NTNU

Norwegian University of
Science and Technology

A Committee of One

Using Dropout for Active Learning in Deep Networks

Martin Gammelsæter

Master of Science in Computer Science
Submission date: August 2015
Supervisor: Keith Downing, IDI
Co-supervisor: Joshua Auerbach, EPFL - LIS

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In many of the problem domains typically tackled by deep learning, data is plentiful and cheap – but labeling of the data is tedious and expensive. Letting a model actively select the data instances it is uncertain about to train on – and ignore others – can reduce the percentage of instances that must be labeled to achieve satisfactory results. To this end, this project presents a novel semi-supervised active learning algorithm called Active Deep Dropout networks (ADD-networks). It is based on evaluating a deep neural network's uncertainty on unlabeled instances, through measuring disagreement within a committee of networks derived from the original network. The committee members are Monte-Carlo-sampled from the full network using the concept of dropout. Experiments on classifying handwritten digits show that ADD-networks are comparable to a state-of-the-art method, and vastly outperforms random selection of instances.

Sammendrag

I mange av problemdomenene hvor dyp læring typisk blir brukt er innhenting av data enkelt og billig, men merking av den samme dataen med fasit er både dyrt og tidkrevende. Om man lar modellen aktivt velge datapunkter som den er usikker på – og ignorerer andre – kan mengden datapunkter som må merkes for å oppnå ønskede resultater reduseres kraftig. Dette prosjektet presenterer en ny aktiv læringsalgoritme kalt Aktive Dype Dropout-nettverk. Algoritmen er basert på å evaluere et dypt nevralt nettverks usikkerhet om den korrekte merkingen av umerkede datapunkter, gjennom å måle uenighet i en komité bestående av nettverk avledet fra det orginale nettverket. Komitémedlemmene er tilfeldig generert fra det originale nettverket gjennom bruk av konseptet "dropout". Forsøk med klassifisering av håndskrevne siffer viser at ADD-nettverk gir sammenlignbare resultater med andre moderne tilnærminger, og gir betraktelig bedre resultater enn tilfeldig utvelging av datapunkter.

Preface

This thesis concludes my Master of Science in Computer Science at the Norwegian University of Science and Technology (NTNU). The final year of my degree was spent as an exchange student at the École Polytechnique Fédérale de Lausanne (EPFL), in Lausanne, Switzerland – where the project described in this thesis was carried out in the period between February and August, 2015. The work was performed in Professor Dario Floreano's Laboratory of Intelligent Systems (LIS), under the supervision of Joshua Auerbach and Giovanni Iacca.

Ever since I was first introduced to the concept, artificial neural networks in all forms have fascinated me tremendously. This fascination led me through courses from logic to neuroscience to robotics, it led me to a specialization in artificial intelligence at NTNU, to Switzerland, EPFL, and the LIS, and ultimately – for now – it led me to this project, and this day. Throughout this adventure I have had lots of support – from colleagues and teachers alike. In the context of this project, I would most of all like to thank the main supervisor, Joshua Auerbach. Not only did you come up with the core idea, and generously handed it to me to explore – you have provided invaluable support and guidance to a novice in the realm of science. Thanks also to Giovanni Iacca, the co-supervisor, for prized discussions and advice.

But it is not an adventure without obstacles. For keeping me sane through times of frustration and feelings of insurmountable obstacles ahead, a big thank you to my friends and family, in both Norway and Switzerland. But, most of all I wish to thank my parents for their unending support, in all my endeavours. Without you, this day would never have come.

Trondheim, August 24, 2015

Martin Gammelsæter

Contents

1 Introduction

1.1 Motivation

During the last decade, machine learning has become an increasingly important part of many companies' business processes, products, and services. Examples of this range from predicting sales figures and customer demands, to automatic speech recognition of customer queries, designing new drugs, image recognition, and robotics. In many of the mentioned problem domains, data for training is often abundant – for example are millions of images uploaded for public consumption on the internet every day. The bigger problem is accurate labeling of this data, that is, providing the true classification or observed value. Depending on the level of annotation, transcription of speech recordings can take up to 400 times longer than the source recording [89], for example. In other domains, such as building forward models in robotics, the number of possible actions may well be infinite, and learning the results of these actions means actually performing them on the robot – an expensive process. Finding ways to cut down on the amount of labeling that has to be performed to achieve satisfactory results in training then, can be an obvious benefit. Achieving this goal by letting an algorithm actively choose the data it finds ambiguous, is the domain of active learning.

A great deal of this recent surge in interest into machine learning comes on the back of the emergence of so-called deep learning [44], with its deep hierarchical neural networks. These networks typically require training on huge amounts of labeled data to work well. For researchers, many benchmark datasets are available, but for many real-world needs, there is a significant cost to amassing and labeling data. Methods from active learning can serve well here, reducing the number of labeled data points required to achieve good performance, but few active strategies exist that work well in conjunction with deep neural networks.

Active learning traces its roots back to the statistical field of optimal experimental design [69], and contains many general approaches to the problem of choosing data to label. The problem with many of them in the context of deep learning is that they are mostly designed with other models in mind,

typically Support Vector Machines or simple models such as decision trees or Gaussian Mixture Models, and are often either restrictively computationally expensive, or ineffective in many of the problem domains in which deep learning is typically applied.

We believe that a computationally feasible and effective active learning algorithm for deep networks, which can fill this gap, could help bring the benefits of deep learning to an even more diverse set of applications than previously feasible.

1.2 Research Questions

This project seeks to explore an algorithm that could possibly help fill the void discussed in the previous section, based on a core idea proposed by Joshua Auerbach – the main advisor to this project. It revolves around a novel combination of a staple active learning algorithm, *Query by Committee*, and ideas originating from a recent neural network regularization technique, *dropout*.

The Query by Committee (QBC) algorithm [71] will be described in detail in Chapter 2, but in brief, it keeps a committee of models, and for each data instance to be evaluated for potential labeling, each member of the committee provides a prediction. The algorithm then quantifies disagreement amongst the committee members, and uses this "disagreement score" to decide whether or not to label the instance.

Dropout regularization [75] – also detailed further in Chapter 2 – is based on randomly "dropping out" neurons from a neural network during training, to mitigate co-adaptation of features. To drop out a neuron means that that neuron is not allowed to contribute to the network. Different neurons are dropped out for each iteration of training, having the effect that neurons cannot depend on other specific neurons being present to be able to perform their functions. The key consequence of dropout in the context of this project is the realization that stochastically dropping neurons in effect generates a completely different network, one with a different topology – both from the full network, and from all other networks with a different subset of dropped neurons. The idea then, is to Monte-Carlo-sample multiple such networks,

and treat them as completely distinct models making up a committee, to be used by the QBC algorithm whenever an active selection is to be made.

To explore this idea, the following research questions have been formulated:

1. Does the variance of predictions made by dropout networks – all generated from the same full network – provide a heuristic for the confusion of the full network on the data example being predicted?

2. Can the Query by Committee algorithm, using such a group of networks as the committee, outperform random selection of examples?

3. Can this algorithm perform comparably to other active learning algorithms that are applicable to deep neural networks?

The first question is the crux, as a negative answer to this would invalidate the entire idea. It is certainly conceivable that the variance is too small to provide any information at all, or that all of the variance stems from other factors than confusion.

The second question is more practical. Assuming the variance *does* contain information about the confusion, are we able to formulate an algorithm that can take advantage of it?

While positive answers to both questions is preferable, a positive answer to the first question, and a negative to the second would still be valuable. Having a good way to estimate the uncertainty of a network on a prediction can be useful in many scenarios, ones where we only want to act if the network is sufficiently certain in its prediction for example.

As for the third question, this project will be unable to definitively answer how the proposed algorithm compares to others in all scenarios. However, even ballpark figures on a single problem can help shed light on the general feasibility of the algorithm.

An attempt to answer these three questions form the basis of the project.

1.3 Thesis Structure

Chapter 2: Presents the background theory on active learning and deep learning that is needed to understand the project.

Chapter 3: Describes the proposed architecture, and describes the dataset and experiments used to examine the posed research questions.

Chapter 4: Presents and analyzes the results from these experiments.

Chapter 5: Discusses whether the research questions were answered, the novel contributions of the project, the proposed architecture's possible uses, and interesting directions for further work, before concluding the thesis.

2 Theoretical Background

2.1 Overview

In this chapter, an overview of the fields and ideas that underpin the proposed architecture of Chapter 3 will be given. The most important background concepts come from two different subfields of machine learning, namely active learning, described in Section 2.2, and deep learning, described in Section 2.3. None of these sections will cover their respective field in full, but will describe the ideas that are necessary to fully grasp the concepts introduced later. At the end of the chapter, earlier attempts to combine these two fields, and other research with similar goals as that described in this thesis is discussed in the Related Work section (Section 2.4).

2.2 Active Learning

The concept of *Active Learning* rests on the idea that some machine learning models are able to learn more effectively if they can *actively* select what data to learn from, instead of being passively fed data as is the norm with machine learning approaches. In domains tackled by supervised machine learning, the act of labeling the training data is often a complex operation, potentially requiring domain experts or other costly and time-consuming processes. By only selecting a subset of the available data to train from, the total cost of labeling is reduced. These are the main motivations behind research into active learning.

The main hypothesis of active learning is that by being able to analyze and ask questions about the data available, the model can form an opinion about which data points will provide the most useful information, and through that achieve better results with less training. In some problem domains this approach can be very effective, greatly reducing the number of data points that have to be labeled in order to learn a concept sufficiently well [69]. Any domain where data is readily available but labeling is costly might be a good fit for employing active learning. Some examples:

- *Medical imaging*. Making diagnoses from medical images – such as those stemming from X-ray- or MR-imaging – is an important part of modern medical treatment. It is however an inherently error-prone process. For example, in studies on mammography, it has been observed that among those with suspicious lesions sent to biopsy, only 15% – 34% are found to actually have malignancies [85, 39, 40]. Using machine learning systems to aid medical professionals in analysis of the images is therefore an important and interesting field of current study [85, 87, 53, 49]. Labeling the data reasonably accurately in this case often requires performing biopsy or other invasive methods of diagnosis on the patient, usually both risky and time-consuming, while acquiring the images themselves is cheap in comparison. Here, lowering the number of labels needed for a machine learning model to generalize is obviously beneficial.

- *Document classification*. Classifying documents such as e-mails or newspaper articles into classes such as *spam*, *not-spam*; *interesting*, *not-interesting*; or determining the topic of the document are tasks in which machine learning is already very popular [61, 1, 68]. Some of these tasks, like spam-filtering, are all but solved by most e-mail providers. Since most users agree on what is spam and what is not, the task of labeling it can be distributed among millions of users (through functionality such as "Mark as Spam"), reducing the cost. But not all document classification problems can be distributed in this manner. For example, if one is trying to create a system which recommends articles to a user based on what that user has found to be interesting in the past, the distributed approach is less effective. In such a system, being able to provide good suggestions with as little explicit labeling as possible is vital for the user experience. Active learning for document classification has shown promising results in multiple studies [78, 62, 50].

- *Speech recognition*. Modern machine learning systems for speech recognition are trained using manual transcriptions as the labels for recorded utterances [58]. While getting hours and hours of speech recordings is rather simple, transcription on the other hand is a very time consuming task – taking on the order of ten times longer than the source recording to annotate at word level, and up to 400 times longer if annotating at phoneme level [89]. Research on applying active learning to speech recognition tasks has been going on for decades [58, 28, 13], and has

been one of the driving forces between active learning advances in general.

- *Modeling complex systems*. In many fields, from systems biology to robotics, complex nonlinear systems are abundant, and creating good mathematical models is often hard and tedious. Many different approaches to automatic modeling have been proposed, but active ones have been shown to be very effective. For example, Bongard and Lipson [8] present an active approach to automatically generate symbolical models from time series data, and shows its efficiency on problems from a wide range of fields. Cohn et al. [15] uses active learning to allow a robotic hand to predict its position given the joint angles in the hand. Similarly, Bongard et al. [9] uses an active self-modelling process that allows a robot to be resilient to unexpected damage.

2.2.1 Different Scenarios

Depending on the specifics of the problem at hand, active learning can be approached in different ways. At its core, the active learning process boils down to repeatedly choosing an unlabeled data instance that the model evaluates to be especially likely to be informative, then querying some labeling oracle for the actual label of that instance, so that the model can use it for supervised learning. The oracle can take many forms – it can be a human domain expert, another program that performs some expensive computation, or a machine that performs an experiment, for example. How the model acquires the instances it evaluates is an important distinction between systems, and the literature defines roughly three different approaches to this question [69]:

De Novo Synthesis

In some scenarios it makes sense for the model to be able to ask queries about any and all points in the input space. That is, the model only has to know the number of dimensions and their ranges, and can within that space query its oracle for the real target value(s) (in regression problems) or label (in classification problems) of any specific point that it thinks is likely to increase its knowledge. These kind of queries were some of the first to be researched – for example by Dana Angluin in 1988 [2]. They are typically useful in domains where all points in the input space can be labeled, and the

input space is finite. One example is the problem of predicting a robot hand's coordinates in some space given its joint angles, as explored by Cohn et al. in 1996 [15]. In this context, there is no point in restricting the learner to some predefined and pre-sampled set of points in the space to learn from. In domains such as the one mostly explored in this thesis – handwritten digit recognition – de novo synthesis of queries is all but useless however, as most points in the input space are not possible for a human oracle to label as a certain digit [5]. The core problem of this method is that the model does not necessarily have any idea of the distribution of real world data, and may generate points that are devoid of meaning, or focus on areas of the input space which are extremely unlikely. A flowchart depicting the active learning loop in de novo synthesis can be seen in Fig. 2.1.

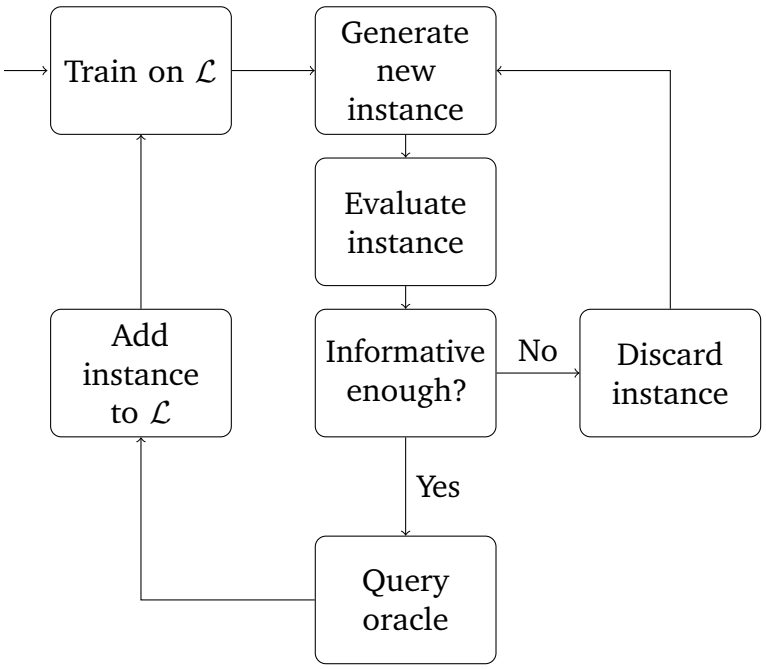


Fig. 2.1: Flowchart for the de novo synthesis loop. How to evaluate instances and determine whether or not they are informative (enough) will be covered in the next section. $\mathcal{L}$ is the set of labeled training examples.

Stream-sampling

One response to the shortcomings of de novo synthesis is stream-sampling, pioneered by Atlas et al. [3]. Here, instead of the learner generating data points by itself, it is fed data, usually one point at a time, sampled from the actual distribution. The learner must then – for every data point it receives – decide whether to query the oracle for the label of this data point and add

it to its training set, or discard it. This method avoids the problem of de novo synthesis where the data points it considers might not be representative of the actual distribution in real world data. Obviously, if the real data is uniformly distributed across the input space, this might not bring any advantages over the de novo approach. Stream-sampling might be especially useful in online learning scenarios where new data is continuously provided – but labeling is expensive – so one ideally wants to only label data that will improve generalization a lot. The flowchart for stream-sampling is practically identical to that for de novo synthesis (Fig. 2.1), only instead of generating a new instance in the top node, one is sampled.

Pool-sampling

A natural extension from stream-sampling is pool-based sampling, first proposed by Lewis and Gale in 1994 [48]. In many cases, a large amount of data is collected long before any learning is started, which an active learning algorithm can later take advantage of. By evaluating every single data point in the unlabeled pool, it can choose one or more examples that are maximally likely to improve performance, query the labels, add them to the training set, and then repeat the process with the now improved model. This approach is much more computationally expensive than sequentially evaluating and querying or discarding examples, but can yield better results because every example is compared before making a decision. The core assumption for pool-based sampling to make sense is that the cost of labeling dominates the computational cost of evaluating every example in the unlabeled data pool every time it is sampled. This assumption appears to often hold in real-world scenarios, and pool-based sampling is often seen in applied research on active learning [48, 70, 30]. Fig. 2.2 shows an illustration of the pool-based sampling loop.

While these three scenarios are the main foci of the literature, some combinations of the above are also possible. Examples include batching stream-sampling, i.e. waiting for some set number of examples to be presented before deciding on which of those examples, if any, are to be added to the training pool; or batching de novo synthesis, i.e. generating multiple examples at once, before evaluating, etcetera. These combinations might be worth considering if finding a good threshold for querying is difficult, since they allow for setting in advance an explicit percentage of examples to be

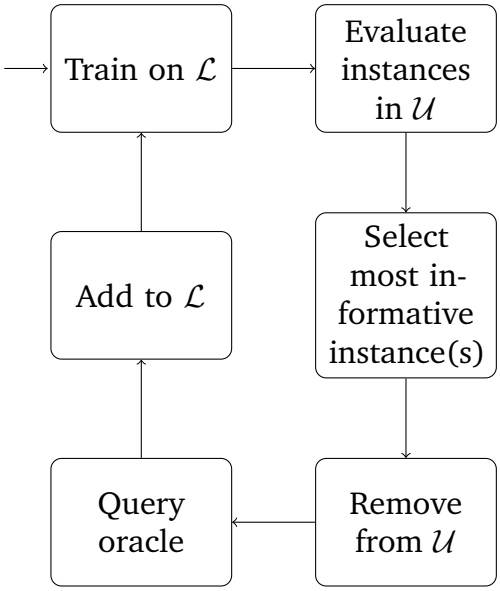


Fig. 2.2: Flowchart for the pool-based sampling scenario. How to evaluate instances will be covered in the next section. $\mathcal{L}$ is the set of labeled training examples, and $\mathcal{U}$ is the unlabeled set.

queried, controlled by the batch size and the number of examples to query per batch.

Throughout the literature and this short review, the set of labeled training examples used for training is referred to as $\mathcal{L}$. In the case of pool-based sampling, the set of unlabeled training examples in the "pool" is referred to as $\mathcal{U}$.

2.2.2 Evaluating Examples

When it comes to actually evaluating the examples, the literature provides many possible options. The choice of which method to use is often guided by the combination of the choice of model and the specifics of the problem at hand. While there are many ways to approach this issue, the following sections will focus on the methods most relevant to the project described in this thesis. Specifically, some particular methods for evaluating examples in classification and regression problems will be highlighted, although methods for other problem classes exist [17, 70]. For the same reason, mostly methods applicable to neural networks models are considered. For a more comprehensive review, refer to Settles, 2002 [69].

Uncertainty Sampling

The most straightforward and yet perhaps one of the most popular general approaches is *uncertainty sampling*, as described by Lewis and Catlett, 1994 [47]. It is a very intuitive concept – the main idea is to choose those examples that the model is the most uncertain about, as those examples are presumably the ones which would yield the most information. Conversely, examples for which the model has high confidence in its prediction do probably not provide much new information. For classification problems, this reduces to selecting those data points which are closest to the decision boundary of the model. To illustrate the concept, an example of a simple algorithm using the concept of uncertainty sampling in a pool-based scenario can be seen in Alg. 1. Tweaking this algorithm for the other scenarios is simple: Instead of looping through $\mathcal{U}$, one simply receives a single instance (generated or presented), evaluates it, then decides whether or not to query based on some threshold. Pool-based sampling will be used in examples, as it is the most complex of the three.

Algorithm 1 A basic algorithm for pool-based active learning using uncertainty sampling

$\mathcal{M} \leftarrow$ the model to be trained
$\mathcal{L} \subset \mathcal{U}$ ▷ $\mathcal{L}$ is initialized as some nonempty subset of $\mathcal{U}$
$train(\mathcal{M}, \mathcal{L})$
while not finished training **do**
    $highestUncertainty \leftarrow -\infty$
    for all $x \in \mathcal{U}$ **do**
        $uncertainty \leftarrow evaluate(x)$ ▷ $evaluate$ estimates the uncertainty of $\mathcal{M}$ on x
        if $uncertainty > highestUncertainty$ **then**
            $mostUncertain \leftarrow x$
            $highestUncertainty \leftarrow uncertainty$
        end if
    end for
    $y \leftarrow query(mostUncertain)$
    $\mathcal{L} = \mathcal{L} \cup \langle mostUncertain, y \rangle$
    $train(\mathcal{M}, \mathcal{L})$ ▷ Either training from scratch or incrementally
end while

Assuming the model can present a probability distribution over the possible classifications, there are multiple methods available to approximate the distance from the decision boundary. When dealing with binary classification it is very simple – the decision boundary is simply where the posterior

probability is 0.5. If using pool-based sampling, one simply chooses the example(s) with a posterior prediction closest to 0.5 to query. In a stream or de novo synthesis scenario, one can choose a threshold, so if some example has a prediction that is closer to the margin than the threshold, it is queried. When the problem at hand requires three or more classes however, we need some different methods to approximate the distance from the margin. Three popular examples of such methods are the *least confident*, *margin*, and *Shannon-entropy* measures of uncertainty:

- *Least Confident*. The simplest measure is obtained by only looking at the class with the highest probability of being correct, that is, the model's prediction for this example. Since we assume a probabilistic classifier, which provides a probability distribution over the possible classes, the probability assigned to the predicted class can be interpreted as a proxy of the uncertainty of the model on that particular example. One then chooses the example(s) in which the model is the *least confident* in its prediction to query. In mathematical terms:

$$x^* = \underset{x}{\operatorname{argmin}}\, P(\hat{y} \mid x) \tag{2.1}$$

  where x^* is the example to query, $\hat{y}$ is the predicted class, and $P(\hat{y} \mid x)$ is the probability assigned to that class by the model.

- *Margin*. Going one step further, the *margin* measure includes the top two classes as ranked by the assigned probability by the model. Instead of only looking at the probability of the predicted class, one looks at the difference between these top two classes, essentially determining how "close a call" the prediction was. One then assumes that the closer the call, the more information is gained by querying that example:

$$x^* = \underset{x}{\operatorname{argmin}}\, P(\hat{y}_1 \mid x) - P(\hat{y}_2 \mid x) \tag{2.2}$$

  where $\hat{y}_1$ is the most likely class as deemed by the model, and $\hat{y}_2$ is the second most likely class.

- *Shannon Entropy*. Using Shannon's concept of information theoretic entropy [72], this uncertainty measure considers the probabilities assigned to all of the classes. It takes the entropy (denoted by H) over these probabilities, which is a measure of uncertainty – a quantification

of how unpredictable a random variable is. H can be defined as follows:

$$H(Y \mid x) = -\sum_{y}^{Y} P(y \mid x) \log P(y \mid x) \tag{2.3}$$

which means that e.g.

$$H([0.5, 0.5]) > H([1.0, 0.0]) \tag{2.4}$$

That is, the information content of a probability distribution of $[0.5, 0.5]$ is higher than that of $[1.0, 0.0]$, which is exactly what we want. The example selection is then defined as:

$$x^* = \underset{x}{\operatorname{argmax}}\, H(Y \mid x) \tag{2.5}$$

where Y is the full prediction distribution provided by the model on example x.

All of the uncertainty measures above assume a classification problem, because they all require either a specific label prediction, or a probability distribution over the possible labels. While the listed measures are useless for regression problems, the general idea of uncertainty sampling is still useful in the context of regression. But, there are some immediate problems. How do you estimate uncertainty when you do not have a probability distribution to go from? Some models, like a Bayesian model, outputs a distribution over possible values, but most others – including neural networks – do not. Since the outputs of a regression model is usually simply real values then, and not labels, we need a different way. One obvious candidate is estimating the *output variance* of the model for different regions of the input space. As shown by e.g. Cohn [14], this is possible using techniques from *optimal experiment design*. These calculations are very computationally expensive however, and therefore unsuitable when dealing with deep neural networks with an immense number of weights.

In addition to the problems with uncertainty sampling and regression, studies have observed that for some practical domains, active learning by uncertainty sampling performs worse than random selection [67, 77, 82]. For many problem domains and classes then, we need a different approach.

Hypothesis-search

A related, but different way of working with evaluating examples than uncertainty sampling is methods based on searching the hypothesis space. These methods rise from looking at the problem from a different angle: Instead of just looking at the uncertainty of a single (partly) trained model, consider the full hypothesis space, $\mathcal{H}$, and the version space subset, $\mathcal{V}$. $\mathcal{V}$ is the set of hypotheses which correctly classifies the data in the training set.

Query by Disagreement Some classic machine learning algorithms work by gradually reducing the size of $\mathcal{V}$ through training [52, Chapter 2], and Cohn et al. presented in 1994 [13] an active learning algorithm based on this concept, *Query by Disagreement* (QBD). It assumes a stream-based scenario (as described in section 2.2.1), and works roughly as described in Alg. 2. Basically, for every instance x it receives, it loops through all of the hypotheses $h \in \mathcal{V}$, and if any two predictions $h(x)$ differ, that is we have disagreement, it queries the label of that instance. That instance, with its label, is then added to the training set $\mathcal{L}$, and $\mathcal{V}$ is rebuilt to only contain hypotheses that correctly classify all of (the new) $\mathcal{L}$.

Algorithm 2 Query by Disagreement [13]

$\mathcal{L} \leftarrow \emptyset$
$\mathcal{V} \leftarrow \mathcal{H}$
while not finished training **do**
    $x \leftarrow receiveInstance(\mathcal{D})$
    for all $h \in \mathcal{V}$ **do**
        if $h(x) \neq previous$ **then**
            $y \leftarrow query(x)$
            $\mathcal{L} \leftarrow \mathcal{L} \cup \langle x, y \rangle$
            $\mathcal{V} \leftarrow \{h \mid h(x^*) = y^* \text{ for all } \langle x^*, y^* \rangle \in \mathcal{L}\}$
            break for
        end if
        $previous \leftarrow h(x)$
    end for
end while

Depending on the size of $\mathcal{V}$ after training, classifying unseen instances can be done in different ways. If it is 0, the training data is inconsistent, and QBD does not work. If it is 1, one simply uses this hypothesis for prediction. If it is > 1, one can either choose some hypothesis $h \in \mathcal{V}$ by some heuristic, or use some voting mechanism.

Unfortunately, there are some game-breaking issues with this algorithm – especially in real-world scenarios, many of which stem from its inspiration, *concept learning* [52, Chapter 2]. Most of its drawbacks are well detailed in Mitchell, 1997 [52] and Settles, 2002 [69], but some of the most problematic issues are worth discussing here. For most problems, maintaining the entire version set in memory is problematic or even impossible. $\mathcal{V}$ may very well be infinite. Many ways have been proposed to mitigate this issue, such as so-called $\mathcal{SG}$-based QBD [13]. This works by instead of keeping track of the full $\mathcal{V}$, we merely keep two specific hypotheses: One from the set of most specific or conservative hypotheses ($\mathcal{S} \subseteq \mathcal{V}$), and one from the set of most general hypotheses ($\mathcal{G} \subseteq \mathcal{V}$). If the two hypotheses disagree, the instance is queried for its label. The wrong hypothesis must then be replaced: If the most specific hypothesis was wrong, it was *too* specific, and must be replaced by the new most specific hypothesis that correctly classifies $\mathcal{L}$, and vice versa. The core idea is that these two hypotheses represent $\mathcal{V}$ by defining its "borders", or extremities. Unfortunately, it has been shown that maintaining $\mathcal{S}$ and $\mathcal{G}$ is still prohibitively expensive in most cases [31].

Query by Committee There are fortunately some additional assumptions in QBD that can be relaxed. First, and most critically is the assumption that *every* $h \in \mathcal{V}$ must be considered when measuring disagreement, either directly or indirectly (through simplifications such as $\mathcal{SG}$-based QBD). In many popular types of models, such as neural networks and other non-discrete models, the concept of a well-defined, non-infinite version space all but breaks down. Maintaining $\mathcal{S}$ and $\mathcal{G}$ becomes likewise impossible or practically so. It seems reasonable, however, to assume that simply randomly sampling $\mathcal{V}$ for n hypotheses between which disagreement is measured, can provide a useful heuristic for the disagreement between hypotheses in $\mathcal{V}$. This idea was first formulated by Seung et al. in 1992 [71] as the *Query by Committee*-algorithm (QBC), where the committee (denoted $\mathcal{C}$) refers to the set of sampled hypotheses.

The original version of this algorithm is merely a tweak of QBD, and assumes that randomly sampling from $\mathcal{V}$ is computationally much cheaper than maintaining it. While this alleviates the need to maintain the full $\mathcal{V}$ (or variants like $\mathcal{SG}$), there are still some issues. If we are dealing with a noisy problem, with possibly noisy labeling, the idea of using the version space as a base breaks down. Several approaches to avoiding this issue have been proposed. One example is taking a likelihood-based approach as proposed by among

others Dagan and Engelson, 1995 [18]: Instead of sampling from some explicit $\mathcal{V}$, one samples from a posterior probability distribution, $P(h \in \mathcal{H} \mid \mathcal{L})$, biasing samples from $\mathcal{H}$ towards hypotheses that are more likely given the training data, $\mathcal{L}$. Unfortunately, this process is not always so simple either, motivating an even more generalized approach; namely thinking of the committees in QBC as simply *ensembles* like in ensemble learning [22]. This allows for using any type of ensemble learning method, like *bagging* [10] or *boosting* [64] for example, to construct committees for QBC. As models trained on $\mathcal{L}$ or some subset of $\mathcal{L}$ will have a higher likelihood given the data than a randomly sampled hypothesis, this method acts similarly as the likelihood-biased sampling. Just as in ensemble learning, it is important to make sure that there is some diversity among the models in $\mathcal{C}$.

While this approach requires some number of labeled examples in $\mathcal{L}$ ahead of time to bootstrap the process, it generalizes QBC to all model classes, and even allows multiple different types of model classes to "sit on" the same committee. An additional effect of this approach is that instead of seeing disagreement as some binary measure, we are able to differentiate between examples, so that unlabeled examples can be ranked, allowing for usage of QBC in a pool-based scenario. All it requires is a heuristic for disagreement in the committee. Furthermore, it generalizes QBD beyond classification, allowing its use on regression problems as long as the disagreement heuristic can handle real-numbered predictions.

QBC now refers to all algorithms that are based on measuring disagreement between committee-members as the heuristic for choosing which examples to query, and a general illustration of the algorithm, in this case for the pool-based scenario, is seen in Alg. 3.

Using the trained committee to predict unseen examples can be done in many different ways, for example through voting mechanisms for classification or averaged outputs for regression. The best way to achieve this usually depends highly on the type of models used, and a discussion of these alternatives is outside the scope for this thesis. The reader is referred to the ensemble learning literature for details [22].

Analogous to the uncertainty measures of the previous section, many different ways to quantify disagreement exists, their effectiveness and applicability depend on the problem specifics. Some popular examples of such disagreement measures:

Algorithm 3 Generalized Query by Committee for pool-based scenarios

```
1: C = the committee of models
2: L ⊂ U                    ▷ L is initialized as some nonempty labeled subset of U
3: train(C, L)
4: while not finished training do
5:     highestDisagreement ← −∞
6:     for all x ∈ U do
7:         predictions ← ∅
8:         for all h ∈ C do
9:             predictions ← predictions ∪ h(x)
10:        end for
11:        disagreement ← evaluateDisagreement(predictions)
12:        if disagreement > highestDisagreement then
13:            highestDisagreementExample ← x
14:            highestDisagreement ← disagreement
15:        end if
16:    end for
17:    y ← query(highestDisagreementExample)
18:    L = L ∪ ⟨highestDisagreementExample, y⟩
19:    train(C, L)
20: end while
```

- *Vote Entropy*. With an obvious connection to the uncertainty measures discussed earlier, vote entropy works by considering the entropy in the votes cast by committee members. Since we are dealing with label votes, it is obviously only applicable in this form on classification problems. There are two versions of the measure, *hard* and *soft*. The hard vote entropy is defined as follows:

$$v_h(x, y) = \begin{cases} 1 & \text{if hypothesis } h \text{ voted for label } y \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$v(x, y) = \sum_{h}^{\mathcal{C}} v_h(x, y) \tag{2.7}$$

$$x^* = \underset{x}{\operatorname{argmax}} - \sum_{y} \frac{v(x, y)}{|\mathcal{C}|} \log \frac{v(x, y)}{|\mathcal{C}|} \tag{2.8}$$

  If the models in $\mathcal{C}$ can provide a probability distribution over all the possible labels, we can use the soft version:

$$P_{\mathcal{C}}(y \mid x) = \frac{\sum_h^{\mathcal{C}} P_h(y \mid x)}{|\mathcal{C}|} \tag{2.9}$$

$$x^* = \underset{x}{\operatorname{argmax}} - \sum_y P_{\mathcal{C}}(y \mid x) \log P_{\mathcal{C}}(y \mid x) \tag{2.10}$$

As we can see, the vote entropy disagreement measure is basically the same as the Shannon's entropy uncertainty measure, except this version smoothes the estimation over multiple samples, which should prove more robust.

Although we will not go through the trouble of properly defining them here, committee-based versions could easily be imagined from the other uncertainty measures defined earlier as well.

- *Kullbach-Leibler Divergence*. Proposed and formalized by Kullbach and Leibler in 1951 [43], KLD is a measure of the difference between two different probability distributions. The most obvious way of using it as a disagreement measure is when the committee is of size 2. In that case, we can simply use the divergence between the two predictions, where KLD is defined as follows:

$$KLD(P_1(Y \mid x), P_2(Y \mid x)) = \sum_y^Y P_1(Y \mid x) \log \frac{P_1(Y \mid x)}{P_2(Y \mid x)} \tag{2.11}$$

And the selection mechanism simply:

$$x^* = \underset{x}{\operatorname{argmax}}\, KLD(P_{h_1}(Y \mid x), P_{h_2}(Y \mid x)) \tag{2.12}$$

where h_1 and h_2 are the two models/hypotheses in the committee. Generalizing this to more than two models can be done by taking the mean of each committee-member's divergence from the mean distribution:

$$P_{\mathcal{C}}(Y \mid x) = \frac{\sum_h^{\mathcal{C}} P_h(Y \mid x)}{|\mathcal{C}|} \tag{2.13}$$

$$x^* = \underset{x}{\operatorname{argmax}} \frac{\sum_h^{\mathcal{C}} KLD(P_h(Y \mid x), P_{\mathcal{C}}(Y \mid x))}{|\mathcal{C}|} \tag{2.14}$$

- *Prediction Variance*. The two preceding disagreement measures can both be effective in classification scenarios, but we need something different for regression where we usually do not have votes or probability distributions available. Intuitively, the variance among the committee's predictions is a good candidate. There are multiple ways to calculate

this, and the results are usually similar. Here, one way is presented – calculating the mean Euclidean distance from the mean. First take the mean prediction:

$$h_{\mathcal{C}}(x) = \frac{\sum_h^{\mathcal{C}} h(x)}{|\mathcal{C}|} \tag{2.15}$$

Then the mean Euclidean distance from that point as the score, selecting the highest scoring example:

$$x^* = \underset{x}{\operatorname{argmax}} \frac{\sum_h^{\mathcal{C}} \sqrt{\sum_i (h(x)_i - h_{\mathcal{C}}(x)_i)^2}}{|\mathcal{C}|} \tag{2.16}$$

Note that there is nothing preventing the use of a variance-based measure like this for classification as well.

2.2.3 Summary

While this section far from covers the field of active learning, it attempts to explain the core ideas and algorithms that make up part of the foundation of the project this thesis describes.

We looked at different scenarios in which active learning can be performed, like query synthesis, stream-sampling, and pool-based sampling. Throughout the rest of this thesis, unless explicitly stated otherwise, pool-based active learning is assumed. Furthermore, some different ways of determining which examples to query, the core problem of active learning, have been discussed: The general ideas of uncertainty sampling and searching the hypothesis-space, and the specific algorithms motivated by them. The query by committee algorithm, one of several discussed in this section, is especially important to understand, as it lays the groundwork for the novel ideas this project is built on.

2.3 Deep Learning

The *perceptron*, proposed by Rosenblatt in 1958 [59] and inspired by the biological neuron, laid the foundation for artificial neural networks (ANNs) as we know them today. In the time since Rosenblatt's landmark paper, countless improvements have been proposed, and theoretical properties investigated. The proof that multilayer feedforward neural networks are

universal function approximators [36] is one such theoretical advance. On the more practical side; the invention of the *backpropagation* algorithm [63] provided an efficient and fairly straightforward way to train neural networks (while there is some controversy around its invention, specifically by who and when it was invented [66], there is no neglecting its monumental impact on the popularity of artificial neural networks). Perhaps the most important development in the practical use of ANNs in recent years cannot be credited to the neural network research community however: the exponential increases in computing power available, even further boosted by the advent of the use of graphical processing units (GPUs) for massively parallel general purpose processing, has "suddenly" made ANNs applicable to problems that were earlier out of reach. This has led to a renaissance in neural network research, especially in nets with more than one hidden layer of nonlinear transformations, popularly dubbed *deep neural networks* (DNNs). The branch of machine learning based on these networks and their descendants is often referred to as *deep learning*. Deep networks are, for complex problems, able to outperform their single-layered cousins, even though, as we have seen, they are proven to be universal function approximators. The reason for this is that the problem is not in a neural networks representational power, but in the problem of training them. The idea is that deep networks can represent hierarchical relationships such as those observed in the human brain, for example in the visual cortex. That is, lower level concepts such as simple edge detection can be represented by features in the early layers, with features of increasing complexity made up by inputs by the simpler features as you get closer to the output. Much research remains before the theory catches up with what is achieved in practice, however.

Because of the practicality of deep learning in many real world problems, and perhaps because of their abundant computational resources, the deep learning renaissance has to some extent been fueled by large technology companies such as *Google*, *Baidu*, *Facebook*, and others. Many real-world problems have been tackled by deep learning, and it is in many cases the most effective approach known [65]. Some examples of domains in which deep learning has been shown effective:

- *Image recognition*. Perhaps the most researched and discussed problem tackled by deep learning is that of image recognition. Many deep learning architectures are inspired by how the vision system in the human brain works, and the problem is interesting both from a theoretical and

practical viewpoint. Benchmark problems like the MNIST handwritten digit dataset [46], the street view house number (SVHN) dataset [54], and the ImageNet natural image database [20] are examples of important benchmarks which have allowed researchers to directly compare results and compete on standardized ground, facilitating progress. In 2012, Krizhevsky et al. [41] used a deep neural architecture to win a competition on the ImageNet dataset with a large margin over the state-of-the-art algorithms previously dominating the field of computer vision, showing the usefulness of neural networks on even such extremely large scale problems as ImageNet, which consists of millions of natural images and thousands of different classes. Ever since, progress with deep networks on large scale image recognition tasks has been steady, in no small part to the ever-increasing computational resources available to researchers.

- *Automatic speech recognition*. Also one of the domains in which deep learning has been most successful, the problem of automatic speech recognition is of great interest and use to many different fields – from interview transcription to speech-based user interfaces. As a problem with many obvious applications within industry, speech recognition with deep models is widely researched and applied by the likes of Microsoft [21], Baidu [29], IBM [32], and Google [32].

  The TIMIT dataset [24] is an important benchmark, providing recordings of many spoken English sentences for transcription.

- *Natural language processing*. There are many subfields of natural language processing (NLP), many of which deep learning has been successfully applied, such as sentiment analysis [26], information retrieval [37], scene description [81] (a very interesting combination of image recognition and NLP), and others [16].

- *Drug discovery*. Recently, some researchers have turned to deep learning for applications within the field of drug design, in an attempt to better predict the effectiveness and possible toxicity of a new drug, among other aspects of drug discovery [19, 56, 79].

These are just some examples of applications of deep learning, and new ones appear regularly. It has in other words been shown to be effective in a wide array of complex and many-dimensional pattern recognition tasks.

There are more variations and variations of variations of deep neural networks than this thesis can possibly cover, so this section will focus only on those variations and advances most relevant for the rest of the thesis. Specifically, we will only consider feed-forward neural nets trained by back-propagation, along with some techniques for improving performance. For some recent, rigorous reviews of the deep learning literature, refer to LeCun, Bengio, and Hinton, 2015 [44]; and Schmidhuber, 2015 [65], both of which cover interesting subjects missing from this section.

2.3.1 Multilayer Perceptrons

The simplest incarnation of DNNs are multilayer perceptrons, or MLPs. MLPs have a long history, and is what we usually refer to when speaking about feed-forward neural networks. It is an extension of the simple linear two-layer perceptron, providing non-linearity in one or more hidden layers between the input and output layers. Normally only MLPs with more than one hidden layer are considered deep, but for simplicity we consider a network of a single hidden layer here, as illustrated in Fig. 2.3.

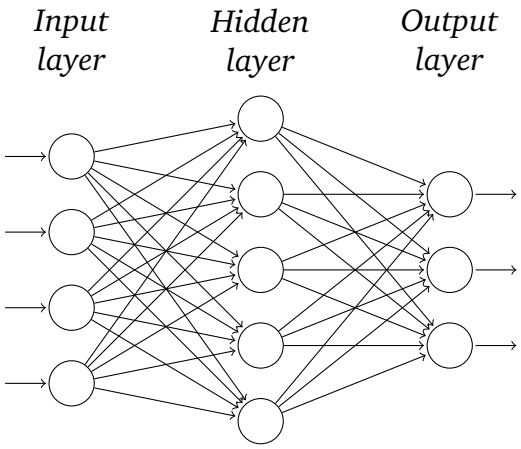


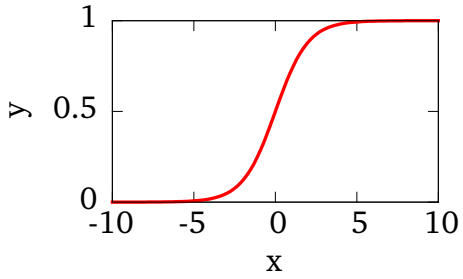
Fig. 2.3: A simple multilayer perceptron with one hidden layer.

The Forward Pass

The network computes the output values, a so-called forward pass, by (at each layer but the input layer) summing the inputs (which are weighted by the weight of the connection), then putting that sum through a non-linear transfer function. There are many different such functions used in the literature. Some of the most used such functions are:

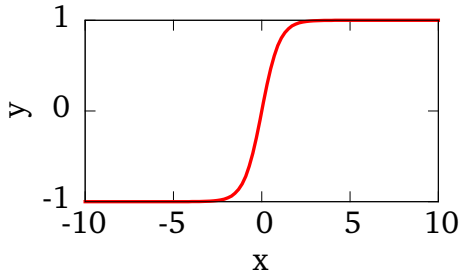
- *The logistic function*:

$$y = \frac{1}{1 + e^{-x}} \tag{2.17}$$



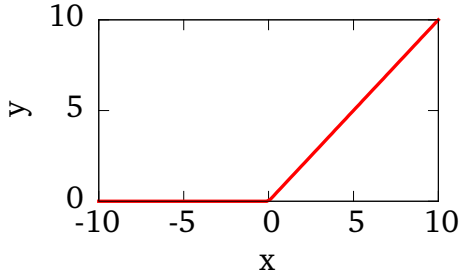
- *The hyperbolic tangent function*:

$$y = tanh(x) \tag{2.18}$$



- *The rectified linear function (ReLU)*:

$$y = \max(0, x) \tag{2.19}$$



- *The softmax function*:

$$y = \frac{e^x}{\sum_i^l e^{x_i}} \tag{2.20}$$

  where l is the layer the neuron is a part of, so the sum is over all the neurons in the layer. A bit impractical to plot, the softmax function has the effect of providing a probability distribution over a set of inputs, and is therefore often used in the output layer.

All of these functions have different characteristics, and the best choice is dependent on the training scheme and the problem domain.

The output from each node's transfer function is used as input for the next layer (or read off as output in the case of the output layer), and so on. Mathematically, we can describe this process for any hidden unit i in layer l as follows:

$$y_i^{l+1} = f(\boldsymbol{w}_i^{l+1}\boldsymbol{y}^l + b_i^{l+1}) \tag{2.21}$$

where $f(\cdot)$ is the transfer function, and $\boldsymbol{w}$ is the weight vector. b is a bias term, which can be learned along with the weights. It is usually seen as a input neuron for each layer that is always active, making certain activity possible even if the network's input vector is $\vec{0}$.

The Backward Pass

To learn to model some target concept, we need to gradually update the weights and biases – the parameters of the network – to correctly compute the target values from the input values. This is done by alternating between forward passes and *backward* passes, in which the parameters are adjusted to reduce the error between the values obtained from the forward pass and the target. These adjustments are done with the backpropagation algorithm. While describing the algorithm in detail is outside the scope of this thesis, it basically boils down to computing for each weight w_{ij} the following:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} \tag{2.22}$$

where Δw_{ij} is the change in the weight of the connection between neurons i and j, α is the learning rate, and E is the output error. In other words, we want to find that weight's contribution to the error, then counteract it. The error can be computed in different ways depending on the problem at

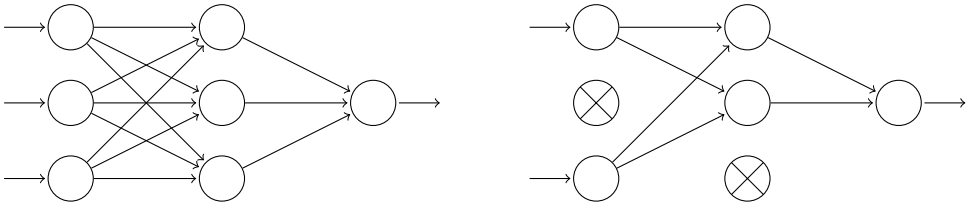
hand (*negative log likelihood* is popular for classification for example), but for illustrative purposes we can think of the simple error function $\frac{1}{2}(y - \hat{y})^2$ where y is the target value and $\hat{y}$ is the network's prediction, assuming a single output neuron. The computation of $\frac{\partial E}{\partial w_{ij}}$ is performed layer-wise from the output layer and backwards towards the input with the help of the chain rule, hence the name of the algorithm.

Backpropagation is a form of gradient descent optimization. In normal, non-stochastic gradient descent, update vectors are based on the full training set, while *stochastic* gradient descent updates based on randomly chosen samples from the training set (typically the whole set is used, but updates are computed for every example individually). In deep learning, small mini-batches – typically 5 to 20 examples to a batch – are used as a compromise between efficient training and reasonably averaged update vectors that do not oscillate too much on the error surface. This is called *batch learning*, and the size of these batches is a hyperparameter that needs to be set.

2.3.2 Dropout

One of the core ideas underlying the work described in this thesis is the concept of *dropout*, a regularization technique invented by Srivastava, Hinton et al. in 2012 [35, 75, 76]. Put simply, dropout means randomly "dropping" neurons when training. For each data instance that is put through the forward-backward passes of the previous section, some of the neurons in each layer are randomly nullified, so that they do not produce any output. The idea is that this prevents too much co-adaptation by neurons – they must be able to perform their function in the network without relying on the existence of others. When one wants to actually use the network for predictions, no neurons are dropped, and instead all the weights in the network are scaled inversely proportional to the dropout rate, to avoid over-saturation. Srivastava et al. were able to show that dropout greatly reduces overfitting, and offers major improvements over other methods of regularization. An illustration of a dropout network can be seen in Fig. 2.4, which clearly shows the concept.

Mathematically, each neuron has some probability p of being dropped, using a Bernoulli distribution. That is, a neuron is either dropped or not dropped, never partly dropped (introducing other types of noise than Bernoulli-noise has been done, such as in the context of denoising autoencoders [80], but



(a) Standard network.

(b) Two nodes have been dropped, and no longer contribute to the network.

Fig. 2.4: A dropout network, before and after randomly dropping neurons.

this is not considered dropout). p is often set differently for the input layer and the hidden layers – Srivatstava et al. finds that, respectively, $p = 0.2$ and $p = 0.5$ works well for a wide range of problems. Neurons in the output layer are never dropped, for obvious reasons. Revisiting Eq. 2.21 then, the formula for neuron output during training becomes:

$$y_i^{l+1} = f(\boldsymbol{w}_i^{l+1}(\boldsymbol{r}^{p^l} * \boldsymbol{y}^l) + b_i^{l+1}) \tag{2.23}$$

where $\boldsymbol{r}^{p^l}$ is a binary vector sampled from a Bernoulli-distribution with the probability p set for layer l. During prediction, the full network is used and the weights are scaled according to the probability of dropout in that layer, leading to the following output calculation:

$$y_i^{l+1} = f((\boldsymbol{w}_i^{l+1} * (1 - p^{l+1}))\boldsymbol{y}^l + b_i^{l+1}) \tag{2.24}$$

This makes intuitive sense: If the dropout probability is high, fewer connections are involved in each computation and they are therefore stronger. The full weight vector must then be scaled down significantly to avoid oversaturation, and vice versa.

An important realization, especially for understanding the novel ideas in this thesis, is that what really happens when dropout is applied to a network is that a different, thinned network is sampled each time, from an exponential number of possible networks. In fact, Srivastava et al. claim that their proposed method for prediction – using the whole network with scaled weights – is simply an approximation of the geometric mean of the predictions of an exponential number of different networks, and they are able to show that taking more than a certain number of prediction samples from dropped out networks and computing their geometric mean outperforms the approximation (although at a cost of increased computational complexity).

Max-norm Regularization

While virtually any regularization technique (like *L1* and *L2* for example) can be combined with dropout, Srivastava et al. found that max-norm regularization, previously used in collaborative filtering [74], was especially effective in conjunction with dropout. It works by constraining the norm of the incoming weights for every neuron i by some upper bound c:

$$\forall i \ \|\boldsymbol{w}_i\|_2 \leq c \tag{2.25}$$

If this constraint is violated, the weights are scaled to have a norm of exactly c. This technique allows a initially high decaying learning rate without the weight mangitudes "blowing up", but brings significant improvement even when the learning rate is static.

2.3.3 Unsupervised Pre-Training

Unsupervised learning is an umbrella term for mechanisms where the network's weights are trained on the available training data, X – without considering target values, Y. The idea is that after this unsupervised step, the weights will somehow represent the structure of X better than random initialization can. Unsupervised learning has many uses, but in the context of unsupervised *pre-training*, normal supervised learning with backpropagation on the $\langle X, Y \rangle$ pairs is performed after the unsupervised training. This last step is referred to as *fine-tuning*.

It is conjectured that one of the central problems that makes training deep neural networks difficult is the strong dependencies between the weights of different layers [23]. Recall that the backpropagation algorithm works by taking the partial derivative of the error with respect to the connection weight being updated. The gradient of the objective function is then a strictly local measure, and single weights are always just adjusted to reduce the error *given that the rest of the weights are not changed*. With such a local approach, the network may be prone to get stuck in local minima close to the point in weight space where the random initialization of weights place it [6]. This is one of the areas in which unsupervised pre-training is believed to help, by providing an initialization point for the network that restricts the search to a region of weight space that corresponds to the structure of the available data. Additionally, pre-training increases the weights' magnitude and thus

also the non-linearity of the network. This makes the local error surface more complex, which has a regularization effect [23].

Restricted Boltzmann Machines

Initially named "Harmonium" by their inventor (Smolensky, 1986 [73]), Restricted Boltzmann Machines (RBMs) are able to learn a probability distribution over its inputs. They are a special type of neural networks that contains only two layers of neurons. These layers make up a symmetric bipartite graph, that is, there are no intralayer connections (which is the restriction referenced in the name), and all visible neurons are connected to all hidden neurons, as illustrated in Fig. 2.5. The connections are bidirectional,

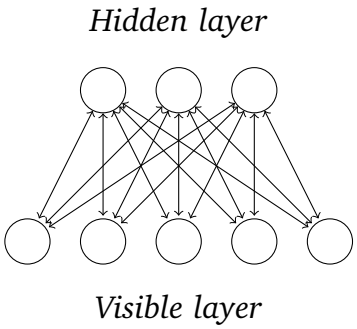


Fig. 2.5: The basic architecture of an RBM.

so the weight of the connection from some visible neuron i to some hidden neuron j is the same as that from j to i. Neuron activations are calculated in a similar manner as for the regular neural networks discussed earlier (see Eq. 2.21), but with an important distinction: RBMs usually uses stochastic binary neurons, that is, the output of a single neuron is either 0 or 1. To calculate the output of some neuron i, first calculate its *activation energy*:

$$a_i = \sum_j w_{ij} y_j + b_i \tag{2.26}$$

where w_{ij} is the weight of the connection between neurons i and j, y_j is the output of neuron j, and b_i is a bias term. Then put a_i through the logistic function:

$$p_i = \sigma(a_i) \tag{2.27}$$

The activation of neuron i is then decided stochastically:

$$y_i = \begin{cases} 1, & \text{with probability } p_i \\ 0, & \text{otherwise} \end{cases} \tag{2.28}$$

The connection weights are trained with *contrastive divergence* [33], an approximation of gradient descent. Training works roughly as follows assuming binary inputs and randomly initialized weights:

1. Set the states of the visible neurons to mirror the example to train on. One visible unit per dimension in the data.

2. Update hidden neuron states according to Eqs. 2.26, 2.27, and 2.28.

3. For each connection e_{ij}, compute $Positive(e_{ij}) = y_i y_j$, which is 1 if both neurons i and j are activated, else it is 0.

4. "Reconstruct" the visual layer by computing the activations of all the visual neurons like in step 2, only this time the opposite way.

5. For each connection e_{ij}, compute $Negative(e_{ij}) = y_i y_j$.

6. Then, the weight change becomes:

$$\Delta w_{ij} = \alpha(Positive(e_{ij}) - Negative(e_{ij})) \tag{2.29}$$

   with learning rate α. This update rule increases the weight if the hidden unit is activated and "helped" in a correct reconstruction, and decreases the weight if it helped in an erroneous reconstruction.

7. Repeat steps 1 - 6 for every training example until convergence or some other stopping criteria.

One such cycle from visual neurons to hidden neurons to reconstruction of the visual neurons is called a *Gibbs step*, a single iteration of a process called *Gibbs sampling* [25] from the field of statistics. Contrastive divergence can be done with multiple such steps, although a single step turns out to work well in practice. See Fig. 2.6 for an illustration.

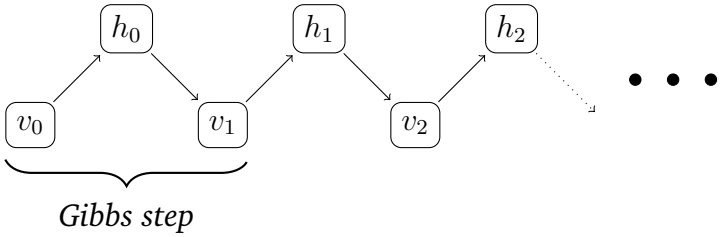


Fig. 2.6: The contrastive divergence Markov chain. The state of the hidden layer at $t = 2$ (h_2) depends on the state of the visual layer at $t = 2$ (v_2), which depends on h_1, and so on. A Gibbs step corresponds to one cycle from visual layer to hidden layer to reconstruction of the visual layer.

Having only two layers, RBMs by themselves are not directly useful as a deep learning pre-training technique, but it is an interesting energy-based model with a wide range of uses.

Deep Belief Networks

In 2006, Hinton et al. [34] proposed a way to use and efficiently train RBMs for deep learning purposes, the *Deep Belief Network* (DBN). A breakthrough in efficient pre-training of deep networks, DBNs are trained in a greedy, layer-wise manner. The core component is a stack of RBMs. The weights learned during this greedy process are used as the initial weights for a normal feed-forward network, ready to be fine-tuned. Training a DBN is roughly done as follows:

1. Train the initial RBM, consisting of visual layer v^0 and hidden layer h^0, on the training data as previously described. One visual-layer neuron per dimension in the data, and as many hidden neurons as wanted.

2. Generate a second-level representation of the data, by taking either samples from h^0 activations, or the mean activations of h^0, after one or more Gibbs steps.

3. Train the second RBM, using the chosen representation as training data.

4. Repeat steps 2 - 3 until the desired number of layers are obtained.

5. When using the DBN for semi-supervised learning, add a final output layer of neurons (with weights randomly initialized), and use normal

back-propagation with a training signal to fine-tune the weights until desired performance or some other stopping criteria.

The general idea of this training procedure is illustrated in Fig. 2.7.

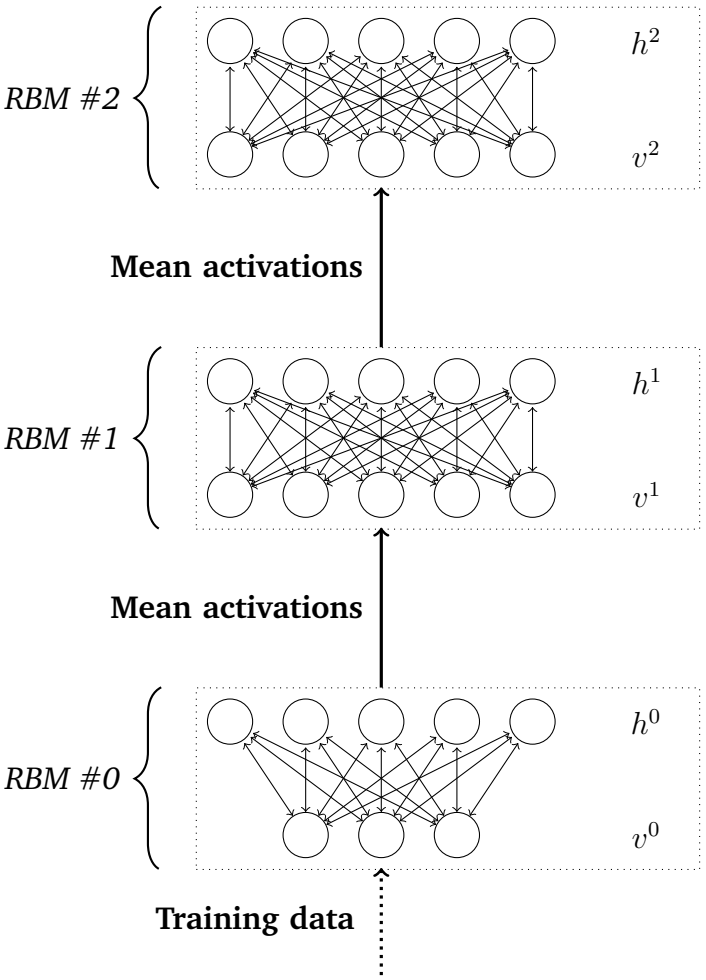


Fig. 2.7: Training a DBN with three hidden layers. First *RBM #0* is trained on the training data. Mean activations from h^0 in that RBM are propagated to *RBM #1*, which trains on those activations, and so on. The weight matrices learned are then used for the hidden layers of a normal MLP for fine-tuning.

2.3.4 Summary

Deep learning is a field with an enormous amount of research activity. This is probably due to a number of factors, such as the very impressive results seen in the last decade, and the heavy involvement from industry giants. Thus, this section does not in any way cover the entirety of the field, but provides an introduction to the core concepts, along with a slightly more

detailed overview of the specific advances and ideas that acts as part of the foundation of the innovations presented in this thesis. Assuming the core concepts of backpropagation learning and deep networks in general are understood, the deceptively simple idea of dropout is the most important to grasp in this context: The idea that a single network can be seen as a collection of exponentially many different networks is extremely powerful and is a crucial component of the ideas presented later in this thesis.

Unsupervised pre-training is also a powerful idea, but for the rest of this thesis it suffices to understand *what* it does, but not necessarily understanding *how* it works. The reason for this is that it is a perfect fit for *combining* with active learning as it can make use of unlabeled examples, but it is in this case only a tool that can increase performance. Only DBNs (and their building blocks, RBMs) were covered here, since it is the only pre-training technique used in this project. Several other options, such as autoencoders [55] do exist however.

2.4 Related Work

While active learning has a long history, especially when you consider its roots in optimal experiment design, not much work has been done on combining it with deep learning. Deep learning also has a long history, but its current popularity is a rather recent development, which may partly explain the lack of research on combining the two. However, most research on active learning with neural networks in general, that is if we include shallow nets, is still applicable to deep networks. Some approaches, like the variance estimation of Cohn, 1994 [14] or the $\mathcal{SG}$-based QBD with neural nets of Cohn et al., 1994 [13] may unfortunately be too computationally complex for large scale application. For classification, however, neural networks are able to present a probability distribution over all the possible labellings through use of a *soft-max* output layer, and hence the uncertainty sampling methods presented in Sec. 2.2.2 are all applicable. These methods have been shown to outperform random selection when used with deep networks (Wang and Shang, 2014 [84], experimenting on the MNIST dataset; and Zhou et al., 2010 [88] in the context of sentiment classification).

As to regression, Krogh and Vedelsby, 1995 [42] considered committees of neural networks for active learning in regression problems. Using a mea-

sure of *ambiguity* for each data point, $a(x)$, based on the variance between committee predictions similar to that mentioned under QBC in Sec. 2.2.2:

$$a(x) = \sum_{h}^{\mathcal{C}} (h(x) - h_{\mathcal{C}}(x))^2 \tag{2.30}$$

where the mean prediction $h_{\mathcal{C}}(x)$ is computed like in Eq. 2.15. They were able to show that this QBC-heuristic outperformed random selection when trying to approximate the univariate square wave function. The disagreement among the committee in this study came simply from the random initializations of the networks. RayChaudhuri and Hamey, 1995 [57] extended this work by training each network on a random sample of the available data to promote more diversity in the committee. For a short review of the literature on using QBC for active learning in regression problems, see Burbidge et al., 2007 [11]. Using ensembles of neural networks in *non-active* scenarios has also shown to be effective, e.g. for handwritten digit recognition [51].

To summarize, while not much work has been done on combining active learning and deep learning specifically, much more has been done on active learning and neural networks in general, from many different perspectives. Importantly, the approaches considered in this project, uncertainty sampling and committee-based methods, have both been used with success in combination with neural models.

3 Methods

3.1 Overview

This chapter will describe the proposed architecture and its implementation, and the experiments conducted to attempt to answer the research questions of Section 1.2. First, Section 3.2 describes in detail the main novel idea of the project – the Active Deep Dropout networks, as well as its various hyperparameters and options, and possible variations. The dataset used is introduced in Section 3.3, and the design of the experiments performed on this dataset in Section 3.4. Finally, Section 3.5 describes the specifics of the implementation.

3.2 Active Deep Dropout Networks

Unsupervised pre-training methods facilitates use of unlabeled instances along with the labeled set – the unlabeled dataset used in the pre-training step can very well be a superset of the set of labeled data points used in the fine-tuning. By combining active and deep learning, the goal is to allow successful training on even fewer labels than what the semi-supervised learning of unsupervised pre-training allows. Here, we propose a new method for combining these two fields, potentially making active learning applicable in a wider range of deep learning problems than previously feasible.

The core idea presented and explored in this thesis is a novel approach to using the QBC algorithm (described in Alg. 3) with deep neural networks, where the committee is composed of n samples from the exponential number of networks that arise from applying dropout to a single network. We call these committees of sampled networks used to guide active learning *Active Deep Dropout* networks, or ADD-nets.

Since the number of different networks that can be generated by applying dropout is enormously large – many orders of magnitude larger than any plausible dataset – the chance of any one such network being trained on more than one instance is minuscule, and the vast majority will not be explicitly

trained at all. Hopefully then, these networks will provide diverse enough predictions, despite the massive weight sharing, that the QBC algorithm can perform well and select informative examples to label.

3.2.1 Generating Networks

When using ADD-nets for active selection, the first step is to generate the networks. The source from which they are all generated is a MLP which has been trained on the examples that exist in $\mathcal{L}$ so far. In this step, dropout (as described in Section 2.3.2), is used. Applying dropout to the full MLP yields a network with the same number of output neurons (which is crucial, as disagreement in the committee will be deceptively high if output neurons are dropped) as the MLP, but with some percentage of the hidden and input neurons removed from the equation. This network is then one member of the committee.

Since dropout is stochastic, the same process can just be repeated for every number of desired networks in the committee. As already noted, dropout exposes an exponential number of possible networks from a single MLP, so this process is basically just Monte-Carlo-sampling the space of possible networks.

This process of generating networks is illustrated in Fig. 3.1.

3.2.2 Scoring

After the committee has been constructed, the unlabeled instances in $\mathcal{U}$ can be scored. This happens by sequentially feeding each instance to be scored to each network in the committee, then using some QBC heuristic to evaluate disagreement between the predictions of the committee members. The predictions of member in the committee will be different to some degree from its "parent" MLP, and its "sister" networks derived from the same process since they have different topologies. When using pool-based sampling as is the case here, it is important that each example in $\mathcal{U}$ is evaluated by the *same* committee – disagreement values stemming from different committees cannot necessarily be directly compared.

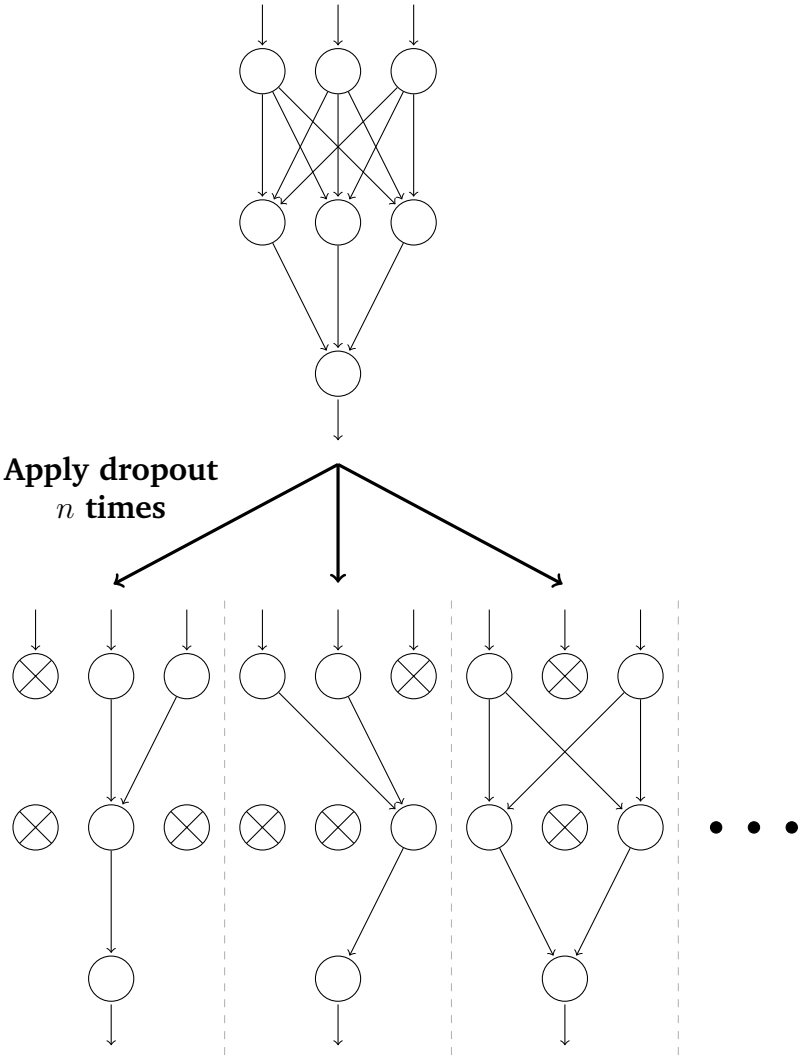


Fig. 3.1: Generating ADD-networks. Dropout is applied to a full MLP n times, generating n distinct networks that together make up a committee.

An illustration of using ADD-nets to score a single instance can be seen in Fig. 3.2.

3.2.3 The Full Architecture

Putting the pieces together into the full active learning algorithm is now trivial: Periodically during normal supervised training of the MLP, generate a dropout committee, evaluate all examples in $\mathcal{U}$, and move the most promising to $\mathcal{L}$. The full algorithm, derived from Alg. 3, is specified in Alg. 4. As we can see, aside from the committee creation, QBC for ADD-nets works just like normal QBC, and puts no restrictions on the types of evaluation measures used, which has to be selected based on the nature of the problem at hand.

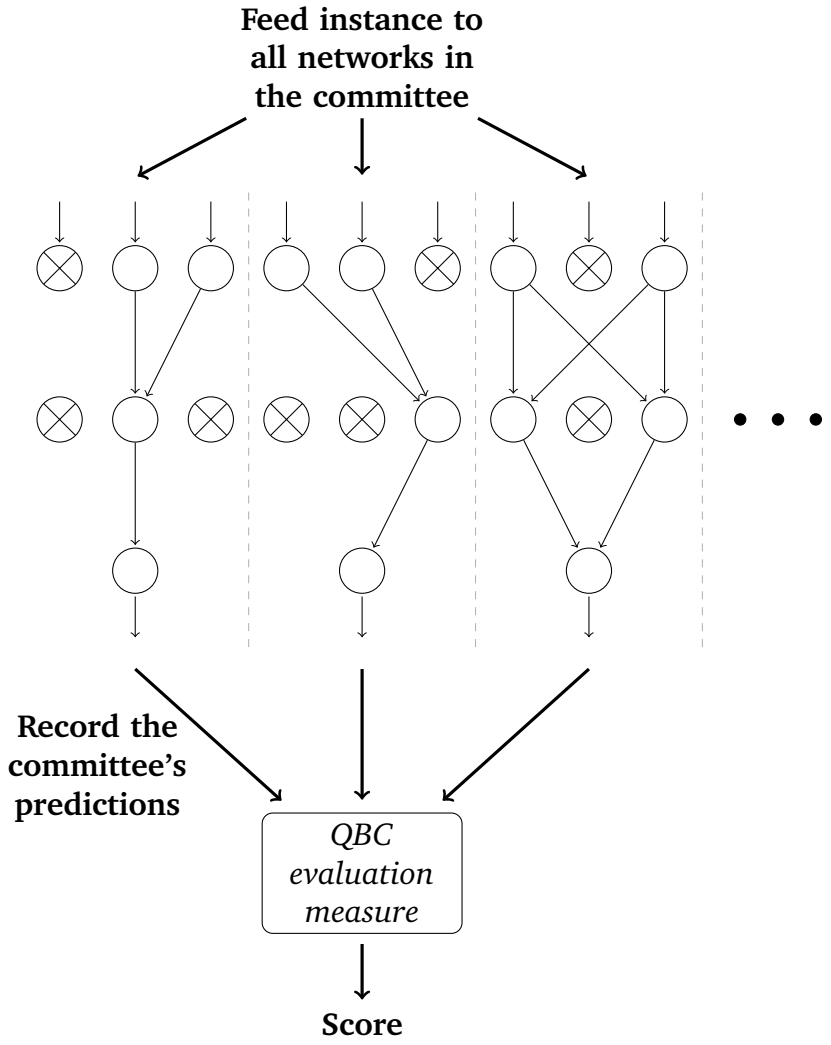


Fig. 3.2: Evaluating instances with ADD-networks. First, the instance is fed to all dropout-networks in the committee, then the committee's predictions are evaluated for disagreement by some QBC heuristic.

To get a better understanding of how the ADD-networks fit in with the larger picture, it can be useful to see its place in the full architecture, illustrated in Fig. 3.3. Here, we can clearly see how ADD-networks augments the normal semi-supervised training process.

Importantly, note that while it is conceptually simpler to view the committee creation and scoring processes as separate, there is a more computationally efficient way to perform these calculations: Since the MLP is trained with dropout during the supervised phase, the machinery to compute a forward pass through the network with dropped out neurons is already in place. Simply running one such forward pass for each member of the committee (the dropped out neurons will obviously be different each time) and recording

Algorithm 4 Query by Committee with Active Deep Dropout networks.

$\mathcal{M} \leftarrow$ the full network
$\mathcal{L} \subset \mathcal{U}$ ▷ $\mathcal{L}$ is initialized as some nonempty labeled subset of $\mathcal{U}$
$train(\mathcal{M}, \mathcal{L})$
while not finished training **do**
    $highestDisagreement \leftarrow -\infty$
    for all $x \in \mathcal{U}$ **do**
        $\mathcal{C} \leftarrow \emptyset$
        for n iterations **do** ▷ Sample n dropout networks from $\mathcal{M}$
            $\mathcal{C} \leftarrow \mathcal{C} \cup dropout(\mathcal{M})$
        end for
        $predictions \leftarrow \emptyset$
        for all $h \in \mathcal{C}$ **do**
            $predictions \leftarrow predictions \cup h(x)$
        end for
        $disagreement \leftarrow evaluateDisagreement(predictions)$
        if $disagreement > highestDisagreement$ **then**
            $highestDisagreementExample \leftarrow x$
            $highestDisagreement \leftarrow disagreement$
        end if
    end for
    $y \leftarrow query(highestDisagreementExample)$
    $\mathcal{L} = \mathcal{L} \cup \langle highestDisagreementExample, y \rangle$
    $train(\mathcal{M}, \mathcal{L})$
end while

the results. This way the need to keep $n + 1$ large networks in memory while doing the evaluations is avoided. Without this optimization, ADD-networks would require the same amount of memory as keeping an ensemble of n different networks as a committee (but would still be computationally simpler, as only one network would need to be trained).

3.2.4 Hyperparameters and Options

When training deep neural networks with dropout, there is already a large amount of hyperparameters and options to set, the most important of which are detailed in Table 3.1. There are additional parameters when taking different regularization methods into account, but these will not be considered here as only max-norm regularization is used in the experiments.

When doing active learning with ADD-nets, an additional set of choices must be made, shown in Table 3.2.

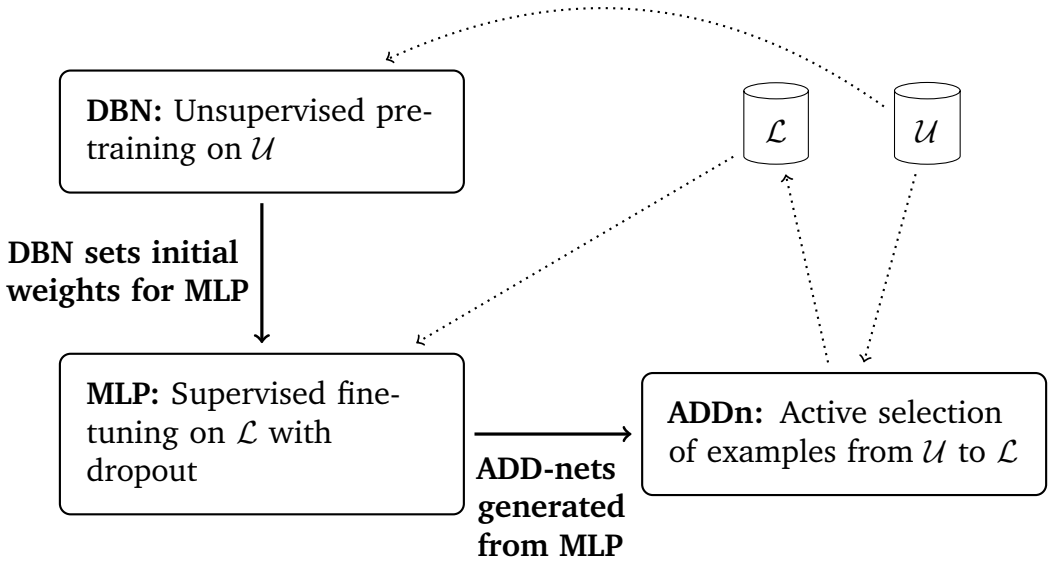


Fig. 3.3: An overview of the implemented architecture and how the components depend on each other. Initially, a DBN is trained on $\mathcal{U}$ that contains all the examples. $\mathcal{L}$ is bootstrapped by labeling some small, fixed percentage of the examples in $\mathcal{U}$, before the fine-tuning process begins. This is performed with a normal MLP training on $\mathcal{L}$ with dropout. Periodically, ADD-networks are generated from the MLP, which evaluates the examples in $\mathcal{U}$. The examples deemed most informative are queried, and moved from $\mathcal{U}$ to $\mathcal{L}$, before fine-tuning the MLP is resumed on (the now greater) $\mathcal{L}$.

3.2.5 The Unbalanced Batch Problem

When combining batch learning of DNNs with active learning, a problem arises: When active selection is performed, and one or more instances from the unlabeled set is queried and added to the labeled set, we can potentially get uneven batches. That is, the last batch in the training set may be of a different size than the rest. This problem is illustrated in Fig. 3.4.

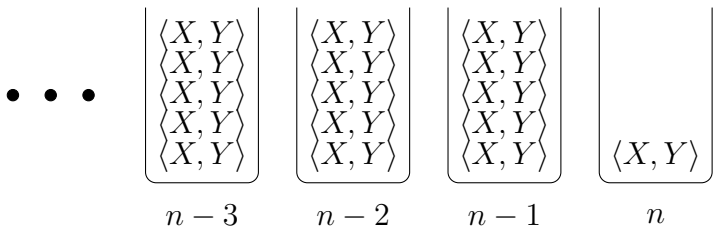


Fig. 3.4: The unbalanced batch problem. This figure shows the last four batches of training examples in some training set, with a batch size of 5. Because of active selection, the last batch only has a single example in it, which will get disproportionally weighted in training.

Table 3.1: General hyperparameters and options when training deep neural networks with dropout.

PARAMETER	DESCRIPTION
Layers	How many hidden layers and the number of neurons in each one.
Transfer functions	The transfer functions used in each layer.
p	The probability of a single neuron dropping out, one value per layer. Typically set to 0.5 for hidden layers and 0.2 for the input layer.
Cost function	The cost function to minimize when training. Often *negative log likelihood* (NLL) or *root mean square error* (RMSE).
α	The learning rate. Must usually be set through empirical testing, but less than 1.
b	The batch size. Usually set in the 5 to 20 range. Optimal values for a certain context must be found empirically.
# of epochs	The number of epochs to train for (an often used regularization technique in normal deep learning is *early stopping*, where training is terminated before the max number of epochs is reached if performance is not improving to avoid overfitting, but this is not done here).
c	The max norm of the incident weights on any neuron, used in max-norm regularization as described in Sec. 2.3.
k	The number of Gibbs steps to take when doing contrastive divergence for training RBMs. A value of 1 usually works well in practice.

Depending on how the copying is implemented, this might theoretically create an issue where the newly added example(s) are given disproportionate weight in training, compared to the rest of $\mathcal{L}$, since the error gradient of that batch will only be based on a single example. In practice this will probably not mean much, and in some instances might even be beneficial. However,

Table 3.2: Set of additional hyperparameters and options when doing active learning with ADD-nets.

PARAMETER	DESCRIPTION
QBC evaluation measure	What heuristic to use with QBC to go from n output vectors to a scalar that signifies level of disagreement in the committee.
n	Number of models in the committee. In the case of ADD-nets, this refers to the number of dropout samples taken.
Initial size of $\mathcal{L}$	The number of labeled examples with which $\mathcal{L}$ is bootstrapped, by randomly sampling this many examples from $\mathcal{U}$ which are then labeled.
s	The number of unlabeled examples selected for querying and addition to $\mathcal{L}$ at a time. This is a computational trade-off, and the optimal value depends greatly on the cost of labeling.
Selection interval	The number of epochs to train for between each active selection. This can also be determined dynamically, if one for example uses early stopping.

for our purposes we want to avoid another variable in testing, and hence avoid this problem. There are multiple ways to achieve this:

- *Variable Batch Sizing*. One obvious idea is to vary the batch size after each round of active selection such that all actual batches are of equal size. This might bring other unforeseen problems however, and depending on s is not even always possible to do perfectly.

- *Learning Rate Scaling*. Instead of varying batch sizes, simply scale the learning rate according to the actual number of examples in the last training batch. For example if the batch size is 10 and $s = 1$, after the first selection the last batch will only contain one instance (assuming $\mathcal{L}$ was bootstrapped with a number of instances divisible by the batch size). To avoid this instance being disproportionally weighted then, we can simply scale the learning rate for this specific batch by the ratio of instances in it to the normal batch size.

- $s = b$. The simplest, and therefore the solution chosen here is simply setting s to be equal to the batch size. As long as $\mathcal{L}$ was bootstrapped to a size divisible by b, there will never be variable sized batches.

While the simple $s = b$ option was chosen for its simplicity here, it might not be the best choice in all contexts. If the cost of labeling is extremely high for example, one could prefer to have a very small s and (re)train to convergence after every active selection, in which case this approach might not be optimal.

3.2.6 Variations

There are many possible variations to the use of ADD-networks that are not mentioned here. The core idea is simply the use of dropout networks for evaluation, which is applicable in many scenarios. For one, ADD-networks can be applied in all the main different active learning scenarios, both the de novo generation and stream-based sampling approaches described in Section 2.2.1 for example. While the pool-based approach was chosen for simplicity in testing here, there is nothing tying ADD-nets to this scenario specifically. Furthermore, using a DBN for pre-training was a rather arbitrary choice in this case, chosen simply for being relatively well understood, being relatively simple to implement, and most importantly because of its proven effectiveness on the dataset used. But, the pre-training step is not connected to the use of ADD-networks in any way, and is simply used for increasing final performance. Any other pre-training technique can be used – or even dropping pre-training altogether, for example in online scenarios where pools of unlabeled data are not available. To summarize, ADD-nets can be used in any active learning scenario where neural networks are used as a model.

3.2.7 Summary

To summarize, ADD-networks are based on a rather simple concept, but have some interesting and desirable features. ADD-nets alleviate some of the problems with using QBC with deep networks, namely the need to keep and train multiple networks, which is expensive in both memory and computational resources. Additionally, no two sampled networks have the same topography, nor are any of them explicitly trained on the same examples, providing variety in the models in the committee without having to resort

to techniques like e.g. *bagging* or *boosting*. Avoiding this is beneficial when working with few labeled instances, because many ensemble techniques require splitting the labeled set (which in the case of active learning is already small) among models to get enough diversity in the ensemble. With ADD-nets however, all networks benefit indirectly from training on every labeled instance, through weight sharing. As described in the preceding section, they are applicable in most active learning scenarios, as long as neural networks are used as models.

The ADD-networks approach does however have some drawbacks. Compared to uncertainty sampling methods it is more computationally taxing, especially if a large committee size is required. Additionally, it introduces quite a few more hyperparameters that must be set and optimized, although this is a problem that most active learning methods have.

3.3 The MNIST Dataset

For testing the proposed architecture, the MNIST handwritten digit dataset [46] was used. It consists of a training set of 60,000 images of digits, with another 10,000 in a testing set, that are accompanied by correct labels – in the range 0 - 9 – for each example. All the digits are normalized on size, and are centered in an image with 28×28 pixels. Each of the 784 dimensions have a scalar value between 0 and 1, signifying the intensity at that pixel. Some examples of random samples from MNIST can be seen in Fig. 3.5. The dataset is designed for classification, where the task is to correctly identify a handwritten digit with its symbolic counterpart.

MNIST was chosen for multiple reasons. Most importantly, it has enough dimensions and complexity and little enough noise to be a good fit for deep learning. Earlier research on modeling MNIST with neural networks has shown (see LeCun's MNIST resources for a comprehensive list [46]) that efficiency is significantly increased with multiple layers of hidden layers, as in many other image recognition tasks. This makes intuitive sense, and is biologically plausible, considering the human visual system's hierarchical nature. Another motivating factor for choosing MNIST is its widespread use as a benchmark in the deep learning literature, making comparisons to and reproductions of other research simple. Finally, while the dataset is complex enough for deep learning to make sense, it is still small and simple

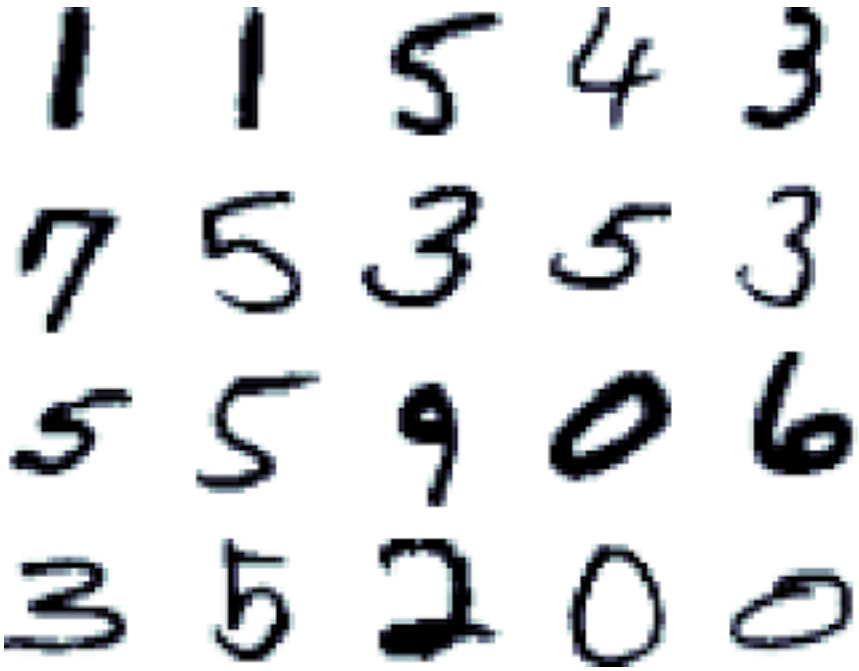


Fig. 3.5: Examples of instances in the MNIST handwritten digit dataset. There are clear differences in style, thickness, skewing, etcetera, making prediction a non-trivial problem. A successful model must be able to isolate the spatial features that make up certain digits, and cannot rely on certain pixels having certain values.

enough to be relatively computationally cheap to work with, compared to other staple deep learning benchmarks like ImageNet [20] and others. Since the computational resources available to this project was rather limited, this was a crucial aspect that allowed lots of experimenting.

MNIST is not an active learning dataset in the sense that labels are unavailable, on the contrary, MNIST supplies labels for each example. For active learning then, these labels are held back during training, until the learner queries for them, in which case the labels are supplied instantaneously. That labeling is instantaneous is obviously unrealistic, as one of the core assumptions of active learning is that labeling is expensive. This is not needed for showing that an active learning strategy works however, and having labels already available makes experimenting much easier. Furthermore, classifying handwritten digits is a problem domain where it could be realistic to use active learning, as labeling needs a human oracle (overlooking the fact that the exact task of classifying handwritten digits is mostly a solved problem, and large datasets with labeled examples are available).

3.4 Experiment Design

To test the proposed architecture's performance, a series of experiments are performed. Before testing different active strategies, three baselines are established:

- *Supervised, non-active learning*. That is, with all the examples and all the labels. This baseline should be impossible to beat with an active strategy that only uses a small percentage of the labels, but it provides a floor-value for which to strive, and is a valuable comparison.

- *Random selection*. The classic baseline in active learning, included as a "ceiling", an upper limit. Obviously, if an active learning strategy cannot beat random selection, it is useless. It is important to note that it is possible that an active strategy that is very efficient on one dataset is unable to beat random selection in another, due to the specifics of how the dataset is distributed.

- *Supervised, non-active learning with reduced, randomized* $\mathcal{L}$. During an experiment, an *active* strategy will select a total of n examples. $\mathcal{L}$ is bootstrapped with m examples before training starts, so the total number of examples in $\mathcal{L}$ after training has ended is $m+n$. This baseline is the same as the first listed (i.e. non-active), but instead of training on the full dataset, it is only given $m + n$ randomly sampled examples. In other words, it trains on the same number of examples as the active strategies do, but does so from the first epoch and never adds new examples to $\mathcal{L}$. If an active strategy can beat random selection, but not this baseline, it shows that the cost in efficiency of gradually adding examples instead of starting with them is higher than what the active strategy is adding. Note that this can be remedied by *retraining* the network from scratch either between each selection, or after the active training is finished.

For each different strategy, most hyperparameters are kept constant – only active selectors and selector hyperparameters are varied – to keep the comparisons fair and accurate, and have as few variables as possible. The constant hyperparameters were set to values based on what has been proven to work in the supervised non-active scenarios in earlier work. To simplify experiments and try to further eliminate the number of variables, only dropout

and max-norm regularization is used, as dropout is crucial to ADD-networks, and max-norm is crucial to dropout. Removing these two regularization methods from the baselines – and other active selectors used as comparison – would unfairly skew results towards being more favorable for ADD-networks, because of the extreme effectiveness of dropout.

To mitigate statistical anomalies leading to incorrect results, all experiments were performed five times with identical settings, and the data averaged across all runs, unless explicitly stated otherwise.

It is important to note that the main goal with the experiments performed at this stage is to prove the feasibility of ADD-networks as an active strategy, and not a lot of time was spent maximizing performance and optimizing parameters. The reason for this is that most of the alloted time for the project was spent implementing and tuning the core ideas themselves. Ideas for possible improvements and additions that may improve performance or applicability are discussed in Section 5.4.

3.4.1 Hyperparameters

For testing ADD-nets on classification, the normal MNIST digit prediction task described in Section 3.3 was used. 10,000 of the examples in the training set are removed and used as a validation set, meaning the maximum number of training examples is 50,000. For all tested methods – active and non-active – the core hyperparameters were set to the same, and DBN was used for pre-training in all cases. The hyperparameter values are listed in Table 3.3. For an explanation of the parameters, refer to Table 3.1 in Section 3.2.4. All of these settings are fairly standard, apart from the number of epochs used – in a normal supervised learning scenario, the network can usually converge to a good performing solution in less than 100 epochs. With active learning however, we start with very few examples, and periodically add new examples throughout the learning process. The network must get time to readjust between each active selection. Normally, the network would badly overfit the training data if allowed to train for so long, but dropout alleviates this problem so that the error on the validation set simply tapers off instead of starting to increase after some number of epochs have passed.

For all the active strategies compared, all the common active learning-specific parameters were kept equal as well, as shown in Table 3.4.

Table 3.3: Core hyperparameters used for MNIST classification task.

PARAMETER	VALUE / SETTING
Layers	784 - 1000 - 1000 - 1000 - 10
Transfer functions	$\tanh$ for hidden layers, $softmax$ for the output layer.
p	0.2 for the input layer, and 0.5 for the hidden layers.
Cost function	Negative log likelihood
α	0.1
b	10
# of epochs	30,000
c	15
k	1

Table 3.4: Active learning-specific hyperparameters shared between all active strategies tested.

PARAMETER	VALUE
Initial size of $\mathcal{L}$	240
s	10
Selection interval	100 epochs

With these numbers, since training is done for 30,000 epochs, the total number of examples in $\mathcal{L}$ by the end of (active) training is:

$$240 + (30000/100) \times 10 = 3240 \tag{3.1}$$

or in other words,

$$(3240/50000) \times 100 \approx 6.48\% \tag{3.2}$$

of the full training set is labeled.

Apart from the random selection baseline already mentioned, the active strategies tested here are the uncertainty sampling based *Shannon entropy* strategy described in Section 2.2.2, and, most importantly, the ADD-networks. Two different QBC disagreement measures are used for scoring instances with the ADD-networks:

- the *Kullbach-Leibler divergence* measure,
- and the *Euclidean mean distance* variance-based measure.

These are also both described in Section 2.2.2. Output Shannon entropy takes no additional hyperparameters than those mentioned in Tables 3.3 and 3.4 (recall that it simply scores examples by the entropy of the network's predictions on that example). ADD-networks take an additional parameter however, the committee size. The values used in the experiments are shown in Table 3.5.

Table 3.5: Committee-sizes used for experiments with ADD-networks.

PARAMETER	VALUES USED
n	2 / 10 / 30 / 50

3.5 Implementation Details

The implementation of the ADD-nets, other active strategies for comparison, and their common normal supervised learning infrastructure, was done in the *Python* programming language [60]. The numerical computation libraries *Theano* [7, 4] and *NumPy* [83] were heavily relied upon. Theano makes it possible to compile computational graphs down to code that can be run in parallel on the many cores of GPUs, which is critical for making the implemented architecture computationally feasible. The experiments were performed on machines with Nvidia GTX 480 GPUs at the Department of Computer and Information Science at the Norwegian University of Science and Technology.

The code is publicly available at `http://github.com/martingms/dan`, and licensed under the *MIT* license, a copy of which is supplied with the source code. Some of the code, most notably the RBM implementation, is di-

rectly borrowed from the Theano tutorials at `http://deeplearning.net/tutorial/` as allowed by the license, and much of the model implementation is inspired by the same. The license for the Theano code used is included where appropriate and required.

4 Results and Analysis

4.1 Overview

In the following sections, the results of the experiments outlined in Section 3.4 will be presented and analyzed. First, Section 4.2 will present the core results, with analysis. Then, Section 4.3 will discuss how performance relates to the committee size used, before Section 4.5 will discuss how prediction variance changes through training.

4.2 Core Results

The final results of the MNIST classification experiments can be seen in Table 4.1, and plots of the change in validation error across epochs can be seen in Figs. 4.1 and 4.2. The first plot (Fig. 4.1) contains all the experiments with ADD-nets. The second (Fig. 4.2) contains the baselines, the output Shannon entropy, and the best and worst experiments with ADD-nets. The results were split across the two plots to reduce clutter.

We can clearly see that the baselines we described in Section 3.4 provide the bounds we predicted, namely that the supervised approach, with all examples and labels available, has the lowest errors, while the supervised approach with only 3240 examples available and the active random strategy has the highest error. That the 3240 example supervised approach and the random selection has approximately equal final errors show that the selection interval (see Table 3.4) has been set high enough: The network is allowed to fully adjust to the newly selected examples added to $\mathcal{L}$ before a new batch is selected – in other words, the supervised training process is allowed to "catch up".

Of the active strategies tested, the output Shannon entropy strategy does best in this case, with an average error on the test set of about 0.25 percentage points lower than ADD-nets with a committee size of 50. Relatively however, ADD-nets are not far behind, beating the random selection with a large margin. This clearly shows that the concept of ADD-networks – while not

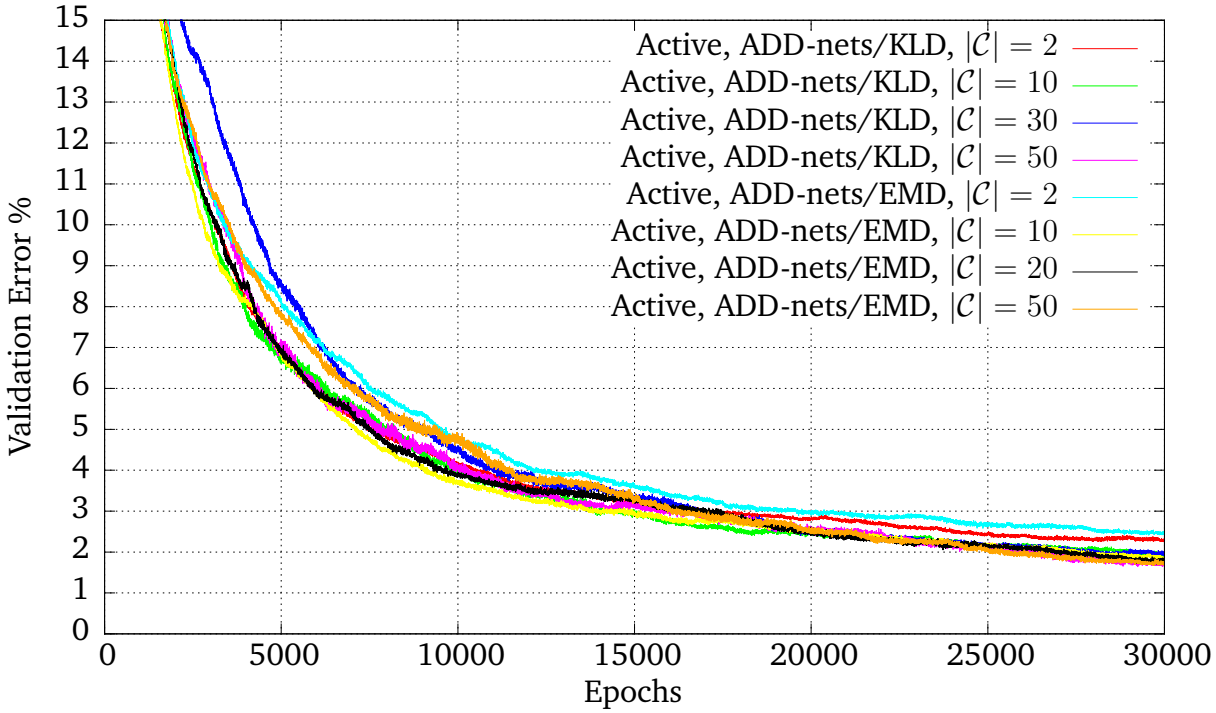


Fig. 4.1: Prediction error on the validation set as training progresses. This plot shows the results for all the experiments with ADD-networks. Validation error is tested after each epoch trained.

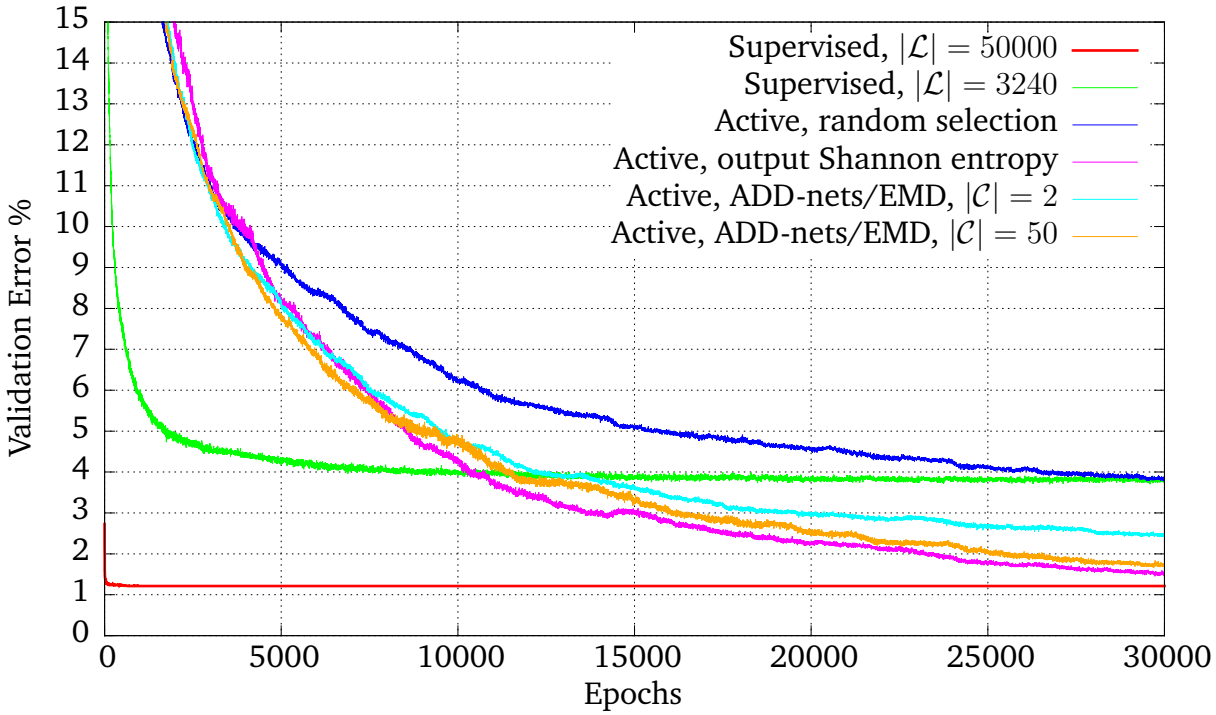


Fig. 4.2: Same as above, but with the baselines discussed in Section 3.4 and the alternative active strategy of uncertainty sampling with output Shannon entropy included. All ADD-results from the above plot has been removed except for the best and worst performing, to reduce clutter.

Table 4.1: Results of classification experiments. The three first rows are the baselines described in Section 3.4. KLD stands for Kullbach-Leibler Divergence, EMD for Euclidean Mean Distance.

Experiment	Validation Error	Test Error
Supervised, $\|\mathcal{L}\| = 50000$	$1.21 \pm 0.01\%$	$1.27 \pm 0.02\%$
Supervised, $\|\mathcal{L}\| = 3240$	$3.70 \pm 0.03\%$	$4.26 \pm 0.02\%$
Active, random selection	$3.75 \pm 0.06\%$	$4.06 \pm 0.03\%$
Active, output Shannon entropy	$1.44 \pm 0.05\%$	$1.46 \pm 0.01\%$
Active, ADD-nets/KLD, $\|\mathcal{C}\| = 2$	$2.17 \pm 0.01\%$	$2.27 \pm 0.07\%$
Active, ADD-nets/KLD, $\|\mathcal{C}\| = 10$	$1.85 \pm 0.07\%$	$1.85 \pm 0.02\%$
Active, ADD-nets/KLD, $\|\mathcal{C}\| = 30$	$1.80 \pm 0.06\%$	$1.89 \pm 0.12\%$
Active, ADD-nets/KLD, $\|\mathcal{C}\| = 50$	$1.69 \pm 0.05\%$	$1.76 \pm 0.05\%$
Active, ADD-nets/EMD, $\|\mathcal{C}\| = 2$	$2.38 \pm 0.03\%$	$2.34 \pm 0.08\%$
Active, ADD-nets/EMD, $\|\mathcal{C}\| = 10$	$1.76 \pm 0.03\%$	$1.75 \pm 0.04\%$
Active, ADD-nets/EMD, $\|\mathcal{C}\| = 30$	$1.68 \pm 0.07\%$	$1.75 \pm 0.02\%$
Active, ADD-nets/EMD, $\|\mathcal{C}\| = 50$	$1.64 \pm 0.03\%$	$1.72 \pm 0.02\%$

currently able to outperform the state of the art on this exact task – is a viable active strategy.

When it comes to disagreement measures used for the ADD-networks, the two tested alternatives – Kullbach-Leibler divergence and Euclidean mean distance – perform approximately equally well, with a slight edge for the Euclidean mean distance when the committee size is larger than 2. While the Euclidean measure has slightly better results here, it is hard to conclude definitively which is better for this exact task. In general, what disagreement measure performs best depends greatly on the problem. An important point however, is that the Euclidean measure is not dependent on the problem being a classification problem. Since Kullbach-Leibler, like other alternatives such as vote entropy, is dependent on either probability distributions or

explicit votes, it is useless on most regression tasks. Euclidean mean distance (and other variance-based measures) on the other hand, can work on all kinds of real-valued outputs.

4.3 Effect of Committee Size

The results further show that the performance of ADD-networks improve with larger committee sizes, as expected. But as the plot in Fig. 4.3 suggests, there are diminishing returns when increasing the number of members in the committee, which is consistent with the results seen in Srivastava et al. [76, Fig. 11]. That the benefits of increasing committee sizes become gradually smaller is fairly intuitive. With more samples, the chance that taking another sample will meaningfully skew the variance becomes smaller and smaller.

The computational overhead of a larger committee is significant, so in real-world scenarios a trade-off has to be made between what is more important – faster execution or more accurate committees.

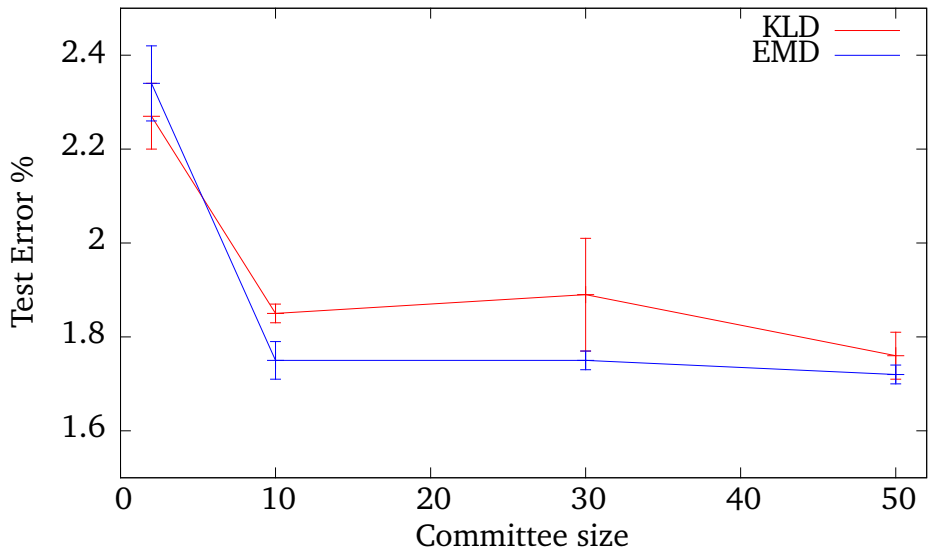


Fig. 4.3: Final test error compared to committee size, for both the Kullbach-Leibler divergence (KLD) and Euclidean mean distance (EMD) disagreement measures. The plot shows that there are diminishing returns when increasing committee sizes.

(a)Epoch: 100
Label: 0

(b)Epoch: 2400
Label: 0

(c)Epoch: 4700
Label: 3

(d)Epoch: 7000
Label: 5

(e)Epoch: 9300
Label: 1

(f)Epoch: 11600
Label: 3

(g)Epoch: 13900
Label: 7

(h)Epoch: 16200
Label: 8

(i)Epoch: 18500
Label: 9

(j)Epoch: 20800
Label: 3

(k)Epoch: 23100
Label: 2

(l)Epoch: 25400
Label: 4

(m)Epoch: 27700
Label: 8

(n)Epoch: 30000
Label: 7

Fig. 4.4: Examples of instances selected for labeling by ADD-nets at different stages in learning. Each instance is captioned with the epoch at which it was selected, and the label.

4.4 Selected Examples

Fig. 4.4 shows examples of digits selected for labeling by one of the runs of the ADD-nets. Some of the digits (like Subfig. 4.4d, 4.4h, and 4.4l) look close to normal, which suggests that the network was struggling on discerning between some different digits at the time, for example 5's and 8's, or 4's and 9's. Most of the others looks much harder to discern, some of them would be hard even for humans. Figuring out the difference between the 9 in Subfig. 4.4i and the 8 in Subfig. 4.4m is obviously tricky for example, let alone even recognizing that Subfig. 4.4m is an 8 at all. Coincidentally, this tricky 8 is an example that is often misclassified even with state of the art fully supervised approaches. One might intuitively assume that the active selector will select progressively more difficult examples as training progresses, but this is not necessarily the case. After training for 100 epochs (the time at which the first active selection is made) on the examples in the boostrapped $\mathcal{L}$, the network will already have a rough idea about how the different digits look. Very ambiguous examples will then already have a higher variance in predictions on average than more clear-cut ones. Most importantly, the figure (Fig. 4.4) clearly shows that the ADD-net-based approach is able to select especially ambiguous examples, a key characteristic of a working active learning strategy.

4.5 Committee Prediction Variance

The plot in Fig. 4.5 shows how the mean variance of predictions within a committee change as training takes place. During each selection phase, all of the unlabeled examples in $\mathcal{U}$ are evaluated according to the active strategy, in this case ADD-networks with Euclidean mean distance as the disagreement measure. The red line shows the *mean* evaluation score of $\mathcal{U}$, for each selection phase through the entire training process. As can clearly be seen, the variance quickly increases until around 25 selections (or 2500 epochs), after which it climbs linearly till the end of training. This rise is despite the fact that the 10 examples with the highest variance is removed from $\mathcal{U}$ (and moved to $\mathcal{L}$) each selection. The increase can possibly be explained as follows: As training progresses, the neurons will get increasingly specialized. As an illustration, the difference between predicting a 4 and a 9 might be dependent on a small set of features that can detect specific angles,

or whether or not the “loop” on top is closed. If some crucial subset of these features are dropped out in one of the dropout networks, that might tip the scales all the way towards predicting a 4 with high confidence, for example. Another network, that has these neurons intact, might make the opposite prediction, and these two predictions will then be far away from each other geometrically. This effect, in combination with the fact that as the network becomes increasingly adept it becomes increasingly certain in its predictions (and thus having output vectors that are further away from the geometrical center), might explain how the variance grows.

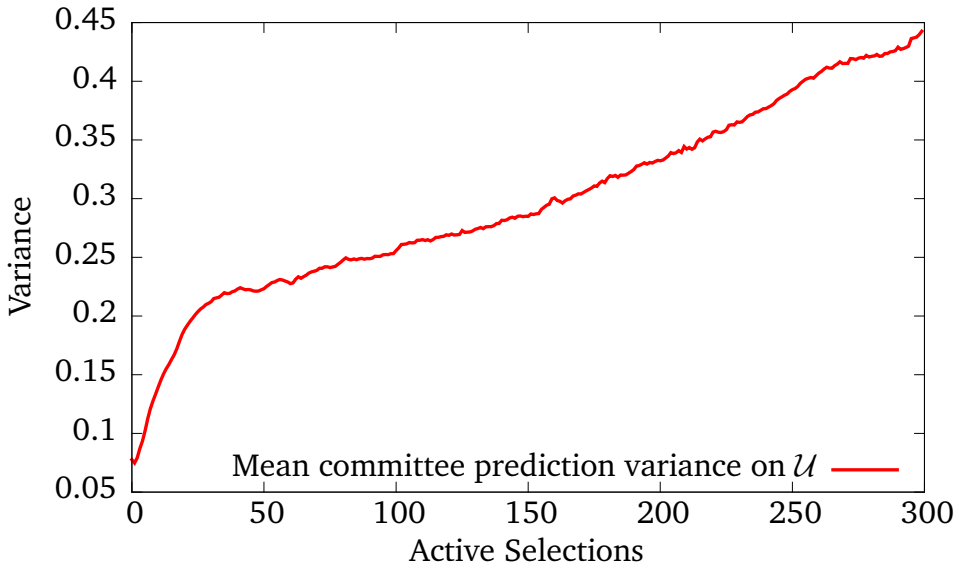


Fig. 4.5: Change in mean variance of predictions on $\mathcal{U}$ over the 300 selections of one experiment. Variance is calculated as the Euclidean mean distance between 10 dropout networks. In other words, this plot shows the mean disagreement score of all the unlabeled examples in $\mathcal{U}$ when using ADD-networks with EMD and $|\mathcal{C}| = 10$, recorded each selection.

It is important to note that the plot in Fig. 4.5 only shows the aggregate variance of the committee predictions, with no information about its constituent parts. Specifically, we cannot tell how the part of the variance that is useful in active selection – the uncertainty – is changing. We can assume that it decreases with training, however – as training reduces uncertainty by definition.

5 Discussion

5.1 Overview

This chapter will wrap up the thesis. First, by broadly discussing whether or not the research questions were answered, in Section 5.2. Then, by discussing the academic contributions of the project in Section 5.3, discussing promising directions for further research in Section 5.4, before concluding the thesis in Section 5.5.

5.2 Results

For convenience, the project's research questions are repeated here:

1. Does the variance of predictions made by dropout networks – all generated from the same full network – provide a heuristic for the confusion of the full network on the data example being predicted?

2. Can the Query by Committee algorithm, using such a group of networks as the committee, outperform random selection of examples?

It turned out that the simplest way to answer the former question was through answering the latter – if the algorithm, which used the variance of predictions made by dropout networks, is able to outperform random selection, it follows that the variance provides a useful heuristic. As the results in the previous chapter clearly shows, the algorithm was in fact able to outperform random selection, justifying positive answers to both of these research questions. Had this not been the case, some different experiment would have to be designed, that could test the first question in a different way. Fortunately, this was not necessary.

3. Can this algorithm perform comparably to other active learning algorithms that are applicable to deep neural networks?

The other active strategy tested in the experiments was uncertainty sampling through taking the output Shannon entropy, which performed slightly better than the best ADD-network configuration (which used Euclidean mean distance as the disagreement measure, with 50 networks in the committee), in addition to being computationally cheaper. On the other hand, the ADD-networks can in theory function in a regression scenario, which the output Shannon entropy approach cannot. To definitively answer this research question, however, more experiments have to be performed. While ADD-networks must be said to be comparable to the entropy strategy on the MNIST dataset, a wider range of both datasets and alternative active strategies must be tested, before anything but a tentative positive answer can be given.

5.3 Contributions

As discussed in the related work section (Section 2.4), active learning and neural networks have been combined in many different contexts – both using singular networks with uncertainty measures [88, 84], and ensembles of networks with disagreement measures [13, 42, 57]. But all of these approaches have some problems. Uncertainty sampling has been shown to work poorly in many scenarios [69], and is mostly useless for regression tasks. Ensemble based methods can work better, but in a deep learning scenario with large networks, the computational and memory overhead of training multiple networks can be insurmountable. The main contribution of this project then, is a novel way to get the benefits of committee-based methods while removing most of the overhead involved. The Active Deep Dropout networks proposed allow generation of an arbitrarily sized committee from one singular network, and as we have shown, this committee is diverse enough that the disagreement of its members provide enough information to be viable for use in active learning.

Many of the domains typically tackled and dominated by deep learning – such as image recognition or automatic speech recognition – are domains in which data is cheap and plentiful, but labeling often laborious and expensive. Effective active learning techniques can then be powerful tools. ADD-networks have shown that they can be a useful addition to the deep learning toolbelt.

A more abstract contribution of this project is further developing the general idea of using dropout networks as ensembles. The original dropout papers hint to this possibility – showing that taking the geometric mean prediction of a (sufficiently large) number of samples of networks provides more accurate predictions than the network as a whole, although they provide a heuristic using only the full network that works well. In other words, dropout can be seen as a type of efficient ensemble training system, and this ensemble can be used for various purposes. This is a powerful idea, as this thesis shows, and might be useful in other situations apart from active learning as well.

5.4 Further Work

Since this project merely investigates whether the general idea of ADD-networks works for active learning, there is a myriad of directions ripe for further research, both in ADD-nets specifically, and deep active learning generally.

5.4.1 Optimizations

The implementation here assumes a lot of simplifications, in an effort to isolate answers to the core question of the feasability of ADD-nets. Ergo, there are many possible optimizations that may further improve the results seen in Chapter 4:

- Weighting newly selected instances. Depending on how much training is done between each selection, the network may already be trained to convergence on the "old" instances when a new selection occurs. Weighting new instances, with a higher learning rate, for example, might reduce the training needed to reach convergence again.

- Transforming instances to increase dataset size. As Cireşan et al., 2010 [12] shows, state-of-the-art results on MNIST are achievable with a fairly simple feed-forward neural network and no pre-training when the instances of the dataset are transformed with both affine (rotation, scaling, shearing) and elastic deformations to provide more examples to learn from. Since the deformed instances have the same label as their origin, this seems like a good fit for active learning, as each query provides multiple new instances to train on, as well as a larger

unlabeled pool on which unsupervised pre-training can be performed. However, this technique is highly domain specific, as most datasets cannot be transformed in this way and still retain correct instance to label mappings.

- Dynamic active selection intervals. For the experiments performed in this project, the selection interval (the number of epochs between active selections) was fixed to a value that was big enough for random selection to perform as well as simple supervised training with the same number of randomly selected instances, just chosen ahead of time. There may be better ways to do this, such as letting the interval be dynamic. For example, if convergence (or something close to it) is detected, a new selection can occur etcetera. Depending on the scenario, experimenting with full retraining (from the weights initialized by pre-training) between each selection can possibly improve results, although with a large penalty in training time.

While these are active learning-specific possible improvements, there are obviously many improvements that can be made to the core MLP network and the supervised training phases, such as using more advanced regularization techniques in addition to dropout and max-norm regularization, experimenting with convolutional networks such as the *LeNet* of LeCun et al., 1998 [45], different pre-training techniques, etcetera.

5.4.2 Validation of Concept

Due to the rather limited computational resources available to this project, there are numerous things left to do to further validate the general ideas behind ADD-networks:

- Testing on more complex datasets, such as the ImageNet database [20]. It would be especially interesting to apply the concept to a largescale regression dataset.

- More comparisons to other active learning algorithms and architectures.

- Relaxing the assumption that we are only dealing with pool-based active scenarios. For example, testing ADD-nets on an application like the self-modelling robots of Bongard et al., 2006 [9], a de novo

synthesis type scenario. Robotics in general could be an interesting field for ADD-networks, as many robots use neural networks to predict and model events in their environment, but are typically not aware of their uncertainty, and using ensembles is often not realistic because of strict resource constraints. Being able to consult ADD-nets about the uncertainty of particular predictions on demand – without having to train multiple networks – could be very useful.

- Using ADD-nets to quantify uncertainty in non-active settings. In some scenarios, knowing the confidence a neural network has in its prediction is very valuable. Consider a network that predicts the pricing of some financial instrument [86, 38, 27]. Due to the extremely noisy nature of financial markets, such a network will have high error rates, and the cost of acting on wrong predictions can be very high (of course, if the network is correct more than half the time, the owner can profit in the long run). If the system has access to an approximation of the uncertainty of the network, it can choose to act only on those predictions the network is most certain about, which may possibly increase the ratio of profitable trades to unprofitable ones.

5.4.3 Dealing With Noise

Since QBC disagreement measures (and other typical active learning strategies) directly or indirectly represent the variance of the different predictions, performance will degrade in datasets where the dataset is more noisy in different regions of the input space: In regions where the dataset is more noisy, supervised training will be less effective, and predictions within this region will vary more than they will in less noisy areas. Active selection will then prefer to choose examples from these areas, as they will always have high variance / uncertainty. The core problem is that the active selectors are unable to differentiate between variance stemming from noise in the dataset and variance from actual uncertainty in the model.

There are multiple ways to attack this problem, but one interesting direction to explore is to include a measure of distance (in the input space) from data points already in the training set in the evaluation of unlabeled instances. This way, when clustering of selected instances in certain regions appear, new instances in that area will be increasingly discouraged until examples in other areas will be selected instead.

5.5 Conclusion

This project sought to test and validate the idea of ADD-networks – using multiple samples of dropout networks (from a single network) to estimate uncertainty for use in active learning. As the results clearly show, this goal was reached. But, so far it has only been shown to work on a relatively simple problem, and as the previous section details, there is more work to be done.

Applying active learning to deep learning is so far fairly uncharted territory, but with the types of datasets on which deep learning is typically employed, it is a possibly potent combination, especially when one considers the possibilities of crowdsourced active labeling. Hopefully then – with the help of tools like ADD-networks – active learning can become a tool available to deep neural network-users and researchers alike, facilitating the application of deep learning on even larger datasets than what is typical today.

Bibliography

[1] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, George Paliouras, and Constantine D Spyropoulos. „An evaluation of naive bayesian anti-spam filtering“. In: *arXiv preprint cs/0006013* (2000) (cit. on p. 6).

[2] Dana Angluin. „Queries and concept learning“. In: *Machine learning* 2.4 (1988), pp. 319–342 (cit. on p. 7).

[3] Les E Atlas, David A Cohn, and Richard E Ladner. „Training connectionist networks with queries and selective sampling“. In: *Advances in neural information processing systems*. 1990, pp. 566–573 (cit. on p. 8).

[4] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, et al. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. 2012 (cit. on p. 49).

[5] Eric B Baum and Kenneth Lang. „Query learning can work poorly when a human oracle is used“. In: *International Joint Conference on Neural Networks*. Vol. 8. 1992 (cit. on p. 8).

[6] Yoshua Bengio, Yann LeCun, et al. „Scaling learning algorithms towards AI“. In: *Large-scale kernel machines* 34.5 (2007) (cit. on p. 27).

[7] James Bergstra, Olivier Breuleux, Frédéric Bastien, et al. „Theano: a CPU and GPU Math Expression Compiler“. In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX, June 2010 (cit. on p. 49).

[8] Josh Bongard and Hod Lipson. „Automated reverse engineering of nonlinear dynamical systems“. In: *Proceedings of the National Academy of Sciences* 104.24 (2007), pp. 9943–9948 (cit. on p. 7).

[9] Josh Bongard, Victor Zykov, and Hod Lipson. „Resilient machines through continuous self-modeling“. In: *Science* 314.5802 (2006), pp. 1118–1121 (cit. on pp. 7, 62).

[10] Leo Breiman. „Bagging predictors“. In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on p. 16).

[11] Robert Burbidge, Jem J Rowland, and Ross D King. „Active learning for regression based on query by committee“. In: *Intelligent Data Engineering and Automated Learning-IDEAL 2007*. Springer, 2007, pp. 209–218 (cit. on p. 33).

[12] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. „Deep, big, simple neural nets for handwritten digit recognition“. In: *Neural computation* 22.12 (2010), pp. 3207–3220 (cit. on p. 61).

[13] David Cohn, Les Atlas, and Richard Ladner. „Improving generalization with active learning“. In: *Machine learning* 15.2 (1994), pp. 201–221 (cit. on pp. 6, 14, 15, 32, 60).

[14] David A Cohn. „Neural network exploration using optimal experiment design“. In: (1994) (cit. on pp. 13, 32).

[15] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. „Active learning with statistical models“. In: *Journal of artificial intelligence research* (1996) (cit. on pp. 7, 8).

[16] Ronan Collobert and Jason Weston. „A unified architecture for natural language processing: Deep neural networks with multitask learning“. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167 (cit. on p. 21).

[17] Aron Culotta and Andrew McCallum. „Reducing labeling effort for structured prediction tasks“. In: *AAAI*. 2005, pp. 746–751 (cit. on p. 10).

[18] Ido Dagan and Sean P Engelson. „Committee-based sampling for training probabilistic classifiers“. In: *Proceedings of the Twelfth International Conference on Machine Learning*. 1995, pp. 150–157 (cit. on p. 16).

[19] George E Dahl, Navdeep Jaitly, and Ruslan Salakhutdinov. „Multi-task neural networks for QSAR predictions“. In: *arXiv preprint arXiv:1406.1231* (2014) (cit. on p. 21).

[20] Jia Deng, Wei Dong, Richard Socher, et al. „Imagenet: A large-scale hierarchical image database“. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255 (cit. on pp. 21, 45, 62).

[21] Li Deng, Jinyu Li, Jui-Ting Huang, et al. „Recent advances in deep learning for speech research at Microsoft“. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8604–8608 (cit. on p. 21).

[22] Thomas G Dietterich. „Ensemble learning“. In: *The handbook of brain theory and neural networks* 2 (2002), pp. 110–125 (cit. on p. 16).

[23] Dumitru Erhan, Yoshua Bengio, Aaron Courville, et al. „Why does unsupervised pre-training help deep learning?“ In: *The Journal of Machine Learning Research* 11 (2010), pp. 625–660 (cit. on pp. 27, 28).

[24] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett. „DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1“. In: *NASA STI/Recon Technical Report N* 93 (1993), p. 27403 (cit. on p. 21).

[25] Stuart Geman and Donald Geman. „Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1984), pp. 721–741 (cit. on p. 29).

[26] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. „Domain adaptation for large-scale sentiment classification: A deep learning approach“. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 513–520 (cit. on p. 21).

[27] Erkam Guresen, Gulgun Kayakutlu, and Tugrul U Daim. „Using artificial neural network models in stock market index prediction“. In: *Expert Systems with Applications* 38.8 (2011), pp. 10389–10397 (cit. on p. 63).

[28] Dilek Hakkani-Tur, Giuseppe Riccardi, and Allen Gorin. „Active learning for automatic speech recognition“. In: *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*. Vol. 4. IEEE. 2002, pp. IV–3904 (cit. on p. 6).

[29] Awni Hannun, Carl Case, Jared Casper, et al. „DeepSpeech: Scaling up end-to-end speech recognition“. In: *arXiv preprint arXiv:1412.5567* (2014) (cit. on p. 21).

[30] Alexander G Hauptmann, Wei-Hao Lin, Rong Yan, Jun Yang, and Ming-Yu Chen. „Extreme video retrieval: joint maximization of human and computer performance“. In: *Proceedings of the 14th annual ACM international conference on Multimedia*. ACM. 2006, pp. 385–394 (cit. on p. 9).

[31] David Haussler. „Learning conjunctive concepts in structural domains“. In: *Machine learning* 4.1 (1989), pp. 7–40 (cit. on p. 15).

[32] Geoffrey Hinton, Li Deng, Dong Yu, et al. „Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups“. In: *Signal Processing Magazine, IEEE* 29.6 (2012), pp. 82–97 (cit. on p. 21).

[33] Geoffrey E Hinton. „Training products of experts by minimizing contrastive divergence“. In: *Neural computation* 14.8 (2002), pp. 1771–1800 (cit. on p. 29).

[34] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. „A fast learning algorithm for deep belief nets“. In: *Neural computation* 18.7 (2006), pp. 1527–1554 (cit. on p. 30).

[35] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. „Improving neural networks by preventing co-adaptation of feature detectors“. In: *arXiv preprint arXiv:1207.0580* (2012) (cit. on p. 25).

[36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. „Multilayer feedforward networks are universal approximators“. In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on p. 20).

[37] Po-Sen Huang, Xiaodong He, Jianfeng Gao, et al. „Learning deep structured semantic models for web search using clickthrough data“. In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 2013, pp. 2333–2338 (cit. on p. 21).

[38] James M Hutchinson, Andrew W Lo, and Tomaso Poggio. „A nonparametric approach to pricing and hedging derivative securities via learning networks“. In: *The Journal of Finance* 49.3 (1994), pp. 851–889 (cit. on p. 63).

[39] Anders M Knutzen and John J Gisvold. „Likelihood of malignant disease for various categories of mammographically detected, nonpalpable breast lesions“. In: *Mayo Clinic Proceedings*. Vol. 68. 5. Elsevier. 1993, pp. 454–460 (cit. on p. 6).

[40] Daniel B Kopans. „The positive predictive value of mammography.“ In: *AJR. American journal of roentgenology* 158.3 (1992), pp. 521–526 (cit. on p. 6).

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 21).

[42] Anders Krogh, Jesper Vedelsby, et al. „Neural network ensembles, cross validation, and active learning“. In: *Advances in neural information processing systems* 7 (1995), pp. 231–238 (cit. on pp. 32, 60).

[43] Solomon Kullback and Richard A Leibler. „On information and sufficiency“. In: *The annals of mathematical statistics* (1951), pp. 79–86 (cit. on p. 18).

[44] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. „Deep learning“. In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on pp. 1, 22).

[45] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 62).

[46] Yann LeCun, Corinna Cortes, and C Burges. „The MNIST database of handwritten digits“. In: *Available electronically at http://yann.lecun.com/exdb/mnist* (1998) (cit. on pp. 21, 44).

[47] David D Lewis and Jason Catlett. „Heterogeneous uncertainty sampling for supervised learning“. In: *Proceedings of the eleventh international conference on machine learning*. 1994, pp. 148–156 (cit. on p. 11).

[48] David D Lewis and William A Gale. „A sequential algorithm for training text classifiers“. In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc. 1994, pp. 3–12 (cit. on p. 9).

[49] Ulrike Lueken, Kevin Hilbert, Hans-Ulrich Wittchen, Andreas Reif, and Tim Hahn. „Diagnostic classification of specific phobia subtypes using structural MRI data: a machine-learning approach“. In: *Journal of Neural Transmission* 122.1 (2015), pp. 123–134 (cit. on p. 6).

[50] Andrew Kachites McCallumzy and Kamal Nigamy. „Employing EM and pool-based active learning for text classification“. In: *Proc. International Conference on Machine Learning (ICML)*. Citeseer. 1998, pp. 359–367 (cit. on p. 6).

[51] Ueli Meier, Dan Claudiu Ciresan, Luca Maria Gambardella, and Jürgen Schmidhuber. „Better digit recognition with a committee of simple neural nets“. In: *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE. 2011, pp. 1250–1254 (cit. on p. 33).

[52] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997 (cit. on pp. 14, 15).

[53] Elaheh Moradi, Antonietta Pepe, Christian Gaser, et al. „Machine learning framework for early MRI-based Alzheimer's conversion prediction in MCI subjects“. In: *NeuroImage* 104 (2015), pp. 398–412 (cit. on p. 6).

[54] Yuval Netzer, Tao Wang, Adam Coates, et al. „Reading digits in natural images with unsupervised feature learning“. In: *NIPS workshop on deep learning and unsupervised feature learning*. Vol. 2011. 2. Granada, Spain. 2011, p. 5 (cit. on p. 21).

[55] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, et al. „On optimization methods for deep learning“. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 265–272 (cit. on p. 32).

[56] Bharath Ramsundar, Steven Kearnes, Patrick Riley, et al. „Massively Multitask Networks for Drug Discovery“. In: *arXiv preprint arXiv:1502.02072* (2015) (cit. on p. 21).

[57] Tirthankar RayChaudhuri and Leonard GC Hamey. „Minimisation of data collection by active learning“. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. Vol. 3. IEEE. 1995, pp. 1338–1341 (cit. on pp. 33, 60).

[58] Giuseppe Riccardi and Dilek Z Hakkani-Tür. „Active and unsupervised learning for automatic speech recognition.“ In: *INTERSPEECH*. Citeseer. 2003 (cit. on p. 6).

[59] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 19).

[60] Guido Rossum. *Python Reference Manual*. Tech. rep. Amsterdam, The Netherlands, The Netherlands, 1995 (cit. on p. 49).

[61] Anton C Rothwell, Luke D Jagger, William R Dennis, and David R Clarke. *Intelligent SPAM detection system using an updateable neural analysis engine*. US Patent 6,769,016. 2004 (cit. on p. 6).

[62] Nicholas Roy and Andrew McCallum. „Toward optimal active learning through monte carlo estimation of error reduction“. In: *ICML, Williamstown* (2001), pp. 441–448 (cit. on p. 6).

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. „Learning representations by back-propagating errors“. In: *Cognitive modeling* 5 (1988), p. 3 (cit. on p. 20).

[64] Robert E Schapire. „The boosting approach to machine learning: An overview“. In: *Nonlinear estimation and classification*. Springer, 2003, pp. 149–171 (cit. on p. 16).

[65] Jürgen Schmidhuber. „Deep Learning in Neural Networks: An Overview“. In: *Neural Networks* 61 (2015). Published online 2014; based on TR arXiv:1404.7828 [cs.NE], pp. 85–117 (cit. on pp. 20, 22).

[66] Jürgen Schmidhuber. *Who Invented Backpropagation?* 2014. URL: `http://people.idsia.ch/~juergen/who-invented-backpropagation.html` (visited on July 27, 2015) (cit. on p. 20).

[67] Hinrich Schütze, Emre Velipasaoglu, and Jan O Pedersen. „Performance thresholding in practical text classification“. In: *Proceedings of the 15th ACM international conference on Information and knowledge management*. ACM. 2006, pp. 662–671 (cit. on p. 13).

[68] Fabrizio Sebastiani. „Machine learning in automated text categorization“. In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47 (cit. on p. 6).

[69] Burr Settles. „Active Learning“. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (2012), pp. 1–114 (cit. on pp. 1, 5, 7, 10, 15, 60).

[70] Burr Settles and Mark Craven. „An analysis of active learning strategies for sequence labeling tasks“. In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2008, pp. 1070–1079 (cit. on pp. 9, 10).

[71] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. „Query by committee“. In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 287–294 (cit. on pp. 2, 15).

[72] Claude E Shannon. „A note on the concept of entropy“. In: *Bell System Tech. J* 27 (1948), pp. 379–423 (cit. on p. 12).

[73] Paul Smolensky. „Information processing in dynamical systems: Foundations of harmony theory“. In: (1986) (cit. on p. 28).

[74] Nathan Srebro and Adi Shraibman. „Rank, trace-norm and max-norm“. In: *Learning Theory*. Springer, 2005, pp. 545–560 (cit. on p. 27).

[75] Nitish Srivastava. „Improving neural networks with dropout“. PhD thesis. University of Toronto, 2013 (cit. on pp. 2, 25).

[76] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: A simple way to prevent neural networks from overfitting“. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 25, 54).

[77] Katrin Tomanek, Florian Laws, Udo Hahn, and Hinrich Schütze. „On proper unit selection in active learning: co-selection effects for named entity recognition“. In: *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*. Association for Computational Linguistics. 2009, pp. 9–17 (cit. on p. 13).

[78] Simon Tong and Daphne Koller. „Support vector machine active learning with applications to text classification“. In: *The Journal of Machine Learning Research* 2 (2002), pp. 45–66 (cit. on p. 6).

[79] Thomas Unterthiner, Andreas Mayr, Günter Klambauer, and Sepp Hochreiter. „Toxicity Prediction using Deep Learning“. In: *arXiv preprint arXiv:1503.01445* (2015) (cit. on p. 21).

[80] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. „Extracting and composing robust features with denoising autoencoders“. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103 (cit. on p. 25).

[81] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. „Show and tell: A neural image caption generator“. In: *arXiv preprint arXiv:1411.4555* (2014) (cit. on p. 21).

[82] Byron C Wallace, Kevin Small, Carla E Brodley, and Thomas A Trikalinos. „Active learning for biomedical citation screening“. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2010, pp. 173–182 (cit. on p. 13).

[83] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. „The NumPy Array: A Structure for Efficient Numerical Computation“. In: *Computing in Science and Engg.* 13.2 (Mar. 2011), pp. 22–30 (cit. on p. 49).

[84] Dan Wang and Yi Shang. „A new active labeling method for deep learning“. In: *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE. 2014, pp. 112–119 (cit. on pp. 32, 60).

[85] Liyang Wei, Yongyi Yang, Robert M Nishikawa, and Yulei Jiang. „A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications“. In: *Medical Imaging, IEEE Transactions on* 24.3 (2005), pp. 371–380 (cit. on p. 6).

[86] Jingtao Yao, Yili Li, and Chew Lim Tan. „Option price forecasting using neural networks“. In: *Omega* 28.4 (2000), pp. 455–466 (cit. on p. 63).

[87] Evangelia I Zacharaki, Sumei Wang, Sanjeev Chawla, et al. „Classification of brain tumor type and grade using MRI texture and shape in a machine learning scheme“. In: *Magnetic Resonance in Medicine* 62.6 (2009), pp. 1609–1618 (cit. on p. 6).

[88] Shusen Zhou, Qingcai Chen, and Xiaolong Wang. „Active deep networks for semi-supervised sentiment classification“. In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics. 2010, pp. 1515–1523 (cit. on pp. 32, 60).

[89] Xiaojin Zhu and Andrew B Goldberg. „Introduction to semi-supervised learning“. In: *Synthesis lectures on artificial intelligence and machine learning* 3.1 (2009), pp. 1–130 (cit. on pp. 1, 6).