, we discuss the learning rate α and damping factor ζ in more detail.

The learning rate (α) has a significant impact on the learning process as shown in Fig. I.4. The x-axis corresponds to the learning rate, which we chose to tune in the range of 10^{-5} to 10^{-1} . The y-axis in the upper plot corresponds to the yearly utilized energy (normalized by the maximum possible utilization), the y-axis in the lower plot corresponds to the mean-variance in duty cycle over the whole year. Each dot in the figure represents a single agent, and its color indicates the number of times the agent has emptied the energy buffer, i.e., failed. Training with low learning rates ($\alpha < 10^{-3}$) results in agents that learn poorly or not at all. These agents pick constant and low duty cycles, which results in low variance and few power failures, thus fulfilling application goals (I.11) and (I.12), but failing to achieve the goal of maximizing utilized energy in (I.10).

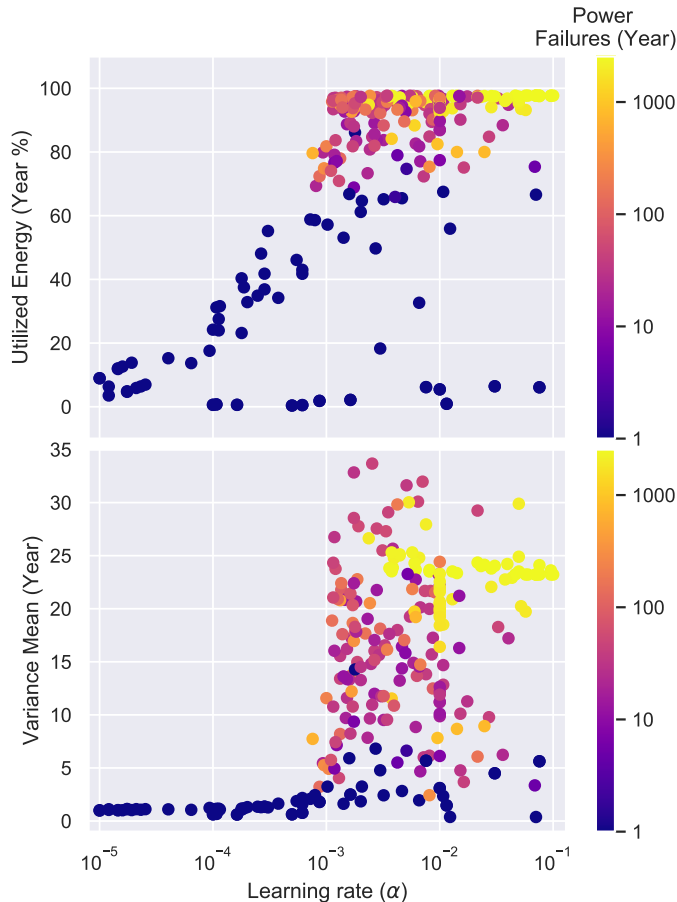


Figure I.4: Influence of the learning rate (α) on agent's learning process in terms of variance in duty cycle and utilized energy.

The choice of damping factor ζ in the reward function also has a significant impact on the resulting behavior. Figure I.5 shows the results of all trained agents running over the whole year of 2011 for different damping factors ζ on the x-axis. We observe that agents trained with values of ζ in the approximate range

($10^{-2} < \zeta < 10^{-1}$) display the desired behavior of less variability and better energy utilization, which translates to an appropriate, continuous measurement coverage of the IoT node. We also observe that training with ζ in the approximate range ($\zeta \geq 10^{-1}$) leads to overdamped agents, which have less variance and fewer power failures, but also less energy utilization. Contrarily, most agents trained with values of ζ in the approximate range ($\zeta \geq 0.1$) are underdamped, with few exceptions as outliers. These underdamped agents have high utilized energy, but also higher variability in the duty cycle and more frequent power failures.

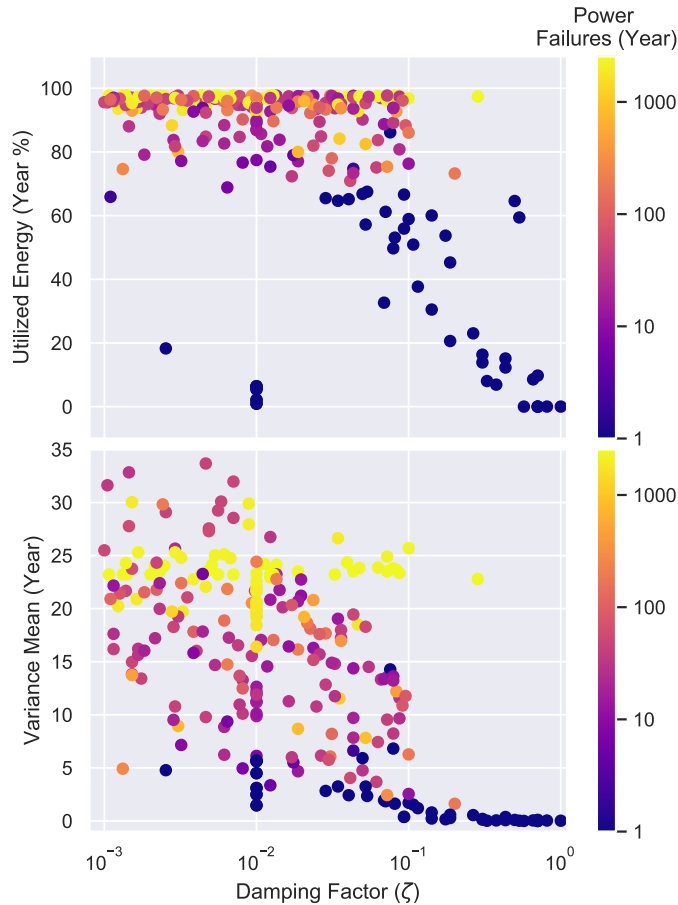


Figure I.5: Influence of the damping factor (ζ) in $R_A(t)$ on an IoT node's behavior in terms of utilized energy and variance in duty cycle.

Depending on the choice of damping factor ζ and failure penalty F in the reward function R_A , agents try to maximize rewards by prioritizing among the conflicting objective (I.10), (I.11), (I.12). Accordingly, agents learn various policies that achieve different performance on each objective, as shown in Fig. I.6. This enables choosing between different trade-off agents based on the context of individual IoT applications.

In comparison, using the same reward function R_A for SARSA-trained agents,

achieves mediocre results at best, as shown in Fig. I.6. The SARSA agents (shown as crosses) score poorly in terms of energy utilization and have a high variance. We attribute their relatively low failure rate to low utilization. This is as expected since the discretization necessary for SARSA does not offer the granularity needed for effective learning.

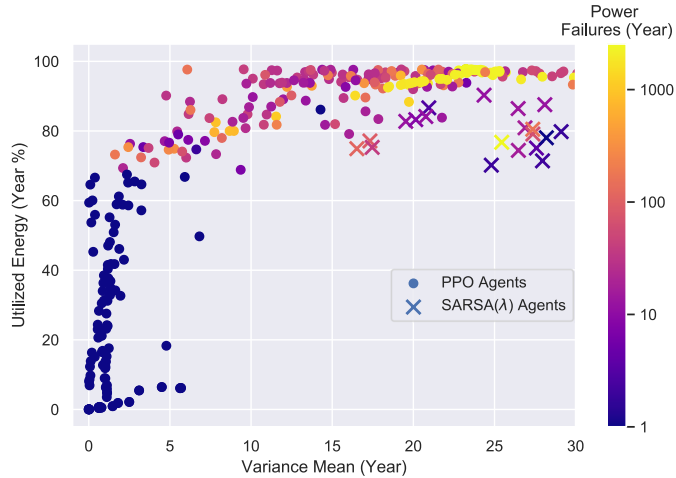


Figure I.6: Performance comparison of SARSA(λ) agents with PPO agents using the reward function R_A

We also take a more detailed look at the behavior over a one-week window. Figure I.7 shows the performance of four selected agents trained with different damping factors ζ . The first day yields much solar power, so all agents utilize this energy by setting their duty cycles to high values. Since the weather forecast for the next day indicates less energy, all agents reduce their duty cycles. However, the pattern of the decrease differs according to the damping factor. Efficiently damped agents show the desired gradual decrease, while underdamped agents increase sharply and oscillate between maximum and minimum duty cycles. When more energy is available again, we see analogous behavior when the duty cycles increase. The overdamped agents appear unaffected by the variance in energy supply, but they exhibit an overall low energy utilization. For example, the agent trained with $\zeta = 0.1$ avoids a high variance penalty by setting its duty cycle to a constant value, which leaves energy unutilized in times when much solar energy is available. Table I.1 summarizes the overall performance of the agents in Fig. I.7 over the whole year.

To investigate the effect of the neural network architecture of the policy approximation on performance, we trained with a different number of hidden layers and units. Our results show that shallower networks perform as well as deeper networks. We ended up using an architecture layout with two hidden layers of 64 units and a tanh activation function. The output layer has two units computing the mean and standard deviation of the Gaussian policy (I.5). Therefore, we substantiate that an agent's policy can be approximated with a neural network that requires low computational effort and memory footprint, which makes it feasible for deployment also in resource-constrained sensor nodes.



Figure I.7: Performance results over one week (Tokyo 2011) of four selected agents trained with different damping factors (ζ) in their reward function $R_A(t)$

Table I.1: Summary of performance results over a year of four selected agents trained with different damping factors (ζ)

ζ	Utilized Energy	Variance Mean	Power Failures
0.1	77%	2.5	11
0.05	94%	9.7	14
0.01	95%	12	17
0.001	98%	25.6	23

I.8 Conclusion

Using reinforcement learning (RL) to manage constrained IoT nodes provides a path to learn optimal behavior without careful human attention. To the best of our knowledge, this is the first application of a policy-gradient method to this problem. We have shown that state-of-the-art policy-gradient RL methods such as PPO which use neural networks as function approximators are suitable for the use in IoT, that they outperform older RL approaches, and that they can solve more difficult problems which better describe and encourage desired behavior. Additionally, their suitability for continuous problems removes another manual design step of discretization. We also investigated the influence of hyperparameters on the resulting policies. While we have focused on the maximization of duty cycle with minimal variance, we believe there is potential to solve much more complex problems. The work presented hence leads to more autonomy in IoT systems, in which RL takes care of technicalities so that engineers can focus on the real

application goals. This will allow a broader application of IoT in complex domains.

Acknowledgments

This research was supported by the Office of Naval Research (N0001418WX01582) and the Department of Defense High Performance Computing Modernization Program.

References

- [7] Chaoming Hsu, R., Liu, C. T., and Lee, W. M. “Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Vol. 5579 LNAI. 2009.
- [12] Aoudia, F. A., Gautier, M., and Berder, O. “RLMan: an Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks”. In: *IEEE Transactions on Green Communications and Networking* vol. 2, no. 2 (2018), pp. 1–1.
- [13] Shresthamali, S., Kondo, M., and Nakamura, H. “Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning”. In: vol. 16, no. 5s (2017), pp. 1–21.
- [14] Fraternali, F., Balaji, B., and Gupta, R. “Scaling configuration of energy harvesting sensors with reinforcement learning”. In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ACM. 2018, pp. 7–13.
- [17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. “Human-level control through deep reinforcement learning”. In: *Nature* vol. 518, no. 7540 (2015), p. 529.
- [36] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017), pp. 1–12. arXiv: [arXiv:1707.06347v2](https://arxiv.org/abs/1707.06347v2).
- [89] Kansal, A., Hsu, J., Zahedi, S., and Srivastava, M. B. “Power management in energy harvesting sensor networks”. In: *ACM Transactions on Embedded Computing Systems* vol. 6, no. 4 (2007), 32–es.
- [94] Hsu, R. C., Liu, C.-T., Wang, K.-C., and Lee, W.-M. “QoS-aware power management for energy harvesting wireless sensor network utilizing reinforcement learning”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 2. IEEE. 2009, pp. 537–542.
- [95] Liu, C. T. and Hsu, R. C. “Dynamic power management utilizing reinforcement learning with fuzzy reward for energy harvesting wireless sensor nodes”. In: *IECON Proceedings (Industrial Electronics Conference)* (2011), pp. 2365–2369.

- [96] Hsu, R. C., Liu, C. T., and Wang, H. L. “A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node”. In: *IEEE Transactions on Emerging Topics in Computing* vol. 2, no. 2 (2014), pp. 181–191.
- [97] Hsu, R. C., Lin, T. H., Chen, S. M., and Liu, C. T. “Dynamic energy management of energy harvesting wireless sensor nodes using fuzzy inference system with reinforcement learning”. In: *Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015* (2015), pp. 116–120.
- [115] Dias, G. M., Nurchis, M., and Bellalta, B. “Adapting sampling interval of sensor networks using on-line reinforcement learning”. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 2016, pp. 460–465.
- [133] Ng, A., Harada, D., and Russell, S. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287.
- [134] Rioual, Y., Moullec, Y. L., Laurent, J., Khan, M. I., and Diguët, J.-p. “Reward Function Evaluation in a Reinforcement Learning Approach for Energy Management”. In: *2018 16th Biennial Baltic Electronics Conference (BEC)*. 2018, pp. 1–4.
- [149] Braten, A. E. and Kraemer, F. A. “Towards Cognitive IoT: Autonomous Prediction Model Selection for Solar-Powered Nodes”. In: *2018 IEEE International Congress on Internet of Things (ICIOT)*. Seattle, USA, 2018, pp. 118–125.
- [153] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016), pp. 1–4. arXiv: [arXiv:1606.01540v1](https://arxiv.org/abs/1606.01540v1).
- [154] Ortiz, A., Al-Shatri, H., Li, X., Weber, T., and Klein, A. “Reinforcement Learning for Energy Harvesting Decode-and-Forward Two-Hop Communications”. In: *IEEE Transactions on Green Communications and Networking* vol. 1, no. 3 (2017), pp. 309–319.
- [155] *uTensor Project*. <https://github.com/utensor>. Accessed: 2019-02-28.
- [156] Kraemer, F. A., Ammar, D., Braten, A. E., Tamkittikhun, N., and Palma, D. “Solar energy prediction for constrained IoT nodes based on public weather forecasts”. In: *the Seventh International Conference*. New York, New York, USA, 2017, pp. 1–8.

Paper II: IoT Sensor Gym: Training Autonomous IoT Devices with Deep Reinforcement Learning

Abdulmajid Murad¹, Frank Alexander Kraemer¹, Kerstin Bach¹,
Gavin Taylor²

¹Norwegian University of Science and Technology, ²United States Naval Academy

In proceedings of the *9th International Conference on the Internet of Things*, Bilbao, Spain, 22–25 October 2019, pp. 1-4, DOI: 10.1145/3365871.3365911.



Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

IoT 2019, October 22–25, 2019, Bilbao, Spain

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7207-7/19/10.

<https://doi.org/10.1145/3365871.3365911>





Abstract

We describe *IoT Sensor Gym*, a framework to train the behavior of constrained IoT devices using deep reinforcement learning. We focus on the main architectural choices to align problems from the IoT domain with cutting-edge reinforcement learning algorithms and exemplify our results with the autonomous control of a solar-powered IoT device.

II.1 Introduction

With constrained resources and the need for scalable solutions, individual and autonomous control through behavior learning is crucial for IoT devices so that they can optimally act in dynamic and non-stationary environments. However, current IoT solutions often rely on manual, static configurations or fined-tuned algorithms that fail to suit all nodes of large-scale IoT systems. Fortunately, reinforcement learning emerged as a promising approach to automate IoT control, and it has already been used in various IoT applications [14] to solve simpler problems. Lately, deep reinforcement learning (DRL) proved to be effective in also solving harder problems due to its ability to model continuous observation spaces, as well as improved function approximation, which lead to its application in many fields, from continuous control tasks [157] to Atari games [158]. In this work, we explore how to apply DRL to the domain of IoT, inspired by the analogy between gaming and IoT, illustrated in Table II.2.¹

Table II.2: Analogy between gaming concepts and resource-constrained IoT devices.

Games	→	IoT
Player 	→	IoT Device Agent
World 	→	Weather and other relevant conditions.
Reward 	→	Data measurements
Control 	→	Duty cycle, sensing frequency,...

II.2 The Sensor-Gym Design

We built the IoT Sensor Gym as an extension to the OpenAI Gym framework [153]. Sensor Gym provides an environment specific to constrained IoT devices, with an emphasis on their energy budget. In DRL, an agent is built using a neural network and learns a policy through experience by interacting with the environment. At

¹The source code is available at <https://github.com/Abdulmajid-Murad/IoT-Sensor-Gym>. The code provides the functionality to train agents using OpenAI baseline implementations of reinforcement learning algorithms [159], including PPO [36]. Further, it provides functionality to evaluate the trained agents, as well as saving and restoring them.

each time step, the agent observes the environment’s state \mathbf{s}_t and selects an action \mathbf{a}_t . Depending on the state and the selected action, the agent receives a reward value \mathbf{R}_t . This reward is used for learning via various learning algorithms. One recent algorithm is Proximal Policy Optimization (PPO) [36]. It is a policy-gradient method able to solve also harder learning problems with little problem-specific engineering and is capable of using continuous values for states and actions.

Figure II.8 illustrates the architecture of the IoT Sensor Gym framework and the process of training and deploying RL agents. IoT devices are simulated using a variety of models which can be combined and configured to match various use cases. The energy-buffer model simulates an energy buffer of a maximum capacity \mathbf{B}_{max} , and provides the level of energy at each time-step \mathbf{B}_t . The energy-harvesting model calculates the harvested energy \mathbf{Eh}_t , using a selected solar panel and location-dependent solar radiation data. The energy-consumption model simulates a load that consumes energy at each time step \mathbf{Ec}_t according to a selected duty cycle \mathbf{D}_t (i.e., $\mathbf{Ec}_t \propto \mathbf{D}_t$). Accordingly, the next level of the energy buffer is given by $\mathbf{B}_{t+1} = \min(\mathbf{B}_t + \mathbf{Eh}_t - \mathbf{Ec}_t, \mathbf{B}_{max})$. The weather forecast model provides general information about the expected weather in terms of estimated solar energy, and it can be acquired from external sources or prediction algorithms as in presented in [156]. More specific models can be added to suite also other settings.

The *Update Step* calculation acts as a bridge between Sensor Gym and an external DRL agent to run one step of the environment’s dynamics. The interaction between Sensor Gym and an RL agent starts with creating a simulated IoT environment and initializing it with the default values. It also defines the boundaries of the environment, such as allowed actions (action space), the environment’s possible states (state space), as well as sampling and returning the first state. Then, the interaction proceeds through *Update Step* by running one epoch of the environment’s dynamics after receiving an action from the agent. The *training* of agents occurs off-board on a server, for instance as a part of the IoT device management [149]. The trained agents are then deployed to real IoT devices and updated regularly. The agent’s policies can usually be approximated with neural networks that require acceptable computational effort and memory footprint, so that they can also be deployed in modern, energy-efficient IoT devices. The models required for the simulation in the IoT Sensor Gym can be updated with data observed by the sensor device. This requires corresponding instrumentation, for instance by measuring its energy intake and consumption, indicated by the feedback in Figure II.8.

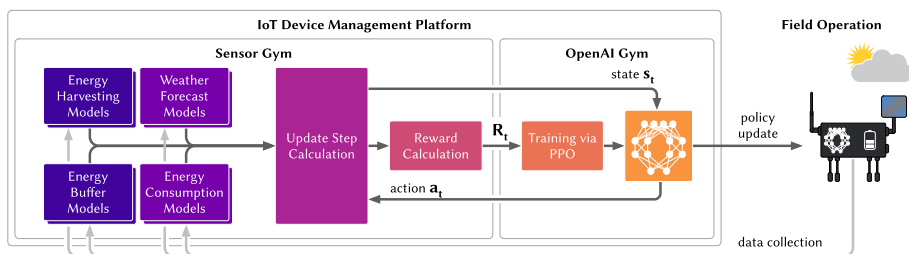


Figure II.8: Architecture of the IoT Sensor Gym as a framework to train the behavior of constrained IoT devices using deep reinforcement learning.

II.3 Use Case: Duty-Cycle Optimization Under Uncertain Energy-Harvesting

In this use case, a solar-powered IoT device with relatively small energy buffer should maximize its duty cycle, i.e., utilize as much of the incoming solar energy as possible, but without failing by depleting its buffer. At the same time, the duty cycle should have a low variance to ensure steady data collection. For that, we designed a corresponding reward function that reflects these application goals [98], using hyperparameters to balance between them. Figure II.9 shows the performance of more than 300 different agents over a whole year, using different hyperparameters.

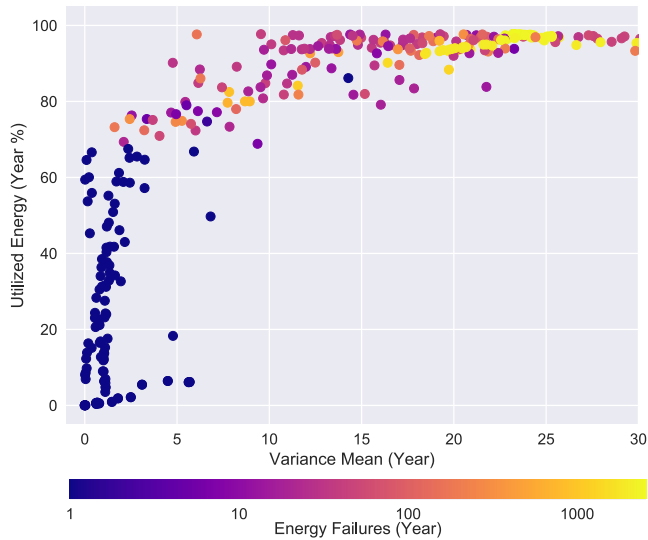


Figure II.9: Performance results of over 300 agents. The x-axis corresponds to the mean variance of the duty cycle, the y-axis to the yearly utilized energy. Each dot represents an agent, and the color indicates the number of times an agent has emptied its energy buffer, i.e., failed.

Figure II.10 shows the resulting duty cycles of two selected agents over six days. The first day is sunny, so the agents utilize this energy by selecting a high duty-cycle. Then, they reduce their duty cycles, since the weather forecast indicates less energy. The agents have different hyperparameters that model the tradeoff between energy utilization and low variance, hence illustrating the tradeoffs that the approach enables.

II.4 Conclusion

The IoT Sensor Gym provides the basic mapping of behavior control of constrained IoT devices using energy harvesting to cutting-edge reinforcement learning algorithms. This opens for the control IoT devices by autonomous agents that are

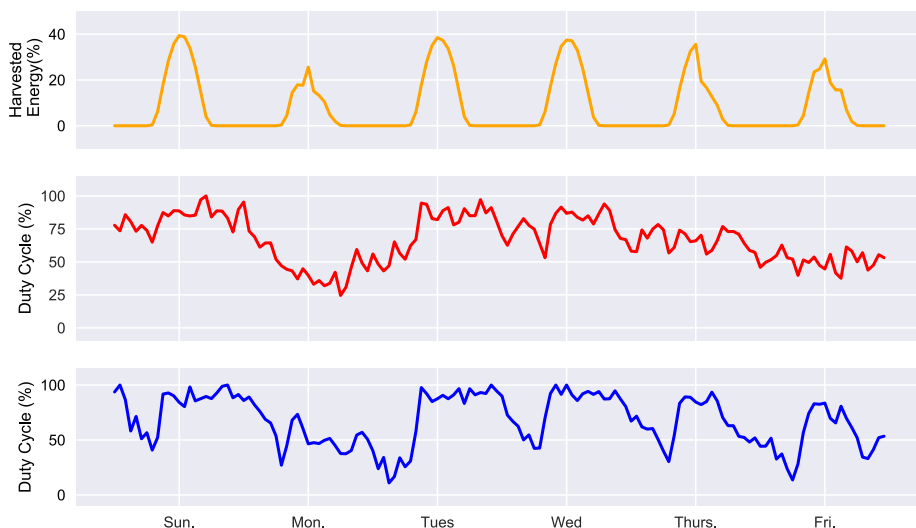


Figure II.10: Solar energy intake and resulting duty cycle selection of two agents. The red agent receives more penalty for duty cycle variance than the blue one and is hence smoother, but utilizes slightly less energy.

able to make complex decisions and learn in non-stationary environments also for more difficult problems.

References

- [14] Fraternali, F., Balaji, B., and Gupta, R. “Scaling configuration of energy harvesting sensors with reinforcement learning”. In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ACM. 2018, pp. 7–13.
- [36] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017), pp. 1–12. arXiv: [arXiv:1707.06347v2](https://arxiv.org/abs/1707.06347v2).
- [98] Murad, A., Kraemer, F. A., Bach, K., and Taylor, G. “Autonomous management of energy-harvesting IoT nodes using deep reinforcement learning”. In: *proceedings of IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Umeå, Sweden*. 2019.
- [149] Braten, A. E. and Kraemer, F. A. “Towards Cognitive IoT: Autonomous Prediction Model Selection for Solar-Powered Nodes”. In: *2018 IEEE International Congress on Internet of Things (ICIOT)*. Seattle, USA, 2018, pp. 118–125.
- [153] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016), pp. 1–4. arXiv: [arXiv:1606.01540v1](https://arxiv.org/abs/1606.01540v1).

-
- [156] Kraemer, F. A., Ammar, D., Braten, A. E., Tamkittikhun, N., and Palma, D. “Solar energy prediction for constrained IoT nodes based on public weather forecasts”. In: *the Seventh International Conference*. New York, New York, USA, 2017, pp. 1–8.
- [157] Duan, Y., Chen, X., Houthoofd, R., Schulman, J., and Abbeel, P. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *International Conference on Machine Learning*. ICML’16. New York, NY, USA, 2016, pp. 1329–1338.
- [158] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *Journal of Artificial Intelligence Research* vol. 47, no. 1 (May 2013), pp. 253–279.
- [159] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. *OpenAI Baselines*. <https://github.com/openai/baselines>. 2017.

Paper III: Information-Driven Adaptive Sensing Based on Deep Reinforcement Learning

Abdulmajid Murad¹, Frank Alexander Kraemer¹, Kerstin Bach¹, Gavin Taylor²

¹Norwegian University of Science and Technology, ²United States Naval Academy





Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Request permissions from permissions@acm.org.

IoT '20, October 6–9, 2020, Malmö, Sweden

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-8758-3/20/10. .

<https://doi.org/10.1145/3410992.3411001>

Abstract

In order to make better use of deep reinforcement learning in the creation of sensing policies for resource-constrained IoT devices, we present and study a novel reward function based on the Fisher information value. This reward function enables IoT sensor devices to learn to spend available energy on measurements at otherwise unpredictable moments, while conserving energy at times when measurements would provide little new information. This is a highly general approach, which allows for a wide range of use cases without significant human design effort or hyperparameter tuning. We illustrate the approach in a scenario of workplace noise monitoring, where results show that the learned behavior outperforms a uniform sampling strategy and comes close to a near-optimal oracle solution.

III.1 Introduction

A fundamental use case in the Internet of Things (IoT) is to represent a phenomenon of the real world with an estimation or prediction model informed by measurements done by IoT sensing devices. When the phenomenon is measured and predicted over time, the sensor devices therefore need to decide *when* they should make a measurement. More frequent measurements often allow for more accurate predictions; however, to spend their constrained resources as frugally as possible, sensor devices should avoid acquiring, processing, and transmitting measurements which are already accurately and confidently predicted by the representation model.

Generally, the optimal placement of measurements depends on the phenomenon to be observed. In noise monitoring of working environments, for example, measurements during nights or holidays are less relevant and more predictable than those during office hours. An optimal sensing strategy may also depend on the local environment and situation of each sensor individually, such as its energy budget. Learning these strategies is a complex task, which may also vary with each use case and specific sensor instance, due to local differences. We therefore see a need for more autonomy through learning, so sensor devices can exhibit appropriate behavior for their specific situation without human guidance [160, 161, 162, 163].

Reinforcement learning (RL) is one way to achieve such individual autonomous behavior. It has proven suitable for sequential decision-making under uncertainty, and has been applied in IoT and wireless sensor networks for a variety of problems [12, 11, 14, 7, 13, 98]. Current approaches, however, focus on generic performance measures, such as maximum duty cycle or optimal network traffic. Such metrics can be useful if they are well correlated with the goals the system should achieve, but such correlation is not guaranteed, and their optimization may diverge from optimizing the true goals of the system.

Instead, we consider a system in which a continuous prediction is made of a phenomenon only sporadically measured by the IoT device in its entirety. The goal is not to measure as frequently as possible, as in optimizing duty cycle or network activity; the goal is to make the prediction as accurate as possible. Therefore, we want to make the IoT device learn how to schedule measurements so that they contribute in times when the prediction is unconfident or inaccurate, contributing optimally to the quality of the overall prediction.

For that, we define a novel reward function inspired by the work of Kho et al. [113] that is based on the Fisher Information of the observed phenomena. This constitutes a novel and general approach for the application of RL in IoT that can be applied to a wide variety of use cases. Results show that the learned policies outperform uniform sampling for a case study with regard to model quality, and come close to oracle solutions.

In Sect. III.2, we will outline related work, before defining our system setup and problem in Sect. III.3. We then present our prediction model and evaluation criteria in Sect. III.4, which is the basis for the frugal operation policy presented in Sect. III.5. We close with an evaluation of our results and a discussion.

III.2 Related Work

Adaptive sampling addresses the problem of *when*, *where*, or *what* to sample, subject to some sampling budget constraints. This corresponds to, respectively, monitoring temporally-varying, spatially-varying, or multiple data types-varying environmental fields. Within the context of adaptive sampling, a large number of prior works have applied RL for power management of constrained, wireless sensors. In energy-harvesting sensors, one objective may be to increase the number of samples while maintaining perpetual operation [11, 14, 7, 13]. These works use tabular methods, such as SARSA or Q-learning, in which the state and action spaces are discrete and small enough to express the value function in tabular form. Other works [12, 154] use linear functions to approximate power-managing policies. Most of the previous works include energy directly in the reward function, while work in [98] proposes a utilitarian reward function that maximizes a system's utility based on the achieved duty cycle.

With respect to battery-powered sensors, the main objective is to minimize energy consumption by reducing the number of samples while maintaining an acceptable level of service. For example, Dias et al. [115] use a Q-learning algorithm to learn an adaptive sampling policy to minimize power consumption while not missing environmental changes. They define the action space to be a range of possible sampling intervals, and the reward function to be proportional to the amount of energy saved, subject to a constraint that the difference between consecutive measurements is less than a threshold value. Cobb et al. [116] apply RL to learn a sampling policy on accelerometer data on lions and show that it is feasible to achieve a reconstruction accuracy of 51 % with 73% reduction in energy consumption. They propose a reward function based on the sampling rate and data variance, then test tabular Q-learning, Deep Q-learning with NN, and Deep Q-learning with LSTM. They also present a heuristic sampling algorithm that outperforms these RL methods. However, they argue that using a heuristic algorithm would be restrictive and not very adaptable.

We observe an increasing attention in IoT towards incorporating prediction models with adaptive sampling, i.e., model-driven adaptive sampling. Ling et al. [123] address adaptive sampling when predicting an environmental field by presenting an adaptive GP planning framework. They integrate planning and learning by framing the problem as a Bayesian sequential decision with a value function. However, they do not derive an exact planning policy due to the

uncountable set of candidate measurements, and thus large possible sequences of posterior GPs. Instead, they use a Lipschitz-continuous reward function to derive an asymptotically-optimal policy. They evaluate their framework, among others, on a simulated energy-harvesting task in which a rover harvests wind energy while exploring a polar region. Muttreja et al. [121] model sensor data using sparse Gaussian processes and propose an active-learning based sampling algorithm. The objective of the sampling algorithm is to maintain the model confidence within pre-specified bounds while minimizing energy consumption. Similarly, Monteiro et al. [124] combine a data prediction model with adaptive sampling, where they build a model based on an extension of Holt’s Method. The algorithm greedily samples until it keeps the prediction error under a threshold value. Most similar to our work are those that use Gaussian processes in each sensor for prediction models and Fisher information as a basis for the sampling algorithms [113, 122]. In contrast to our work, where we use a learned sampling policy, these works use manually fine-tuned sampling algorithms.

In a somewhat different setting, some other works use variance-based adaptive sampling without prediction models [102, 103, 104]. They manually designed sampling algorithms that adjust the sampling rate according to the variation in the environment. Salim et al. [105] propose an algorithm that uses analysis of variance with Fisher test to adapt the sensing frequency according to the environmental variation. Harb et al. [104] developed three variance-based sampling algorithms, which, separately, uses a statistical test of data variance, set-similarity functions, or distance function.

III.3 System Setup

The system in our case study measures the noise levels in a working environment, and we study the operation of a single conceptual sensor device. The data that is the basis for the case study was collected using a commercial system and is further detailed in [86]. The noise levels are aggregated as so-called *equivalent continuous sound levels* L_{Aeq} , which is a standard indicator used for noise measurement [164]. For this case study, we select time slots with a length of 15 minutes. This seems an appropriate resolution for use cases where the working environment should be evaluated. This means there are 96-time slots during a day, or 672 per week.

Often, we are not just interested in measuring a phenomenon, such as the noise levels here. We rather want a model of the phenomena which can predict or estimate noise levels based on explanatory variables, even when the IoT device has not recently made measurements, providing a service as close as possible to a continually-monitoring sensor. For many application use cases, such predictive power is more useful than only retrospective logging: A noise monitoring application can for instance recommend silent workplaces, or detect significant outliers in noise corresponding to what is usual. A forecast model implies uncertainty; we do not expect it to represent only points of actual observations, but estimated values that correspond to the actual ones with some certainty. This different view on an IoT sensing system allows us to optimize the behavior of a sensing device. When its energy budget is constrained so that it can only sample a subset of the time slots, the sensor device must be strategic about which time slots to observe. The

hypothesis—which we confirm in this paper—is that once the sensor device selects the time slots to sample wisely, it can produce prediction models with high accuracy and high certainty with only a subset of possible observations. Intuitively speaking, the sensor should only spend energy on observations that are “interesting” with respect to the prediction model, where the prediction is uninformed or unconfident. The challenge is developing such a sampling policy that can anticipate time slots worth covering.

An overview of our process is illustrated in Fig. III.11.² The sensor takes a noise measurement during a 15-minute time slot and stores the observed value. As a prediction model, we use a Gaussian process (GP) with explanatory variables as inputs. Given the explanatory variables and the output of the Gaussian process, the sensor then chooses the number of time steps to sleep before the next measurement; this decision-making is the frugal policy π . It outputs $a_t = 1$ to observe the immediate next time slot, or $a_t = 2$ to hop over one-time slot, and so on. The higher a_t , the longer the sensor device can stay in sleep mode. After the sensor wakes up, it makes a measurement, and the GP is updated with the latest observation to make a new prediction.

The policy π is the result of reinforcement learning, encoded as a neural network. We will detail this step in Sect. III.5.1. Training a reinforcement learning agent is a computationally intensive phase and hence best executed on a server with sufficient computational resources, for instance, as part of the device management. The training step requires training data as input, both for the reinforcement step that creates the policy and to train an initial version of the Gaussian process as a prediction model. For that, we use two weeks of data that the sensor collected uniform sampling before going into frugal mode.

To be explicit, a sample during the frugal policy has the following three purposes:

- The data point provides an entirely accurate measurement for that time period.
- The data point provides a training point for the refinement of the Gaussian process for future predictions.
- The data point provides a training point for the refinement of the RL policy.

In the following, we have a close look at the Gaussian process as a prediction model, the use of RL for building the frugal policy and especially the design of the reward function.

III.4 Gaussian Processes as Predictors

We use a Gaussian process (GP) as a representation model of the monitored phenomenon [50]. The GP produces a Bayesian probabilistic model of the monitored phenomenon over a period of time by predicting a Gaussian distribution over possible measurements with expected value μ_* and variance $\pm\sigma_*^2$ at every time slot.

Figure III.12 shows the estimation of noise levels of a GP over several days. The blue line shows the true value, the red line shows the mean of the GP’s estimation,

²Data and code is available at <https://github.com/Abdulmajid-Murad/adaptive-sensing>

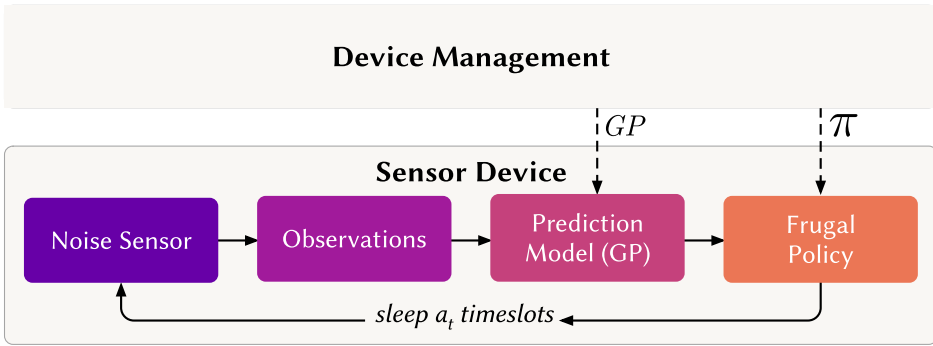


Figure III.11: Overview of the system and the sensor operation.

and the red shading indicates the standard deviation of the GP's estimation. The GP is trained on the observations of the true value marked with filled circles until t_{now} . Until that time, the GP's estimation works as an interpolation between the observations, deviating from occasional outliers. Beyond t_{now} , the GP's estimation works as a forecast. As the uncertainty of the GP increases, the variance becomes larger after t_{now} .

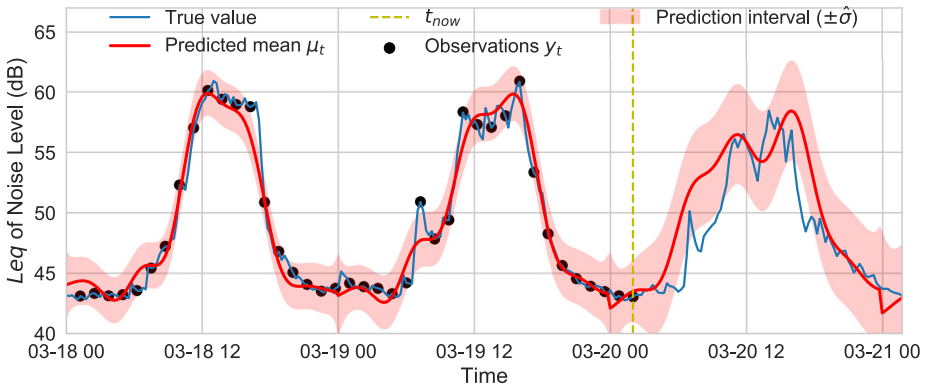


Figure III.12: Gaussian Process regression applied to the noise monitoring and performing short-term prediction

Before using the GP for estimation, it must be trained with a training dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, which consists of measurement collection $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_N]^T$, and their corresponding inputs $\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_N]^T$. Every entry of the input \mathbf{x}_k is a feature vector that consists of measurement time and other explanatory variables [86]. The GP produces a prediction distribution based on similarity with training examples, where similarity is defined by the kernel function. In our case, we chose the commonly used Matern kernel function, which is a generalization of the radial

basis function (RBF kernel):

$$\mathbf{K}_{mat}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_{mat}^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{\mathbf{d}}{\rho} \right)^\nu \mathbf{K}_\nu \left(\sqrt{2\nu} \frac{\mathbf{d}}{\rho} \right) \quad (\text{III.18})$$

Where $\mathbf{d} = \|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance between the two measurement points \mathbf{x}_i and \mathbf{x}_j . The kernel's parameters σ^2 , ρ , and ν can be hand-tuned or estimated by maximizing the likelihood of the observations in the training dataset. Γ is the gamma function, and \mathbf{K}_ν is the Bessel function. Due to its finite differentiability, the Matern kernel is better able to capture a phenomenon than RBF. The Bessel and gamma functions are derived from the Fourier transform of a finite positive measure. The measure is a stochastic differential equation of Laplace type that works well in more than one dimension.

Additionally, we added a periodic kernel function to model the periodicity in the environmental variable.

$$\mathbf{K}_{per}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_{per}^2 \exp \left(-\frac{2 \sin^2(\frac{\pi \mathbf{d}}{p})}{\ell^2} \right) \quad (\text{III.19})$$

Where ℓ is the length scale, σ^2 is the variance, and p is the period of the kernel. Again, these parameters can be hand-tuned or estimated by maximizing the likelihood. The resulting overall kernel is a combination of the Matern and the periodic kernels.

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{K}_{mat}(\mathbf{x}_i, \mathbf{x}_j) + \mathbf{K}_{per}(\mathbf{x}_i, \mathbf{x}_j) \quad (\text{III.20})$$

III.4.1 Evaluating Prediction Models

To assess the quality of the representation model generated by a GP, we can use the root mean square error (*RMSE*) between the predicted mean μ_t and the true values y_t over a period of time T .

$$RMSE_T = \sqrt{\frac{1}{T} \sum_T (\mu_t - y_t)^2} \quad (\text{III.21})$$

However, the *RMSE* metric overlooks an important feature of a Bayesian probabilistic model, which is its prediction interval $\pm\sigma_t^2$. In other words, a prediction should be both accurate and confident. Therefore, we want a metric that also quantifies the uncertainty represented by the model. Figure III.13 shows two different GP predictive distributions of noise levels over one day. The lower model was built with more observations, resulting in more certain predictions with smaller uncertainty σ_t^2 (or higher precision $\frac{1}{\sigma_t^2}$). As a metric for the certainty of a model over a time period T we can take the precision mean:

$$FI = \frac{1}{T} \sum_T \frac{1}{\sigma_t^2} \quad (\text{III.22})$$

This metric is called *Fisher information*, and has been used in many previous works in the field WSN for data fusion, sensor selection, or adaptive sensing [165, 113, 166, 167].

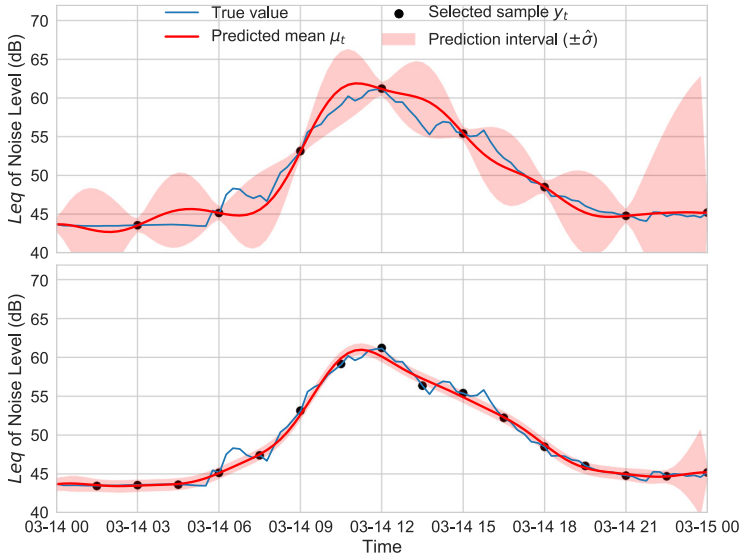


Figure III.13: Comparison of two different predictive models. The top model has a less certain predictions with $FI = 167.84$, and $RMSE = 1.42$. The bottom model has more certain prediction with $FI = 1111.73$, and $RMSE = 0.70$.

III.5 Frugal Operation Mode

In the frugal operation mode, the sensor device only has a limited energy budget over a period T . We assume that the sensor can make no more than N observations in T , and each observation costs a constant amount of energy. By executing a policy, the sensor device decides which time slots will be sampled.

A straightforward sampling policy is uniform sampling, in which a time period T is divided by N equidistant measurements. Figure III.14 shows a GP prediction model fitted from 100 observations distributed uniformly over a one-week period. It results in an RMSE of 0.94 and an FI of 4826.50. This approach provides a steady flow of information, but may result in samples being taken where the prediction is already confident and accurate, and therefore sufficient without the sample.

To gain an understanding of the potential optimal sampling, Kho et al. [113] describe *greedy optimal adaptive sampling*. It is an offline oracle solution that requires knowledge of future observations and is hence not usable during operation, but establishes a near-optimal baseline. (The authors also outline an optimal solution, which is computationally unfeasible, even offline.) Greedy optimal adaptive sampling greedily allocates samples based on the mean FI gain over a day. Figure III.15 shows the same GP prediction model, again fitted with 100 observations. When trained with observations with this oracle policy, the GP's RMSE is reduced to 0.85 compared to that of one trained with samples from the uniform policy; additionally, the uncertainty is reduced, i.e., the predictor has a higher FI of 7331.03. This is a considerable improvement over the uniform policy.

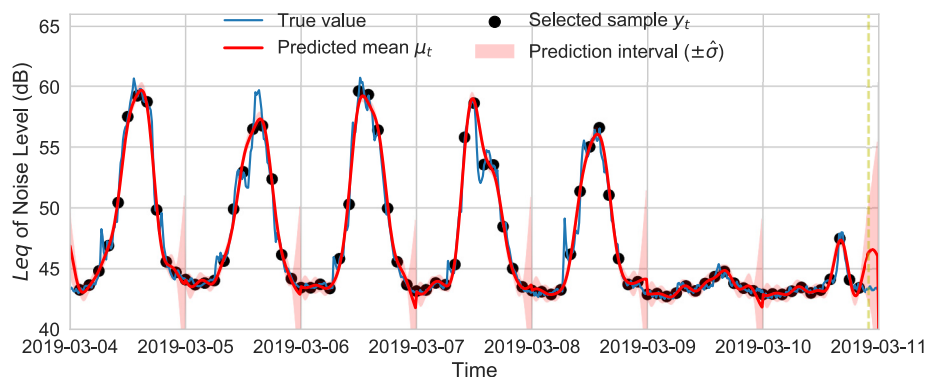


Figure III.14: GP model updated with uniform non-adaptive sampling, having FI = 4826.50, and RMSE = 0.94

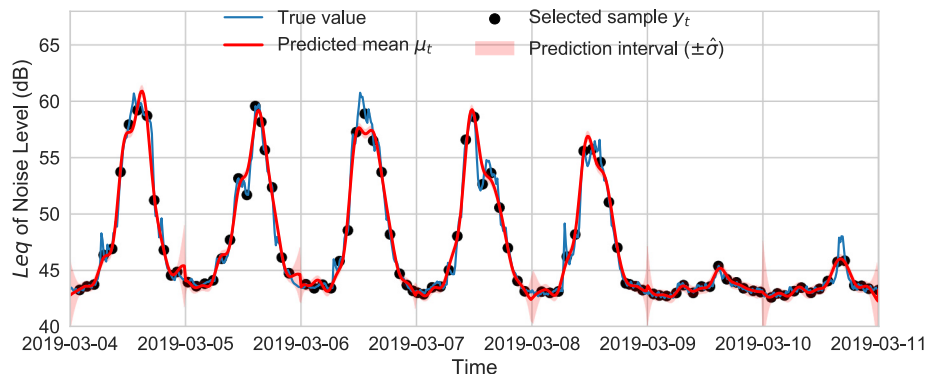


Figure III.15: GP model updated with an offline oracle sampling, having FI = 7331.03, and RMSE = 0.85

III.5.1 Deep Reinforcement Learning Policies

The sampling policies described above outline the task for a reinforcement learning agent in the system: it should learn a sampling policy that is close to the optimal one, but without using knowledge of the future.

Figure III.16 illustrates the process of training a reinforcement learning agent to learn an adaptive sampling task, which is based on the IoT Sensor Gym [168]. Generally, a reinforcement learning task is modeled as a Markov decision process $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$, where \mathcal{S} is the set of the problem's states; \mathcal{A} is the set of actions an agent can take; \mathcal{P} is the states' transition probability; $R : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function; and $\gamma \in (0, 1)$ is a discount factor. In our case study, at each time t , the state ($s_t \in \mathcal{S}$) consists of: the GP's future prediction (μ_t^h, σ_t^h) over a prediction horizon h ; the battery level B_t ; and the explanatory variables \mathbf{x}_t^h , which

consist of weekday, time of day, and whether a given day is a holiday.

$$s_t = [\mu_t^h \quad \sigma_t^h \quad B_t \quad \mathbf{x}_t^h] \quad (\text{III.23})$$

In response to receiving the state s_t , the agent takes an action ($a_t \in \mathcal{A} = (1, h)$) which is the index for the next time step to make a sample, i.e., sleep a_t time slots. The environment, then, transits to a new state according to the transition probability ($p(s'|s, a) \forall s, s' \in \mathcal{S}, a \in \mathcal{A}$), and the agent receives a reward $r_t = R(s_t)$ according to the reward function R , explained later in detail. By interacting with the environment, the agent learns a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which can be evaluated using the value function of a state $V^\pi(s)$:

$$V^\pi(s) = R(s) + \int_{\mathcal{S}} p(s'|s, \pi(s)) V^\pi(s') ds' \quad (\text{III.24})$$

By learning, the agent tries to find a behavior policy that maximizes the value for all states; there exist a variety of learning algorithms in RL [169], which intend to produce such policies. In this paper, we use a policy gradient algorithm, specifically Proximal Policy Optimization (PPO) [36]. In PPO, policies (π_θ) are represented by deep neural networks, parameterized by θ , which can produce actions for any state from a continuous state space. Compared to tabular methods such as SARSA or Q-learning, which are sometimes seen in literature applying RL to IoT, PPO eliminates the need for manual state-space discretization, allowing for better generalization and potentially less hand-tuning by the user.

III.5.2 Information-Based Reward Function

A key requirement for a successful application of reinforcement learning is to design a reward function that frames the goal of an application and guides the learning towards a desirable behavior. However, many goals are difficult to translate into scalar values or are intractable to learn. To overcome this difficulty, many of the previous works use reward shaping, which is a way of incorporating domain knowledge by rewarding easily quantified subgoals to accelerate learning, such as energy-based reward [115, 7, 154, 13]. Although effective in specific domains, energy-based rewards can be restrictive and may lead to undesired emergent behavior in new domains, such as high variance in duty cycle which is not appropriate for IoT nodes. This is because energy is only a resource to be managed, not a goal to be optimized. In our previous work [98], we addressed this issue by introducing a utilitarian reward function that maximizes the utility of a system by rewarding it for high duty cycles. This means, a system gets more reward the more samples it manages to take, no matter if these samples are valuable or not.

The reward functions outlined above only utilize metrics that are more or less correlated with the application goal, but not identical to it. In our setting, the system's goal is to construct a model that represents the underlying phenomenon as closely as possible. Since the sensor has a limited sampling budget, and not all samples are valuable, the objective of the reinforcement learning agent is to select the most informative observations concerning the representation model. Therefore, we design a new reward function that reflects this goal, i.e., rewarding the agent based on the information content of its selected observations. By doing so, we judge

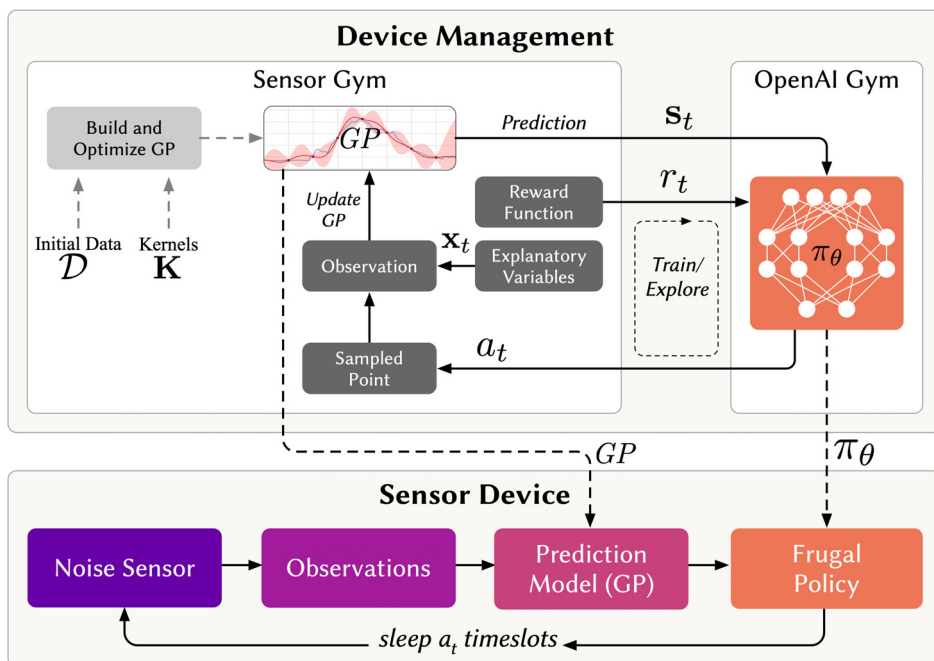


Figure III.16: Architecture for training a model-driven sampling policy with reinforcement learning

the agent’s performance by the metric that we are trying to maximize, and there is a good alignment between maximizing reward and the true goal of the system. Our hypothesis is that this leads to a better performance of the device, as it will learn to spend its constrained energy budget where it has the best-expected effect. Going back to Sect. III.4.1 where we defined the metric for the prediction model, we see that the proposed metric can also work as a reward function. At the end of a period of time T , a day in our case, the RL agent receives a reward based on the quality of the representation model, which is fitted by observations selected by the agent. The quality of the model is abstracted by the mean Fisher information over T :

$$R(\mathbf{s}_T) = \frac{1}{T} \sum_T \frac{1}{\sigma_t^2} \quad (\text{III.25})$$

where $\frac{1}{\sigma_t^2}$ is the prediction confidence at point t . Accordingly, we propose to use the Fisher information directly as a reward function. Fisher information does not depend on unobserved samples as compared to the RMSE, and decodes the application goal into a scalar value that is easy to learn. Hence, it eliminates the need for reward shaping, and it can be applied to large, scalable applications without requiring a system-specific knowledge.

III.6 Evaluation Results

After training approximately 120 RL agents with different hyperparameters, we tested their performance over a different one-week period. Figure III.17 shows a GP prediction model with observation selected using an RL agent. The GP trained on data chosen using an RL sampling policy achieved a better information gain ($FI = 7274.84$) and a lower RMSE ($RMSE = 0.86$) compared to the uniform sampling, and approaches the near-optimal policy implemented by the oracle.

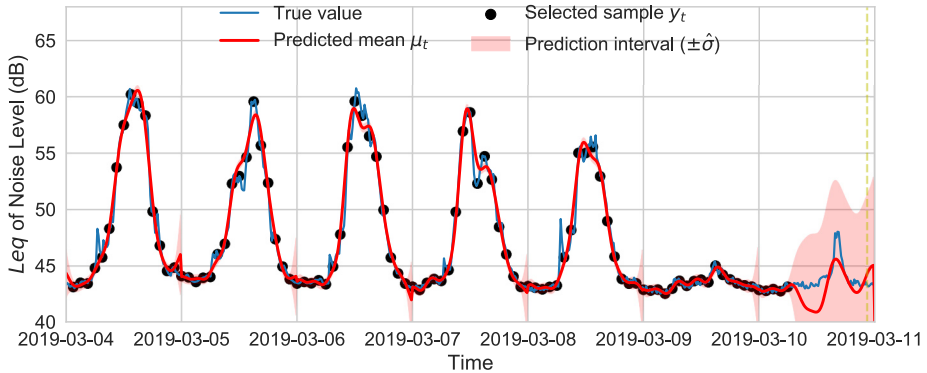


Figure III.17: GP model fitted with observations selected using an RL sampling policy, collecting information value of $FI = 7274.84$, and $RMSE = 0.86$

We also take a more detailed look at the progress of training and how the reward function guides the agents to reach their goals. Figure III.18 shows smoothed learning curves of four sampled agents, trained with different hyperparameters. The x-axis shows the number of training episodes experienced so far, and the y-axis shows the corresponding reward achieved in each episode. The original curves are in shaded colors, while the smoothed versions are on top in solid colors. After sufficient episodes, all four agents reach near-optimal policies that achieve rewards close to the oracle solution.

III.7 Discussion

The results show that using the mean Fisher information as a reward function is effective in learning desirable policies. To investigate the effectiveness and constraints of using this reward function, we can take a close look at its definition in (III.22). Most important for IoT in a setting where taking samples is energy-intensive, the reward function does not depend on unobserved samples, only on the confidence of the resulting model over a period of time. Because model confidence can only come about when measurements taken at similar moments have low variance, we assume the model is close to correct when it is confident. This assumption may occasionally be unfounded in outlier moments, but measuring and hoping for an outlier which is, by definition, unlikely to occur, is not intelligent or desirable behavior.

The main contribution of our paper regards the development of the frugal policy, which is encoded as a neural network. This network can be easily implemented on IoT hardware. For example, one exemplar of a good performing policy was a network with 4 hidden layers with 32 neurons and ReLU activation functions. We have implemented this network on an Arduino NANO 33 BLE Sense, where it only requires 5 ± 1 ms to evaluate. Compared with the estimated execution time to measure, aggregate, calculate and send noise data, this overhead is acceptable. The compiled flash memory consumption was 115 KB (11%) and the RAM Memory was 50 KB (20%). With regards to the GP, learning involves an inversion of the kernel matrix, which has an asymptotic complexity of $\mathcal{O}(n^3)$, where n is the number of training examples. However, the GP can be trained off-device before deployment, so it doesn't have to store its history of measurements. Additionally, sparse online GP can be exploited to train incrementally without retaining the entire model when new data are observed [51]. Furthermore, there are many model approximations or approximate GP inference with cheaper complexity that can be utilized such as Nystrom approximation [52], FITC approximation [53], or variational sparse GP [54].

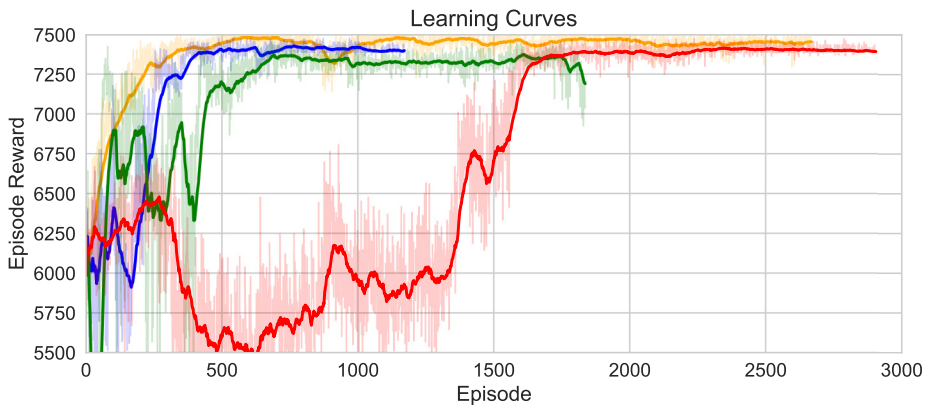


Figure III.18: Smoothed learning curves of four agents, where the episode reward is the collected Fisher information.

In general, finding good values for hyperparameters can be challenging during training. In our setting, we first required suitable types and parameters for the kernels of the GP. We found these through optimization by maximizing the likelihood of the data, using the GPy framework [170]. Secondly, we had to dimension the environment's parameters, such as energy capacity, prediction horizon, range and resolution for states and actions. For that, we used an iterative process of tuning with randomized values. Finally, learning an RL policy can be unstable due to the algorithm's sensitivity to hyperparameters and their intricate interplay. Hence only a subset of the policies performs well, and not all of them are useful. This, however, is not a problem since the training takes place on a device management server, and we can choose the best performing policy before deployment.

III.8 Conclusion

In this work, we have introduced a novel reward function for learning adaptive sensing policies with deep reinforcement learning. This reward function is based on the mean Fisher information value of a probabilistic model of an environmental phenomenon. The model is fitted with observations selected by the learning agent. Using a case study of workplace noise monitoring, we demonstrated that this reward function led to a learned sampling policy that outperforms a uniform strategy and is close to a near-optimal oracle solution. Our results indicate that using this information-based reward function along with policies approximated by neural networks can achieve more generalization and autonomy in IoT applications. Finally, we discussed the implementations and the constraints of the proposed framework. While the presented work uses a case study with a temporally-varying phenomenon, it can be applied in a wide range of adaptive sensing applications. Overall, this approach leads to an increased level of autonomy in IoT by reducing manual design effort through information-driven behavior learning.

Acknowledgements

We would like to thank Finn Julius Stephansen-Smith and Amund Askeland for their implementation of the neural network.

References

- [7] Chaoming Hsu, R., Liu, C. T., and Lee, W. M. “Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Vol. 5579 LNAI. 2009.
- [11] Lei, Z., Tang, H., Li, H., and Jiang, Q. “Dynamic power management strategies for a sensor node optimised by reinforcement learning”. In: *Int. J. Computational Science and Engineering* vol. 13, no. 1 (2016).
- [12] Aoudia, F. A., Gautier, M., and Berder, O. “RLMan: an Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks”. In: *IEEE Transactions on Green Communications and Networking* vol. 2, no. 2 (2018), pp. 1–1.
- [13] Shresthamali, S., Kondo, M., and Nakamura, H. “Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning”. In: vol. 16, no. 5s (2017), pp. 1–21.
- [14] Fraternali, F., Balaji, B., and Gupta, R. “Scaling configuration of energy harvesting sensors with reinforcement learning”. In: *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. ACM. 2018, pp. 7–13.
- [36] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017), pp. 1–12. arXiv: [arXiv:1707.06347v2](https://arxiv.org/abs/1707.06347v2).

- [50] Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*. Vol. 2. 2006.
- [51] Csató, L. and Oppor, M. “Sparse on-line Gaussian processes”. In: *Neural computation* vol. 14, no. 3 (2002), pp. 641–668.
- [52] Williams, C. K. and Seeger, M. “Using the Nyström method to speed up kernel machines”. In: *Advances in neural information processing systems*. 2001, pp. 682–688.
- [53] Bui, T. D., Yan, J., and Turner, R. E. “A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation”. In: *The Journal of Machine Learning Research* vol. 18, no. 1 (2017), pp. 3649–3720.
- [54] Titsias, M. “Variational learning of inducing variables in sparse Gaussian processes”. In: *Artificial Intelligence and Statistics*. 2009, pp. 567–574.
- [86] Kraemer, F. A., Alawad, F., and Bosch, I. M. V. “Energy-Accuracy Tradeoff for Efficient Noise Monitoring and Prediction in Working Environments”. In: *Proceedings of the 9th International Conference on the Internet of Things*. 2019, pp. 1–8.
- [98] Murad, A., Kraemer, F. A., Bach, K., and Taylor, G. “Autonomous management of energy-harvesting IoT nodes using deep reinforcement learning”. In: *proceedings of IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Umeå, Sweden*. 2019.
- [102] Laiymani, D. and Makhoul, A. “Adaptive data collection approach for periodic sensor networks”. In: *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE. 2013, pp. 1448–1453.
- [103] Silva, J. M. C., Bispo, K. A., Carvalho, P., and Lima, S. R. “LiteSense: An adaptive sensing scheme for WSNs”. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2017, pp. 1209–1212.
- [104] Harb, H. and Makhoul, A. “Energy-efficient sensor data collection approach for industrial process monitoring”. In: *IEEE Transactions on Industrial Informatics* vol. 14, no. 2 (2017), pp. 661–672.
- [105] Salim, C., Makhoul, A., Darazi, R., and Couturier, R. “Adaptive sampling algorithms with local emergency detection for energy saving in wireless body sensor networks”. In: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2016, pp. 745–749.
- [113] Kho, J., Rogers, A., and Jennings, N. R. “Decentralized control of adaptive sampling in wireless sensor networks”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 5, no. 3 (2009), pp. 1–35.
- [115] Dias, G. M., Nurchis, M., and Bellalta, B. “Adapting sampling interval of sensor networks using on-line reinforcement learning”. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE. 2016, pp. 460–465.
- [116] Cobb, A. and Markham, A. “Adaptive sampling of lion accelerometer data”. In: *Oxford* (2016).

-
- [121] Muttreja, A., Raghunathan, A., Ravi, S., and Jha, N. K. “Active learning driven data acquisition for sensor networks”. In: *11th IEEE Symposium on Computers and Communications (ISCC’06)*. IEEE. 2006, pp. 929–934.
- [122] Osborne, M. A., Roberts, S. J., Rogers, A., and Jennings, N. R. “Real-time information processing of environmental sensor network data using bayesian gaussian processes”. In: *ACM Transactions on Sensor Networks (TOSN)* vol. 9, no. 1 (2012), pp. 1–32.
- [123] Ling, C. K., Low, K. H., and Jaillet, P. “Gaussian process planning with Lipschitz continuous reward functions: Towards unifying Bayesian optimization, active learning, and beyond”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.
- [124] Monteiro, L. C., Delicato, F. C., Pirmez, L., Pires, P. F., and Miceli, C. “Dpcas: Data prediction with cubic adaptive sampling for wireless sensor networks”. In: *International Conference on Green, Pervasive, and Cloud Computing*. Springer. 2017, pp. 353–368.
- [154] Ortiz, A., Al-Shatri, H., Li, X., Weber, T., and Klein, A. “Reinforcement Learning for Energy Harvesting Decode-and-Forward Two-Hop Communications”. In: *IEEE Transactions on Green Communications and Networking* vol. 1, no. 3 (2017), pp. 309–319.
- [160] Chincoli, M. and Liotta, A. “Self-learning power control in wireless sensor networks”. In: *Sensors* vol. 18, no. 2 (2018), p. 375.
- [161] Udenze, A. and McDonald-Maier, K. “Indirect Reinforcement Learning for autonomous power configuration and control in Wireless Networks”. In: *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE. 2009, pp. 297–304.
- [162] Fraternali, F., Balaji, B., Agarwal, Y., and Gupta, R. K. “ACES—Automatic Configuration of Energy Harvesting Sensors with Reinforcement Learning”. In: *arXiv preprint arXiv:1909.01968* (2019).
- [163] Khan, U. A. and Rinner, B. “Online learning of timeout policies for dynamic power management”. In: *ACM Transactions on Embedded Computing Systems (TECS)* vol. 13, no. 4 (2014), pp. 1–25.
- [164] 1996-1:2016, I. “Acoustics — Description, measurement and assessment of environmental noise — Part 1: Basic quantities and assessment procedures”. In: *International Organization for Standardization* (2016), pp. 1–47.
- [165] Rogers, A., Dash, R. K., Jennings, N. R., Reece, S., and Roberts, S. “Computational mechanism design for information fusion within sensor networks”. In: *2006 9th International Conference on Information Fusion*. IEEE. 2006, pp. 1–7.
- [166] Wang, F., Bai, X., Guo, B., and Liu, C. “Dynamic clustering in wireless sensor network for target tracking based on the fisher information of modified Kalman filter”. In: *2016 3rd International Conference on Systems and Informatics (ICSAI)*. IEEE. 2016, pp. 696–700.

- [167] Yilmaz, Y. and Wang, X. “Sequential decentralized parameter estimation under randomly observed fisher information”. In: *IEEE transactions on information theory* vol. 60, no. 2 (2013), pp. 1281–1300.
- [168] Murad, A., Bach, K., Kraemer, F. A., and Taylor, G. “IoT Sensor Gym: Training Autonomous IoT Devices with Deep Reinforcement Learning”. In: *Proceedings of the 9th International Conference on the Internet of Things*. 2019, pp. 1–4.
- [169] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2011.
- [170] GPy. *GPpy: A Gaussian process framework in python*. <http://github.com/SheffieldML/GPy>. 2012.

Paper IV: Probabilistic Deep Learning to Quantify Uncertainty in Air Quality Forecasting

Abdulmajid Murad¹, Frank Alexander Kraemer¹, Kerstin Bach¹,
Gavin Taylor²

¹Norwegian University of Science and Technology, ²United States Naval Academy

IV

Abstract

Data-driven forecasts of air quality have recently achieved more accurate short-term predictions. Despite their success, most of the current data-driven solutions lack proper quantifications of model uncertainty that communicate how much to trust the forecasts. Recently, several practical tools to estimate uncertainty have been developed in probabilistic deep learning. However, there have not been empirical applications and extensive comparisons of these tools in the domain of air quality forecasts. Therefore, this work applies state-of-the-art techniques of uncertainty quantification in a real-world setting of air quality forecasts. Through extensive experiments, we describe training probabilistic models and evaluate their predictive uncertainties based on empirical performance, reliability of confidence estimate, and practical applicability. We also propose improving these models using "free" adversarial training and exploiting temporal and spatial correlation inherent in air quality data. Our experiments demonstrate that the proposed models perform better than previous works in quantifying uncertainty in data-driven air quality forecasts. Overall, Bayesian neural networks provide a more reliable uncertainty estimate but can be challenging to implement and scale. Other scalable methods, such as deep ensemble, Monte Carlo (MC) dropout, and stochastic weight averaging-Gaussian (SWAG), can perform well if applied correctly but with different tradeoffs and slight variations in performance metrics. Finally, our results show the practical impact of uncertainty estimation and demonstrate that, indeed, probabilistic models are more suitable for making informed decisions. Code and dataset are available at https://github.com/Abdulmajid-Murad/deep_probabilistic_forecast.

IV.1 Introduction

Monitoring and forecasting real-world phenomena are fundamental use cases for many practical applications in the Internet of Things (IoT). For example, policymakers in municipalities can use forecasts of ambient air quality to make decisions about actions, such as informing the public or starting emission-reduction measures. The problem is that forecasts are both uncertain and intended for human interpretation, so decisions about specific actions should take forecast confidence into account. For instance, it may be best to only start a costly initiative for cleaning streets of dust when the forecast of air pollutants exceeds a certain threshold and the reported confidence is high. Therefore, quantifying the predictive confidence is crucial for learning, providing, and interpreting reliable forecasting models.

Progress in probabilistic machine learning [171] and, more recently, in probabilistic deep learning led to the development of practical tools to estimate uncertainty about models and predictions [63, 58, 68, 70, 172, 73]. These tools have been successfully used in various domains, such as computer vision [173, 174], language modeling [175, 176], machine translation [177] and autonomous driving [178]. All of these tools address quantifying predictive uncertainty but differ in techniques, approximations, and assumptions when representing and manipulating uncertainty.

The successful application of probabilistic models to real human problems requires us to bridge the gap from the theory of these disparate approaches to

practical concerns. In particular, we need an empirical evaluation and comparison of these many techniques. We want to know how they perform with respect to prediction accuracy, uncertainty quantification, and other requirements. Specifically, we want to know the reliability of their confidence estimate, meaning if they are actually more accurate and trustworthy when their confidence is high. This is practically important for policymakers to make risk-informed decisions. Such an empirical evaluation, especially in the domain of air quality, is currently lacking.

Existing studies already incorporate aspects of uncertainty into their data-driven forecasts, but they often only quantify aleatoric uncertainty. This uncertainty is inherent in the observed data and can be quantified by a distribution over the model’s output using softmax or Gaussian. There is also epistemic uncertainty that we can and should quantify as well. This type of uncertainty represents how much the model does not know, for instance, in regions of the input space with little data. Probabilistic models can capture both types of uncertainties, which makes them practically appealing since they convey more information about the reliability of the forecast. They also provide a wider area of control over the decision boundary and the risk profile of a decision. For instance, Figure IV.19 illustrates the potential of quantifying both types of uncertainties in decision making. It shows the decision F1 score as a function of normalized aleatoric and epistemic confidence thresholds in a probabilistic model. Each point on the surface represents the resulting score of accepting the model predictions only where its confidence is above specific thresholds (τ_1 and τ_2). Expressing both uncertainty dimensions gives more control regarding false positives and false negatives and their associated risks. We will come back to this Figure in Section IV.5.3.

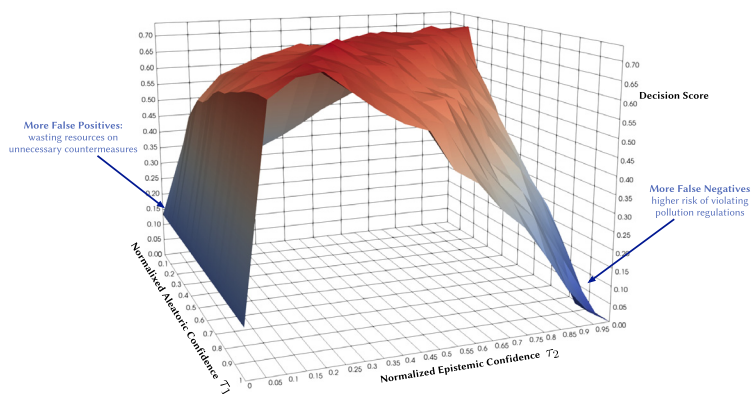


Figure IV.19: Illustrative example: decision F1 score as a function of normalized aleatoric and epistemic confidence thresholds.

In this work, we develop a set of deep probabilistic models for air quality forecasting that quantify both aleatoric and epistemic uncertainties and study how to represent and manipulate their predictive uncertainties. Our contributions are the following:

- We conduct a broad empirical comparison and exploratory assessment of state-of-the-art techniques in deep probabilistic learning applied to air quality

forecasting. Through exhaustive experiments, we describe training these models and evaluating their predictive uncertainties using various metrics for regression and classification tasks.

- We improve uncertainty estimation using adversarial training to smooth the conditional output distribution locally around training data points.
- We apply uncertainty-aware models that exploit the temporal and spatial correlation inherent in air quality data using recurrent and graph neural networks.
- We introduce a new state-of-the-art example for air quality forecasting by defining the problem setup and selecting proper input features and models.

Our results show that all the considered probabilistic models perform well on our application, with slight variations in different performance metrics. Bayesian neural networks perform slightly better in proper scoring rules that measure the quality of probabilistic predictions. In addition, we show that smoothing predictive distributions by adversarial training improves metrics that punish incorrect, overconfident predictions, especially in regression tasks, since the forecasted phenomena are inherently smooth.

The rest of this paper is organized as follows: In Section IV.2, we outline the related work in uncertainty estimation and air quality forecasting. We then define the problem setup and introduce non-probabilistic baselines and evaluation metrics in Section IV.3. In Section IV.4, we present deep probabilistic models applied to the air quality forecast, describe their training, evaluation, and how to improve their uncertainty estimate. We close with a comparative analysis to compare the selected models in Section IV.5 and a conclusion in Section IV.6.

IV.2 Related Work

Forecasting ambient air quality is crucial to support decision-making in urban management. Therefore, a sizeable body of work addresses building air quality forecasting models. For example, MACC (Monitoring Atmospheric Composition and Climate) [179] is a European project that provides air quality analysis and forecasting services for the European Continent. It uses physics-based modeling that combines transport, chemistry, and satellite data to provide a multi-model ensemble forecast of atmospheric composition (daily forecasts with hourly outputs of 10 chemical species/aerosols). Walker et al. [180] used the output of MACC ensemble-based probabilistic forecasting of air pollution and performed statistical post-processing, calibrated predictive distribution using Box-Cox transformation for correcting the skewness of air pollution data. Additionally, they discussed model selection and verification using Akaike and Bayesian information criteria. To obtain the ensemble forecast from MACC, they introduced stochastic perturbations to the emissions. Garaud et al. [181] performed a posterior calibration of multi-model ensembles using a mixed optimization algorithm to extract a sub-ensemble.

The official air quality forecasting service in Norway [182]³ uses a Gaussian dispersion modeling that provides a 2-day hourly forecast with high-resolution coverage (between 250 and 50 m grid) over the entire country [183, 184]. The forecast is based on weather conditions, polluting emissions, and terrain. In particular, it uses the chemical transport model uEMEP (urban European Monitoring and Evaluation Program) [183] and a road dust emission model [185]. Denby et al. [186] analyze the accuracy of the Norwegian air quality forecasting by comparing model calculations with measurements. They show that the model’s forecast on particle dust is marginally better than the persistent forecast and give some assumptions about why model calculations deviate from the observations.

Although physics-based models [187, 183, 184] can provide long-range air pollution information, they require significant domain knowledge and complex modeling of dispersion, chemical transport, and meteorological processes. Additionally, physics-based models involve structural uncertainty, low spatial resolution, and do not capture abrupt and short-term changes in air pollution. Data-driven modeling based on historical data [188, 189, 190] can complement physics-based modeling by learning directly from air quality measurements and providing a more reliable short-term prediction. For example, Lepperod et al. [188] deployed stationary and mobile micro-sensor devices and used Narrowband IoT (NB-IoT) to aggregate air quality data from these sensors. Then, they applied machine learning methods to predict air quality in the next 48 hours using observations of sensors’ measurements, traffic, and weather data. Zhou et al. [190] used long short-term memory (LSTM) to forecast multi-step time-series of air quality, while Mokhtari et al. [191] proposed combining a convolutional neural network (CNN) with LSTM for air quality prediction and quantified the uncertainty using quantile regression and MC dropout. Tao et al. [192] used 1D CNNs and a Bidirectional gated recurrent unit (GRU) for a short-term forecast of fine air particles. Pucer et al. [193] used Gaussian Processes (GP) to forecast daily air-pollutant levels, and Aznarte et al. [194] proposed using quantile regression for probabilistic forecasting of extreme nitrogen dioxide (NO_2) pollution.

Most of the current data-driven forecasts give point predictions of a deterministic nature. Thus, they lack useful estimates of their predictive uncertainty that convey more information about how much to trust the forecast. Recently, quantifying prediction uncertainty has garnered increasing attention in machine learning fields, including deep learning. Bayesian methods are among the most used approaches for uncertainty estimation in neural networks. Given a prior distribution over the parameters, Bayesian methods use training data to compute a posterior distribution. Using the obtained distribution, we can easily quantify the predictive uncertainty. This approach has been extended to neural networks, theoretically allowing for the accuracy of modern prediction methods with valid uncertainty measures; however, modern neural networks contain many parameters, and obtaining explicit posterior densities through Bayesian inference is computationally intractable. Instead, there exist a variety of approximation methods that estimate the posterior distributions. These methods can be decomposed into three main categories, either based on variational inference [57, 58, 59], Markov chain Monte Carlo (MCMC) [60, 61, 62],

³<https://luftkvalitet.miljodirektoratet.no/kart/59/10/5/aqi>

or Laplace approximation [63, 64]. In this paper, we will use variational Bayesian methods and approximate Bayesian inference methods.

IV.3 Air Quality Prediction, Base Models and Metrics

IV.3.1 Problem Setup

We are trying to build a model for forecasting air quality trends at pre-defined locations and for a specified forecast horizon. In our case study, we want to forecast the level of microscopic particles in the air, known as particulate matter (PM). These are inhalable particles with two types: coarse particles with a diameter less than $10\ \mu\text{m}$ (PM_{10}) and fine particles with a diameter less than $2.5\ \mu\text{m}$ ($PM_{2.5}$). The forecast predicts pollutant levels for the next 24h at four monitoring stations in the city of Trondheim, as shown in Figure IV.20. The stakeholders, policymakers of the municipality, would like to estimate if the concentration of air particles exceeds certain thresholds following the Common Air Quality Index (CAQI) used in Europe [195], as shown in Table IV.3. This can be achieved through value regression or by directly classifying threshold exceedance levels. We will explore probabilistic models that forecast air quality values and predict threshold exceedance events. Using probabilistic models provides more qualitative information since decision-making largely depends on the credibility intervals of specific predictions. The forecast is based on explanatory variables, such as historical air quality measurements, meteorological data, traffic, and street-cleaning reports from the municipality.

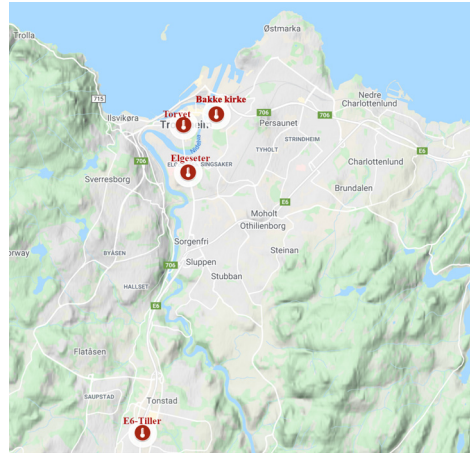


Figure IV.20: Air quality monitoring stations in Trondheim, Norway.

Although the CAQI (Table IV.3) specifies five levels of air pollutants, the air quality in the city of Trondheim is usually at *Very Low* and rarely exceeds the *Low* level. For example, Figure IV.21 shows the air quality level over one year of a representative monitoring station (Elgeseter). Therefore, instead of a multinomial classification task (with five classes), we transform the problem into a threshold exceedance forecast task in which we try to predict the points in time where the air quality exceeds the *Very Low* level.

Table IV.3: European Common Air Quality Index

Index	$PM_{10}(\mu\text{g}/\text{m}^3)$	$PM_{2.5}(\mu\text{g}/\text{m}^3)$
Very low	0–25	0–15
Low	25–50	15–30
Medium	50–90	30–55
High	90–180	55–110
Very High	>80	>110



Figure IV.21: Air quality level over one year of one representative monitoring station in Trondheim, where the air pollutant is commonly at a *Very Low* level and rarely exceeds *Low*.

The air quality dataset we use is a part of the official measuring network in Europe. Specifically, we use the open database of air quality measurements offered by the Norwegian Institute for Air Research (NILU) [196]⁴. The meteorological data are based on historical weather and climate data offered by the Norwegian Meteorological Institute [197]⁵. The traffic data are based on aggregated traffic volumes offered by the Norwegian Public Roads Administration [198]⁶. A more detailed description of the used datasets can be found in Appendix IV.7.1.

IV.3.2 Epistemic and Aleatoric Uncertainty

Before quantifying the predictive uncertainty, it is worth distinguishing the different sources of uncertainty and the appropriate actions to reduce them. The first source is model or epistemic uncertainty, which is uncertainty in the model parameters in regions of the input space with little data (i.e., data sparsity). This type of uncertainty can be reduced given enough data. By estimating the epistemic uncertainty of a model, we can obtain its confidence interval (CI). The second source of uncertainty is data or aleatoric uncertainty. It is essentially a noise inherent in the observations (i.e., input-dependent) due to either sensor noise or entropy in the true data generating process. By estimating the aleatoric and epistemic uncertainties, we can obtain the prediction interval (PI) [199, 174]. Accordingly, prediction intervals are wider than confidence intervals. The third source of uncertainty is model

⁴<https://www.nilu.com/open-data/>

⁵<https://frost.met.no>

⁶<https://www.vegvesen.no/trafikkdata/start/om-api>

misspecification, i.e., uncertainty about the general structure of the model, such as model type, number of nodes, number of layers. It is also related to the bias-variance tradeoff [200].

IV.3.3 Non-probabilistic Baselines

As a baseline for comparison and to motivate the use of evaluation metrics, we test the performance of non-probabilistic models, such as persistence, XGBoost, and gradient boosting models. A simple baseline predictor is a persistence forecast, which uses the diurnal patterns of the observations. To predict a value in the future, we use the value observed 24 hours earlier. Figure IV.22 shows the results of the persistence model when forecasting the PM-value over one month (January 2020). Suppose $\hat{y}_t \in \mathbb{R}$ is the forecast value, while $y_t \in \mathbb{R}$ is the true observed value at time t , then we can evaluate the aggregated accuracy over a time period T using the root-mean-square error:

$$RMSE = \sqrt{\frac{1}{T} \sum_{t=1}^T (y_t - \hat{y}_t)^2}. \quad (\text{IV.26})$$

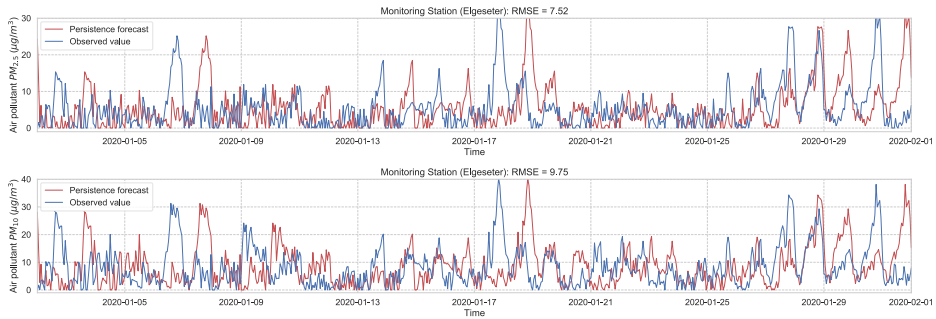


Figure IV.22: Persistence forecast of air pollutant over one month in one representative monitoring station.

Next we evaluate the performance of XGBoost (*eXtreme Gradient Boosting*) [49] as a non-probabilistic baseline in our problem setting. XGBoost uses the same model representation and inference as Random forests (i.e., gradient-boosted decision trees) but has a different training mechanism since it uses the second-order approximation of the training objective. We train the model over one year of data (2019) and test its performance over one month (January 2020) of a 24-hour forecast horizon. Figure IV.23 shows the results of the XGBoost model in one representative monitoring station. The results show an improved prediction accuracy compared to the persistence forecast, which illustrates the value of a learned predictor for time-series forecasting of air quality.

One advantage of using an XGBoost model is the feasibility of retrieving the feature importance by assigning a score to each input feature, which indicates how useful that feature is when making a prediction, thus contributing to prediction interpretation. Figure IV.24 shows the feature importance of the XGBoost forecast

Paper IV: Probabilistic Deep Learning to Quantify Uncertainty in Air Quality Forecasting

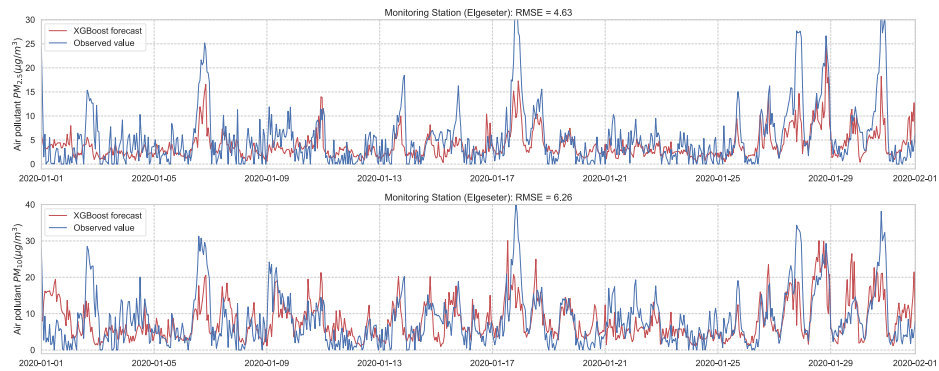


Figure IV.23: PM-value regression using the XGBoost model over one month in one representative monitoring station.

shown in Figure IV.23 (a more detailed description of the features can be found in Appendix IV.7.1.

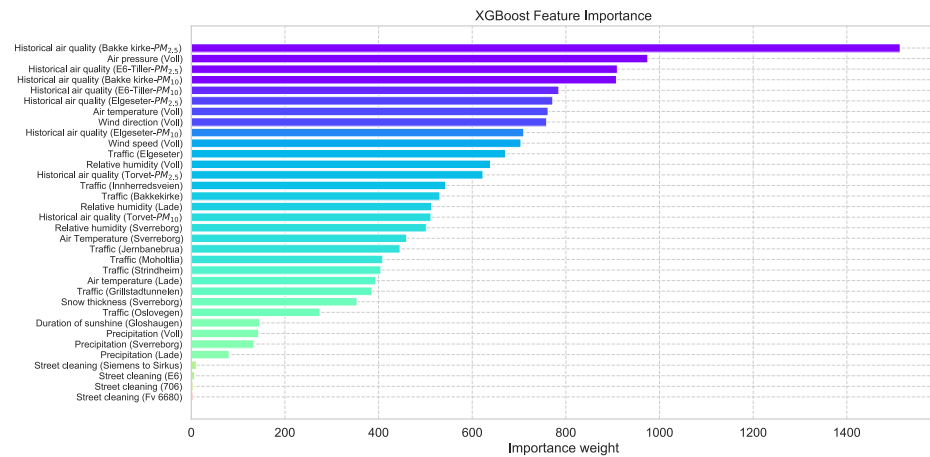


Figure IV.24: Feature importance of a trained XGBoost model indicating how useful each input feature when making a prediction.

For the threshold exceedance prediction task, we can use XGBoost to train a binary classifier to predict the probability of threshold exceedance. By using a proper scoring rule [55], such as the cross-entropy as a training criterion, we obtain predictive probabilities. Notably, these probabilities do not represent the model (epistemic) uncertainty. Rather, they represent data (aleatoric) uncertainty. In addition to cross-entropy, we need to evaluate performance using another metric that the model was not optimizing. We can use the Brier score [201], which is a strictly proper scoring rule [55] and metric to evaluate the reliability of predictive probabilities. Suppose at a time, t , the true class label is represented by o_t (1 when pollutant level exceeds the threshold, 0 if not), while the predicted probability is

represented by $\hat{p}_t \in [0, 1]$. Then we calculate the cross-entropy (CE) and Brier Score (BS) as follows:

$$CE = -\frac{1}{T} \sum_{t=1}^T (o_t \times \log(\hat{p}_t)) \quad (\text{IV.27})$$

$$BS = \frac{1}{T} \sum_{t=1}^T (o_t - \hat{p}_t)^2 \quad (\text{IV.28})$$

Both metrics heavily punish overconfident, incorrect predictions. Cross-entropy uses exponential punishment (heavily emphasizes tail probabilities) since it is a negative log-likelihood loss. Thus, it is sensitive to the predicted probabilities of the infrequent class (i.e., threshold exceedance events). In contrast, the Brier score uses quadratic punishment since it is a mean square error in the probability space. Thus, it treats the predicted probabilities of both classes equally.

Additionally, we can use the commonly used metrics for classification tasks [202] to evaluate the performance of a deterministic prediction. By converting the predictive probabilities into predictive class labels, we can calculate the true-positive rate tp , the false-positive rate fp , and the false-negative rate fn . Then, we can use the metrics of $Precision : \mathbb{R} \rightarrow [0, 1]$, $Recall : \mathbb{R} \rightarrow [0, 1]$, and $F1 : \mathbb{R} \rightarrow [0, 1]$ score to evaluate the threshold exceedance prediction as following:

$$Precision = \frac{tp}{tp + fp} \quad (\text{IV.29})$$

$$Recall = \frac{tp}{tp + fn} \quad (\text{IV.30})$$

$$F1 = \frac{tp}{tp + 0.5 \times (fp + fn)} \quad (\text{IV.31})$$

Figure IV.25 shows the results of the XGBoost model when predicting threshold exceedance probability. The blue dots indicate the points in time (hours) when air pollutants actually exceeded the threshold level. The red line represents the predicted probability in percentages. We see that with higher probability, the model predicts a more likely event of threshold exceedance at that specific time.

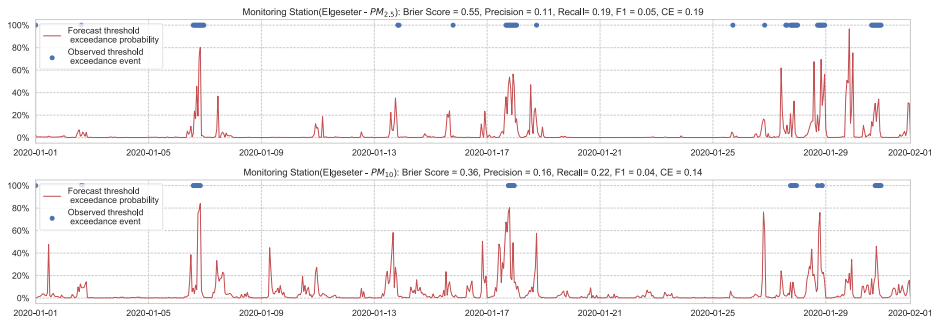


Figure IV.25: Predicting the threshold exceedance probability of the air pollutant level using an XGBoost model in one representative monitoring station.

IV.3.4 Quantile Regression

Using an XGBoost model leads to an improved prediction accuracy (Figure IV.23), but with a caveat that it only produces point estimates of the mean. To address this, we can use quantile regression [203] to estimate a specific percentile (i.e., quantile) of the target variable. The quantile regression estimates the conditional quantile by minimizing the absolute residuals. In contrast, regular regression estimates the conditional mean by minimizing the least squared residuals (focuses on central tendency). The advantages of using quantile regression are that it focuses more on dispersion or variability, does not assume homoscedasticity in the target variable (i.e., having the same variance independent of the input variable), and is more robust against outliers. We use a Gradient Tree Boosting model [204] to estimate the 5% and 95% percentiles, as shown in Figure IV.26.

Given a predicted lower $\hat{L}_t \in \mathbb{R}$ and upper bound $\hat{U}_t \in \mathbb{R}$, we can assess the quality of the generated prediction interval using metrics, such as Prediction Interval Coverage Probability ($PICP : \mathbb{R} \rightarrow [0, 1]$) and Mean Prediction Interval Width ($MPIW : \mathbb{R} \rightarrow [0, \infty)$):

$$PICP = \frac{1}{T} \sum_{t=1}^T \mathbb{1}(y_t - \hat{L}_t) \times \mathbb{1}(\hat{U}_t - y_t) \quad (IV.32)$$

$$MPIW = \frac{1}{T} \sum_{i=1}^T (\hat{U}_t - \hat{L}_t) \quad (IV.33)$$

where $\mathbb{1}$ is the Heaviside step function. $PICP$ indicates how often the prediction interval captures the true values, ranging from 0 (all outside) to 1 (all inside). Intuitively, we seek a model with a narrow prediction interval (i.e., smaller $MPIW$) while capturing the observed data points (i.e., larger $PICP$). For example, when forecasting $PM_{2.5}$ pollutants in Figure IV.26, $PICP = 0.61$ indicates that in 61% of the time-steps, the observed values are inside the predicted intervals, as compared to 72% when forecasting PM_{10} . In contrast, the predicted intervals are smaller ($MPIW = 7.49$) when forecasting $PM_{2.5}$ as compared to wider predicted intervals ($MPIW = 14.67$) when forecasting PM_{10} .

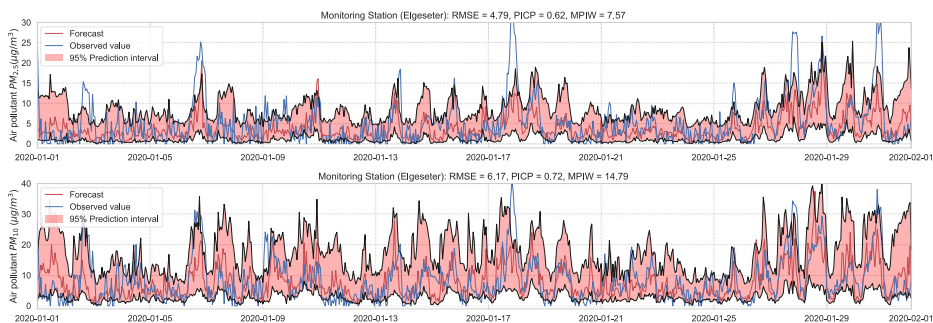


Figure IV.26: Air quality prediction interval using quantile regression of a Gradient Tree Boosting model.

IV.4 Deep Probabilistic Forecast

This section explores probabilistic models for air quality forecasting and describes how to quantify their predictive uncertainty. We assume we have a training dataset $\mathcal{D} = \{\mathbf{x}_t, y_t\}_{t=1}^T$ with an input feature $\mathbf{x}_t \in \mathbb{R}^D$ and a corresponding observation $y_t \in \mathbb{R}$ for each monitoring station. For the time-series forecasting task, we estimate the aleatoric uncertainty by employing the Mean-Variance Estimation method [56], in which we have a neural network with two output nodes (for every individual time-series) that estimates the mean $\hat{\mu}_t \in \mathbb{R}$ and variance $\hat{\sigma}_t^2 \in (0, \infty)$ of the target probability distribution. Additionally, we use the negative log-likelihood ($NLL : (0, \infty) \rightarrow (0, \infty)$) as a training criterion since the RMSE does not capture the predictive uncertainty. By treating the observed value y_t as a sample from the target distribution, we can calculate NLL as follows:

$$NLL = 0.5 \left(\log 2\pi\hat{\sigma}_t^2 + \frac{(y_t - \hat{\mu}_t)^2}{\hat{\sigma}_t^2} \right) \quad (\text{IV.34})$$

Notably, we treat the target distributions as having heteroscedastic uncertainty (i.e., non-constant variance) and diagonal covariance matrices, which simplifies the training criterion of multivariate time-series. For the threshold exceedance prediction task, we estimate the aleatoric uncertainty by having a neural network that outputs the predictive probability in terms of cross-entropy.

Although epistemic uncertainty is model-dependent, it is essentially estimated by measuring the dispersion in predictions when running several inference steps over a specific data point. For the PM-value regression task, every single forward pass (i) outputs a normal distribution (with mean $\hat{\mu}_{t,i}$ and variance $\hat{\sigma}_{t,i}^2$). Thus, multiple forward passes result in a (uniformly weighted) mixture of normal distributions with the mean $\hat{\mu}_{t,mix} \in \mathbb{R}$ and variance $\hat{\sigma}_{t,mix}^2 \in (0, \infty)$ calculated as follows:

$$\hat{\mu}_{t,mix} = \frac{1}{M} \sum_{i=1}^M \hat{\mu}_{t,i} \quad (\text{IV.35})$$

$$\hat{\sigma}_{t,mix}^2 = \frac{1}{M} \sum_{i=1}^M (\hat{\sigma}_{t,i}^2 + \hat{\mu}_{t,i}^2) - \hat{\mu}_{t,mix}^2 \quad (\text{IV.36})$$

Given a predicted mean and variance, we can then estimate a point prediction $\hat{y}_t = \hat{\mu}_{t,mix}$, a lower bound $\hat{L}_t = \hat{\mu}_{t,mix} - z\hat{\sigma}_{t,mix}$, and an upper bound $\hat{U}_t = \hat{\mu}_{t,mix} + z\hat{\sigma}_{t,mix}$, where z is the standard score of a normal distribution. For example, with 95% prediction interval (i.e., $P(L_t < y_t < U_t) = 0.95$), we use $z = 1.96$. For the threshold exceedance prediction task, every single forward pass outputs a predictive probability. Thus, we can combine predictions from multiple forward passes by simply averaging the predicted probabilities.

For empirical evaluation, we use the NLL and cross-entropy as evaluation metrics. Additionally, we use the Continuous Ranked Probability Score (CRPS) [55]. It is a widely used metric to evaluate probabilistic forecasts that generalizes the MAE to a probabilistic setting. CRPS measures the difference in the cumulative distribution function (CDF) between the forecast and the true observation. It can be derived analytically for parametric distributions or estimated using samples if the CDF is

unknown (e.g., originating from VI or MCMC). Given a forecast CDF \mathbf{F} and an empirical CDF of scalar observation y :

$$CRPS(\mathbf{F}, y) = \int_{\mathbb{R}} (\mathbf{F}(\hat{y}) - \mathbb{1}(\hat{y} - y))^2 d\hat{y} \quad (\text{IV.37})$$

IV.4.1 Bayesian Neural Networks (BNNs)

In BNNs, we use Bayesian methods for inferring a posterior distribution over the weights $p(\mathbf{w})$ rather than being constrained to weights of fixed values [63]. These weight distributions are parameterized by trainable variables θ . For example, the trainable variable can represent the mean and variance of a Gaussian distribution $\theta = (\mu, \sigma^2)$, from which the weights can be sampled $\mathbf{w} \sim \mathcal{N}(\mu, \sigma^2)$.

The objective of training is to calculate the posterior, but obtaining explicit posterior densities through Bayesian inference is intractable. Additionally, using Markov chain Monte Carlo (MCMC) to estimate the posterior can be computationally prohibitive. Instead, we can leverage new techniques in variational inference [65, 66] that make BNNs computationally feasible by using a variational approximation $q(\mathbf{w}|\theta)$ to the posterior. Thereby, the goal of learning is to find the variational parameters that minimize the Kullback–Leibler (KL) divergence between the variational approximation $q(\mathbf{w}|\theta)$ and the true posterior distribution given training data $p(\mathbf{w}|\mathcal{D})$. This can be achieved by minimizing the negative variational lower bound of the marginal likelihood [57]:

$$Loss(\theta) = KL[q(\mathbf{w}|\theta)||p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log p(\mathcal{D}|\mathbf{w})] \quad (\text{IV.38})$$

Assuming Gaussian prior and posterior, we can compute the KL divergence in a closed-form:

$$KL[q(\mathbf{w}|\theta)||p(\mathbf{w})] = \log \frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - 0.5 \quad (\text{IV.39})$$

Additionally, the distribution over activations will also be Gaussian. Thus, we can take advantage of the local reparameterization trick [67], in which we sample from the distribution over activations rather than sampling the weights individually. Consequently, we reduce the variance of stochastic gradients, resulting in faster and more stable training. However, to allow for more flexibility and adaptations to a wide range of situations, we can use non-Gaussian distributions over the weights and MC gradients, as proposed by [58]:

$$Loss(\theta) \approx \sum_{i=1}^M \log q(\mathbf{w}^i|\theta) - \log P(\mathbf{w}^i) - \log P(\mathcal{D}|\mathbf{w}^i) \quad (\text{IV.40})$$

where $-\log P(\mathcal{D}|\mathbf{w}^i)$ is the NLL for time-series forecasting (Equation IV.34) or the cross-entropy for the threshold exceedance prediction (Equation IV.27).

Our implementation uses Laplace distributions as priors and Gaussian as approximate posteriors in time-series forecasting, resulting in better empirical performance. For threshold exceedance prediction, using Gaussian for both the

priors and posteriors and using the local reparameterization trick results in better performance. During inference, we sample the weight distributions and perform a forward pass to obtain a prediction. We use ($M = 1000$) samples for every data point to estimate the model’s uncertainty. This corresponds to sampling from infinite ensembles of neural networks. Therefore, combining the outputs from different samples gives information on the model’s uncertainty.

We train the BNN model using training data of one year (2019). Figure IV.27 shows the learning curves for the PM-value regression task. Then we evaluate the model using data of one month (January 2020). Figures IV.28 and IV.29 show the results of PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.4 shows a summary of performance results in all monitoring stations. The arrows alongside the metrics indicate which direction is better for that specific metric.

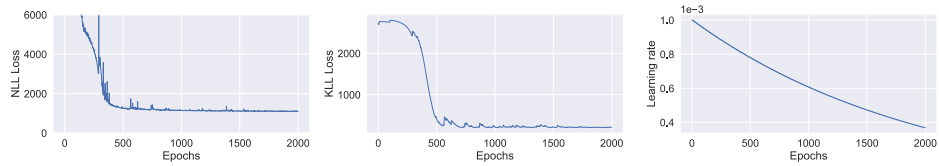


Figure IV.27: Learning curve of training a BNN model to forecast PM-values. **Left:** negative log-likelihood loss; **Center:** KL loss estimated using MC sampling; **Right:** learning rate of exponential decay.

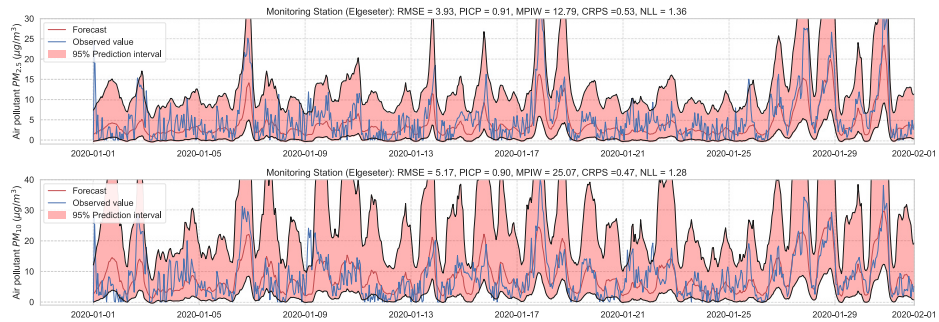


Figure IV.28: Probabilistic forecasting of multivariate time-series of air quality using a BNN model in one representative monitoring station.

IV.4.2 Standard Neural Networks with MC Dropout

Although BNNs are more flexible in reasoning about the model’s uncertainty with Bayesian analysis, they are computationally less efficient and take longer to converge than standard (non-Bayesian) neural networks. Additionally, they require double the number of parameters at deployment compared to neural networks of the same size. A possible solution is to gracefully prune the weights with the lowest signal-to-noise ratio [58], but this leads to a loss in uncertainty information.

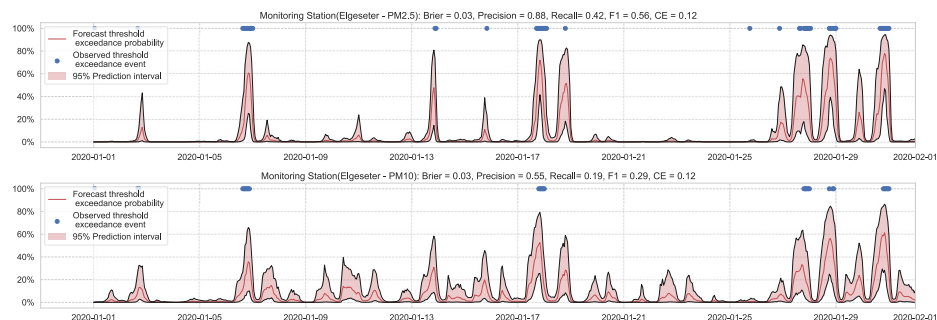


Figure IV.29: Predicting threshold exceedance probability of air pollutant level using a BNN model.

Table IV.4: Summary of performance results when forecasting the PM-value and threshold exceedance using a BNNs model.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification					
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓	
Using The s	$PM_{2.5}$	4.81	0.99	17.62	0.51	1.29	0.04	1.00	0.44	0.61	0.13	
	PM_{10}	5.86	0.94	26.12	0.50	1.28	0.03	1.00	0.30	0.47	0.09	
E6-Tiller	$PM_{2.5}$	3.77	0.92	13.25	0.54	1.39	0.02	0.00	0.00	0.00	0.08	
	PM_{10}	9.40	0.92	34.18	0.48	1.26	0.06	0.00	0.00	0.00	0.23	
Elgeseter	$PM_{2.5}$	3.93	0.91	12.79	0.53	1.36	0.03	0.88	0.42	0.56	0.12	
	PM_{10}	5.17	0.90	25.07	0.47	1.28	0.03	0.55	0.19	0.29	0.12	
Torvet	$PM_{2.5}$	4.07	0.90	10.83	0.48	1.30	0.03	0.75	0.46	0.57	0.13	
	PM_{10}	5.25	0.93	18.47	0.43	1.17	0.03	0.50	0.23	0.32	0.10	

Therefore, it would be more convenient to obtain uncertainty directly from standard neural networks. One simple approach we can use is Monte Carlo dropout as an approximate Bayesian method for representing model uncertainty. As shown in [68], MC dropout can be interpreted as performing variational inference. More specifically, it is mathematically equivalent to an approximation of a probabilistic deep Gaussian process.

Essentially, we train a standard neural network model with dropout. Then we keep the dropout during inference and run multiple forward passes using the same data input. This corresponds to sampling nodes, which is equivalent to sampling from ensembles of neural networks [69]. By measuring the spread in predictions, we estimate the predictive uncertainty. In our implementations, we train a standard neural network model with a 50% dropout rate. Then we evaluate it with 50% dropout and ($M = 1000$) samples. Figures IV.30 and IV.31 show the results of the PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.5 shows a summary of performance results in all monitoring stations.

IV.4.3 Deep Ensembles

An established method to improve performance is to train an ensemble of neural networks with different configurations [71]. Additionally, many model types can be interpreted by using ensemble methods. For example, sampling weights of BNNs is equivalent to sampling from an infinite ensemble of networks, while sampling the

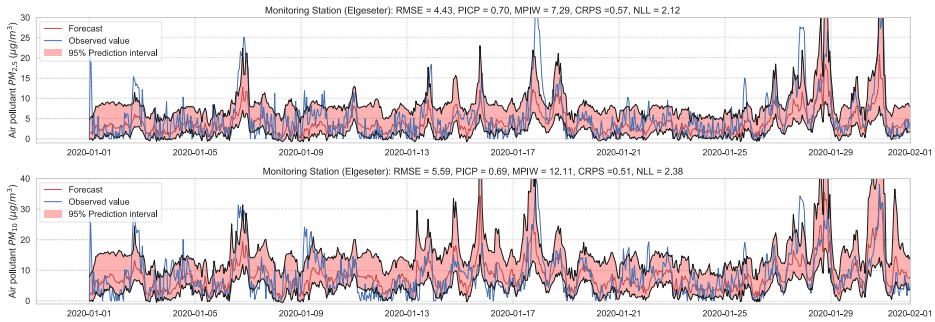


Figure IV.30: Probabilistic forecasting of multivariate time-series air quality using a standard neural network model with MC dropout.

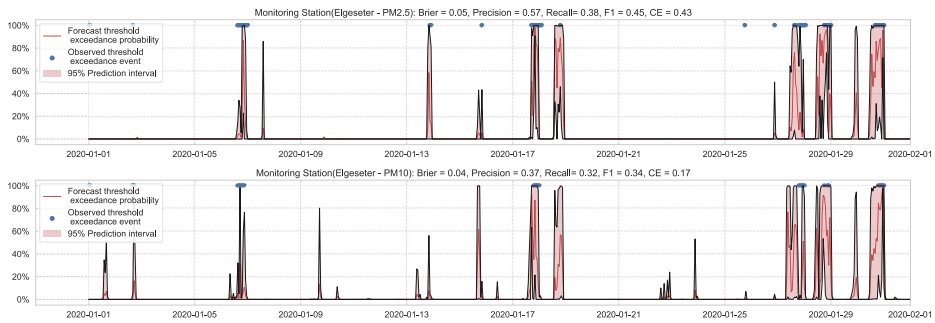


Figure IV.31: Predicting the threshold exceedance probability of air pollutants level using a standard neural network with MC dropout.

Table IV.5: Summary of performance results when forecasting the PM-value and threshold exceedance using a standard neural network with MC dropout.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification				
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓
Bakke kirke	$PM_{2.5}$	5.34	0.69	9.40	0.60	2.30	0.04	0.65	0.51	0.57	0.31
	PM_{10}	6.42	0.66	12.45	0.59	3.35	0.03	0.67	0.48	0.56	0.10
E6-Tiller	$PM_{2.5}$	3.75	0.72	7.26	0.60	2.24	0.01	0.00	0.00	0.00	0.24
	PM_{10}	9.49	0.71	16.62	0.51	2.30	0.07	0.18	0.04	0.06	0.57
Elgeseter	$PM_{2.5}$	4.43	0.70	7.29	0.57	2.12	0.05	0.57	0.38	0.45	0.43
	PM_{10}	5.59	0.69	12.11	0.51	2.38	0.04	0.37	0.32	0.34	0.17
Torvet	$PM_{2.5}$	4.60	0.55	5.26	0.57	2.91	0.04	0.68	0.44	0.53	0.33
	PM_{10}	5.63	0.62	8.94	0.51	2.51	0.03	0.56	0.35	0.43	0.14

Paper IV: Probabilistic Deep Learning to Quantify Uncertainty in Air Quality Forecasting

nodes in MC dropout compares to sampling from a finite ensemble [69]. Remarkably, we can also use (deep) ensembles to estimate the predictive uncertainty in neural network models [70].

Essentially, we train multiple neural networks with different parameter initialization on the same data. The stochastic optimization and random initialization ensure that the trained networks are sufficiently independent. During inference, we run a forward pass on the multiple neural networks using the same data input. We estimate the uncertainty by measuring the dispersion in predictions resulting from the multiple neural networks. In our implementation, we train an ensemble of 10 networks. Figures IV.32 and IV.33 show the results of PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.6 shows a summary of performance results in all monitoring stations.

The main drawback of the deep ensemble method is the computational cost and number of parameters required at run time compared to standard neural networks, which is an order of magnitude in our case. One solution is to use knowledge distillation [72] to compress the knowledge from the ensemble into a single model, but this also leads to a loss in uncertainty information.

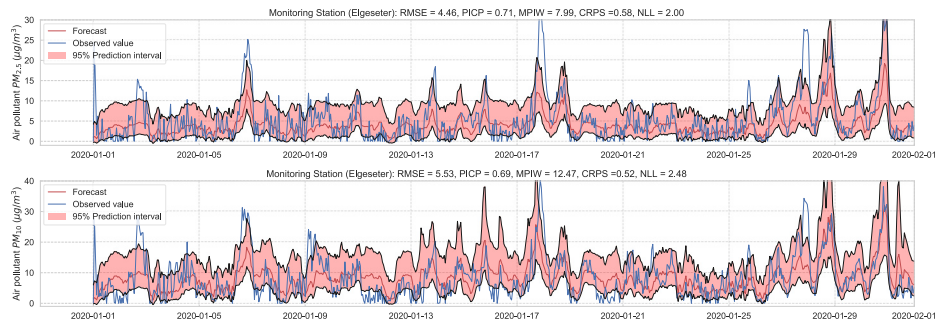


Figure IV.32: Probabilistic forecasting of multivariate time-series air quality using a deep ensemble.

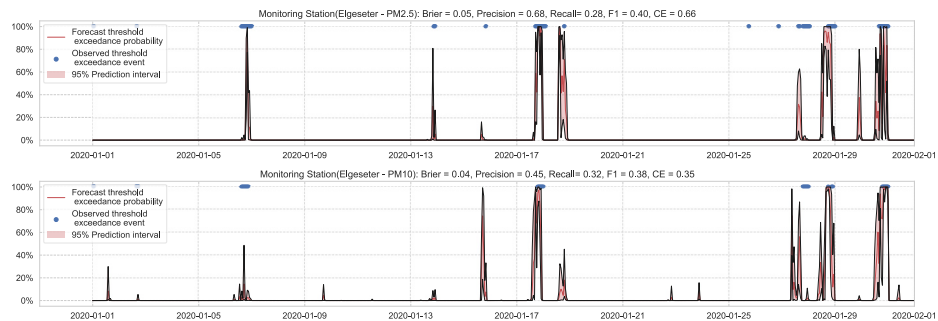


Figure IV.33: Predicting threshold exceedance probability of air pollutants level using a deep ensemble.

Table IV.6: Summary of performance results when forecasting the PM-value and threshold exceedance using a deep ensemble.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification				
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓
Bakke kirke	$PM_{2.5}$	5.29	0.77	11.65	0.57	1.67	0.05	0.69	0.53	0.60	0.55
	PM_{10}	6.21	0.70	14.00	0.57	2.46	0.03	0.60	0.36	0.45	0.26
E6-Tiller	$PM_{2.5}$	3.78	0.77	8.46	0.58	1.84	0.01	0.00	0.00	0.00	0.34
	PM_{10}	9.44	0.72	16.07	0.50	2.14	0.07	0.31	0.08	0.12	1.16
Elgeseter	$PM_{2.5}$	4.46	0.71	7.99	0.58	2.00	0.05	0.68	0.28	0.40	0.66
	PM_{10}	5.53	0.69	12.47	0.52	2.48	0.04	0.45	0.32	0.38	0.35
Torvet	$PM_{2.5}$	4.45	0.57	5.13	0.56	2.66	0.04	0.73	0.31	0.43	0.55
	PM_{10}	5.39	0.64	8.68	0.49	2.19	0.03	0.62	0.19	0.29	0.30

IV.4.4 Recurrent Neural Network with MC Dropout

While standard neural networks are powerful models at representational learning, they do not exploit the inherent temporal correlation in air quality data since they act only on static, fixed contextual windows. To address this shortcoming, we can use recurrent neural networks (RNNs), which have cyclic connections from previous time steps, to learn the temporal dynamics of sequential data. Specifically, a hidden state from the last time step is stored and used in addition to the input to generate the current state and output. One class of RNNs is the long short-term memory (LSTM), which is used extensively in sequence modeling tasks, such as modeling language [205], forecasting weather [206] and traffic [207], recognizing human activity [208], and recently forecasting COVID-19 transmission [209]. An LSTM has gated memory cells to control how much information to forget from previous states and how much information to use to update current states [76].

A simple approach to capture uncertainty within an LSTM model is to use Monte Carlo dropout to approximate Bayesian inference [172]. To implement dropout, we apply a mask that randomly drops some network units with their inputs, output and recurrent connections [210]. Our implementation trains a model of two LSTM layers with a 50% dropout rate and evaluates it with 50% dropout and ($M = 1000$) samples. Figures IV.34 and IV.35 show the results of the PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.7 shows a summary of performance results in all monitoring stations.

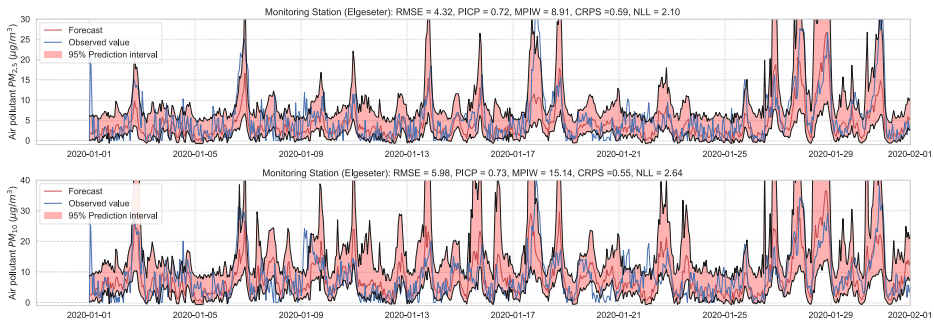


Figure IV.34: Probabilistic forecasting of multivariate time-series air quality using an LSTM model with MC dropout.

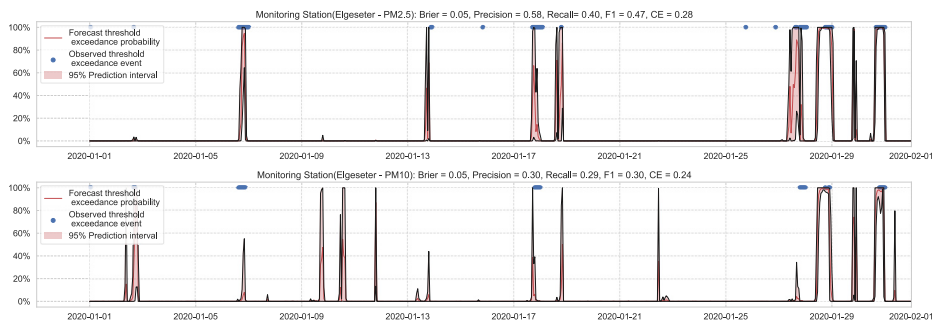


Figure IV.35: Predicting threshold exceedance probability of air pollutants level using an LSTM model with MC dropout.

Table IV.7: Summary of performance results when forecasting PM-value or threshold exceedance using an LSTM model with MC dropout.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification					
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓	
Bakke kirke	$PM_{2.5}$	5.01	0.88	14.01	0.53	1.47	0.05	0.66	0.53	0.58	0.25	
	PM_{10}	6.25	0.82	19.28	0.54	1.78	0.03	0.59	0.48	0.53	0.14	
E6-Tiller	$PM_{2.5}$	3.90	0.72	7.45	0.62	2.31	0.02	0.00	0.00	0.00	0.11	
	PM_{10}	9.68	0.74	18.96	0.53	2.03	0.08	0.24	0.12	0.16	0.43	
Elgeseter	$PM_{2.5}$	4.32	0.72	8.91	0.59	2.10	0.05	0.58	0.40	0.47	0.28	
	PM_{10}	5.98	0.73	15.14	0.55	2.64	0.05	0.30	0.29	0.30	0.24	
Torvet	$PM_{2.5}$	4.19	0.56	6.88	0.58	4.79	0.05	0.58	0.42	0.49	0.30	
	PM_{10}	5.81	0.61	11.33	0.54	4.03	0.03	0.43	0.35	0.38	0.1	

IV.4.5 Graph Neural Networks with MC Dropout

By using RNNs, we can capture the temporal (i.e., intra-series) correlations in the time-series of air quality data. However, we need also to exploit the inherent structural (i.e., inter-series) correlations between multiple sensing stations. We can use Graph Neural Networks (GNNs) to address this, which operate on graph-structured data. In essence, GNNs update the features of a graph node by aggregating the features of its adjacent nodes. For example, a node can be a single monitoring station in our case. In the end, GNNs apply a shared layer on each node to obtain a prediction for each node.

In our setting, we assume each sensing station to be a node in a weighted and directed graph, represented by a learnable adjacency matrix. We use a slight variation of the GNNs suggested by Cao et al. [78] to forecast multivariate time-series of air quality. The main idea is to learn a correlation graph directly from data (i.e., learn a graph of sensor nodes without a pre-defined typology) and then learn the structural and temporal correlation in the frequency domain. To capture the temporal correlations, we use a layer of 1D convolution, and three sub-layers of Gated Linear Units (GLUs) [80], while we use Graph Convolutional Networks (GCNs) [79] to capture the structural correlations. In the end, we use two shared layers of fully connected neural networks to predict each node’s output.

We use the MC dropout applied to the GLUs, GCNs, and to the fully connected layers to estimate model’s uncertainty [211]. We train the model with a 50% dropout rate and evaluate it with 50% dropout and ($M = 1000$) samples. Figures IV.36

and IV.37 show the results of the PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.8 shows a summary of performance results in all monitoring stations.

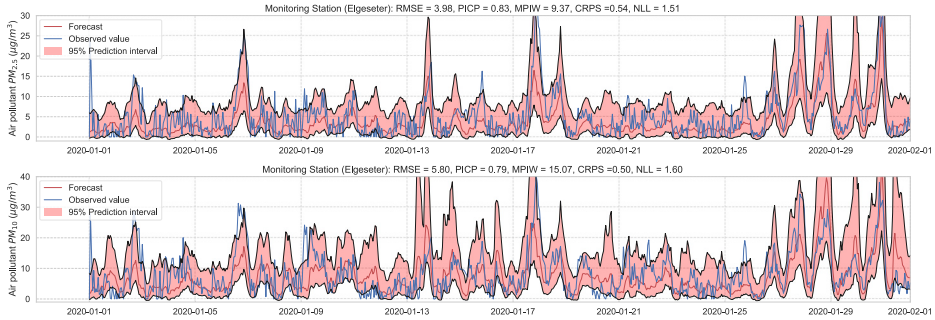


Figure IV.36: Probabilistic forecasting of multivariate time-series air quality using a GNN model with MC dropout.

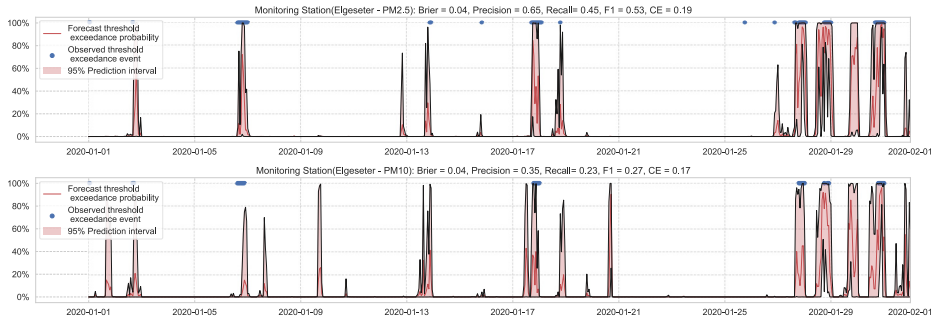


Figure IV.37: Predicting threshold exceedance probability of air pollutants level using a GNN model with MC dropout.

Table IV.8: Summary of performance results when forecasting the PM-value or threshold exceedance using a GNN model with MC dropout.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification				
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓
Bakke kirke	$PM_{2.5}$	4.70	0.88	12.33	0.52	1.41	0.05	0.61	0.53	0.56	0.21
	PM_{10}	6.26	0.79	16.10	0.54	1.83	0.03	0.43	0.36	0.39	0.11
E6-Tiller	$PM_{2.5}$	3.80	0.83	9.14	0.57	1.60	0.02	0.00	0.00	0.00	0.11
	PM_{10}	9.46	0.80	19.89	0.48	1.59	0.07	0.19	0.06	0.09	0.35
Elgeseter	$PM_{2.5}$	3.98	0.83	9.37	0.54	1.51	0.04	0.65	0.45	0.53	0.19
	PM_{10}	5.80	0.79	15.07	0.50	1.60	0.04	0.35	0.23	0.27	0.17
Torvet	$PM_{2.5}$	4.27	0.68	6.19	0.50	2.04	0.05	0.55	0.46	0.50	0.22
	PM_{10}	5.55	0.70	10.39	0.47	1.83	0.03	0.36	0.35	0.35	0.11

IV.4.6 Stochastic Weight Averaging–Gaussian (SWAG)

An alternative approach to estimate uncertainty in a neural network is to approximate (Gaussian) posterior distributions over the weights using the geometric information in the trajectory of a stochastic optimizer. This approach is named Stochastic Weight Averaging–Gaussian (SWAG) [73]. Notably, SWAG does not optimize the approximate distributions directly, such as in BNNs. Instead, it estimates the mean by calculating a running average of the weights traversed by the optimizer with a modified learning rate schedule (stochastic weight averaging [74]). In addition, SWAG estimates the standard deviation by a diagonal covariance plus of a low-rank deviation matrix using information from a running average of the second moment of the traversed weights.

During inference, we run multiple forward passes using the same data input while drawing samples of the weights from the approximate posterior. By measuring the spread in predictions, we estimate the predictive uncertainty. In our implementations, we train a simple feed-forward neural network model with SWAG and evaluate it with ($M = 1000$) samples. Figures IV.38 and IV.39 show the results of the PM-value regression and threshold exceedance classification in one representative monitoring station. Table IV.9 shows a summary of performance results in all monitoring stations.

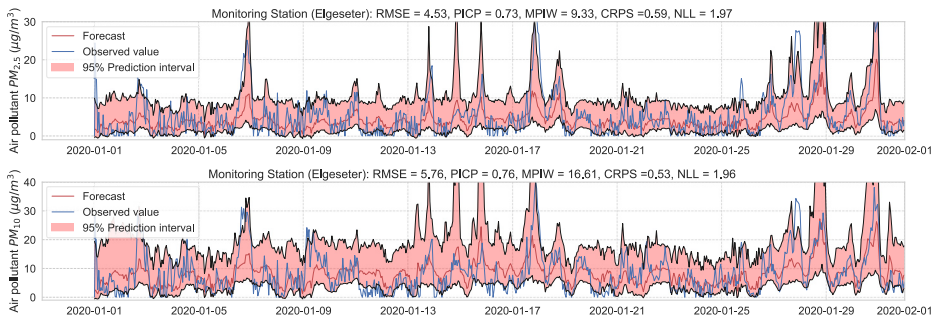


Figure IV.38: Probabilistic PM-value regression using a SWAG model.

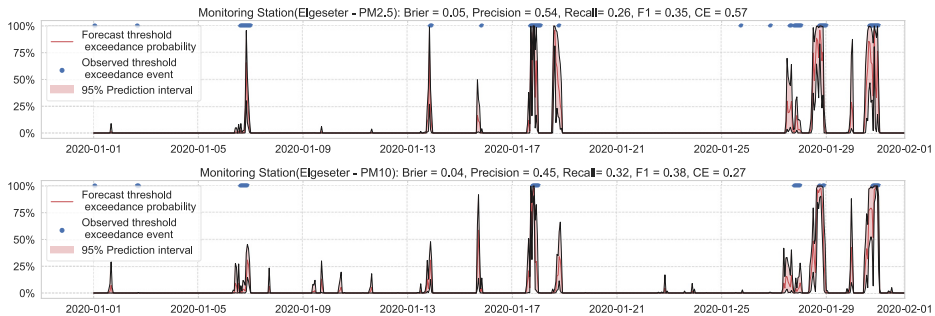


Figure IV.39: Probabilistic threshold exceedance classification using a SWAG model.

Table IV.9: Summary of performance results when using a SWAG model with adversarial training.

Station	Particulate	PM-Value Regression					Threshold Exceedance Classification				
		RMSE↓	PICP↑	MPIW↓	CRPS↓	NLL↓	Brier↓	Precision↑	Recall↑	F1↑	CE↓
Bakke kirke	$PM_{2.5}$	5.51	0.79	13.13	0.58	1.64	0.04	0.66	0.64	0.65	0.20
	PM_{10}	6.66	0.78	17.95	0.57	2.03	0.04	0.49	0.61	0.54	0.12
E6-Tiller	$PM_{2.5}$	3.76	0.79	9.25	0.59	1.82	0.01	0.00	0.00	0.00	0.10
	PM_{10}	9.35	0.82	21.28	0.49	1.73	0.08	0.19	0.08	0.11	0.49
Elgeseter	$PM_{2.5}$	4.53	0.73	9.33	0.59	1.97	0.04	0.60	0.45	0.52	0.21
	PM_{10}	5.76	0.76	16.61	0.53	1.96	0.04	0.37	0.45	0.41	0.18
Torvet	$PM_{2.5}$	4.58	0.79	10.33	0.54	1.63	0.04	0.67	0.50	0.57	0.20
	PM_{10}	5.62	0.71	12.48	0.50	1.76	0.03	0.50	0.42	0.46	0.13

IV.4.7 Improving Uncertainty Estimation with Adversarial Training

Generally, it is desirable to have a model with a smooth conditional output distribution with respect to its input because most measured phenomena are inherently smooth. This idea of distributional smoothing has been used as a regularization technique by encouraging a model to be less overconfident, for example, using label smoothing [212, 213] or virtual adversarial training [214].

For uncertainty estimation, distributional smoothing can improve the quality of the predictive uncertainty depending on the direction of smoothing. For example, smoothing along a random direction can be less effective while being computationally expensive in all directions. Lakshminarayanan et al. [70] propose using adversarial training with the fast gradient sign method [215] to smooth the predictive distribution along the direction where the loss is high. Qin et al. [216] investigate the relationship between adversarial robustness and predictive uncertainty. They show that inputs that are sensitive to adversarial perturbations are more likely to have unreliable predictive uncertainty. Based on this insight, they propose a new training approach that smooths training labels based on their input adversarial robustness.

In this paper, we instead propose using the “free” adversarial method [217], which recycles the gradient information from regular training to quickly generate adversarial data. Thus, we locally smooth the prediction distribution along the adversarial direction with virtually no additional cost. Figures IV.40 illustrates the improvement in uncertainty estimation when using adversarial training in PM-value regression task, while figure IV.41 in threshold exceedance classification (using MC dropout as an example). Generally, we observe that adversarial training improves the NLL and CE (i.e., making less overconfident predictions) with negligible effects on other metrics. This, of course, depends on the size of the adversarial perturbation, which can be tuned accordingly. We observe that increasing the perturbation size can have adverse effects on the accuracy metrics, which is expected [218]. We also observe that adversarial training led to more improvements in the PM-value regression than in the threshold exceedance classification. This is because threshold exceedance events are rare, and smoothing the predictive distribution does not help catch these events.

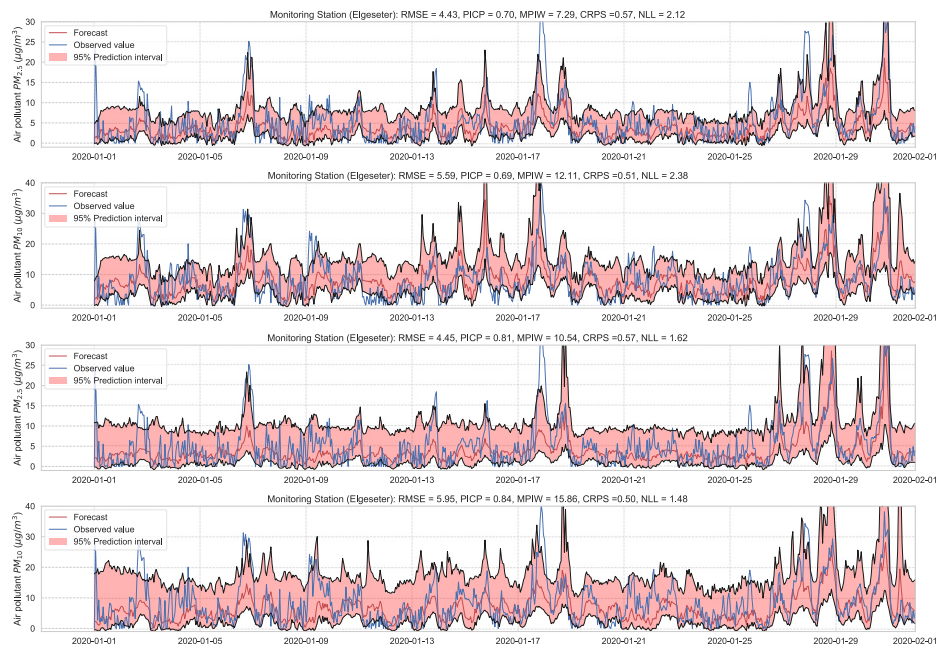


Figure IV.40: Comparison of uncertainty estimation in PM-value regression when training (**top**) without adversarial training versus (**bottom**) with adversarial training. Using Adversarial training leads to smoother predictive distribution; thus, lower NLL (less overconfident predictions).

IV.5 Discussion

In this section, we investigate some of the implications of the study. In particular, we perform a comparative analysis to evaluate the selected probabilistic models based on empirical performance, reliability of confidence estimate, and practical applicability. Then we close by investigating the practical impact of uncertainty quantification on decision-making.

IV.5.1 Empirical Performance

In Section IV.4, we summarized the performance of each model in a tabular format. Here, we compare all the models according to their empirical performance in a single monitoring station.

Figure IV.42 shows a comparative summary of empirical performance in the PM-value regression task. We observe that all models perform consistently well with slight variations. The BNN model performs better in metrics that assess the quality of a probabilistic forecast (i.e., in CRPS and NLL). This is expected since BNNs provide the closest approximation to Bayesian inference, while other models provide only a crude approximation. Interestingly, the GNNs with MC dropout perform very closely to BNNs in CRPS and NLL. By scoring better in CRPS, BNNs

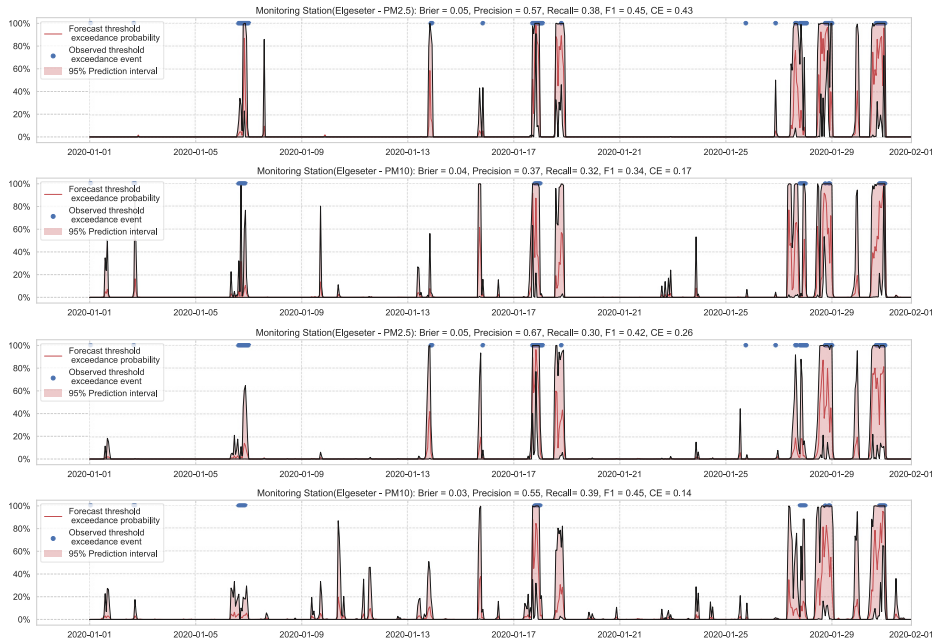


Figure IV.41: Comparison of uncertainty estimation in threshold exceedance classification when training (**top**) without adversarial training versus (**bottom**) with adversarial training. Using Adversarial training leads to smoother predictive distribution; thus, lower CE (less overconfident predictions).

also score better in accuracy metrics (i.e., RMSE) since the CRPS generalizes the MAE to a probabilistic setting.

The PICP and MPIW are conflicting metrics that simultaneously assess the quality of the generated prediction interval. For example, by increasing the width of a prediction interval (higher MPIW), more values will be inside the predicted intervals (higher PICP). Thus, we observe that BNNs perform well in PICP but poorly in MPIW.

Figure IV.43 shows a comparative summary of empirical performance in the threshold exceedance classification task. We observe that the BNN model performs better in metrics that measure the quality of probabilistic predictions (scoring rule): Brier score and cross-entropy. We also observe that the performance is inconclusive in metrics intended for deterministic classification (F1, precision, recall). This shows that these metrics are not appropriate for probabilistic prediction. Additionally, these metrics are biased by class imbalance since threshold exceedance is a rare event.

IV.5.2 Reliability of Confidence Estimate

In Section IV.5.1, we evaluate the selected models using metrics of accuracy and predictive probabilities separately. However, for decision-making, it is crucial to avoid over-confident, incorrect predictions. Therefore, evaluating the reliability of

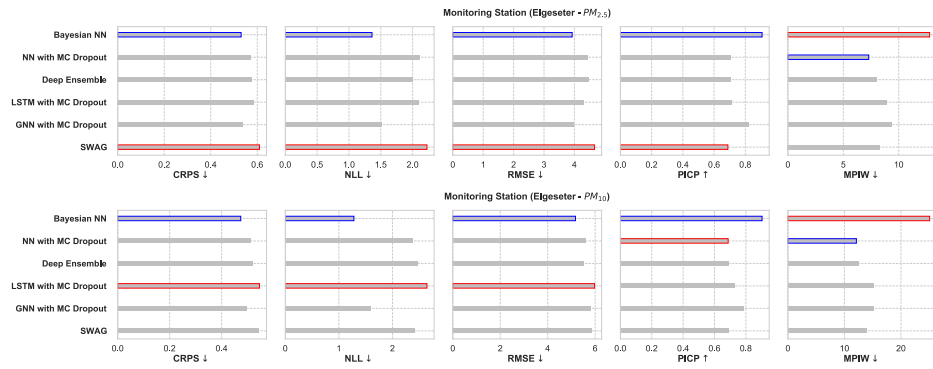


Figure IV.42: Comparison of empirical performance of the selected probabilistic models in the PM-value regression task. The comparison is according to five performance metrics (left to right): CRPS, NLL, RMSE, PICP, and MPIW. Blue highlights the best performance, while red highlights the worst performance. The arrows alongside the metrics indicate which direction is better for that specific metric.

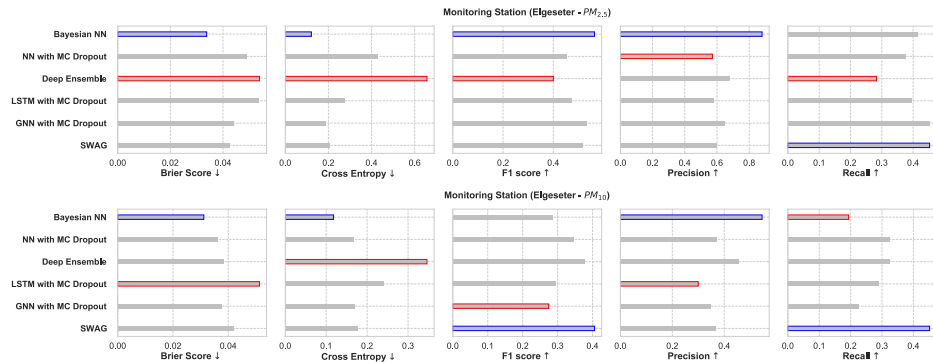


Figure IV.43: Comparison of empirical performance of the selected probabilistic models in the threshold exceedance classification task. The comparison is according to five performance metrics (left to right): Brier score, cross-entropy, FI score, precision, and recall. Blue highlights the best performance, while red highlights the worst performance.

a confidence estimate is indispensable when selecting probabilistic models. One approach to evaluate reliability is to measure the amount of loss (or the number of incorrect predictions) a model makes when its confidence is above a certain threshold. This is a slight variation of the *accuracy-versus-confidence* technique suggested by Lakshminarayanan et al. [70] for the classification task.

For the threshold classification task, the confidence interval is bounded between 0 and 1 ($0 \leq CI \leq 1$). Thus, we can define a model confidence to be: $confidence = 1 - CI$ and a confidence threshold $0 \leq \tau \leq 1$. Then we plot the number of incorrect predictions the model makes when its confidence is above

τ . We expect the curve to be monotonically decreasing for a reliable confidence estimate since a rational model has fewer incorrect predictions at high confidence. Additionally, we plot the total number of predictions (in our case, number of hours in a month of test set) as a function of τ . This curve is monotonically decreasing, but the decreasing rate indicates the amount of confidence in a model. The decreasing rate would be high in a model with lower confidence since it makes fewer predictions with high confidence. Figure IV.44 shows plots of loss vs. confidence and count vs. confidence in the threshold classification task. We observe that the BNN model is more reliable by making fewer incorrect predictions at high confidence but that it has the lowest confidence.

For the PM-value regression task, the confidence interval is unbounded ($0 < CI < \infty$). However, to compare models, we can normalize and calculate their relative confidence to be bounded between 0 and 1. Then we plot the regression loss as a function of the confidence threshold τ . Figure IV.45 show the plots of loss vs. confidence and count vs. confidence in the PM-value regression task. We observe that the selected models produce rational behaviors and that the BNN model is more reliable but has the lowest confidence. The curves are smooth since the loss is continuous in the regression task.

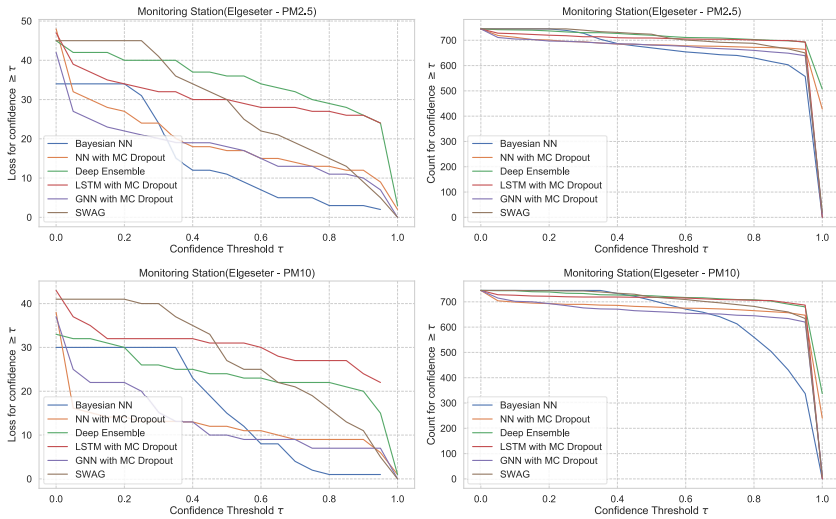


Figure IV.44: Comparison of confidence reliability for the selected probabilistic models in the threshold exceedance task. **Left:** loss versus confidence. **Right:** count versus confidence. The selected models produce are rational, which means the loss-vs-confidence curves are monotonically decreasing.

Figure IV.46 shows the impact of adversarial training in the PM-value regression task, with deep ensembles as an example. We observe that adversarial training can reduce overconfident predictions.

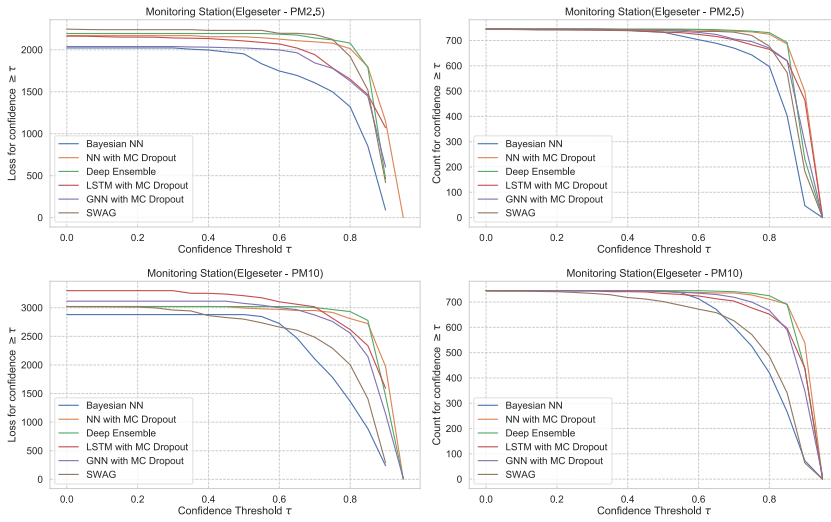


Figure IV.45: Comparison of confidence reliability for the selected probabilistic models in the PM-value regression task. **Left:** loss versus confidence. **Right:** count versus confidence.

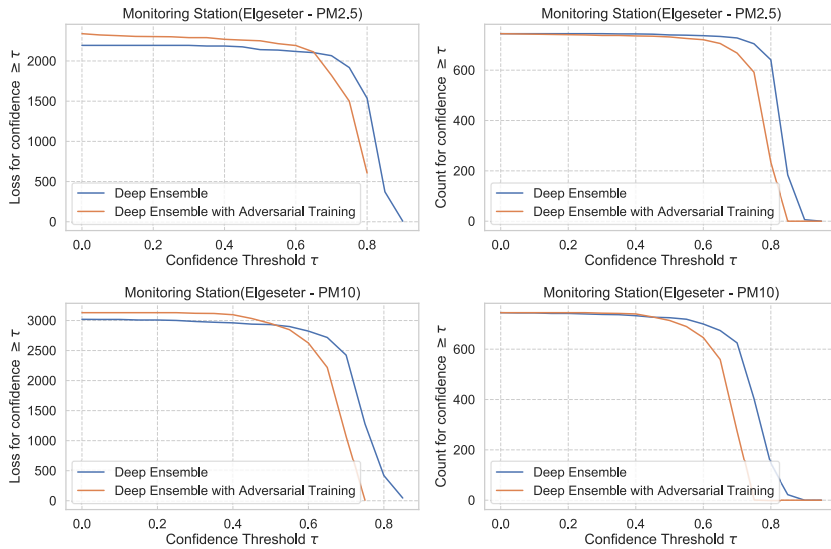


Figure IV.46: Impact of adversarial training on predictive uncertainty in PM-value regression, using deep ensemble as an example. **Left:** loss versus confidence. **Right:** count versus confidence.

IV.5.3 Risk-informed Decisions

The primary motivation of quantifying epistemic uncertainty is to represent how much the model does not know. Evaluating specific decision policies is out of the

scope of this paper, as they depend on the specific costs of countermeasures or the cost of high pollution values. Nonetheless, to show the value of probabilistic models, we investigate the practical impact on decision-making using a case of urban management. Recently, the Norwegian government has proposed strict regulations for particle dust thresholds. That means if the particle dust exceeds a specific threshold, the government will impose a penalty on the municipality for violating pollution regulations. Therefore, the municipality has to decide whether to implement countermeasures in advance if a forecasting model predicts threshold exceedance events.

The challenge is that not all forecasts are correct and occasionally result in false positives and false negatives. A false positive is associated with implementing unnecessary countermeasures, while a false negative is associated with violating pollution regulations. The costs of these two scenarios are context-dependent. Therefore, it is helpful to have a forecasting model that also provides potential flexibility and tradeoffs depending on the cost of false positives and false negatives.

Non-probabilistic models already offer some aspect of this tradeoff by leveraging the aleatoric uncertainty. For example, we can adjust the probability threshold in a binary classification, i.e., implement countermeasures, only if the predictive probability is above τ_1 . Figure IV.48a shows the decision score (in terms of F1, precision, and recall) as a function of aleatoric confidence in a non-probabilistic model. Accepting the model decision even when its predictive probability is low will result in more false positives and thus higher costs of unnecessary countermeasures. On the other hand, accepting the model decisions only when its predictive probability is high will result in more false negatives, thus increasing the risk of violating pollution regulations.

In probabilistic models, we can also leverage epistemic uncertainty to obtain a higher degree of tradeoffs. That means we implement countermeasures only if the predictive probability is above τ_1 **and** the model confidence is above a certain threshold τ_2 . Figure IV.48b shows the resulting decision score as a function of aleatoric and epistemic confidence in a probabilistic model. We use a BNN model as a representative example. To allow a fair comparison, we use a non-probabilistic model with the same architecture and trained with the same conditions. We observe that the probabilistic model provides a wider area of control over the risk profile based on the costs of false positives versus false negatives. In this case, the probabilistic model scores better over a wider range of τ_1 and τ_2 than a non-probabilistic model. This confirms that probabilistic models are more suitable for making more informed and risk-aware decisions.

IV.5.4 Practical Applicability

In Section IV.5.1, we observe that a BNN model performs better in a variety of metrics. Further, BNNs offer an elegant approach to represent model uncertainty without strong assumptions. However, they are computationally demanding, require more parameters, are sensitive to hyperparameter choices, and take longer to converge than non-Bayesian neural networks. Thus, BNNs are difficult to work with, especially on larger datasets. Additionally, the nature of variational approximation and the choice of the prior or posterior distributions bias the predictive uncertainty.

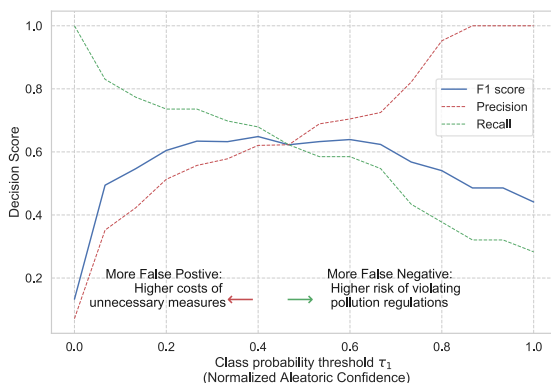


Figure IV.47: Decision score in a non-probabilistic model as a function of class probability threshold (τ_1 corresponding to aleatoric confidence).

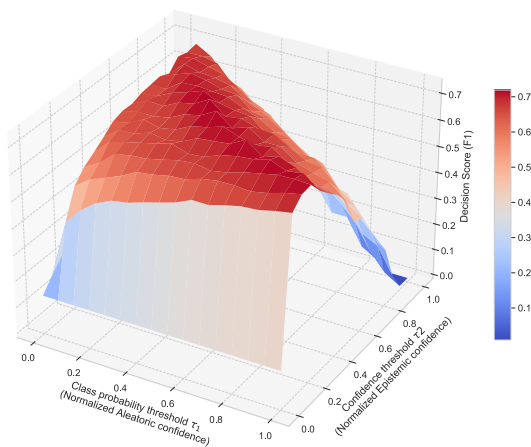


Figure IV.48: Decision score in a probabilistic model as a function of both class probability threshold (τ_1) and model confidence threshold (τ_2 corresponding to epistemic uncertainty).

From a practical perspective, MC Dropout, SWAG, and deep ensemble are convenient and scalable approaches to obtain uncertainty directly from standard neural networks without changing the optimization. Additionally, they have fewer hyperparameters and can be applied independently from the underlying network architecture (such as standard neural network, LSTM, and GNNs). However, these approaches require strong assumptions and provide a crude approximation of the Bayesian inference. During inference, they have high computational costs during inference (proportional to the number of samples M). Further, the uncertainty is estimated by measuring the diversity of predictions that depends on the network’s architecture, size, and properties of the training data and initialization parameters.

Accordingly, there is no guarantee that they produce a diverse prediction for an uncertain input.

The deep ensemble is a convenient non-Bayesian approach that captures aspects of multi-modality. It is better than the bootstrap ensemble in which independent models are trained using independent datasets. The deep ensemble trains every model using the whole dataset since stochastic optimization and random initialization make the models sufficiently independent. However, training deep ensemble is more expensive than SWAG or MC dropout since we need to train M different models.

The underlying assumption used in MC dropout is that it approximates a probabilistic deep Gaussian process. This assumption is based on the idea that a single-layer neural network tends to converge to a Gaussian process in the limit of an infinite number of hidden units [219]. Our results show that MC dropout provides less reliable uncertainty estimates (Figures IV.44 and IV.45), which confirms the results recently presented in other domains, such as computer vision [220] and molecular property prediction [221].

The related works that address quantifying uncertainty in a data-driven forecast of air quality uses either quantile regression (QR) [194], Gaussian processes (GP) [193], or ConvLSTM with MC dropout [191]. We have reimplemented these methods in our problem setting to compare them to the proposed models since we have different datasets and different data inputs. We use a Gradient Tree Boosting [204] for quantile regression and GPyTorch [222] for Gaussian processes. Table IV.10 summarizes the comparison results in the PM-value regression task in one representative monitoring station. We observe that the proposed models perform better, especially in metrics that assess the quality of a probabilistic forecast (i.e., in CRPS and NLL). Quantile regression only estimates the conditional quantile and not a probabilistic distribution. Thus, we can only assess the quality of its prediction interval and not a probabilistic forecast. Gaussian processes offer a Bayesian formalism to reason about uncertainty. However, they have the highest computational cost since they require an inversion of the kernel matrix, which has an asymptotic complexity of $\mathcal{O}(n^3)$, where n is the number of training examples.

Table IV.10: Comparison of the previous works and the proposed models when quantifying uncertainty in data-driven forecast of air quality

Metric	QR[194]	GP [193]	ConvLSTM[191]	BNN	Deep Ensembles	GNN	SWAG
RMSE↓	6.17	6.45	6.46	5.17	5.59	5.80	5.76
PICP↑	0.72	0.92	0.71	0.90	0.69	0.79	0.76
MPIW↓	14.79	36.96	12.18	25.07	12.11	15.07	16.61
CRPS↓	NA	0.53	0.56	0.47	0.51	0.50	0.53
NLL↓	NA	1.36	2.48	1.28	2.38	1.60	1.96

IV.6 Conclusion

This work presents a broad empirical evaluation of the relevant state-of-the-art deep probabilistic models applied in air quality forecasting. Through extensive experiments, we describe training these models and evaluating their predictive uncertainties using various metrics for regression and classification tasks. We

introduce a new state-of-the-art example for air quality forecasting by defining the problem setup and selecting proper input features and models. Then, we apply uncertainty-aware models that exploit the temporal and spatial correlation inherent in air quality data using recurrent and graph neural networks. We propose improving uncertainty estimation using "free" adversarial training to locally smooth the prediction distribution along the adversarial direction with virtually no additional cost. Finally, we show how data-driven probabilistic models can improve the current practice using a real-world example of air quality forecasting in Norway. Particularly, we show the practical impact of uncertainty quantification and demonstrate that probabilistic models are more suitable for making informed decisions.

The results show that the proposed models perform better than previous works in quantifying uncertainty in data-driven air quality forecasts. BNNs provide a more reliable uncertainty estimate but with challenging practical applicability. Additionally, the results demonstrate that MC Dropout, SWAG, and deep ensemble have less reliable uncertainty estimates, but they are more convenient in practical applicability. Our contribution hereof is not about specific model selection but more about navigating the larger design space and the different tradeoffs offered by various probabilistic models.

Future work includes exploring hybrid air quality forecasting combining physics-based and data-driven probabilistic models. Additionally, future work should address uncertainty estimation with out-of-domain or seasonal variations in air quality forecasting. In our experiments, we used a reasonable search space for hyperparameters and network architectures. Thus, future work could target an exhaustive search of hyperparameters, additional data sets, and models (e.g., transformers).

Funding

This work is funded by the European Union's Horizon 2020 research and innovation program, project AI4EU, grant agreement No. 825619.

Acknowledgments

We would like to thank Sigmund Akselsen for reading a draft of the paper and providing detailed feedback on the work. We also thank Tiago Veiga for helping with a script to fetch the datasets.

References

- [49] Chen, T. and Guestrin, C. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.
- [55] Gneiting, T. and Raftery, A. E. "Strictly proper scoring rules, prediction, and estimation". In: *Journal of the American statistical Association* vol. 102, no. 477 (2007), pp. 359–378.

-
- [56] Nix, D. A. and Weigend, A. S. “Estimating the mean and variance of the target probability distribution”. In: *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*. Vol. 1. IEEE. 1994, pp. 55–60.
- [57] Graves, A. “Practical variational inference for neural networks”. In: *Advances in neural information processing systems* vol. 24 (2011).
- [58] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. “Weight uncertainty in neural network”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1613–1622.
- [59] Louizos, C. and Welling, M. “Multiplicative normalizing flows for variational bayesian neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2218–2227.
- [60] Neal, R. M. *Bayesian learning for neural networks*. Vol. 118. 2012.
- [61] Welling, M. and Teh, Y. W. “Bayesian learning via stochastic gradient Langevin dynamics”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 681–688.
- [62] Chen, T., Fox, E., and Guestrin, C. “Stochastic gradient hamiltonian monte carlo”. In: *International conference on machine learning*. PMLR. 2014, pp. 1683–1691.
- [63] MacKay, D. J. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* vol. 4, no. 3 (1992), pp. 448–472.
- [64] Ritter, H., Botev, A., and Barber, D. “A scalable laplace approximation for neural networks”. In: *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*. Vol. 6. International Conference on Representation Learning. 2018.
- [65] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. “Stochastic variational inference.” In: *Journal of Machine Learning Research* vol. 14, no. 5 (2013).
- [66] Kingma, D. P. and Welling, M. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).
- [67] Kingma, D. P., Salimans, T., and Welling, M. “Variational dropout and the local reparameterization trick”. In: *Advances in neural information processing systems* vol. 28 (2015), pp. 2575–2583.
- [68] Gal, Y. and Ghahramani, Z. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.
- [69] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* vol. 15, no. 1 (2014), pp. 1929–1958.
- [70] Lakshminarayanan, B., Pritzel, A., and Blundell, C. “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles”. In: *Advances in Neural Information Processing Systems* vol. 30 (2017).
- [71] Dietterich, T. G. “Ensemble methods in machine learning”. In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

- [72] Hinton, G., Vinyals, O., and Dean, J. “Distilling the knowledge in a neural network”. In: *NIPS 2014 Deep Learning and Representation Learning Workshop*. 2014.
- [73] Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. “A simple baseline for bayesian uncertainty in deep learning”. In: *Advances in Neural Information Processing Systems* vol. 32 (2019), pp. 13153–13164.
- [74] Izmailov, P., Podoprikin, D., Garipov, T., Vetrov, D., and Wilson, A. G. “Averaging weights leads to wider optima and better generalization”. In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*. Association For Uncertainty in Artificial Intelligence (AUAI). 2018, pp. 876–885.
- [76] Sak, H., Senior, A. W., and Beaufays, F. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling”. In: *Google Research* (2014).
- [78] Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., and Zhang, Q. “Spectral Temporal Graph Neural Network for Multivariate Time-series Forecasting”. In: *Proceedings of Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. Vol. 33. 2020, pp. 17766–17778.
- [79] Kipf, T. N. and Welling, M. “Semi-supervised classification with graph convolutional networks”. In: *Proceedings of the 5th International Conference on Learning Representations, ICLR '17*, 2017.
- [80] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. “Language modeling with gated convolutional networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 933–941.
- [171] Ghahramani, Z. “Probabilistic machine learning and artificial intelligence”. In: *Nature* vol. 521, no. 7553 (2015), pp. 452–459.
- [172] Zhu, L. and Laptev, N. “Deep and confident prediction for time series at uber”. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2017, pp. 103–110.
- [173] Kendall, A., Gal, Y., and Cipolla, R. “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491.
- [174] Kendall, A. and Gal, Y. “What uncertainties do we need in Bayesian deep learning for computer vision?” In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 5580–5590.
- [175] Chien, J.-T. and Ku, Y.-C. “Bayesian Recurrent Neural Network for Language Modeling”. In: *IEEE Transactions on Neural Networks and Learning Systems* vol. 27, no. 2 (2016), pp. 361–374.
- [176] Xiao, Y. and Wang, W. Y. “Quantifying uncertainties in natural language processing tasks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 7322–7329.

- [177] Ott, M., Auli, M., Grangier, D., and Ranzato, M. “Analyzing uncertainty in neural machine translation”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3956–3965.
- [178] Meyer, G. P. and Thakurdesai, N. “Learning an uncertainty-aware object detector for autonomous driving”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10521–10527.
- [179] Marécal, V., Peuch, V.-H., Andersson, C., Andersson, S., Arteta, J., Beekmann, M., Benedictow, A., Bergström, R., Bessagnet, B., Cansado, A., et al. “A regional air quality forecasting system over Europe: the MACC-II daily ensemble production”. In: *Geoscientific Model Development* vol. 8, no. 9 (2015), pp. 2777–2813.
- [180] Walker, S.-E., Hermansen, G. H., and Hjort, N. L. “Model selection and verification for ensemble based probabilistic forecasting of air pollution in Oslo, Norway”. In: *60th ISI World Statistics Congress (WSC)*. 2015.
- [181] Garaud, D. and Mallet, V. “Automatic calibration of an ensemble for uncertainty estimation and probabilistic forecast: Application to air quality”. In: *Journal of Geophysical Research: Atmospheres* vol. 116, no. D19 (2011).
- [182] *Air Quality Forecasting Service in Norway*. <https://luftkvalitet.miljodirektoratet.no/kart/59/10/5/aqi>. [Online; accessed on 27 November 2021]. 2021.
- [183] Denby, B. R., Gauss, M., Wind, P., Mu, Q., Grøtting Wærsted, E., Fagerli, H., Valdebenito, A., and Klein, H. “Description of the uEMEP_v5 downscaling approach for the EMEP MSC-W chemistry transport model”. In: *Geoscientific Model Development* vol. 13, no. 12 (2020), pp. 6303–6323.
- [184] Mu, Q., Denby, B. R., Wærsted, E. G., and Fagerli, H. “Downscaling of air pollutants in Europe using uEMEP_v6”. In: *Geoscientific Model Development Discussions* (2021), pp. 1–24.
- [185] Norman, M., Sundvor, I., Denby, B. R., Johansson, C., Gustafsson, M., Blomqvist, G., and Janhäll, S. “Modelling road dust emission abatement measures using the NORTRIP model: Vehicle speed and studded tyre reduction”. In: *Atmospheric environment* vol. 134 (2016), pp. 96–108.
- [186] Denby, B. R., Klein, H., Wind, P., Gauss, M., Pommier, M., Fagerli, H., and Valdebenito, A. *The Norwegian air quality service: Model forecasting*. URL: https://wiki.met.no/_media/airquip/workshopno/denby_17sep2018.pdf. Sept. 2018.
- [187] Simpson, D., Benedictow, A., Berge, H., Bergström, R., Emberson, L. D., Fagerli, H., Flechard, C. R., Hayman, G. D., Gauss, M., Jonson, J. E., et al. “The EMEP MSC-W chemical transport model—technical description”. In: *Atmospheric Chemistry and Physics* vol. 12, no. 16 (2012), pp. 7825–7865.
- [188] Lepperød, A., Nguyen, H. T., Akselsen, S., Wienhofen, L., Øzturk, P., and Zhang, W. “Air Quality Monitor and Forecast in Norway Using NB-IoT and Machine Learning”. In: *International Summit Smart City 360°*. Springer. 2019, pp. 56–67.

- [189] Veiga, T., Munch-Ellingsen, A., Papastergiopoulos, C., Tzovaras, D., Kalamaras, I., Bach, K., Votis, K., and Akselsen, S. “From a Low-Cost Air Quality Sensor Network to Decision Support Services: Steps towards Data Calibration and Service Development”. In: *Sensors* vol. 21, no. 9 (2021), p. 3190.
- [190] Zhou, Y., Chang, F.-J., Chang, L.-C., Kao, I.-F., and Wang, Y.-S. “Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts”. In: *Journal of cleaner production* vol. 209 (2019), pp. 134–145.
- [191] Mokhtari, I., Bechkit, W., Rivano, H., and Yaici, M. R. “Uncertainty-Aware Deep Learning Architectures for Highly Dynamic Air Quality Prediction”. In: *IEEE Access* vol. 9 (2021), pp. 14765–14778.
- [192] Tao, Q., Liu, F., Li, Y., and Sidorov, D. “Air pollution forecasting using a deep learning model based on 1D convnets and bidirectional GRU”. In: *IEEE access* vol. 7 (2019), pp. 76690–76698.
- [193] Pucer, J. F., Pirš, G., and Štrumbelj, E. “A Bayesian approach to forecasting daily air-pollutant levels”. In: *Knowledge and Information Systems* vol. 57, no. 3 (2018), pp. 635–654.
- [194] Aznarte, J. L. “Probabilistic forecasting for extreme NO₂ pollution episodes”. In: *Environmental Pollution* vol. 229 (2017), pp. 321–328.
- [195] Elshout, S. v. d. and Léger, K. *CAQI Air quality index — Comparing Urban Air Quality across Borders - 2012*. Tech. rep. EUROPEAN UNION European Regional Development Fund Regional Initiative Project, 2012.
- [196] *Open database of air quality measurements by the Norwegian Institute for Air Research (NILU)*. <https://www.nilu.com/open-data/>. [Online; accessed on 27 November 2021]. 2021.
- [197] *The meteorological data by the Norwegian Meteorological Institute*. <https://frost.met.no>. [Online; accessed on 27 November 2021]. 2021.
- [198] *Traffic data by the Norwegian Public Roads Administration*. <https://www.vegvesen.no/trafikdata/start/om-api>. [Online; accessed on 27 November 2021]. 2021.
- [199] Heskes, T., Wiegierinck, W., and Kappen, H. “Practical confidence and prediction intervals for prediction tasks”. In: *PROGRESS IN NEURAL PROCESSING* (1997), pp. 128–135.
- [200] Dar, Y., Muthukumar, V., and Baraniuk, R. G. “A Farewell to the Bias-Variance Tradeoff? An Overview of the Theory of Overparameterized Machine Learning”. In: *arXiv preprint arXiv:2109.02355* (2021).
- [201] Brier, G. W. et al. “Verification of forecasts expressed in terms of probability”. In: *Monthly weather review* vol. 78, no. 1 (1950), pp. 1–3.
- [202] Sokolova, M. and Lapalme, G. “A systematic analysis of performance measures for classification tasks”. In: *Information processing & management* vol. 45, no. 4 (2009), pp. 427–437.

- [203] Koenker, R. and Hallock, K. F. “Quantile regression”. In: *Journal of economic perspectives* vol. 15, no. 4 (2001), pp. 143–156.
- [204] Friedman, J. H. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.
- [205] Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. “Exploring the limits of language modeling”. In: *arXiv preprint arXiv:1602.02410* (2016).
- [206] Chen, R., Wang, X., Zhang, W., Zhu, X., Li, A., and Yang, C. “A hybrid CNN-LSTM model for typhoon formation forecasting”. In: *Geoinformatica* vol. 23, no. 3 (2019), pp. 375–396.
- [207] Li, Y., Yu, R., Shahabi, C., and Liu, Y. “Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting”. In: *International Conference on Learning Representations*. 2018.
- [208] Murad, A. and Pyun, J.-Y. “Deep recurrent neural networks for human activity recognition”. In: *Sensors* vol. 17, no. 11 (2017), p. 2556.
- [209] Chimmula, V. K. R. and Zhang, L. “Time series forecasting of COVID-19 transmission in Canada using LSTM networks”. In: *Chaos, Solitons & Fractals* vol. 135 (2020), p. 109864.
- [210] Gal, Y. and Ghahramani, Z. “A theoretically grounded application of dropout in recurrent neural networks”. In: *Advances in neural information processing systems* vol. 29 (2016), pp. 1019–1027.
- [211] Hasanzadeh, A., Hajiramezanali, E., Boluki, S., Zhou, M., Duffield, N., Narayanan, K., and Qian, X. “Bayesian graph neural networks with adaptive connection sampling”. In: *International conference on machine learning*. PMLR. 2020, pp. 4094–4104.
- [212] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.
- [213] Muller, R., Kornblith, S., and Hinton, G. E. “When does label smoothing help?” In: *Advances in Neural Information Processing Systems* vol. 32 (2019), pp. 4694–4703.
- [214] Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. “Virtual adversarial training: a regularization method for supervised and semi-supervised learning”. In: *IEEE transactions on pattern analysis and machine intelligence* vol. 41, no. 8 (2018), pp. 1979–1993.
- [215] Goodfellow, I. J., Shlens, J., and Szegedy, C. “Explaining and harnessing adversarial examples”. In: *International conference on machine learning*. 2015.
- [216] Qin, Y., Wang, X., Beutel, A., and Chi, E. H. “Improving uncertainty estimates through the relationship with adversarial robustness”. In: *arXiv preprint arXiv:2006.16375* (2020).
- [217] Shafahi, A., Najibi, M., Ghiasi, A., Xu, Z., Dickerson, J., Studer, C., Davis, L. S., Taylor, G., and Goldstein, T. “Adversarial training for free!” In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2019, pp. 3358–3369.

- [218] Zhang, H., Yu, Y., Jiao, J., Xing, E., El Ghaoui, L., and Jordan, M. “Theoretically principled trade-off between robustness and accuracy”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7472–7482.
- [219] Williams, C. K. “Computing with infinite networks”. In: *Advances in neural information processing systems* (1997), pp. 295–301.
- [220] Gustafsson, F. K., Danelljan, M., and Schon, T. B. “Evaluating scalable bayesian deep learning methods for robust computer vision”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 318–319.
- [221] Scalia, G., Grambow, C. A., Pernici, B., Li, Y.-P., and Green, W. H. “Evaluating scalable uncertainty estimation methods for deep learning-based molecular property prediction”. In: *Journal of chemical information and modeling* vol. 60, no. 6 (2020), pp. 2697–2717.
- [222] Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. “Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration”. In: *Advances in Neural Information Processing Systems* vol. 2018 (2018), pp. 7576–7586.

IV.7 Supplementary Materials

IV.7.1 Datasets

This section shows additional figures explaining the used dataset of air quality, meteorological data, traffic, and street-cleaning reports from the municipality.

IV.7.1.1 Air Quality Data

Figure IV.49 shows the air quality data of $PM_{2.5}$ and PM_{10} , measured over two years in four different sensing stations in the city of Trondheim (IV.20). The figure shows a clear temporal and spatial correlations between the four sensing stations.

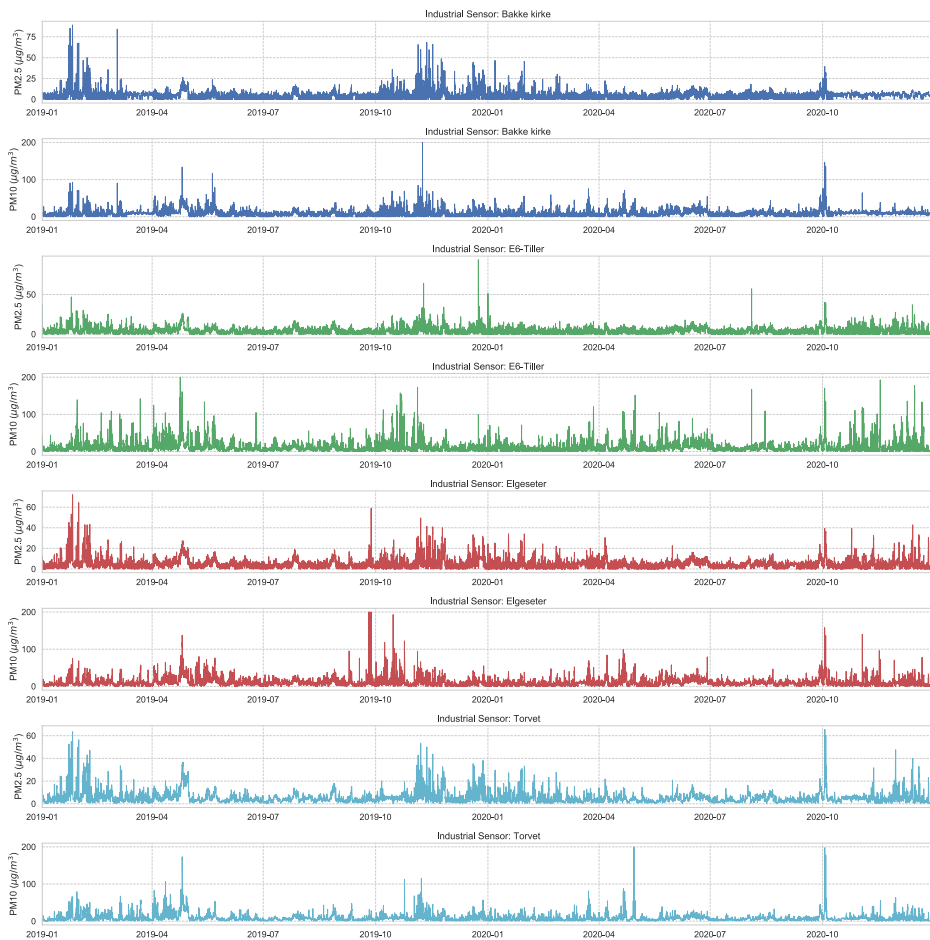


Figure IV.49: Air quality data of $PM_{2.5}$ and PM_{10} , measured over two years in four different sensing stations in the city of Trondheim. These data are offered by the Norwegian Institute for Air Research (NILU) (<https://www.nilu.com/open-data/>).

IV.7.1.2 Weather data

We use weather data observations at four monitoring stations in the city of Trondheim (Voll, Sverreborg, Gloschaugen, Lade). These observations include: air temperature, relative humidity, precipitation, air pressure, wind speed, wind direction, snow thickness, and duration of sunshine. Figure IV.50 summarize all these observations.

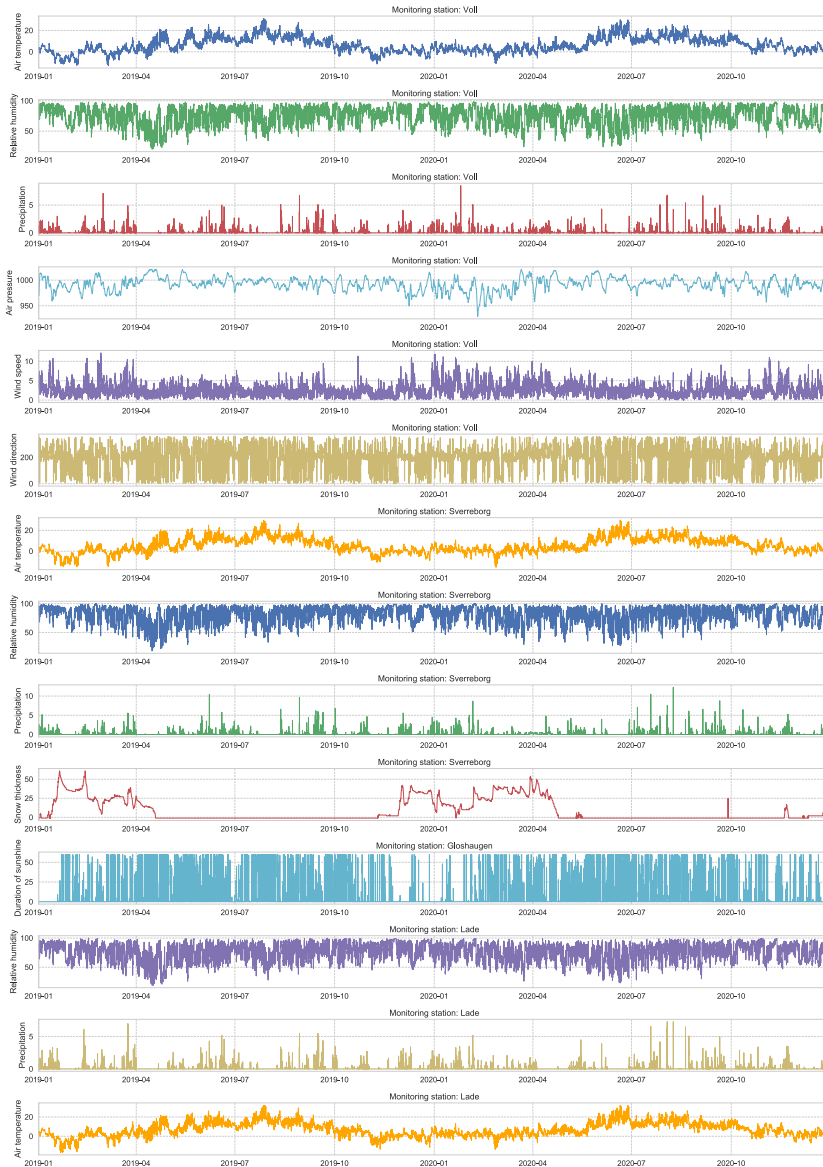


Figure IV.50: Weather data observations over two years at four monitoring station in the city of Trondheim (Voll, Sverreborg, Gloschaugen, Lade). These data are offered by the Norwegian Meteorological Institute (<https://frost.met.no>).

IV.7.1.3 Traffic data

We also used traffic volume as an input feature for the forecasting models. We used the data of the traffic volume recorded at eight streets of Trondheim. Figure IV.51 shows the traffic volume recorded over two years. The figure shows a clear correlation of traffic between streets. It also shows the drop in traffic during the pandemic lockdown in Trondheim (after March 12, 2020).

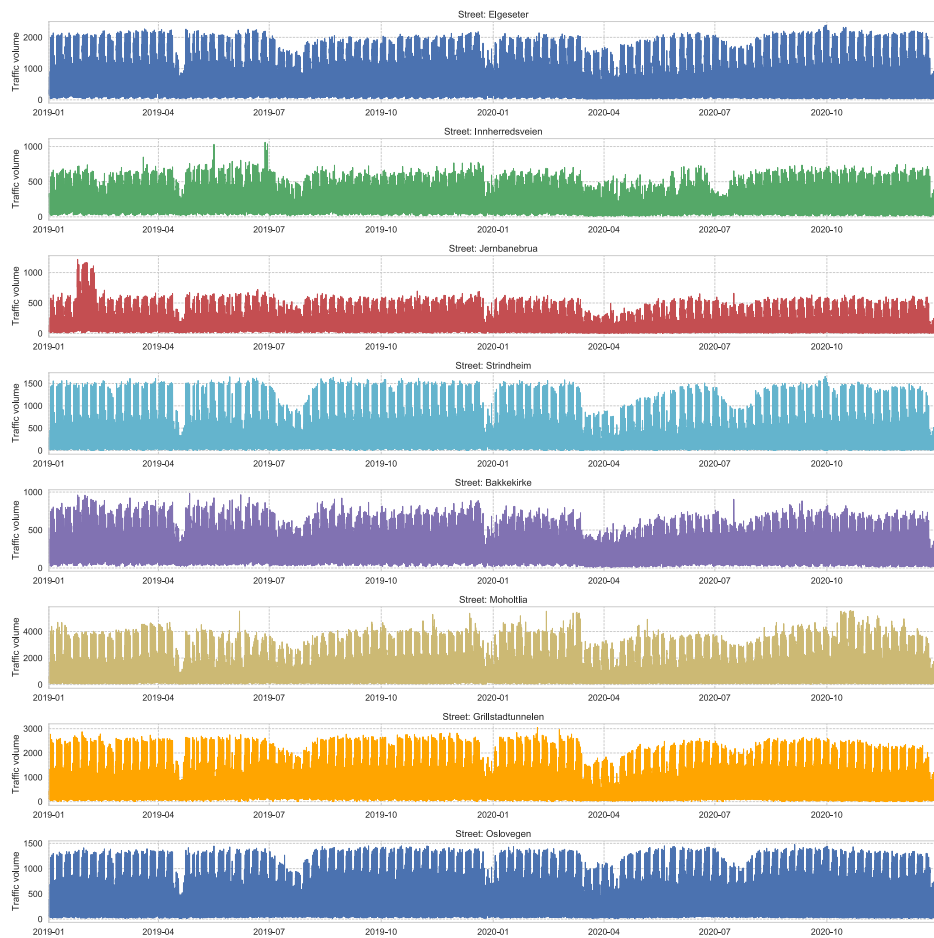


Figure IV.51: Traffic volume recorded at eight streets of Trondheim over two years. These data are offered by the Norwegian Public Roads Administration (<https://www.vegvesen.no/trafikkdata/start/om-api>).

IV.7.1.4 Streets-cleaning data

We also used data of street-cleaning at main streets of Trondheim, as shown in Figure IV.52. These data are reported by the municipality and include the duration of time in which a street-cleaning is taking place.

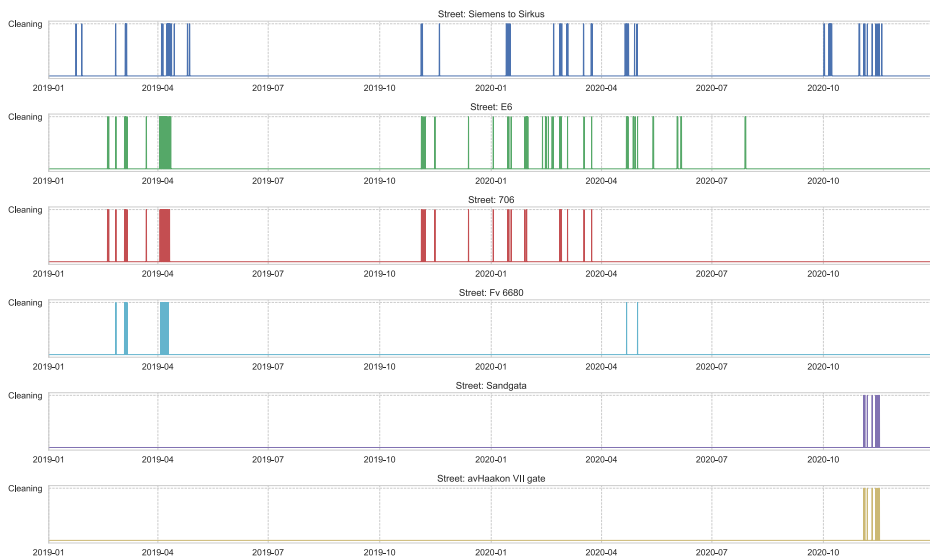


Figure IV.52: Data of the duration of time in which a street-cleaning is taking place on the main streets of Trondheim, reported by the municipality.

IV.7.2 Reliability of Confidence Estimate: Additional Plots

This section presents additional plots for comparison of confidence reliability in all monitoring stations. Figure IV.53 shows the comparison of confidence reliability in the PM-value regression task while Figure IV.54 in the threshold exceedance classification task.

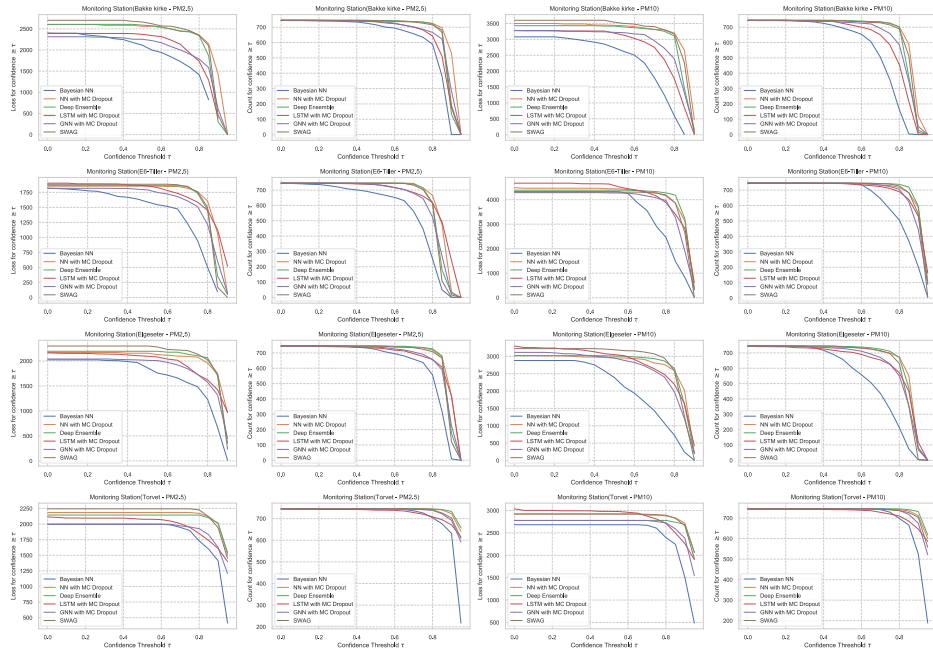


Figure IV.53: Comparison of confidence reliability for the selected probabilistic models in the PM-value regression task in all monitoring stations.

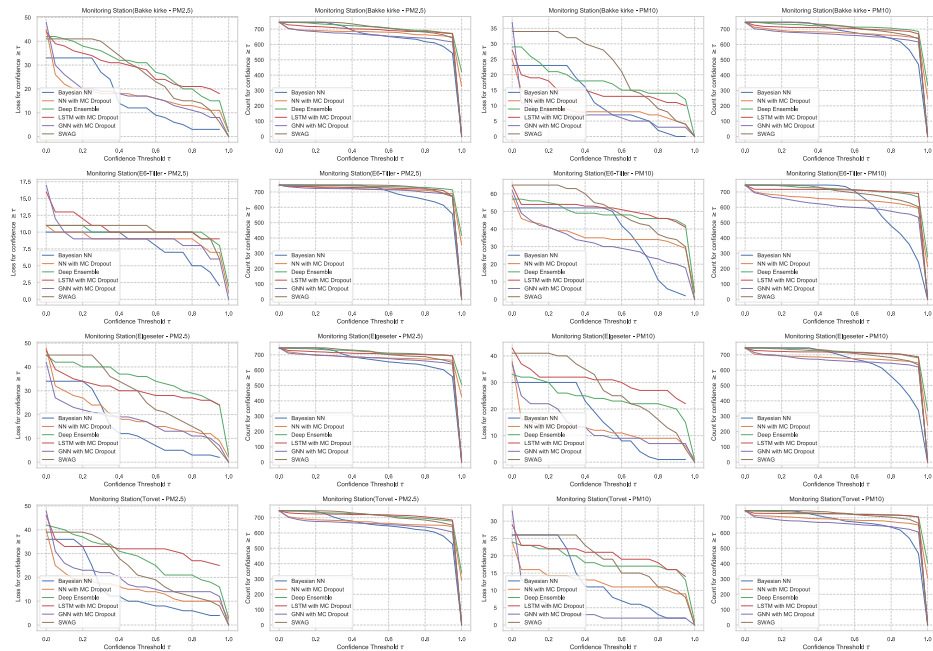


Figure IV.54: Comparison of confidence reliability for the selected probabilistic models in the threshold exceedance classification task in all monitoring stations.

IV.7.3 Justification for Threshold-Exceedance Classification

Recently, the Norwegian government has proposed even stricter regulations for particle dust thresholds. Therefore, we want to estimate if the concentration of air particles exceeds certain thresholds following the Common Air Quality Index (CAQI) used in Europe [195], as shown in Table IV.3.

The CAQI specifies five levels of air pollutants, but the air quality in the city of Trondheim is usually at *Very Low* and rarely exceeds the *Medium* level. Figure IV.56 shows the air quality level over one year in all monitoring stations. Generally, air quality data have right-skewed distributions, with the degree of skewness (asymmetry) differing among different cities. Figure IV.55 shows histograms that approximately represent the distribution of air quality at four monitoring stations in the city of Trondheim. The figure clearly shows heavily right-skewed distributions, in which higher CAQI classes are under-represented. Therefore, instead of a multinomial classification task (with five classes), we transform the problem into a threshold exceedance forecast task in which we try to predict the points in time where the air quality exceeds the *Very Low* level.

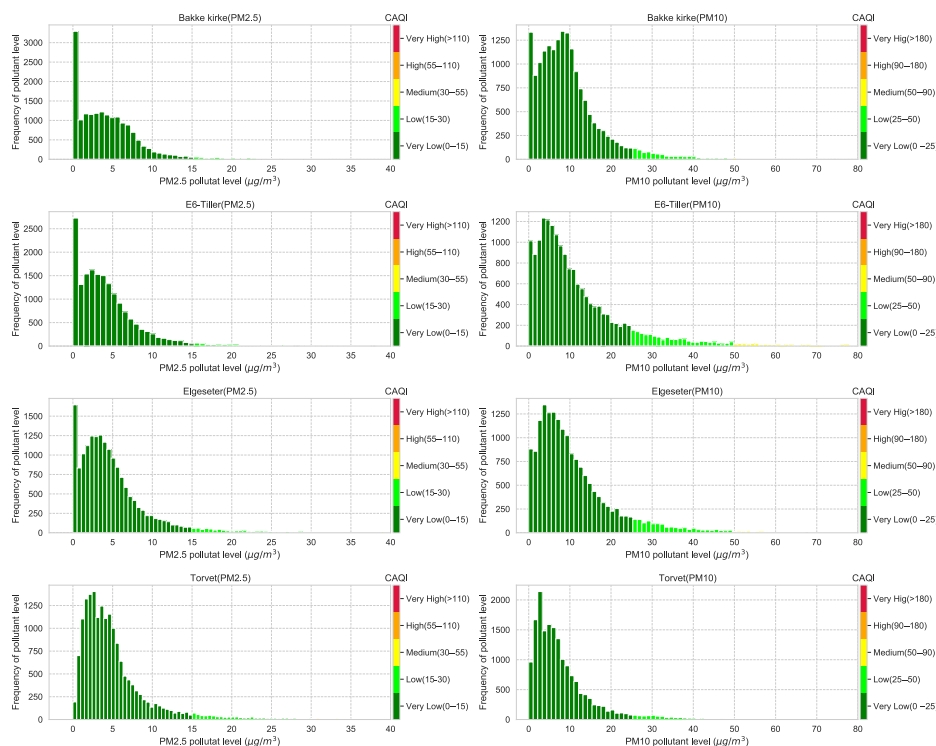


Figure IV.55: Histograms that approximately represent the distribution of air quality at four monitoring stations in the city of Trondheim. The air quality data come from heavily right-skewed distributions, in which higher CAQI classes are under-represented.

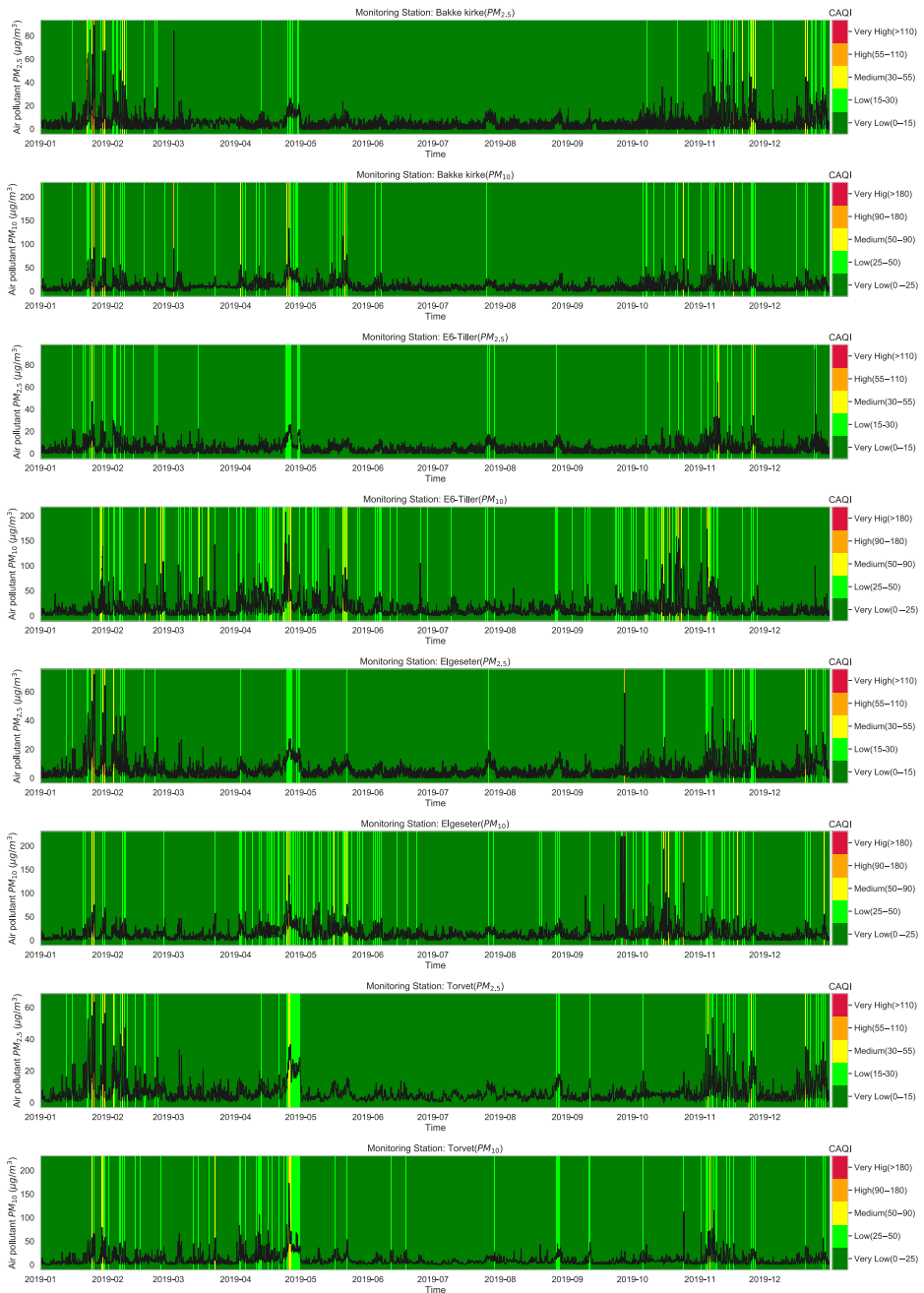


Figure IV.56: Air quality in the city of Trondheim over one year in all monitoring stations. The data are decomposed into the five CAQI levels of air pollutants. It is usually at *Very Low* and rarely exceeds the *Medium* level.

Paper V: Uncertainty-Aware Autonomous Sensing with Deep Reinforcement Learning

Abdulmajid Murad¹, Frank Alexander Kraemer¹, Kerstin Bach¹,
Gavin Taylor²

¹Norwegian University of Science and Technology, ²United States Naval Academy



V

This paper is awaiting publication and is not included

Paper VI: Learning Task Agnostic Skills with Data-driven Guidance

Even Klemsdal*¹, Sverre Herland*¹, Abdulmajid Murad*¹

* Equal Contribution

¹Norwegian University of Science and Technology

Abstract

To increase autonomy in reinforcement learning, agents need to learn useful behaviours without reliance on manually designed reward functions. To that end, skill discovery methods have been used to learn the intrinsic options available to an agent using task-agnostic objectives. However, without the guidance of task-specific rewards, emergent behaviours are generally useless due to the under-constrained problem of skill discovery in complex and high-dimensional spaces. This paper proposes a framework for guiding the skill discovery towards the subset of expert-visited states using a learned state projection. We apply our method in various reinforcement learning (RL) tasks and show that such a projection results in more useful behaviours.

VI.1 Introduction

While autonomous learning of diverse and complex behaviors is challenging, significant progress has been made using deep reinforcement learning (DRL). The progress has been accelerated by the powerful representational learning of deep neural networks [15], and the scalability and efficiency of RL algorithms [17, 18, 19, 20]. However, DRL still involves an externally designed reward function that guides learning and exploration. Manually engineering such a reward function is a complex task that requires significant domain knowledge in such a way that hinders autonomy and adoption of RL. Prior works have proposed using unsupervised skill discovery to alleviate these challenges by using empowerment as an intrinsic motivation to explore and acquire abilities [145, 146, 41, 147, 148].

Although skill discovery without a reward function can be helpful as a primitive for downstream tasks, most of the emergent behaviours in the learned skills are useless or of little interest. This is a direct consequence of *under-constrained* skill discovery in complex and high dimensional state space. One possible solution is to leverage prior knowledge to bias skill discovery towards a subset of the state space through a hand-crafted transformation of the state space [41, 148]. However, utilizing prior knowledge to hand-craft such a transformation contradicts the primary goal of unsupervised RL, which is reducing manual design efforts and reliance on prior knowledge.

Instead, we explore how to learn a parameterized state projection that directs skill discovery towards the subset of expert-visited states. To that end, we employ examples of expert data to train a state encoder through an auxiliary classifier, which tries to distinguish expert-visited states from random states. We then use the encoder to project the state space into a latent embedding that preserves information that makes expert-visited states recognizable. This method extends readily to other mechanisms of learned state-projections and different skill discovery algorithms. Crucially, our method requires only samples of expert-visited states, which can easily be obtained from any reference policy, for example expert demonstrations.

The key contribution of this paper is a simple method for learning a parameterized state projection that guides skill discovery towards a substructure of the observation space. We demonstrate the flexibility of our state-projection method and how it can be used with the skill-discovery objective. We also present empirical results that show the performance of our method in various locomotion tasks.

VI.2 Related Work

Unsupervised reinforcement learning aims at learning diverse behaviours in a task-agnostic fashion without guidance from an extrinsic reward function [239]. This can be accomplished through learning with an intrinsic reward such as *curiosity* [240] or *empowerment* [145]. The notion of curiosity has been utilized for exploration by using predictive models of the observation space and providing a higher intrinsic reward for visiting unexplored trajectories [241]. Empowerment addresses maximizing an agent’s control over the environment by exploring states with maximal intrinsic options (skills).

Several approaches have been proposed in the literature to utilize empowerment for skill discovery in unsupervised RL. Gregor et al. [146] developed an algorithm that learns intrinsic skill embedding and used generalization to discover new goals. They used the mutual information between skills and final states as the training objective and hence used a discriminator to distinguish between different skills. Eysenbach et al. [41] used mutual information between skills and states as an objective while using a fixed embedding distribution of skills. Additionally, they used a maximum-entropy policy [19] to produce stochastic skills. However, most of the previous approaches assume a state distribution induced by the policy itself, resulting in a premature commitment to already discovered skills. Campos et al. [148] used a fixed uniform distribution over states to break the dependency between the state distribution and the policy.

Certain prior work has addressed the challenge of complex and high dimensional state space by constraining the skill-discovery in a subset of the state space. Sharma et al. [147] learned predictable skills by training a skill-conditioned dynamic model instead of a discriminator to model specific behaviour in a subset of the state space. Eysenbach et al. [41] proposed incorporating prior knowledge by conditioning the discriminator on a subset of the state space using a hand-crafted and a task-specific transformation. Our work addresses this challenge by guiding the skill discovery towards the subset of expert-visited states. In contrast to inverse reinforcement learning, [242], we do not explicitly infer the extrinsic reward. Crucially, we do not try to learn the expert policy directly in contrast to behaviour cloning or imitation learning [143]. Our proposed method resembles the algorithm proposed by Li et al. [243] in which they used a Bayesian classifier that estimates the probability of successful outcome states, resulting in a more task-directed exploration. However, their algorithm does not optimize the mutual information; hence it does not learn diverse skills via the discriminability objective.

VI.3 Preliminaries

In this paper, we formalize the problem of skill discovery as a Markov decision process (MDP) without a reward function: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P})$, where \mathcal{S} is the state space, \mathcal{A} is the action space, and $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ is the transition probability density function. The RL agent learns a skill-conditioned policy $\pi_{\vartheta}(a|s, z)$, where the skill z is sampled from some distribution $p(z)$. A skill, or option (as first introduced in [42]), is a temporal abstraction of a course of actions that extends over many time steps. We will also consider the information-theoretic notion of

mutual information between states and skills $\mathcal{I}(S; Z) = \mathcal{H}(Z) - \mathcal{H}(Z|S)$, where $\mathcal{H}(\cdot)$ is the Shannon entropy.

VI.3.1 Skill Discovery Objective

The overall goal of skill discovery is to find a policy π_θ capable of carrying out different tasks that are learned without extrinsic supervision for each type of behavior. We consider policies of the form $\pi_\theta(a|s, z)$ that specify different distributions over actions depending on which skill z they are conditioned on. Although this general framework does not constrain how z should be represented, we define it as a discrete variable since it has been empirically shown to perform better than continuous alternatives [41].

We follow the framework proposed by the "diversity is all you need" (DIAYN) algorithm [41], in which skills are learned by defining an intrinsic reward that promotes diversity. Intuitively, each skill should make the agent visit a unique section of the state space. This can be expressed as maximising the mutual information $\mathcal{I}(S; Z)$ of the state visitation distributions for different skills [145]. To ensure that the visited areas of the state space are spaced sufficiently far apart, we use a soft policy that maximises the entropy of the action distribution. Formally, we maximize the following objective function:

$$\begin{aligned} \mathcal{F}(\theta) &\triangleq \mathcal{I}(S; Z) + \mathcal{H}(A|S) - \mathcal{I}(A; Z|S) \\ &= \mathcal{H}[A|S, Z] - \mathcal{H}[Z|S] + \mathcal{H}[Z] \end{aligned} \quad (\text{VI.53})$$

The first term $\mathcal{H}[A|S, Z]$ means that the policy should act as randomly as possible and can be optimized by maximizing the policy's entropy. The second term $-\mathcal{H}[Z|S]$ dictates that each visited state should (ideally) identify the current skill. The third term is the entropy of the skill distribution, which can be maximized by deliberately sampling skills from a uniform distribution during training. Unfortunately, $-\mathcal{H}[Z|S]$ requires knowledge about $p(z|s)$, which is not readily available. Consequently, we approximate the true distribution by training a classifier $q_\phi(z|s)$, leading to a lower bound:

$$\begin{aligned} \mathcal{F}(\theta) &= \mathcal{H}[A|S, Z] + \mathbb{E}_p[\log p(z|s)] - \mathbb{E}_p[\log p(z)] \\ &\geq \mathcal{H}[A|S, Z] + \mathbb{E}_p[\log q_\phi(z|s)] - \mathbb{E}_p[\log p(z)] \end{aligned} \quad (\text{VI.54})$$

The lower bound follows from the non-negative property of the Kullback-Leibler divergence $D_{KL}(p||q) = \mathbb{E}_p[\log p(z|s) - \log q(z|s)] \geq 0$, which can be rearranged to $\mathbb{E}_p[\log p(z|s)] \geq \mathbb{E}_p[\log q(z|s)]$ [244].

The classifier q_ϕ is fitted throughout training with maximum likelihood estimation over the sampled states and active skills. This leads to a scenario where the policy is rolled out for a (uniformly) sampled skill, and the classifier is trained to detect the skill based on the states that were visited. The policy is given a reward proportional to how well the classifier could detect the skill in each state. In the end, this should make the policy favor visiting disjoint sets of states for each skill, leading to a cooperative game between q_ϕ and π_θ .

VI.3.2 Limitations of existing methods

A major challenge that arises when maximizing the objective in Equation VI.54, particularly in applications with high-dimensional spaces, is that it becomes trivial for each skill to find a sub-region of the state space where it is easy to be recognised by q_ϕ . In preliminary experiments, we observed that the existing methods discovered behaviours that covered small parts of the state space. For the HalfCheetah environment [245] this resulted in many skills generating different types of static poses (see Figure VI.65) and not many skills exhibiting "interesting" behaviour such as locomotion.

Optimising for $\mathcal{H}[A|S, Z]$ should mitigate this issue to some extent. Increasing the policy's entropy incentivises the skills to progressively visit regions of the state space that are so far apart that not even highly stochastic actions will cause them to overlap accidentally. However, it has been shown that mutual information based algorithms have difficulties spreading out to novel states due to low values of $\log q_\phi(z|s)$ for out-of-sample states [148].

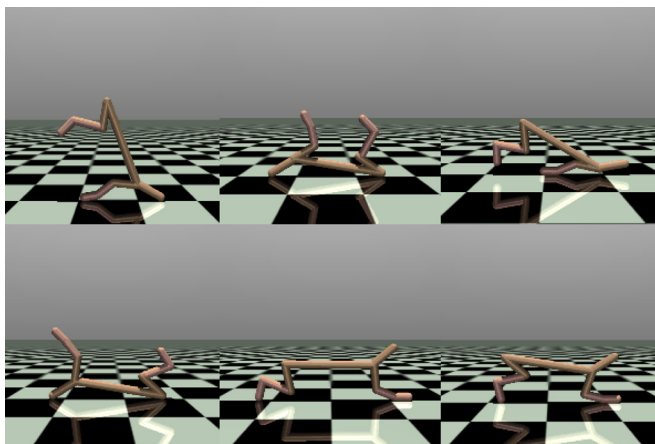


Figure VI.65: Cheetahs frozen in various configurations for 6 unique skills. When sampled, the policy generates actions that "shake" around these positions. However, they are still distinguishable by the classifier q_ϕ with over 90% accuracy.

VI.4 Proposed Method

The main idea of our approach is to focus the skill discovery towards certain parts of the state space by using expert data as a prior. The DIAYN algorithm can be biased towards a user-specified part of the state space by changing the discriminator to maximize $\mathbb{E}[\log q_\phi(z|f(s))]$, where f represents some transformation of the state space [41]. Instead of using a hand-crafted f to improve the skills discovered for a navigation task, we aim to learn a parameterized f_χ by using expert data.

VI.4.1 State Space Projections

We consider linear projections of continuous factored state representations on the form $f_\chi : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|E|}$ with $e = f_\chi(s) = \chi s$, $\chi \in \mathbb{R}^{|E| \times |S|}$, and $|E| < |S|$. In principle, the idea should apply to more complex mappings, such as a multi-layer perceptron. However, we want to limit the scope of skill discovery to a hyperplane within the original state space.

For the same reason, we also omit any non-linearities in the encoder. Squeezing the output through a Sigmoidal function would limit discriminability at the (potentially interesting) extremes of the encoding. Similarly, a ReLU function would effectively eliminate all exploration along the negative direction of f_{χ_i} . In summary, the objective of the DIAYN skill classifier becomes:

$$\max_{\phi} \mathbb{E}[\log q_{\phi}(z|e)] \quad (\text{VI.55})$$

We learn the parameters χ for the projection through an auxiliary discriminative objective. Specifically, a binary classifier $h_{\psi} : \mathbb{R}^{|E|} \rightarrow \{0, 1\}$ is trained to predict whether an (encoded) state was sampled from the marginal state visitation distribution of a random policy π_{rand} or from the distribution of a reference (expert) policy π^* . Let $x \sim \mathcal{D}$ denote whether a state s was visited by the reference policy or not (in dataset \mathcal{D}), then the parameters of f_χ are obtained through joint pretraining with h_{ψ} by maximizing the log likelihood over \mathcal{D} :

$$\max_{\chi, \psi} \mathbb{E}_{x, s \sim \mathcal{D}} [x \log h_{\psi}(x|e) + (1 - x) \log(1 - h_{\psi}(x|e))] \quad (\text{VI.56})$$

where the dataset \mathcal{D} is collected prior to training the main RL algorithm. The first half (random samples) are collected by rolling out π_{rand} whereas the second half (reference samples) are collected by rolling out π^* . After the objective in Equation VI.56 is optimized, the discriminator h_{ψ} is discarded and the projection encoding $f_\chi(s)$ is extracted to be used for the objective in Equation VI.55. Analogous to autoencoders [246], the idea is that the embeddings produced by $f_\chi(s)$ should now contain a more compact representation of the state space without collapsing the dimensions that make "interesting" behaviour stand out.

While the use of a reference data changes our approach from a strictly unsupervised skill discovery algorithm, the discriminative objective in equation VI.56 resembles the objectives used in adversarial inverse reinforcement learning (e.g. [242]). However, it differs in that it makes no attempts at matching the behaviour of a reference policy as it is used only as a prior for simplifying the state space. This approach could also be used with samples from several different reference policies with substantially different marginal state distributions. As long as their variation can be explained sufficiently without full use of the entire state space, a projection should simplify skill discovery.

VI.4.2 Implementation

For learning diverse skills, we use DIAYN as a basis framework. DIAYN uses the Soft Actor-Critic (SAC) algorithm [19] that is optimized using policy gradient style updates in contrast to the reparameterized version (DDPG style updates

[20]). They also use a Squashed Gaussian Mixture Model to represent the policy $a \sim \pi_\vartheta = \tanh GMM(\mu_\vartheta(s), \sigma_\vartheta(s))$. The learning objective is to maximize the mutual information between the state and skill $I(S; Z)$. This objective is optimized by replacing the task rewards with a pseudo-reward

$$r_z(s, a) \triangleq \log p_\phi(z|s) - \log p(z) \quad (\text{VI.57})$$

where q_ϕ is trained to discriminate between skills and $p(z)$ is the fixed uniform prior over skills [41]. A skill is sampled from $z \sim p(z)$ and used throughout a full episode.

In contrast to DIAYN, we use two Q-functions $Q_\theta^1(s, a)$ & $Q_\theta^2(s, a)$ where both Q-functions attempt to predict the same quantity. This allows us to sample differentiable actions and climb the gradient of the minimum of the two Q-functions (DDPG-style update [20]), giving us this objective:

$$J(\pi_\vartheta) = \min_{i \in \{1, 2\}} Q_\theta^i(\pi_\vartheta(a|s)) + \alpha \mathcal{H}(\pi_\vartheta(a|s))$$

Like in DIAYN, we also use a Squashed Gaussian Mixture Model to promote diverse behaviour.

Figure VI.66 illustrates the training process of the proposed expert-guided skill discovery. First, we train the encoder f_χ jointly with the auxiliary classifier h_ψ using the external dataset \mathcal{D} . Secondly, we train the agent using an offline policy algorithm (SAC), in which the agent samples a skill $z \sim p(z)$, and then interacts with the environment by taking action a_t according the skill-conditioned policy $\pi_\vartheta(a_t|s_t, z)$. The environment, then, transits to a new state according to the transition probability $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. We add this transition (s_z, z, a_t, s_{t+1}) to the replay buffer \mathcal{B} . Simultaneously, the policy is updated by sampling a mini-batch from the replay buffer $\mathcal{M} \sim \mathcal{B}$, then encoding the next states $e_{t+1} = f_\chi(s_{t+1})$ and passing them through the discriminator $q_\phi(z|e_{t+1})$ to get the intrinsic reward. This reward is used by the Q-functions $Q_\theta(s_t, a_t, z)$ to minimize the soft Bellman residual and update the policy. A pseudocode 3 for the proposed approach can be found in the supplementary material.

VI.5 Experiments

In our experimental evaluation, we aim to demonstrate the impact of our approach of restricting skill discovery to a projection subspace. We verify our method on both point-mazes and continuous control locomotion tasks. All the code for running the experiments are publicly available on GitHub⁸.

VI.5.1 Point Maze

As an illustrative example, we begin by testing the algorithm on a simple 2D point-maze problem. The term *maze* is used very generously here, as the environment consists of an open plane in \mathbb{R}^2 enclosed by walls that restrict the agent to $\frac{1}{7} < x < \frac{6}{7}$ and $\frac{1}{7} < y < \frac{6}{7}$. At initialization, the agent is dropped down at

⁸Project code base: <https://github.com/sherilan/cs285-project/tree/master>

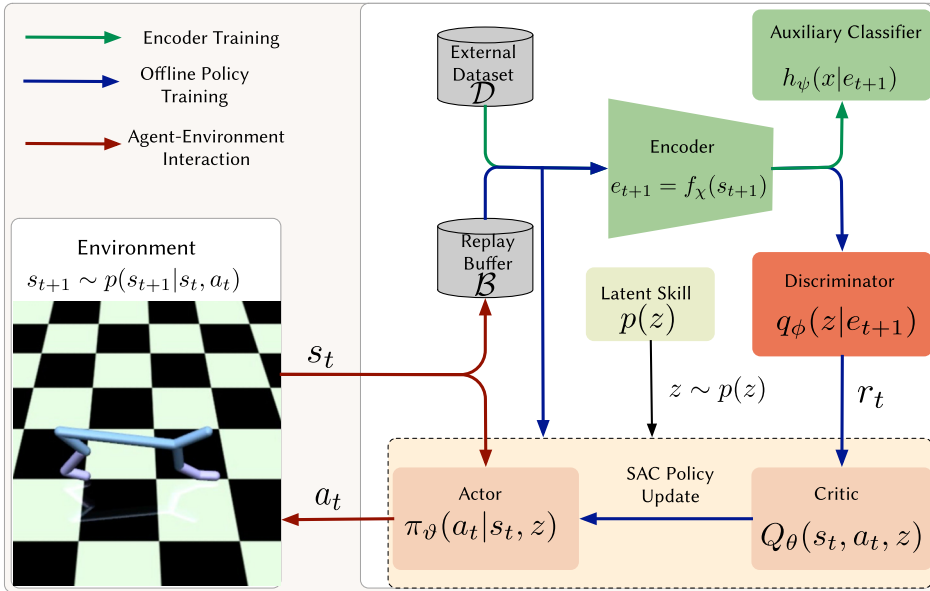


Figure VI.66: Framework for training expert-guided skill discovery. Green arrows show the training of the encoder, red arrows show the agent-environment interaction, while the blue arrows show the interactions for the offline policy training. Note there is no gradients through the encoder in the offline training.

$x, y = \frac{7 \pm \epsilon}{14}, \epsilon \sim U(-1, 1)$ and incentivized to move towards the lower right by a reward proportional to a gaussian kernel centered at $(\frac{9}{14}, \frac{3}{14})$. The agent is free to move by $\pm \frac{1}{70}$ in both the x and y direction.

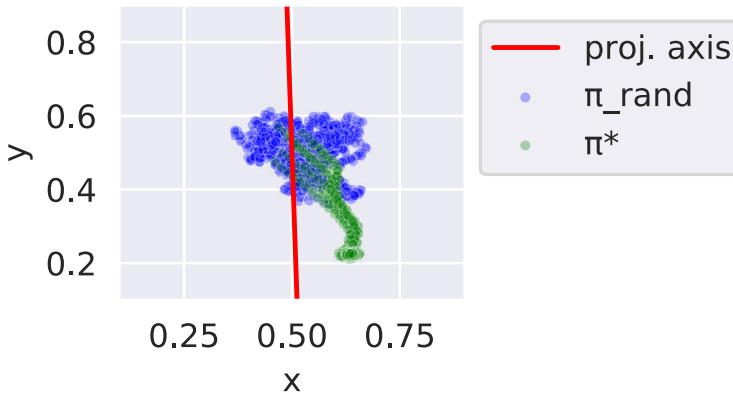


Figure VI.67: The learned projection axis (in red), which is the output of the encoder when trained from the discrimination of the green and blue points.

We train a SAC agent against the extrinsic environment reward to convergence and set its final policy as the reference policy π^* . We then sample 10 trajectories

of length 100 (green dots in Figure VI.67) with π^* , as well as 10 trajectories of length 100 (blue dots in Figure VI.68) from a uniform random policy π_{rand} . The resulting dataset \mathcal{D} consists of 2000 samples and is used to train $h_\psi(x|e)$ until it can distinguish states from π^* and π_{rand} with around 98% accuracy. For this experiment, we project down from 2D to 1D, making $\chi \in \mathbb{R}^{1 \times 2}$. The resulting projection axis is visualized as a red line in Figure VI.67 and is the only thing exported to the next stage of the algorithm.

We then train two versions of the DIAYN algorithm; a baseline using the states as-is in the classifier ($q_\phi(z|s)$) and our proposed method using the state projections ($q_\phi(z|f_\chi(s))$). All other hyperparameters are held equal in the two experiments, and the algorithms are trained for 400,000 environment interactions, each attempting to learn 10 distinct skills.

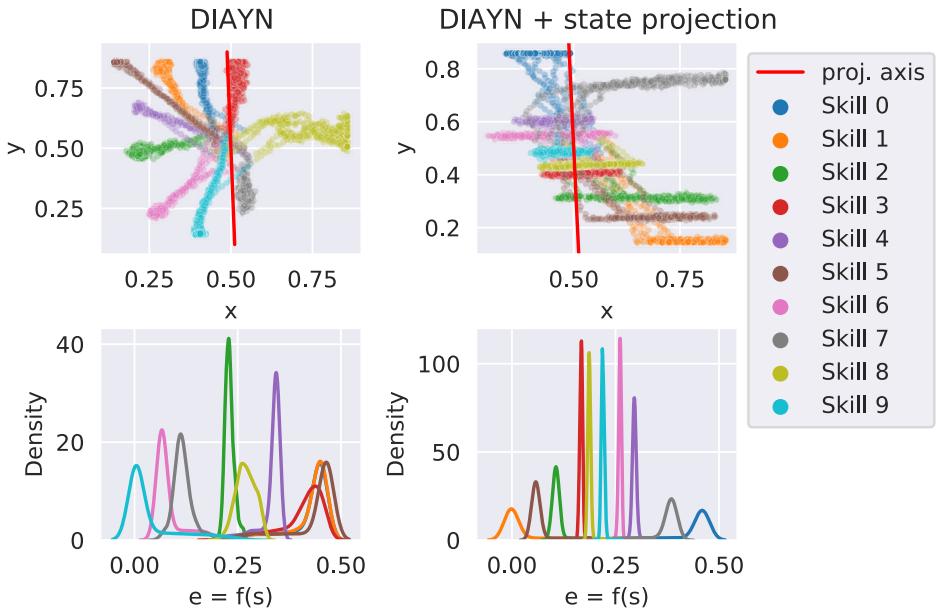


Figure VI.68: State visitation visualization for 10 skills learned with baseline DIAYN (left) and DIAYN with state projection (right). The top row shows the actual states visited in \mathbb{R}^2 , whereas the bottom row shows the same data linearly mapped onto the projection axis obtained in Figure VI.67.

Figure VI.68 visualizes the results of the baseline and the state projection to the left and right, respectively. The top row shows the states that were visited for five rollouts of each skill. As expected, the baseline skills spread out in all directions (albeit slightly more so towards the left) and converge on locations that are easy to distinguish with a 2D state representation. In contrast, the skills generated in the left plot form lines along the state projection axis. Their wide lateral spread follows from the (unbounded) entropy maximization objective. Besides, any movement perpendicular to the projection axis will not affect the 1D vector passed to the classifier.

VI.5.2 Mujoco Environments

Next, we evaluate the algorithm on three continuous control problems from the OpenAI gym suite. [245]. The environments include *HalfCheetah* ($S = \mathbb{R}^{17}$, $A = \mathbb{R}^6$), *Hopper* ($S = \mathbb{R}^{11}$, $A = \mathbb{R}^3$) and *Ant* ($S = \mathbb{R}^{27}$, $A = \mathbb{R}^8$). We choose these environments because they involve substantially different locomotion methods. Additionally, they have observation spaces with different dimensionality, which enable us to better investigate the projection impact.

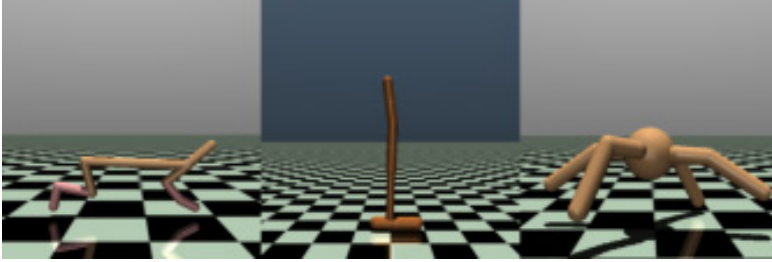


Figure VI.69: Continuous control environments from the Mujoco part of the OpenAI gym suite. Left: HalfCheetah-v2. center: Hopper-v2, right: Ant-v2.

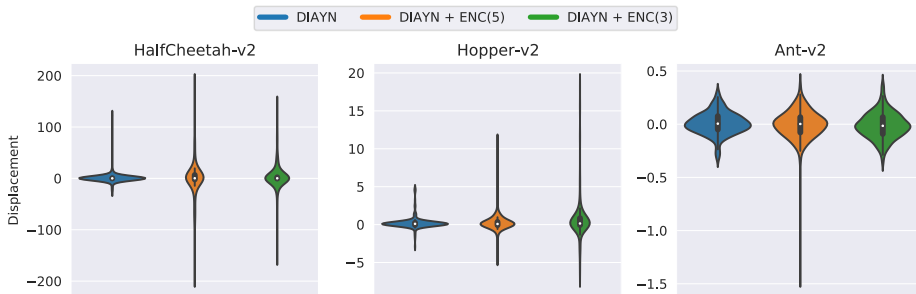


Figure VI.70: Impact of state-space projection illustrated as distribution of displacements along the locomotion axis used to calculate the extrinsic reward of the environments. By introducing a projection step, the skill search gets focused towards locomotion skills, resulting in a larger spread of displacements.

We do one baseline run without any state projection for all three problems, one with a projection down to \mathbb{R}^5 , and one with a projection down to \mathbb{R}^3 . We use our base SAC implementation to obtain reference policies π^* and sample 10 trajectories of length 1000 with fairly high returns (Ant: 5063.9 ± 469.8 , Cheetah: 10656.4 ± 673.5 , Hopper: 3348.0 ± 316.0). The DIAYN algorithm is otherwise identical to [41] in terms of hyperparameters; Q , π , q_ϕ , and h_ψ use MLP architectures with 2 hidden layers of width 300, the entropy bonus weight α is set to 0.1, and the number of skills is set to 50. We limit each skill-discovery run to 2.5 million environment interactions but repeat each experiment 5 times with different random seeds (including training of SAC agents for π^*).

For quantitative evaluation, we look at the displacement along the target locomotion axis for the extrinsic objective. In our approach, we would expect

Table VI.18: Summary statistics for displacement (along the locomotion axis used to calculate the extrinsic reward of the environments) across the 50 skills learned. The values to the right of \pm indicate standard deviation across 5 seeded runs.

		min	25%	50%	75%	max
HalfCheetah-v2	DIAYN	-9.7 ± 11.1	-0.1 ± 0.0	0.0 ± 0.1	0.3 ± 0.1	76.6 ± 46.6
	DIAYN + ENC(3)	-88.6 ± 42.1	-0.4 ± 0.4	0.2 ± 0.1	3.9 ± 4.7	99.0 ± 37.5
	DIAYN + ENC(5)	-129.0 ± 48.5	-6.1 ± 9.5	0.7 ± 1.2	6.6 ± 5.1	121.2 ± 44.7
Hopper-v2	DIAYN	-1.0 ± 1.1	-0.0 ± 0.1	0.1 ± 0.0	0.2 ± 0.1	3.7 ± 1.4
	DIAYN + ENC(3)	-4.3 ± 1.8	-0.0 ± 0.1	0.2 ± 0.2	0.9 ± 0.6	10.1 ± 6.1
	DIAYN + ENC(5)	-3.1 ± 1.4	-0.1 ± 0.0	0.1 ± 0.0	0.4 ± 0.1	7.0 ± 3.2
Ant-v2	DIAYN	-0.3 ± 0.0	-0.1 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.3 ± 0.1
	DIAYN + ENC(3)	-0.3 ± 0.1	-0.1 ± 0.0	-0.0 ± 0.0	0.1 ± 0.0	0.3 ± 0.1
	DIAYN + ENC(5)	-0.5 ± 0.5	-0.1 ± 0.0	-0.0 ± 0.0	0.1 ± 0.0	0.3 ± 0.1

to observe skills that cover this axis well, i.e., skills that run forward and backward at different speeds. To test this, we roll out each skill deterministically⁹, record its movement over 1000 time steps (or until it reaches a terminal state) and observe the inter-skill spread. A similar assessment is possible by only looking at the environment’s rewards. However, the environment reward also includes terms for energy expenditure, staying alive (for Ant/Hopper), and collisions (Ant), which would obscure the results. Figure VI.70 shows the displacement distribution of the 50 skills across all runs. The same information is summarized numerically in Table VI.18.

For a qualitative evaluation, we have also composed a video with every skill across all runs¹⁰.

VI.6 Discussion

For HalfCheetah and Hopper, the runs with state encoding (+ ENC(3|5)) exhibit a substantially larger spread than the baseline. The best forward-moving cheetah skill moves 178 units forward (= 3311 environment return), and the best backwards-moving cheetah skill moves 186 units backwards (= -4025 environment return). For the hopper environment, the best forward-moving skill manages to jump 20 units forward, which corresponds to an environment reward of 3268, which is on the same level as the reference data used to fit its encoder.

The results in the Ant environment are less impressive. There is hardly any difference in how the displacements are distributed for the three approaches, and the total movement is almost negligible. For reference, a good Ant agent trained against the extrinsic reward should obtain displacements in the 100s when evaluated over the same trajectory horizon.

Looking at the generated Ant behaviour, we found that the skills produced with encoders typically moved even less than those generated by the baseline. This is not because it is impossible to generate a linear projection that promotes locomotion at various speeds, as the state representation of all three problems contains a feature

⁹Deterministic sampling from our GMM-based policy implies taking the mean of the component with the highest mixture probability.

¹⁰Video of skills: <https://www.youtube.com/watch?v=Xx7RVNmv1tY>

for linear velocity along the target direction. Moreover, the skill classifier does reach a high accuracy (some breaking 90%), so the algorithm manages to find distinguishable skills. We, therefore, suspect that the procedure used to fit the encoder is insufficient for this environment. While it does pick up on linear velocity, it also picks up on several other features from the state space, which might have made it easier for the algorithm to make the skills distinguishable.

To better understand the results of the Ant experiment, we investigate the projection matrix learned at the start of the algorithm. Figure VI.71 gives a representative example of a projection learned for an ENC(3) run. In the diagram, each bar indicates the impact each feature of the state space has on the final embedding. The orange bar highlights the feature corresponding to linear torso velocity in the x-direction, i.e. the direction in which the extrinsic objective rewards an agent for running in. All the bars to the left correspond to joint configurations, link orientations, and all the bars to the right correspond to other velocities.

The feature for velocity in the target direction is well represented. However, so are the features for the 8 joint velocities (8 rightmost bars in each group). Since it is a lot easier to move a single joint than to coordinate all of them for locomotion, the algorithm might more easily converge to this strategy than figure out a way to walk. Moreover, because the projection mixes features for movement of single joints with features for locomotion of the entire body, it becomes more difficult for the classifier to distinguish the two. For instance, an ant that figures out how to walk may (in the projected space) look similar to one that only twitches some of its joints.

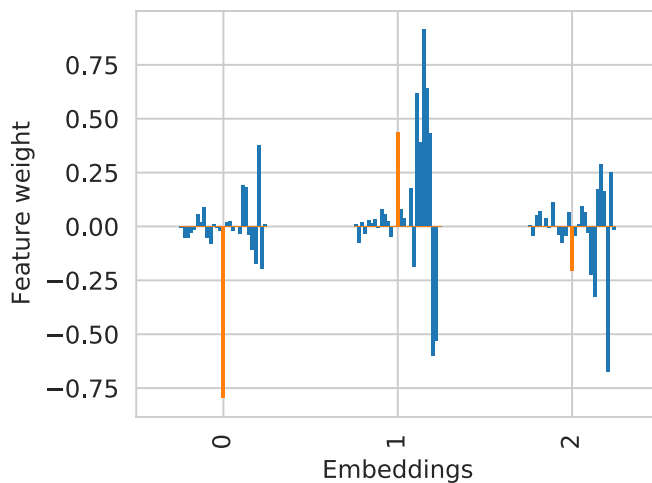


Figure VI.71: Embedding weights for a linear projection of the Ant state space down to \mathbb{R}^3 . Only weights for the first 27 (standardized) state features are visualized since the remaining 84 are always zero. The feature that corresponds to (whole body) velocity in the same direction as the main environment objective is highlighted in orange.

VI.7 Conclusion

In this work, we propose a data-driven approach for guiding skill discovery towards learning useful behaviors in complex and high-dimensional spaces. Using examples of expert data, we fit a state-space projection that preserves information that makes expert behavior recognizable. The projection helps discover better behaviors by ensuring that skills similar to the expert are distinguishable from randomly initialized skills. We show the applicability of our approach in a variety of RL tasks, ranging from a simple 2D point maze problem to continuous control locomotion. For future work, we aim to improve the embedding scheme of the state projection to be suitable for a wider range of environments.

Acknowledgment. We would like to thank Kerstin Bach and Rudolf Mester for their useful feedback.

References

- [15] LeCun, Y., Bengio, Y., and Hinton, G. “Deep learning”. In: *nature* vol. 521, no. 7553 (2015), pp. 436–444.
- [17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. “Human-level control through deep reinforcement learning”. In: *Nature* vol. 518, no. 7540 (2015), p. 529.
- [18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [19] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.
- [20] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. “Continuous control with deep reinforcement learning”. In: *arXiv:1509.02971 [cs, stat]* (July 5, 2019). arXiv: 1509.02971.
- [41] Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. “Diversity is All You Need: Learning Skills without a Reward Function”. In: *International Conference on Learning Representations*. 2018.
- [42] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.
- [143] Ross, S., Gordon, G., and Bagnell, D. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.
- [145] Salge, C., Glackin, C., and Polani, D. “Empowerment—an introduction”. In: *Guided Self-Organization: Inception*. 2014, pp. 67–114.

-
- [146] Gregor, K., Rezende, D. J., and Wierstra, D. “Variational intrinsic control”. In: *arXiv preprint arXiv:1611.07507* (2016).
- [147] Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. “Dynamics-Aware Unsupervised Discovery of Skills”. In: *International Conference on Learning Representations*. 2019.
- [148] Campos, V., Trott, A., Xiong, C., Socher, R., Giro-i-Nieto, X., and Torres, J. “Explore, discover and learn: Unsupervised discovery of state-covering skills”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1317–1327.
- [239] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. “Reinforcement learning with unsupervised auxiliary tasks”. In: *arXiv preprint arXiv:1611.05397* (2016).
- [240] Oudeyer, P.-Y. and Kaplan, F. “What is intrinsic motivation? A typology of computational approaches”. In: *Frontiers in neurorobotics* vol. 1 (2009), p. 6.
- [241] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. “Curiosity-driven exploration by self-supervised prediction”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2778–2787.
- [242] Fu, J., Luo, K., and Levine, S. “Learning Robust Rewards with Adversarial Inverse Reinforcement Learning”. In: *International Conference on Learning Representations*. 2018.
- [243] Li, K., Gupta, A., Pong, V., Reddy, A., Zhou, A., Yu, J., and Levine, S. “Reinforcement Learning with Bayesian Classifiers: Efficient Skill Learning from Outcome Examples”. In: *Deep RL Workshop, NeurIPS 2020* (2020).
- [244] Agakov, D. B. F. “The im algorithm: a variational approach to information maximization”. In: *Advances in neural information processing systems* vol. 16 (2004), p. 201.
- [245] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
- [246] Hinton, G. E. and Salakhutdinov, R. R. “Reducing the Dimensionality of Data with Neural Networks”. en. In: *Science* vol. 313, no. 5786 (July 2006). Publisher: American Association for the Advancement of Science Section: Report, pp. 504–507.
- [247] Jang, E., Gu, S., and Poole, B. “Categorical Reparameterization with Gumbel-Softmax”. In: (Nov. 2016).

VI.8 Supplementary Materials

VI.8.1 Pseudocode

Algorithm 3: Skill Discovery with Data-Driven Guidance

input : Replay buffer \mathcal{B} , dataset of expert states \mathcal{D} .
 Initialize policy π_ϑ , Q-functions $\{Q_\theta^1, Q_\theta^2\}$, discriminator q_ϕ , encoder f_χ .
Function Pre-train-Encoder(f_χ, \mathcal{D}):

- Initialize classifier h_ψ .
- Sample actions from a random policy: $a_t \sim \pi_{rand}(a_t|s_t)$.
- Step environment using the random actions: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$.
- Add visited states to the dataset: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_{t+1}, 0)\}$.
- Update encoder (f_χ) through joint training with the classifier h_ψ to maximize the likelihood of \mathcal{D} .
- Discard the classifier h_ψ .
- return** f_χ

for $epoch \leftarrow 1$ **to** num_of_epochs **do**

- for** $t \leftarrow 1$ **to** $environment_steps_per_epoch$ **do**
- Sample a skill: $z \sim p(z)$.
- Sample action from the skill-conditioned policy: $a_t \sim \pi_\vartheta(a_t|s_t, z)$.
- Step environment: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$.
- Add a transition to the replay buffer: $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, z, a_t, s_{t+1})\}$.
- end**
- for** $i \leftarrow 1$ **to** $train_steps_per_epoch$ **do**
- Sample batch of transitions from the replay buffer: $M \sim \mathcal{B}$.
- Encode states: $e_{t+1} = f_\chi(s_{t+1})$.
- Compute intrinsic reward: $r = \log q_\phi(z|e_{t+1}) - \log p(z)$.
- Update Q-functions $\{Q_\theta^1, Q_\theta^2\}$ to minimize the soft Bellman residual.
- Update policy π_ϑ using the minimum of the two Q-functions.
- Update discriminator q_ϕ with MLE.
- end**

end

output : Learned skill-conditioned policy π_ϑ .

VI.8.2 Additional Experimental Details

This appendix extends VI.5 with additional plots and commentary. Figure VI.72, VI.73, VI.74 show maximum, average and minimum return for the three environments.

VI.8.3 Implementation Details

Conceptually, our skill-discovery algorithm is the same as DIAYN [41]. There are, however, a few implementation differences that we empirically found to work just as well. Below follows a brief rundown of the key implementation details of the algorithm used in the documented experiments.

1. Two Q-functions $Q_\theta^1(s, a)$ & $Q_\theta^2(s, a)$ are used, both with target clones Q_θ^1 & Q_θ^2 , that are continuously updated with polyak averaging. Both Q-functions

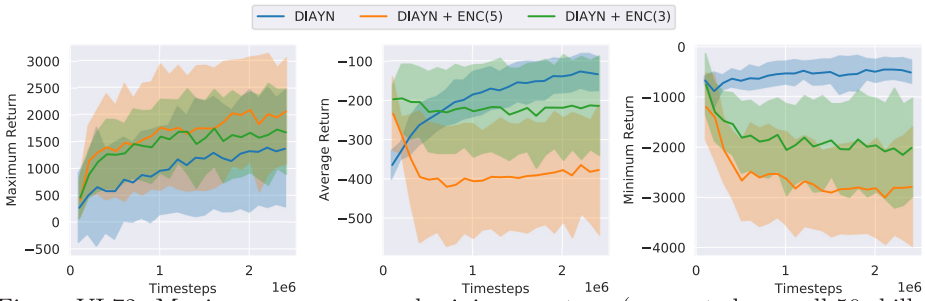


Figure VI.72: Maximum, average, and minimum return (computed over all 50 skills) for **HalfCheetah-v2** during training. Shaded areas correspond to \pm standard deviation across 5 random seeds.

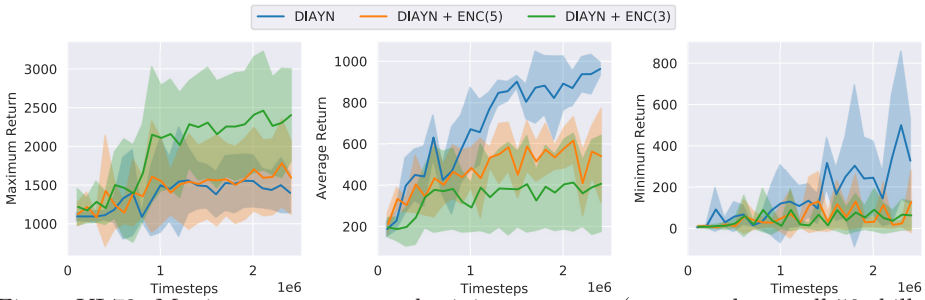


Figure VI.73: Maximum, average, and minimum return (computed over all 50 skills) for **Hopper-v2** during training. Shaded areas correspond to \pm standard deviation across 5 random seeds.

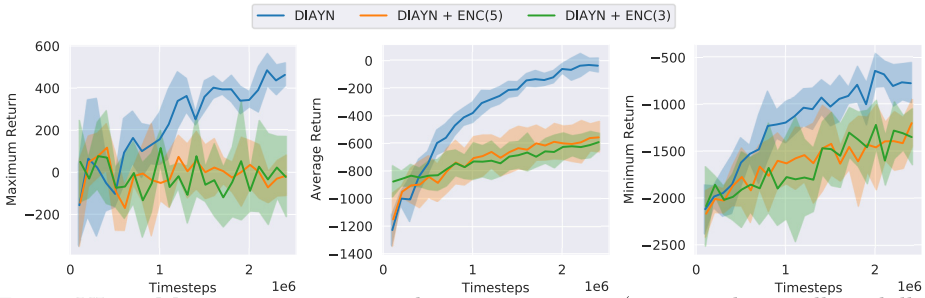


Figure VI.74: Maximum, average, and minimum return (computed over all 50 skills) for **Ant-v2** during training. Shaded areas correspond to \pm standard deviation across 5 random seeds.

attempt to predict the same quantity:

$$Q_{\theta}^{1,2}(s_t, a_t) = \mathbb{E}_{s, a \sim \pi_{\theta}}[r(s_t, a_t) + \sum_{t' > t} \gamma^{t'-t} (\alpha \mathcal{H}(a_{t'}) + r(s_{t'}, a_{t'}))]$$

2. The policy distribution is a mixture of Gaussians with four components. The policy network predicts the mixture logits, as well as the means and log

standard deviations of the Gaussians. The output is squashed through a hyperbolic tangent function, similar to [19].

3. The policy is updated by climbing the gradient of the minimum of the two Q functions (DDPG-style (Lilli-crap et al.)).

$$J(\pi_\theta) = \min_{i \in \{1,2\}} Q_\theta^i(\pi_\theta(a|s)) + \alpha \mathcal{H}(\pi_\theta(a|s))$$

This requires that the actions sampled from the policy are differentiable. Each gaussian component of the mixture is reparametrized the standard way, and the mixture is reparametrized with Gumbel-Softmax [247].

4. $Q_\theta^{1,2}$ is trained by descending on the squared temporal difference (TD) errors generated by the minimum of the target networks $Q_{\theta'}^1$ & $Q_{\theta'}^2$

$$TD(s, a, r, s') = Q_\theta^{1,2}(s, a) - r - \gamma(\min_{i \in \{1,2\}} Q_{\theta'}^i(s', \pi_\theta(a'|s')) + \alpha \mathcal{H}(\pi_\theta(a'|s')))$$

ISBN 978-82-326-6625-6 (printed ver.)
ISBN 978-82-326-6974-5 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)



NTNU

Norwegian University of
Science and Technology