NTNU
Norwegian University of
Science and Technology

Domain general Active Learning strategies using inter-sample similarity and Reinforcement Learning

Bjørn Hoxmark
Jørgen Wilhelmsen

Master of Science in Computer Science
Submission date: August 2018
Supervisor: Massimiliano Ruocco, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

This master thesis was written during the spring of 2018 as a conclusion to our masters degree in computer science with specialization in artificial intelligence at the Norwegian University of Science and Technology (NTNU).

The research was done in collaboration with the Artificial Intelligence Lab at NTNU, provided by Telenor, and the AImageLab at the Engineering Department at the University of Modena and Reggio Emilia (UNIMORE).

We would like to thank our supervisors at NTNU, Massimiliano Ruocco and Erlend Aune, and at UNIMORE, Simone Calderara and Lorenzo Baraldi. We would also like to thank our fellow master students who have joined us in weekly discussion meetings. Finally, we would like to thank our families and friends for their time spent reading and commenting on our work.

Jørgen Wilhelmsen and Bjørn Hoxmark
Trondheim, Norway, August 8, 2018

Abstract

One of the major drawbacks of deep learning is the amount of labeled training data required in order to reach acceptable performance. This labeled data may be difficult and expensive to obtain. The goal of active learning is to reduce the amount of labeled data needed for a model to achieve acceptable performance.

Traditional active learning has limited effectiveness across different domains, and there is no single query strategy that outperforms all others in various domains. In this thesis, we perform research on ways to close the gap between the current state of the art active learning research and a domain general active learning strategy.

This thesis proposes an architecture that utilizes the similarity between samples to improve the performance achieved by traditional active learning algorithms. We test the architecture on datasets with different characteristics, and prove that the utilization of inter-sample similarity enhances the performance in all cases. Our architecture is able to reach and exceed the performance achieved by traditional entropy sampling using only 62.5% and 83.9% of the data on the MR and UMICH dataset. To the best of our knowledge, this is the first analysis of the effects of inter-sample similarity in active learning.

Furthermore, we explore approaches to active learning in a visual semantic embedding setting. First, we provide a qualitative discussion on how model uncertainty may be represented in such an application. Using this representation, we propose a reinforcement learning approach to stream-based active learning. Lastly, we evaluate the effectiveness of the proposed approach, and provide an in-depth discussion of its performance.

Sammendrag

En av de største ulempene med dyp læring er den store mengden annoterte eksempeldata som kreves for å trene modellene. Det kan være både vanskelig og dyrt å fremskaffe slike eksempeldata i det omfanget som kreves for at modellen skal oppnå et tilfredsstillende læringsresultat. Målet med aktiv maskinlæring er å redusere den nødvendige mengden annoterte eksempeldata. I denne oppgaven utforsker vi hvordan det er mulig å minske avstanden mellom nåværende ledende forskning på aktiv maskinlæring og en generell aktiv maskinlæringsstrategi.

Oppgaven introduserer en arkitektur som utnytter likhet innenfor eksempeldataene for å forbedre læringsresultatene til de algoritmene som benyttes ved tradisjonell aktiv maskinlæring. Vi tester arkitekturen på datasett med forskjellige egenskaper. Vi viser at den oppnår bedre resultater enn tradisjonell aktiv maskinlæring i hvert eksperiment. Vår arkitektur oppnår og overgår læringsresultatene til tradisjonelle aktive læringsalgoritmer ved kun å bruke 62.5% og 83.9% av dataen på MR og UMICH datasettene. Så vidt vi vet, er dette den første analysen av effektene av inter-datapunktslikhet i aktiv maskinlæring.

Videre undersøker vi hvordan aktiv maskinlæring kan utnyttes i visuell semantisk embedding. Vi utvikler en metode basert på modellusikkerhet og diskuterer hvordan metoden kan brukes i denne settingen. Vi velger å bruke strøm-basert forsterkende læring som utnytter modellusikkerheten. Til slutt vurderer vi effektiviteten av den tilnærmingen vi har valgt og diskuterer grundig hva det er mulig å oppnå.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter first describes the background and the motivation for this thesis. Then, we present our goals and research questions, before we summarize our main contributions.

1.1 Background and Motivation

Deep learning has achieved amazing results in recent years. In acoustic modeling [18], the error rate improved from 24,4% to 20,7% by replacing Gaussian mixture models with deep neural networks. In sentiment analysis [25], accuracy improved from 80% to 85.4% using a recurrent neural network. Question answering [8] also achieved an improvement with deep neural networks. Deep neural architectures consistently outperform other machine learning techniques, and has rightfully received much attention.

A disadvantage, however, with deep learning is the amount of data required to train the models. Active Learning (AL) is a field within machine learning that aims to reduce the amount of data needed to train machine learning models. A very high percentage of deep learning applications use supervised learning, and gathering and labeling data needed for this kind of learning is a costly process. An example is ImageNet [15] that contains more than 15 million images. This dataset was created by manually classifying the images into 22000 different categories.

Reducing the amount of training data needed to fulfill this kind of task by only a small percentage means significant economical and computational benefits. An example of how the amount of training data may be reduced is [6], where a 5% test error on the MNIST dataset was achieved with active learning using 295 labeled images. On the contrary, 835 labeled images were required to achieve the same percentage using naive random sampling, which means a 283% increase in the amount of data needed to accomplish the same performance.

There is little doubt that reducing the required amount of labeled training data is beneficial for virtually all supervised learning applications. However, most of the existing approaches to active learning are domain specific, or requires the existence of a labeled dataset to learn a query policy. Both circumstances inherently defeat the purpose of active learning. This motivates a work on designing domain general active learning strategies, applicable without using domain-specific knowledge.

Learning joint embeddings for data from different domains is an attractive use of deep learning. Applications of joint embeddings may be recognizing items in a photo [4] and assist self-driving cars in understanding what happens around them using object-based semantic mapping [19], among others. Training models for joint embeddings requires large amounts of data, and motivates research on active learning within such applications. When connecting data from different domains, the model maintains a joint embedding space which exhibits semantic properties. The properties of that space are, in theory, domain independent. As such, it motivates a theme of research that explores approaches to exploit this independence, in order to produce an active learning strategy that works across applications of joint embeddings.

1.2 Goals and Research Questions

Our main goal for this thesis is to reduce the distance between current active learning research and a domain general active learning query strategy.

Our pre-thesis [28] yielded interesting results on the similarity between samples in active learning and warranted further research on how to incorporate inter-sample similarity in an active learning query strategy. This similarity between samples is a property which is domain general, and the utilization of this information may be a step in the right direction towards our main goal of a general active learning strategy.

Research question 1 *How may similarity between samples be utilized in order to improve on existing active learning techniques?*

Deep neural models that connect data from different domains are highly sought after at the present time. To the best of our knowledge, no active learning query strategy has been defined for domains on which such a model can be applied. Active learning algorithms with a basis in uncertainty have traditionally performed well, and this motivates research on how we can capture this uncertainty in the joint embedding space.

Research question 2 *How may uncertainty be captured in Visual Semantic Embedding?*

Using the factors that encapsulate model uncertainty in Visual Semantic Embedding (VSE), we want to develop an active learning query strategy for this domain. A main purpose with this thesis is to explore the possibilities and limitations of using a reinforcement learning agent to learn an active learning strategy in joint embedding settings, moving active learning closer to a general active learning strategy.

Research question 3 *By applying factors that encapsulate model uncertainty in Visual Semantic Embedding, will a reinforcement agent be able to learn which states lead to high rewards?*

1.3 Contributions

This section summarizes our main contributions in this thesis.

1. An effective approach for exploiting similarity between samples in order to improve on existing active learning techniques
2. Insight to how to encapsulate model uncertainty in visual semantic embedding
3. An attempt to utilize model uncertainty to train a reinforcement agent to make labeling decisions in visual semantic embedding
4. A qualitative analysis of the effectiveness of the proposed approach

All our experiments and code are publicly available at Github[1].

1.4 Thesis Structure

The paper is structured as follows: In chapter 2, we give a brief description of the background theory needed for our theoretical discussion. In chapter 3 we refer the state of the art research within the field. In chapter 4 we present the architecture and models we employed. Chapter 5 starts with describing the hardware, frameworks, and datasets we made use of, with some key characteristics of each dataset. Thereafter, we describe our experimental setup and the details surround the experiments we have performed. The chapter is closed off by displaying and describing our results. In the last chapter, we discuss these results on a theoretical basis before we clarify our conclusions and make suggestions for future work.

[1]`https://github.com/hoxmark/Deep_reinforcement_active_learning` (Accessed on 7/08/2018)

Chapter 2

Background Theory

This chapter involves the background theory needed in order to follow the discussion and proposed theories for the remainder of the thesis.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational networks originally inspired by the human brain and how neurons work. A neuron, illustrated in figure 2.1, consists of a core, called the nucleus, which has several incoming connections, called dendrites. Based on the electronic signals on these dendrites, the nucleus may activate and send an output signal on its axon. This output signal is then spread to the incoming dendrites of many other neurons, making the signal propagate forward through the network.

ANNs consist of several layers of *artificial neurons*. Artificial neurons, shown in figure 2.2, are digital replicas of the neuron described above. The first layer is called the input layer and forwards the input values to the nodes on the next layer. Based on the incoming signals, a single node computes an output signal that is passed to the nodes on the next layer. This process repeats until the final output layer, which yields the output of the neural network. Layers between the input and the output layers are called hidden layers.

Each node in the network has weights associated with each incoming signal. These weights are multiplied with the particular input signal, before all the resulting values are summed. This sum is passed through an activation function which computes the value to be passed on to the next layer of nodes. If a neuron j receives input values $x_1, x_2, ..., x_n$, the output o_j of that neuron is given by

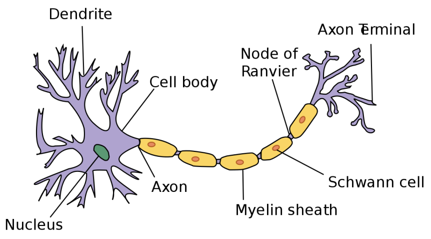


Figure 2.1: Neuron [1]

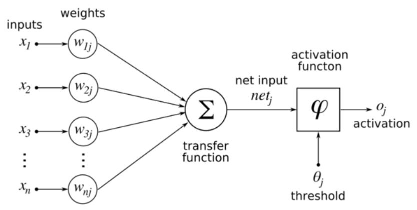


Figure 2.2: Artificial neuron [2]

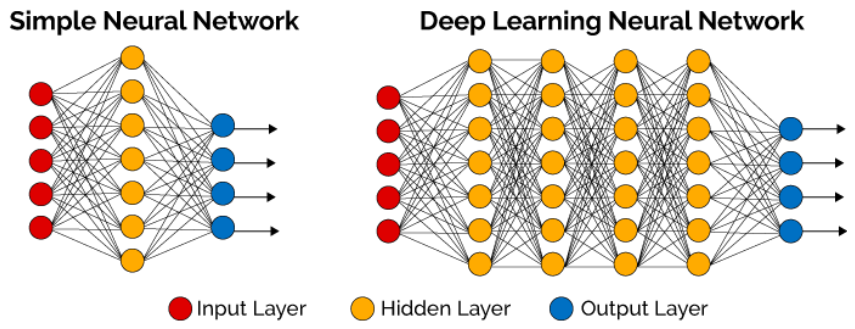


Figure 2.3: Example of a Deep Neural Network [3]

$$o_j = \phi\left(\sum_{i}^{n} x_i \cdot w_{ij}\right) \tag{2.1}$$

where ϕ is the activation function used for the neuron. The choice of activation function matters significantly, and has been extensively studied in the past years [26, 29, 7, 2, 9].

Artificial Neural Networks with multiple hidden layers are called deep neural networks, illustrated in figure 2.3. Traditionally, many hidden layers have made the networks too computationally difficult to work with, but in the later years, hardware development has risen to a level where incredibly large networks are

[1]`https://simple.wikipedia.org/wiki/Neuron` (Accessed on 11/11/2017)

[2]`https://upload.wikimedia.org/wikipedia/commons/thumb/6/60/ArtificialNeuronModel_english.png/600px-ArtificialNeuronModel_english.png` (Accessed on 11/11/2017)

[3]`https://cdn-images-1.medium.com/max/1600/1*5egrX--WuyrLA7gBEXdg5A.png` (Accessed on 11/11/2017)

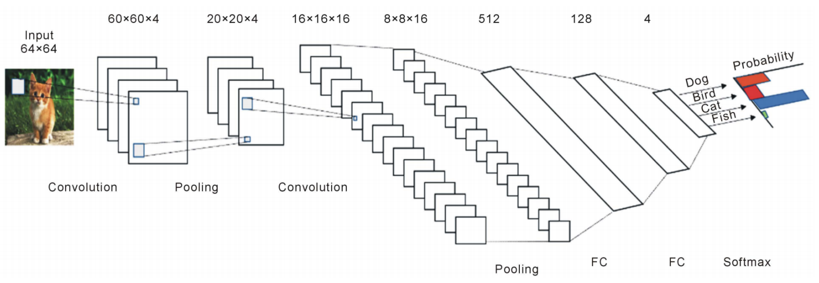


Figure 2.4: Example CNN architecture. Figure presented in [31].

feasible. Deep neural networks may further be split into subcategories depending on what type of computation is done within the layers. Here two particularly essential types of networks are described.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a very popular kind of neural networks, used mostly for image related tasks. CNNs had their breakthrough in image classification, and are also in the center of almost all computer vision systems to date. Essential in CNNs is the notion of *convolutions*. Convolutions may be described as functions applied to a sliding subset of the input values. The output of one application of this sliding function is one value in the resulting convolution vector. An example CNN is shown in figure 2.4.

The convolutions transform a window of lower level features to higher level representations. Each convolution layer extracts different features from different regions, and repeated applications of convolutional- and max-pooling layers transform the low-level pixel values to higher-level information. One filter may detect edges in its sliding window, the next shapes from the detected edges. These edges are constructed into objects, and so on.

Multiple convolutional layers are computed at each level before being fed into a pooling layer. The purpose of the pooling layer is to downsample the data, and to make the results from a convolutional filter more general and robust to scale or orientation changes.

The nature of the convolutional layers is to extract a feature over a sliding window, converting low-level spatial information to information of higher level. This method of computation does not take into account temporal information of previous data, which makes CNNs poorly suited for tasks that are sequential by

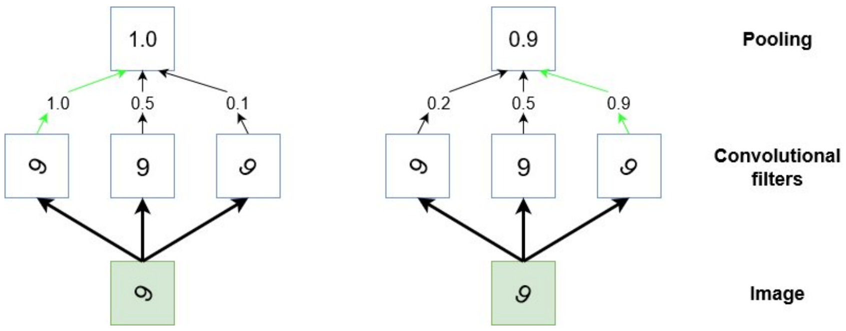


Figure 2.5: Purpose of CNN pooling [4]

nature, but better suited for tasks where spatial locality is significant, e.g. image processing.

2.3 Recurrent Neural Networks

The power behind Recurrent Neural Networks (RNNs) comes from the ability to perform sequence modeling, and the ability to let previous time steps influence the future. This makes RNNs suited for tasks of sequential nature - e.g. natural language processing and text generation. If a sentence x consists of n words

$$x = x_0, x_1, x_2, x_3, ...x_n \tag{2.2}$$

Sequence modeling is made possible by first initializing a hidden state that is combined with the first input x_0. This combined vector is fed to the RNN, which outputs h_0 corresponding to that time step. This hidden state may be decoded to an output at that time step. To compute the hidden state h_1 at $t = 1$, the hidden state h_0 is combined with the input x_1 and then fed to the RNN. This process is repeated for each sequential data part in x. At each time step the model outputs a hidden state h_t that may be used both as an output at that time step, or used in the next time step $t + 1$.

There are several different implementations of RNNs. The general idea of outputting a hidden state to be used for future time steps is ubiquitous, but the internal computation differs between the implementations. A traditional RNN consists of just one *tanh*-layer. A common problem with traditional RNNs is that their performance degrades once the gap between the relevant history and the

[4] `http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/`

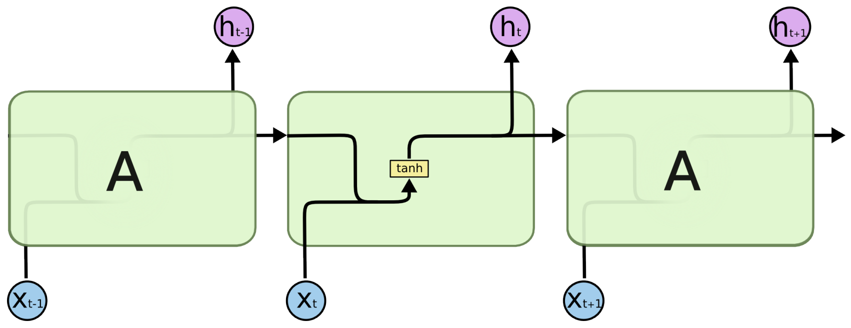


Figure 2.6: Traditional RNN [5]

current time step increases, which means that they suffer from long-term memory loss. Section 2.3.1 and 2.3.2 describe two principal kinds of RNNs.

2.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of RNN that aims to improve the short-term memory found in traditional RNNs. Instead of a simple tanh layer inside the repeating units, an LSTM contains a more sophisticated way of computing the hidden layers at each time step. The key behind LSTM's performance is the cell state that is displayed as the top line running through the LSTM unit, shown in figure 2.7.

$$i = \sigma(x_t U^i + s_{t-1} W^i) \tag{2.3}$$

$$f = \sigma(x_t U^f + s_{t-1} W^f) \tag{2.4}$$

$$o = \sigma(x_t U^o + s_{t-1} W^o) \tag{2.5}$$

$$g = \tanh(x_t U^g + s_{t-1} W^g) \tag{2.6}$$

$$c_t = c_{t-1} \circ f + g \circ i \tag{2.7}$$

$$s_t = \tanh(c_t) \circ o \tag{2.8}$$

An LSTM unit consists of four layers and three gates. The gates are comprised of an input gate i, a forget gate f and an output gate o. All gates apply a sigmoid activation function. This function has the effect of defining how much information you want to let through that particular gate. g combines the previous hidden

[5] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (Accessed on 01/12/2017)

state s_{t-1} and the current input x_t to a hidden state that will be run through the input gate. c_t is calculated by multiplying c_{t-1} with the forget gate f, added to g multiplied by i. The intuition is to control how the previous memory is combined with the new input. Finally, the hidden state s_t is computed by multiplying c_t with the output gate o.

2.3.2 Gated Recurrent Unit

A Gated Recurrent Unit (GRU) is quite similar to an LSTM unit, but there are still significant differences. A GRU has only two gates - a reset gate r and an update gate z. The reset gate r controls how much of the incoming memory h_{t-1} that should be combined with the new input x_t. The update gate controls how much of that previous memory persists to the next time step. GRUs are also missing the internal memory c_t found in LSTMs.

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{2.9}$$
$$r = \sigma(x_t U^r + s_{t-1} W^r) \tag{2.10}$$
$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \tag{2.11}$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1} \tag{2.12}$$

The Gated Recurrent Unit is a more modern implementation of the Recurrent Neural Network, which usage has increased the last couple of years. Because of the reduced amount of layers and gates, GRUs may train faster, and they need less data to achieve a better performance.

2.3.3 Bidirectional RNN

The performance of an RNN may be enhanced by making it bidirectional. In many cases, knowledge about what will happen in the future in addition to what has happened in the past may often be beneficial for the accuracy of an RNN. To make an RNN bidirectional, it will operate with two hidden states. One hidden state works through the input sequentially, and the other works through the input backward. The final outputs of both the hidden states are typically concatenated before being fed to a dense output layer. A visualization is shown in figure 2.10.

2.4 Pre-trained word vectors

A vital part of a deep neural net's performance is the representation of the input presented to the model. Although deep neural networks compute their own internal representations throughout the model, the quality of the intermediate

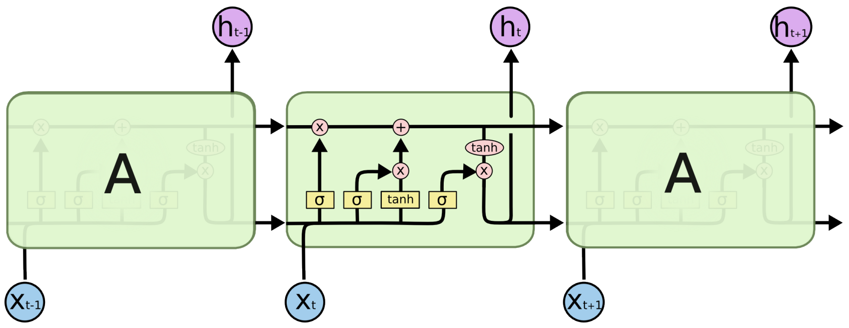


Figure 2.7: LSTM unit [6]

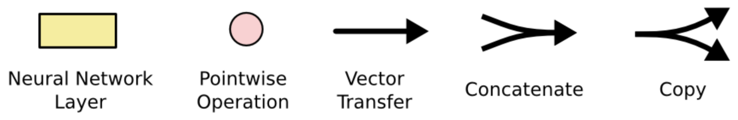


Figure 2.8: RNN Legend [7]

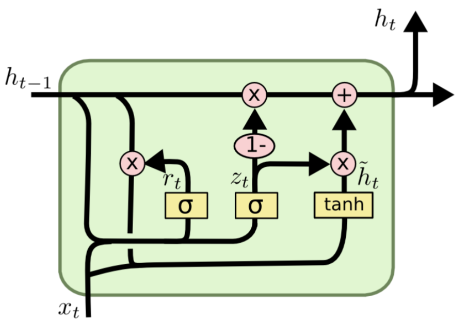


Figure 2.9: GRU [8]

[6]http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed on 05/6/2018)

[7]http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed on 01/12/2017)

[8]http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed on 01/12/2017)

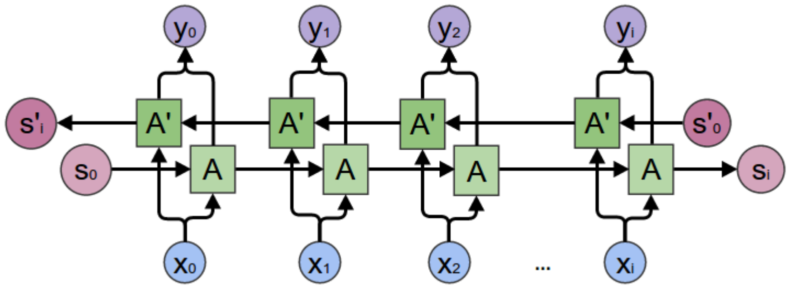


Figure 2.10: Bidirectional RNN [9]

representations depends on the input representation of a given sample. The representation of the input should thus be one that ensures that it is as easy as possible to separate features that are important for the task at hand.

Pre-trained word vectors are a collection of numerical vectors, each representing one word. These vectors are trained on an extremely large dataset, in order to capture the meaning of each word by the context it is normally used in. When making use of deep neural networks for natural language processing, the use of pre-trained word-vectors almost always leads an increase in performance. The most commonly used are Word2Vec[10], GloVE, described in [20], and fastText [11].

These vectors display interesting behavior. When mapped into a multidimensional space, words with similar meaning are grouped together. See figure 2.11. Furthermore, the distance between similar pairs is the same. Vector addition and summation also show particularly interesting properties, see figure 2.13.

$$vector('king') - vector('man') + vector('woman') \approx vector('queen') \tag{2.13}$$

2.5 Active Learning

Active Learning (AL) is a type of semi-supervised machine learning. The goal of active learning is to intelligently select which samples to use when training a deep neural network, in order to reduce the total amount of labeled samples needed to reach acceptable performance. There are two main ways of performing active learning: Pool-based and stream-based. In a pool based scenario, unlabeled

[9]http://colah.github.io/posts/2015-09-NN-Types-FP (Accessed on 14/11/2017)
[11]https://code.google.com/archive/p/word2vec/ (Accessed on 07/12/2017)
[11]https://fasttext.cc/ (Accessed on 10/07/2018)

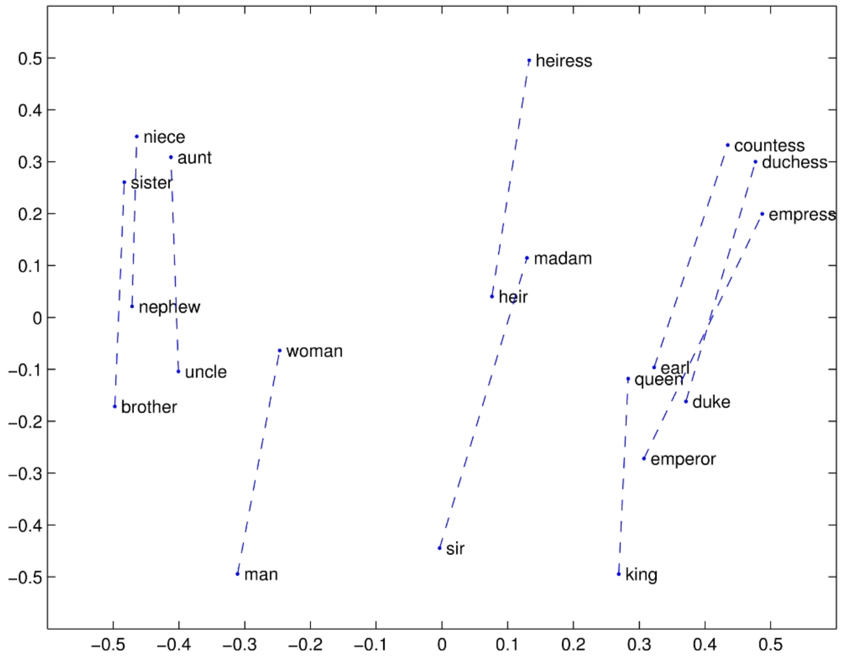


Figure 2.11: Word vectors display interesting behaviour when mapped into a multidimensional space. [12]

samples are scored using a scoring algorithm before greedily selecting samples using the calculated score. The selected unlabeled samples are labeled by an annotator, and then added to the labeled pool. A model is trained on the labeled data pool, before the cycle is repeated. In a stream-based setting, the active learner is presented with samples one at a time, and has to make a decision whether or not to label the current sample in the stream.

Active learning's field of application is areas in which unlabeled data are abundant, but the cost of labeling data using human experts is high. Spending a minor amount of computational power to drastically reduce the number of labeled samples needed promotes a huge financial motivation to apply active learning in a labeling process. An example of increased accuracy as a result of intelligent selection of samples is shown in figure 2.12

The effectiveness of active learning depends on the query strategy chosen for ranking the unlabeled samples. Here a few historically significant active learning scoring strategies are described.

[12]`https://nlp.stanford.edu/projects/glove/` (Accessed on 10/7/2018)

2.5.1 Traditional active learning

Uncertainty sampling is a scoring method concerned with one sample's expected information gain. This method ranks samples by how much new information they are expected to add to the model. A common definition for uncertainty sampling in active learning is Shannons entropy, first proposed in [23]:

$$-\sum_k P(y_k|x_i)\log_2 P(y_k|x_i) \tag{2.14}$$

where $P(y_k|x_i)$ is the predicted probability that sample x_i belongs to class y_k.

It follows from equation 2.14 that a model is least certain when $P(y_k|x_i)$ is equal for all k. If a model assigns an equal probability for all possible classes, it is obviously uncertain as to which label the sample should have. On the other hand, a model is most certain when there is one class y_k that has a much higher predicted probability than the other classes. Computing Shannon's entropy for all unlabeled samples before greedily selecting samples according to their entropy has traditionally performed well for a range of machine learning applications [22].

Expected parameter changes is a different group of active learning scoring methods. They are based on the expected model parameter change after the sample's label has been revealed and applied for training. The intuition is that if the model only needs to perturb its parameters in a small way, the sample lies close to other samples already seen by the model, and does not provide much new information.

On the other hand, if the expected model parameter updates are of greater magnitude, the sample is expected to bring a large quantity of new information. Because the true label for a sample is unknown at the selection stage, the expected parameter changes will have to be calculated for each possible class the sample may belong to. This may lead to a substantial increase in runtime when the number of possible classes grows large.

A deep neural model has many parameters, and the expected parameter changes may be calculated with respect to the different layers of the model. Which layer one decides to choose as a basis for expected parameter changes may be different for each application of the model.

Query by committee is another popular choice of active learning strategy. Query by committee maintains a set of different models that are trained on the labeled data. At the selection stage, samples are ranked by how much the different models disagree on the classification of the sample. If the sample is classified differently, it is more likely to bring new information to the labeled data pool.

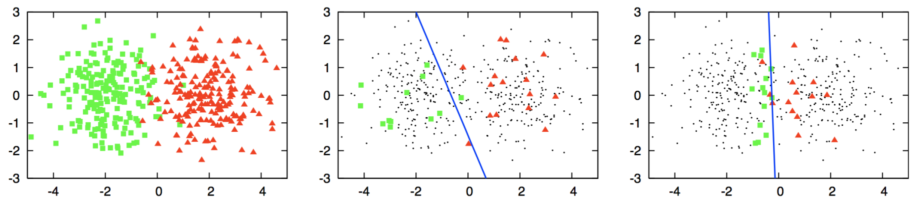


Figure 2.12: The left figure shows two types of data, separated by shape and color. The middle figure shows a decision boundary found when selecting training samples randomly. The right shows a decision boundary found when selecting samples intelligently, resulting in a boundary that more accurately splits the two types of data. Figure from [13].

2.6 Reinforcement Learning

Reinforcement Learning (RL) is an area within machine learning concerned with how agents select actions in an environment with the purpose of maximizing a cumulative reward. A predominant difference between this kind of learning and traditional machine learning is the absence of annotated training data. A reinforcement learning agent is able to collect and learn from experience while it is acting.

The reinforcement agent can be described as an entity playing a game: At each time step the agent must make an action, and by performing the action, it receives a reward, much like what a human player does when playing a game. The process of action selection, action execution and reward observation is repeated until the game ends and reaches a terminal state. Each individual play through is referred to as an episode.

The environment, in which the agent acts, is described as a Markov Decision Process (MDP), where outcomes from actions are partly random, and partly under the control of the agent. A Markov decision process is defined by

- A set of environment and agent states, S
- A set of actions the agent can make, A
- A transition model $P_a(s, s')$ that describes the probability of going from state s to state s' when performing action a
- A reward model $R_a(s, s')$ that describes the reward of going from state s to state s' when performing action a

[13]`https://www.datacamp.com/community/tutorials/active-learning`. (Accessed on 31/07/18)

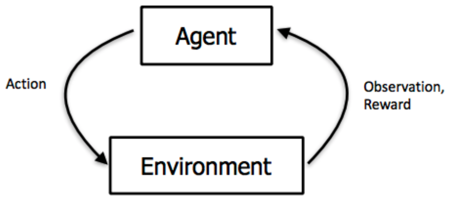


Figure 2.13: Reinforcement Learning cycle. Figure from [14].

- A discount factor γ that controls the importance of future rewards

How the agent selects actions in an environment, is called a *policy*, denoted π. The policy is a function that inputs the current state s and returns the proposed action to make in state s:

$$\pi(s) : \mathbb{S} \rightarrow \mathbb{A} \tag{2.15}$$

The purpose of a reinforcement agent is to learn a policy π that will maximize the rewards received throughout an episode. Each time the agent acts it stores a *state transition* that will be used to improve its policy. A state transition consists of the state the agent was in, the action it performed, the reward it received, and the new state reached as a consequence of making the action, commonly denoted as an (s, a, r, s') tuple. An episode is a sequence of states, actions, and rewards that ends in a terminal state. Given a discount factor γ, the agent's objective is to maximize the rewards throughout the episode

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{2.16}$$

, where r is a function that calculates a reward when the agent chooses action a in state s.

To enable an agent to maximize equation 2.16 it is trained numerous times using its stored state transitions collected during multiple episodes, iteratively improving its policy. If an agent greedily selects actions purely based on its estimate of action and state values, the agent would never explore the possible state-action space and, as a result, perform poorly. On the other hand, a purely stochastic action selection policy will never improve itself. This trade-off between optimal actions and exploration is commonly referred to as the exploitation versus exploration dilemma, and is essential within reinforcement learning. This dilemma

[14]`http://rll.berkeley.edu/deeprlcourse-fa15/`. (Accessed on 31/07/18)

makes it necessary to define an exploration strategy. A common strategy is the ϵ-greedy strategy which selects the action that yields the highest expected future reward with probability $1 - \epsilon$, and a random action with probability ϵ.

There are several methods for an agent to improve its policy as it acts and gathers experience. We will now describe the different central methods.

2.6.1 Value based

Value based reinforcement agents are agents that try to estimate the value of being in any given state. Naturally, the expected value of being in state s depends on our policy π. Denoted $V^{\pi}(s)$, the value of acting according to policy π starting from state s is the expected sum of discounted future rewards perceived by the agent

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{2.17}$$

An agent that always makes optimal actions acts under policy π^*. The optimal value function is the value function that corresponds to the optimal policy:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \tag{2.18}$$

In addition to the value function $V^{\pi}(s)$, that tells us the value of being in state s when acting according to π, it is useful to know the value of making action a in state s. This function is typically called Q:

$$Q : \mathbb{S} \times \mathbb{A} \to \mathbb{R} \tag{2.19}$$

$V^*(s)$ is the value of being in state s when acting according to the optimal policy π^*. This is equivalent to selecting the action that maximizes the state-action function Q(s, a) for all possible states:

$$V^*(s) = \max_{a} Q^*(s, a) \tag{2.20}$$

Having the optimal Q-function Q^*, an agent may act optimally by selecting the action from Q^* with the highest value in each state:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a) \tag{2.21}$$

The search for an optimal policy $\pi*$ is thereby equivalent to the search for the optimal state-action function $Q^*(s, a)$, and the *Bellman equation* can be used to iteratively improve an estimate of $Q^*(s, a)$. The Bellman equation simply states that the value of $Q^*(s, a)$ is equal to the immediate reward of making action a in state s, plus the discounted expected value of being in the state action a puts the agent in, s'. Recall that the outcome of actions is not deterministic, so the expected value of state s' is the sum of probabilities times the value of that particular state.

$$\begin{aligned} Q^*(s,a) &= R(s,a) + \gamma \mathbb{E}_{s'}\left[V^*(s')\right] \\ Q^*(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s') \end{aligned} \tag{2.22}$$

Using the definition above, iteration may take place numerous times, and the state-action function will converge towards optimal values. However, it is required to know the transition and reward models p and r, which is not always the case.

2.6.2 Q-learning

Q-learning is a form of model-free reinforcement learning. A model-free reinforcement learning algorithm is one that does not require knowledge about the transition- or reward-model. The Q-learning algorithm maintains an estimate Q-value for each state-action pair (s, a), and updates these by continuously acting in the environment and observing rewards. At first, the Q-values are initialized to a fixed value. Iterating, the Q-values are updated by

$$Q(s_t, a) = Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right] \tag{2.23}$$

where α is the learning rate, and determines the magnitude of the update to $Q(s_t, a_t)$. r_t is the instantaneous reward for taking action a in state s_t. This is known as Time-Difference Learning. The estimated future reward is the reward when acting optimally from the next state onward. One of the benefits of Q-learning is that it does not require knowledge about either the transition or the reward model.

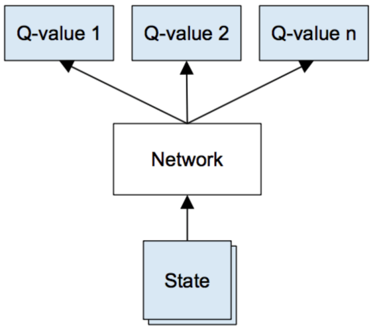
Traditional Q-learning needs to enumerate the entire state-action space that quickly grows unfeasibly large. Modern approaches utilize approximation techniques to circumvent the complexity in large state-action spaces, keeping Q-learning valid as a reinforcement learning technique.

2.6.3 Deep Q-Learning

When the total number of states and the number of possible actions in each state increases, the memory size required to perform traditional Q-learning grows

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 80 & 0 \\ 1 & 0 & 0 & 0 & 64 & 0 & 100 \\ 2 & 0 & 0 & 0 & 64 & 0 & 0 \\ 3 & 0 & 80 & 51 & 0 & 80 & 0 \\ 4 & 64 & 0 & 0 & 64 & 0 & 100 \\ 5 & 0 & 80 & 0 & 0 & 80 & 100 \end{array}$$

(a) Q-learning with table



(b) DQN

Figure 2.14: The left figure shows a table that supports a traditional Q-learning algorithm. There is one entry per state-action pair, which grows in size quickly. The right shows how a neural network is used to reduce the size requirement. Figures from [15].

exponentially and becomes unfeasible for large state-action spaces. Deep neural networks' ability to generalize over a large space may be used to approximate state-action values, and keep Q-learning viable in domains with a large state-action space. A deep neural network used for Q-learning is often called DQN, and takes a representation of the current state as input.

Training Deep Q-networks

In order to train DQNs we use the fact, that every Q-function for some policy obeys the Bellman equation. Because the task of the DQN is to predict Q-values for actions given a state, the Bellman equation may be used as a basis for a regression objective function in DQN training.

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s')) \tag{2.24}$$

The temporal difference error δ is known as the difference between the left and right side of equation 2.24:

$$\delta = Q(s,a) - (r + \gamma \max_{a} Q(s',a)) \tag{2.25}$$

[15] `http://mnemstudio.org/path-finding-q-learning-tutorial.htm` (Accessed 08/08/2018) `https://ai.intel.com/demystifying-deep-reinforcement-learning/` (Accessed 08/08/2018)

δ tells us that the current Q-value for an action in a state should be equal to the reward for making the action added with the maximum Q-value for the resulting state.

As the agent is acting in an environment, it records state transitions, actions, and rewards in a tuple (s, a, r, s'). These are used to iteratively improve its estimate Q-values, using the temporal difference error δ as a loss function.

Unstable learning

DQN suffers from unstable learning. This may partly be attributed to strong correlation between subsequent states. This makes the deep RL network overfit similar situations, and lose the ability to generalize. Several techniques to counteract this have been developed.

Replay Memory Instead of recording all state transitions, the amount of recorded state transitions is limited to a fixed size and stored in the *replay memory.* When performing mini-batch training, the agent samples a mini-batch at random from the replay memory, and uses it to update its estimate Q-values. This makes the state transitions in a mini-batch decorrelated, which is beneficial for the stability of the training.

Prioritized replay memory Traditionally, samples from the replay memory are drawn uniformly at random. In reality, the samples stored in the replay memory are of varying importance, and some samples contribute to the agent's learning more than other samples of worse quality. Prioritized replay memory, presented in [21], is a technique where the agent associates a priority with each transition in the replay memory. This leverages the probability that it will be drawn, with the intention of making items of higher quality drawn more often than those of low quality. Some transitions may be expected, and some may be rare and should contribute more than others.

Assigning priority to state transitions may be done in several ways, but the most common is to use the magnitude of the temporal error δ described in section 2.6.3. The temporal error is used to estimate the amount that the RL agent may learn from the transition. A greedy sampling using this temporal error will make the agent focus on a small subset of the experience, and this is not optimal. A solution to this is to find a trade-off between stochastic and greedy prioritized sampling. The probability of a transition being sampled should be scaled with the transition's priority, while simultaneously ensuring that zero-priority transitions have a chance of being sampled:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{2.26}$$

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_{p} \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

Figure 2.15: The figure shows a Q-learning update when the Q-values of the next state comes from a separate target Q-network. The target Q-network is updated to the current one in fixed intervals. The red rectangle is the values output from the target Q-network.

where p_i is the priority if transition i. α determines how much the priority should contribute towards the sample probability.

Target Network When training the Q-network using the loss function described in section 2.6.3, the target Q-values are constantly changing as we are updating our network weights. Google DeepMind has shown in [17] that having a separate target network that predicts the target Q-values helps stabilize training. The target network is updated in fixed intervals, and there are two typical ways to update it. *Hard*, which sets the target network equal to the Q-network every n-th iteration, and *soft* where the target network weights are updated by a small moving average of the Q-network weights.

2.6.4 Policy gradient

Policy gradient methods try to learn and optimize a policy function $\pi(s)$ in the policy space instead of learning the state-action function $Q(s, a)$. The agent learns how to map states directly to actions, as opposed to learning to estimate values for making actions in certain states. This is normally done using neural networks that directly model the action probabilities. On each agent interaction, the parameters of the neural network are tweaked, with the intention of making beneficial actions sampled more often in the future. This is repeated until the policy converges.

As mentioned in equation 2.16, we want our agent to maximize

$$J_\theta = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.27}$$

, where θ represents the neural network parameters. We want to optimize the parameters in the policy space, and following the gradient of J will make our network predict more beneficial actions in the future. Similar to Q-learning, the agent acts in an environment and stores state transitions (state, action, reward, next state). These samples are used to estimate $\nabla_\theta J$ before updating our models parameters with $\Delta\theta = \alpha \nabla_\theta J$

Policy gradient methods provide several benefits. There are environments in which the Q-function is too complex to be learned, and normal Q-learning will not be able to learn an acceptable policy. Policy gradient methods are still capable of learning an acceptable policy in these environments because they operate directly in policy space. In addition, policy gradient methods normally converge faster, but often reach a local optimum. On the other hand, policy gradient methods possess high variance in calculating $\nabla_\theta J$. This makes the network weights change too much in the wrong directions, and that slows and penalizes the training.

The main problem in policy gradient methods is computing $\nabla_\theta J$. An analytical solution for this gradient is not possible to derive, so one has to resort to approximation methods. A common way to do this is making use of likelihood ratios, that leads to the policy gradient theorem: For any differentiable policy $\pi_\theta(s, a)$ and objective function J, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \; Q^{\pi_\theta}(s, a)\right] \tag{2.28}$$

REINFORCE

The REINFORCE algorithm, from [30], is a policy gradient method that has basis in likelihood ratios. $Q^{\pi_\theta}(s, a)$ in equation 2.28 is the long-term value of making action a in state s. The REINFORCE algorithm uses v_t, the value of being in state t, as an unbiased sample of $Q^{\pi_\theta}(s, a)$. v_t is simply the experienced rewards from state t onward.

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t \tag{2.29}$$

The agent collects state transitions over the course of several episodes. At the end of each episode, the agent loops through all the recorded state transitions for that episode and updates the network parameters according to equation 2.29. Because it has the empirical rewards received at each time step, it is able to use those to estimate $Q^{\pi_\theta}(s, a)$, denoted v_t in equation 2.29.

2.6.5 Actor critic methods

Policy gradient methods have a high variance. This is partly due to the fact that v_t in equation 2.29 in reality is a high variance sample of the true $V(s)$. v_t is the sum of discounted rewards the agent perceived after time step t, but the agent might have followed a different path from the state at t, resulting in a different v_t. This is an intuitive explanation for the high variance of policy gradient methods.

To reduce the variance, actor-critic agents have a separate network for estimating v_t that is updated simultaneously with the policy network. This estimate of v_t is called the critic, and the policy network that outputs actions is called the actor. The critic estimates the action-value function

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a) \tag{2.30}$$

Our parameter update is then

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) Q_w(s, a) \tag{2.31}$$

This results in a decrease in variance, because the long-term value of being in state at time step t is estimated instead of using the unstable empirical rewards perceived by the agent.

Chapter 3

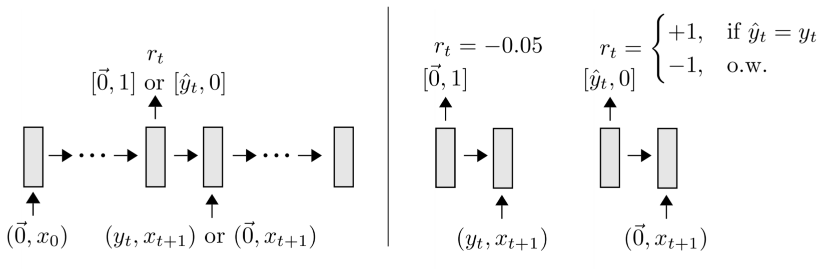
State of the Art

To design active learning strategies, heuristics are needed to determine for which samples to request the true label. These heuristics must be designed ahead of time and will often incur a human bias to the sample distribution. They are in addition often specialized and designed for specific datasets and datatypes. Furthermore, if the process of active learning is framed as a sequential one, each acquired sample changes the model, and thus affects the scoring of the samples in the next acquisition stage. Due to this, several experiments have been carried out where the purpose is to learn an active learner. The sequential nature of active learning supports the use of reinforcement learning to learning an active learning policy. Here we describe related work on learned active learners.

[32] use deep reinforcement learning in stream-based active learning to learn a policy that determines whether or not a label for the current sample should be requested. The model receives a stream of images and should make a choice at each time step whether or not to predict a label for the current sample. The rewards associated with either requesting the label, predicting the correct label or predicting the wrong label are

$$r_t = \begin{cases} R_{req} & \text{if a label is requested} \\ R_{cor} & \text{if predicting and } \hat{y} = y_t \\ R_{inc} & \text{if predicting and } \hat{y} \neq y_t \end{cases}$$

The action produced by the model is a one-hot vector of length $c + 1$, where c is the number of classes in the current training episode. By setting the final bit of the output vector, the model requests the true label for the current image. The next input to the model at time step x_{t+1} includes the next image, along with either the true label for the previous image if the previous action was a



r_t
$[\vec{0}, 1]$ or $[\hat{y}_t, 0]$
$(\vec{0}, x_0)$
(y_t, x_{t+1}) or $(\vec{0}, x_{t+1})$
$r_t = -0.05$
$[\vec{0}, 1]$
(y_t, x_{t+1})
$r_t = \begin{cases} +1, & \text{if } \hat{y}_t = y_t \\ -1, & \text{o.w.} \end{cases}$
$[\hat{y}_t, 0]$
$(\vec{0}, x_{t+1})$

Figure 3.1: The left part of the figure shows the complete task structure. At time step t the model outputs an action corresponding to a request or a prediction. The right part show the possible scenarios for the reward r_t calculated at time step $t+1$. Figure presented in [32].

request, or a 0-vector if the previous action was a prediction. The reward r_t is then determined at time step $t+1$, depending on what action the model chose, and whether the prediction was correct. The goal of each episode is to maximize the sum of future rewards using r_t at each time step. The task structure is shown in figure 3.1.

[5] argues that active learning methods relying on predefined heuristics is far from optimal, and achieve varying performance across different types of datasets. To address these problems, they frame the active learning problem as a reinforcement learning problem, where the learned policy takes the place of predefined heuristics. Their application is Named Entity Recognition(NER). They utilize the power of cross-lingual word-embeddings to train an agent on a target language where labeled data is abundant and apply it to another language with less available data. They claim that learning a policy on a high-resource language is trivial, and that the real gain in active learning comes from being able to apply a query strategy on a target language with considerably less labeled data.

The reinforcement agent is used in a stream-based active learning setting. Here the agent is tasked with deciding whether or not to label the current sample in the stream. The reinforcement agent is implemented using Deep Q-learning. A key decision in any stream-based reinforcement active learner is how to represent the current state. The state representation used in [5] consists of several parts. First, they use a CNN for text classification, similar to what is proposed in [11], to extract a feature vector from each sentence. As stated before, model uncertainty has proven beneficial for a range of wide active learning applications. To quantify this they use a different CNN over their classifier's predictive marginals belonging

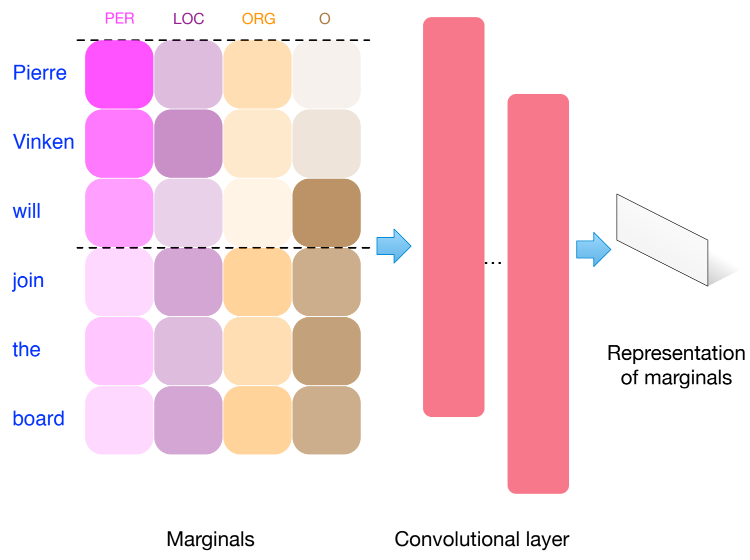


Figure 3.2: The architecture for representing predictive marginal distributions, $p_\phi(\mathbf{y}|\mathbf{x}_i)$, as a fixed dimensional vector, to form part of the MDP state. Figure and description from [5].

to a sentence and include it in the state. Finally, they include the confidence of the sequential predictions using the Viterbi algorithm for most probable path: $C = \sqrt[n]{\max_{\mathbf{y}} p_\phi(\mathbf{y}|\mathbf{x}_i)}$, where $n = |\mathbf{x}_i|$ is the length of the sentence.

[33] also uses reinforcement learning in an active learning setting. Co-training is a commonly applied semi-supervised method where two separate classifiers make use of the unlabeled data to increase each other's performance. Co-training is suited when the data exhibit two or more views, like multilingual word embeddings for Natural Language Processing (NLP), or headline and content for text classification.

Their problem setup is similar to that of [5], but they argue that making the reinforcement agent reason about each individual unlabeled sample is computationally inefficient. To combat this, [33] divides the unlabeled data pool into k groups $U = \{U_1, U_2, U_3, \ldots, U_k\}$ using Jaccard similarity.

A representation vector S_i is selected for each group $U_i \in U$, and the representation vectors $S_1, S_2, S_3, \ldots, S_k$ are used to construct a state for the reinforcement agent. This state should be able to encapsulate the state of the two co-training classifiers. Research has shown [36] that co-training with high-confidence ex-

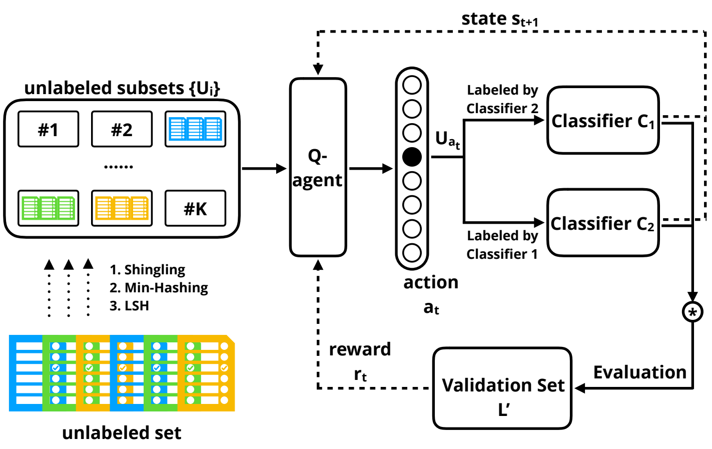


Figure 3.3: The Reinforced Co-Training framework. Figure from [33].

amples only results in improving areas in which the model already has a high performance, instead of learning a distribution that better represents the sample space. That is why the state should also be comprised of unlabeled samples with high uncertainty and diversity. There is a trade-off between these two types of data, and the balance between the two is difficult to determine ahead of time. Focusing on high uncertainty samples makes the classifiers unstable, and focusing too much on diversity shifts the bias towards the unlabeled data pool. For a reinforcement agent to be able to reason about this trade-off, the state needs to fully encapsulate the distribution of the unlabeled data.

Based on this intuition, the state representation is defined as

$$s_t = \{P_1^1 || P_1^2, P_2^1 || P_2^2, ..., P_K^1 || P_K^2\}_t, \tag{3.1}$$

where each P_i^1 and P_i^2 is the probability distribution of the two classifiers C_1 and C_2 on S_i.

A weakness of learned active learning strategies is that the learned policy is usually domain specific. A general active learning strategy needs to generalize across different datasets. Furthermore, it is trivial to learn a brilliant query strategy using deep reinforcement learning with access to many labeled samples. However, in a scenario where one has access to many labeled samples, an active learning approach will not be needed in the first place. This has motivated research on how to learn an active learner that is able to achieve cross-domain generalization.

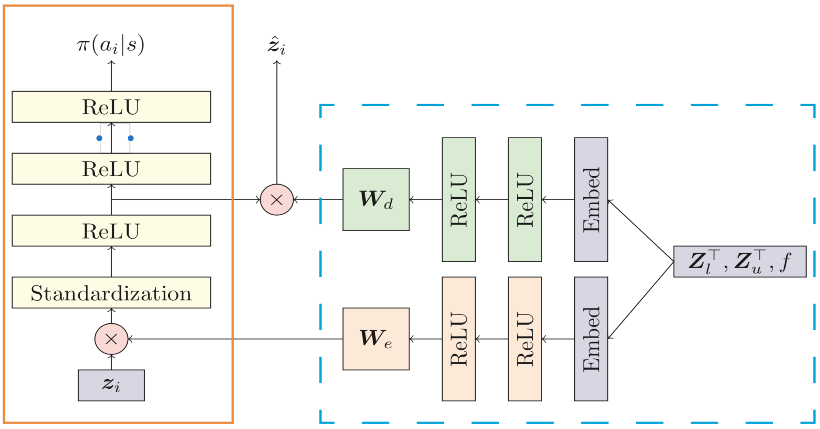


Figure 3.4: Policy and meta-learning network architecture from [1]. The meta-network on the right takes as input a featurization of the unlabeled and labeled samples Z_u^T and Z_l^T, and the current state of the classifier f, and outputs both an encoder W_e and a decoder W_d to be used for policy network to the left. The decoder is used for regularization purposes.

[1] also use deep reinforcement techniques for active learning, and they investigate how to train an active learning policy that may generalize across datasets. This is achieved by using two neural networks: One representing the criterion policy used for reinforcement learning, and a meta-network used for generating a dataset embedding and the weight matrices that are input to the policy network. The policy network outputs a softmax distribution $\pi(a_i|s)$ for selecting samples for which to request a label. The proposed architecture is shown in figure 3.4.

Cross-dataset generalization is achieved by multi-task training the embedding-generating network on multiple datasets. The network learns how to create an embedding for several types of datasets that the policy network may use for reinforcement learning.

Chapter 4

Architecture

This chapter first presents a detailed summary of the architecture applied in the experiments, along with the rationale behind the architectural choices made.

4.1 Similarity in traditional active learning

Our pre-thesis [28] included work on traditional active learning methods, where we explored the effects of employing different methods in a sentiment analysis setting. It showed that if used naively, active learning algorithms often select samples of very high similarity. If one sample is scored high using the scoring strategy in question, similar samples will probably be scored high as well. Labeling many samples of similar characteristics can be detrimental to the model's performance, and we will now describe an extension to the architecture that counteracts this.

The classifier for doing sentiment analysis is a CNN, based on [11]. The embedding layer of the classifier is initialized with word2vec word embeddings to increase performance. Our pre-thesis showed that if this is used naively, an active learner may select a subset of samples similar to one another at each acquisition stage. This may harm the model's performance, and warrants a method that discards samples of sufficient similarity. To further experiment on the effects of inter-sample similarity on model performance we developed an extension of the architecture of our pre-thesis.

The model starts with a pool U containing all the unlabeled samples in the dataset, and an empty pool L of labeled samples. At each acquisition stage, we score the samples in U using entropy (section 2.5.1), before greedily selecting samples with the highest score. Instead of naively adding all the selected samples to L, as done in traditional active learning, we further calculate pairwise similarity between the samples in the selected subset. Any pairs that have a similarity above a given threshold, has its lowest-scoring sample discarded from the selected set, and replaced by a new sample. This is an iterative process that continues until all the samples in the selected subset have a pairwise similarity that is sufficiently low, or no more samples of sufficient distance may be found.

To calculate pairwise similarity, we use cosine similarity. The representation of a sentence used in cosine similarity may be found in different ways. Here we describe two different methods.

w2v When the application is sentiment analysis, and we are using pre-trained word vectors, one way is to calculate the average word vector for each sentence, and use that vector as a basis for calculating similarity between sentences. The average word vector for a sentence is calculated by first adding all the individual vectors for each word that constitutes the sentence, and then dividing each element in the resulting vector by the number of words in the sentence. An advantage of this approach is that the embedding layer is pre-trained. This means that we will have useful representations of sentences without the need to train the classifier. A disadvantage of the approach is that the w2v features may not be an optimal representation for this setting.

CNN Another method is to use the output of the classifier applied in the experiment. First, the sentence may be passed through the CNN. Thereafter the second-last fully-connected layer(before logits) is used as a sentence representation. This as a basis for calculating similarity between the sentences. See figure 4.1 for a graphical representation of the CNN used for text classification. A fault of this approach is that we need to train the classifier in order to be able to produce useful representations. However such representations might be better suited for the task at hand since they are trained in the actual domain.

To summarize the architecture that utilizes similarity information in an active learning setting, we implemented the following procedure:

1. Train the model using the labeled samples in L
2. Calculate entropy for all samples in U
3. Select *topk* samples from U, ordered by entropy score
4. Calculate pairwise similarity within the selected subset
5. Discard lower-ranking item with similarity above the threshold, replace with new samples

6. If the selected subset is too small, go to 3

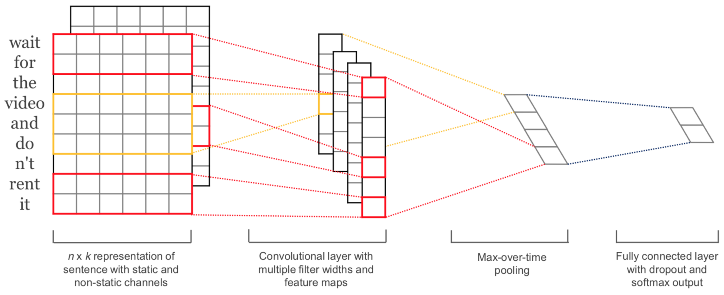

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 4.1: CNN for text classification. An input sentence has each word embedded with its word-vector. Then a series of convolution and max-pooling layers are applied, before final, fully connected layers downsamples the output to the number of classes in the classification problem. Figure from [11].

4.2 Visual Semantic Embedding

Our application for doing active reinforcement learning is Visual Semantic Embedding. This represents an example of problem types where the goal is to compute a joint embedding connecting different domains. The intention is to map data from two different domains into a common vector space, where semantically similar items ideally reside in close proximity to one another. In other words, we need to define a system that may learn to represent the underlying structure of the domains. The first step in this process is to derive a proficient representation in the respective domains, before projecting these into a common space.

4.2.1 Data representation

An important aspect of machine learning is the representation of data. Here we describe how we compute and derive vector representations of the data in our datasets.

VGG19

To compute representations of images, we use a CNN. A specific CNN architecture that has performed well is VGG19 [24]. VGG is a collection of convolutional network configurations that utilize deeper networks with smaller receptive window sizes. VGG-{size} denotes the released architecture with depth=size. Networks of this depth are made possible due to the small window sizes in the convolutional layers. The VGG networks all use 3x3 convolutional filters.

A pre-trained VGG19 model is natively available within the PyTorch framework and eliminates the need to train a model locally. We use the representations before the softmax layer in the VGG19 model as image representations, shown in figure 4.2.

Caption representations are computed using a Gated Recurrent Unit (section 2.3.2). The embedding layer in the GRU inherits weights from pre-trained word vectors, Word2Vec (section 2.4), in order to improve performance.

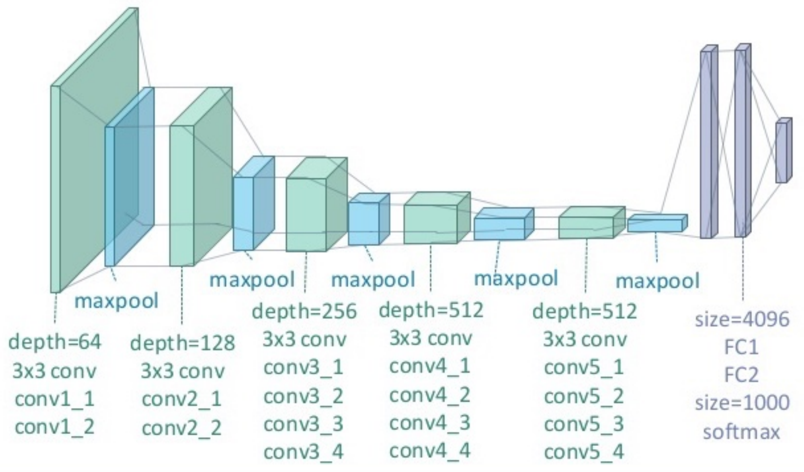


Figure 4.2: VGG19 model with its constituent layers. All convolutional layers are of size 3x3, with increasing number of channels per layer. Figure from [1].

The selected images do not change during the experiment. By using pre-trained encoders, we can save computational time by calculating the representations of the images once, and store these for future use. We use precomputed image features from [14], downloaded from [2].

[1]`https://lihan.me/2018/01/vgg19-caltech101-classification` (Accessed on 08/08/2018)

[2]`https://github.com/fartashf/vsepp` (Accessed on 01/06/2018)

4.2.2 Projections

The purpose of Visual Semantic Embedding is to connect images and descriptions of images. Images and text comprise data types with different properties. The object is to learn projections that map data of similar semantic characteristics from both domains close to one another in a common embedding space. This is a popular approach to semantic-visual joint embeddings [4, 13, 10, 27].

The projection of both images and captions is performed using linear, fully connected layers, which input the representations described in the previous section. Let ψ and ϕ be functions that encode images and captions in their respective domains. The common embedding space projections are then

$$f(i, W_f) = W_f^T \psi(i) \tag{4.1}$$

$$g(c, W_g) = W_g^T \phi(c) \tag{4.2}$$

, where W_f and W_g denotes the weights associated with the image- and caption projection layers. i and c denotes an image and a caption.

Once mapped to a common vector space, we need to define how to calculate similarity to determine if the projections from different domains are semantically analogous. Semantic similarity in the common embedding space is defined by the inner product

$$s(i, c) = f\,(i; W_f) \cdot g\,(c; W_g) \tag{4.3}$$

The parameters included in loss function, θ, need to include error terms from the different projections W_f, W_g. Let $e^i_{m\times d}$ and $e^c_{m\times d}$ be the matrices consisting of the embedded images and captions where the i-th row of $e^c_{m\times d}$ is the embedding of the correct caption for the image embedded in the i-th row of $e^i_{m\times d}$,m is the number of samples, and d is the size of each embedding vector. The pairwise distance between all images and captions is calculated by the inner product:

$$d = e^i_{m\times d} \cdot e^c_{m\times d} \tag{4.4}$$

d is a matrix of size $m \times m$ where elements along the diagonal represent the distance between an image and its correct caption. If we subtract each row in d with its corresponding element in the diagonal, the result is a matrix in which the diagonal is 0, and the other elements are real numbers. Using the sum of all the elements in the matrix as a loss function makes the model push pairwise non-similar items apart while minimizing the distance to the most similar one. The loss for the caption projection layer may be computed by transposing d, and repeating the procedure. The two losses are combined to form the total loss that includes elements from both projection layers.

To evaluate the performance of the model, we resort to using common information retrieval performance evaluation metrics. Recall @K is the fraction of query samples that have their corresponding item among the closest K items. An ideal model projects all pairs closest to each other and would have an R@1 score of 100%, but this is nearly impossible to achieve. The common embedding space is a space of semantical properties, and even though the pairs are not projected closest to each other, a projection that puts them in near vicinity of one another can still be a decent projection. Thus, we define recall at different distances, and use the sum of those as a performance measure.

A visualization of the visual semantic embedding procedure is shown in figure 4.3.

4.3 Reinforcement Active Learning

We apply a reinforcement active learning agent on top of the architecture described in section 4.2. The agent operates in a stream-based active learning environment, where samples are presented one after another. The agent is tasked with determining whether or not to query the label for the current sample in the stream.

Our task contains data belonging to different domains. In order to do stream-based active learning, we need to select one of these domains as the primary domain, to be used as a basis for active learning. When doing this, one can re-frame the problem as a classification problem, where the possible classes for each instance of the primary domain is the possible assignments for the instances in the secondary domain. The proposed architecture uses images as the primary domain and basis for stream-based active learning. Labeling the current image in the stream consists consequently of querying the correct captions for that image. Reinforcement learning requires us to define a state and reward function. We will now describe how we define these in the Visual Semantic Embedding setting.

4.3.1 State generation

The action selected by the agent ought to depend on the current state it can observe. It is crucial to provide a state to the agent that encapsulates important features from the environment, so that the agent is sufficiently informed and is able to make an appropriate action. Here we describe how we construct the state presented to the agent.

First, all images and captions are encoded using the linear projections from equation 4.1 and 4.2. Using the vector representations in the common embedding space, we calculate the pairwise cosine distances between images and captions. This results in a matrix D, where each $x_{i,j} \in D$ is the pairwise cosine distance

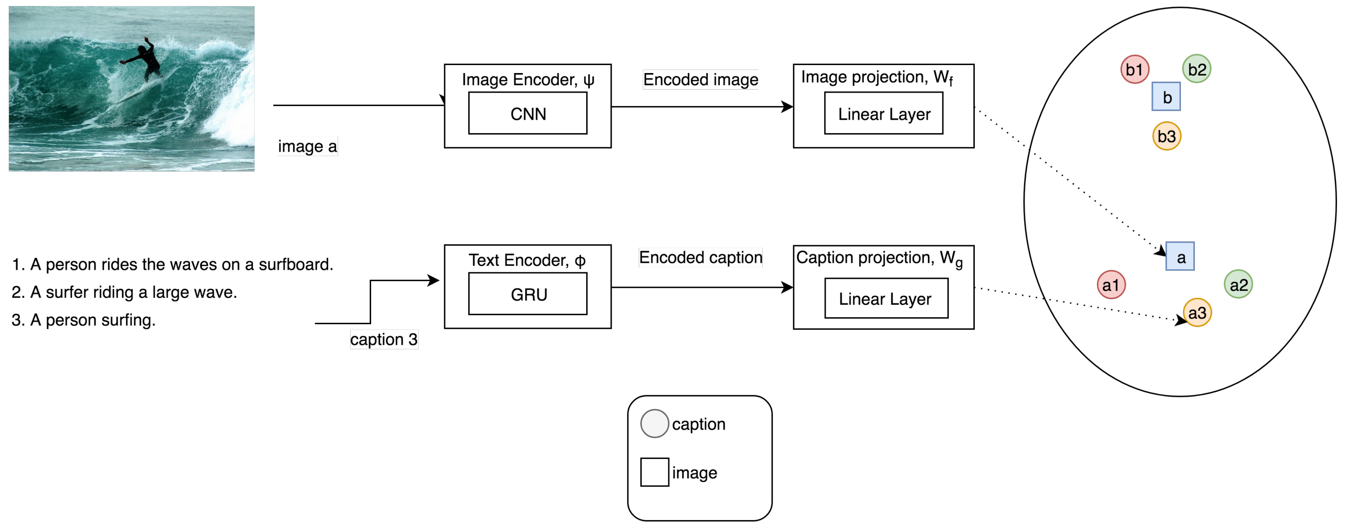


Figure 4.3: Visual Semantic Embedding architecture. Images are first encoded using the CNN described above. The output is projected to the common embedding space using a linear projection. An equivalent process is applied to sentences, with the exception of using a separate GRU encoder as opposed to a CNN.

between image i and caption j.

$$D = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,n} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & \cdots & \cdots & \cdots & x_{n,n} \end{bmatrix} \tag{4.5}$$

Each row in D constitutes a basis for constructing a state to be input to the reinforcement agent. How we construct this state is of great importance and affects the agent's ability to learn and generalize across domains. Research has shown that using model uncertainty as the basis for a query strategy leads to an increase in performance for a wide range of active learning applications [34]. However, quantifying uncertainty in joint embeddings is not straightforward. Our state generation procedure will try to capture differences that make a difference and make it possible for the agent to reason about this uncertainty. Here we describe possible features to include in such a state.

Distance to close captions

The goal of the joint embedding space is to capture semantic information from different domains. An optimal VSE model places images and captions with the same semantic information in the exact same location in the joint embedding space. Because of this, a state should encapsulate the distances to the captions which reside in close proximity to the image in question. Distances between pairs of data types are inherently domain general, which attains our goal of learning a general active learning framework. We select the distance to the k closest captions and include it in the state, where k is a hyperparameter. An intuitive example of how distances may be used to determine uncertainty is shown in figure 4.4.

Naively using the actual distances between the image and captions in the common embedding space is not sufficient. It is not the actual distances from the image to the captions that are of importance, it is the *difference* in distance among the closest captions that should give the agent grounds for reasoning about uncertainty. Distance is, in reality, a relative measure, and the uncertainty for the state

$$[0.1, 0.2, 0.3] \tag{4.6}$$

should intuitively be roughly the same as the uncertainty for the state

$$[10.1, 10.2, 10.3]\,, \tag{4.7}$$

but these states are vastly different. As a result, the agent loses important generalization opportunities and needs more data to converge to a better policy.

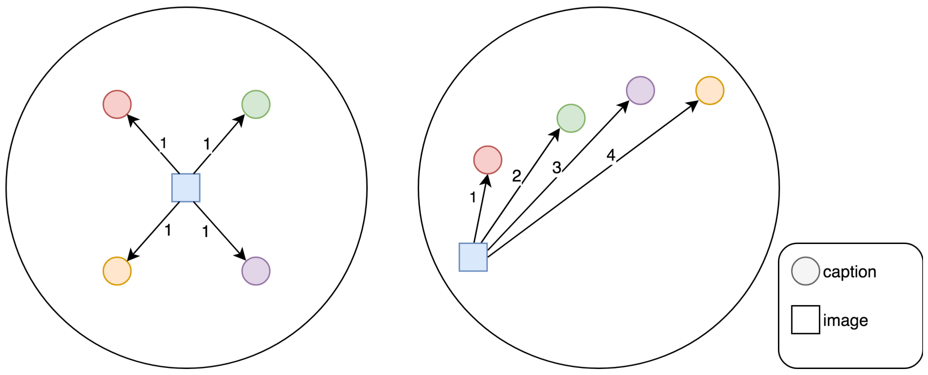


Figure 4.4: The left figure shows 4 captions with the same distance from the image. The right figure shows 4 captions with different distance from the image. Because of the difference in distance in the right figure, the state constructed using the left figure should have a larger uncertainty than the one at the right.

To encapsulate the relative differences between the distances as opposed to the actual distances themselves, we resort to using a softmin(x) = softmax(-x) function over the distances to the closest captions. This moves the caption-distance state values to the range $[0, 1]$, and captures the difference between the numerical values in the state.

$$\text{softmin}([0.1, 0.2, 0.3]) = \text{softmin}([10.1, 10.2, 10.3]) = [0.367, 0.33, 0.3] \tag{4.8}$$

Similarity between close captions

Extending the argument that the state provided to a reinforcement agent should make the agent able to reason about model uncertainty, we also make the case that the semantic difference between the k closest captions should be included in the state. If the closest captions for a given image all reside in a small area of the common embedding space, the model is fairly certain as to what semantic properties that image exhibits. On the other hand, if the closest captions for an image are in vastly different spaces of the embedding space, the model is clearly uncertain as to what semantic properties the image contains. A visual example is shown in figure 4.5.

To calculate the difference between the closest captions, we first extract the *topk* closest captions for each image. Then, for each caption, we calculate the average pairwise distance towards the other closest captions. The resulting vector is of size *topk*, where each element represents that caption's average distance towards the other close captions.

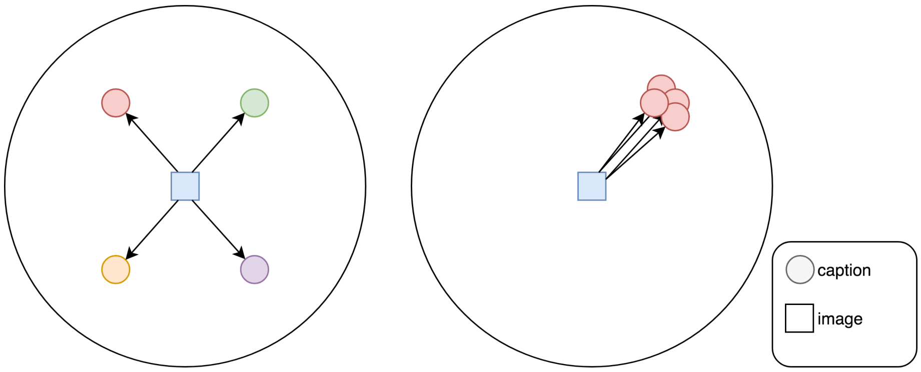


Figure 4.5: Both figures show 4 captions that all have distance 1 from the image. However, all of the captions in the right figure are in close proximity to one another, which means that the model is fairly certain of the semantic properties of the image. In the left figure, the closest captions all lie far from each other. Even though the distance from the image is identical, the distance between the captions should result in an increase in uncertainty.

Image representation

There may be images with semantical properties that are more difficult to learn than other images. Quantifying this difficulty is not a trivial task for humans to do, but a reinforcement agent, however, should be able to make sense of which image representations statistically lead to an increase in performance.

The decision of what to choose as image representation is significant. By using the representation calculated by the image-encoder (CNN, second-last layer of VGG19), we limit our reinforcement agent to only learn about joint embedding problems where the primary domain consists of images. Furthermore, in order to maintain the performance from one dataset to another, the image representation should be the same, which means we must use the same image representation encoder each time.

Another option is to use the actual representation of the image in the common embedding space. This representation is inherently more domain general due to the fact that it lies in the common embedding space. On the other hand, distribution of the common embedding space changes across domains, so this option may not be as general as we want.

We decided to include the representation of the image in the common embedding space in the state.

An example of a resulting state vector is shown in figure 4.6, and the full RL network is shown in figure 4.7.

$$\begin{bmatrix} \overbrace{0.367, 0.33, 0.3}^{\text{Distance to close captions}} \quad , \quad \overbrace{0.012, 0.06, 0.3}^{\text{Similarity between close captions}} \quad , \quad \overbrace{0.13, \ldots, 0.01}^{\text{Image representation}} \end{bmatrix}$$

Figure 4.6: Example of a state used in reinforcement learning for visual semantic embedding. The state depicted consists of image representation, distance to the closest captions and the distance between the closest captions, as explained in section 4.3.1.

4.3.2 Reward

If the agent chooses to query the captions for the current image in the stream, they are added to the labeled pool and the model is retrained. After the model is retrained, we measure the change in model performance and use it as the reward for labeling that particular image. If the agent decides not to label the current image, a reward of 0 is received.

Model performance may be defined in many ways and will vary with the domain and application. In normal classification use-cases, the measured classification accuracy on a held-out dataset is the obvious choice, but for more complex use-cases such as VSE, there are several possible reward choices. Furthermore, the calculation of held-out classification performance may be computationally demanding. The calculation of reward is something that happens frequently in the training of a reinforcement learning agent, and the runtime of this procedure contributes to a large extent to the overall runtime. This motivates defining reward functions that require less computational resources, such as the total loss of the loss function over a held-out dataset. Nonetheless, the reward chosen for the action should always be computed over data not previously seen by the model.

If the sample(s) that are being added to the labeled pool normally leads to an increase in performance, the agent will eventually always learn to request a label for the current sample, regardless of the input state. The intuition is obvious: If the agent doesn't select this sample, the reward will be 0, whereas a selection will, with a very high probability, bring a positive reward. If left as-is, the agent will learn that more data is always beneficial for a model's performance, and will consequently label the current sample with a very high probability. An increase in the amount of labeled data available presumably leads to an increase in performance. We therefore need to incur a penalty for requesting the true

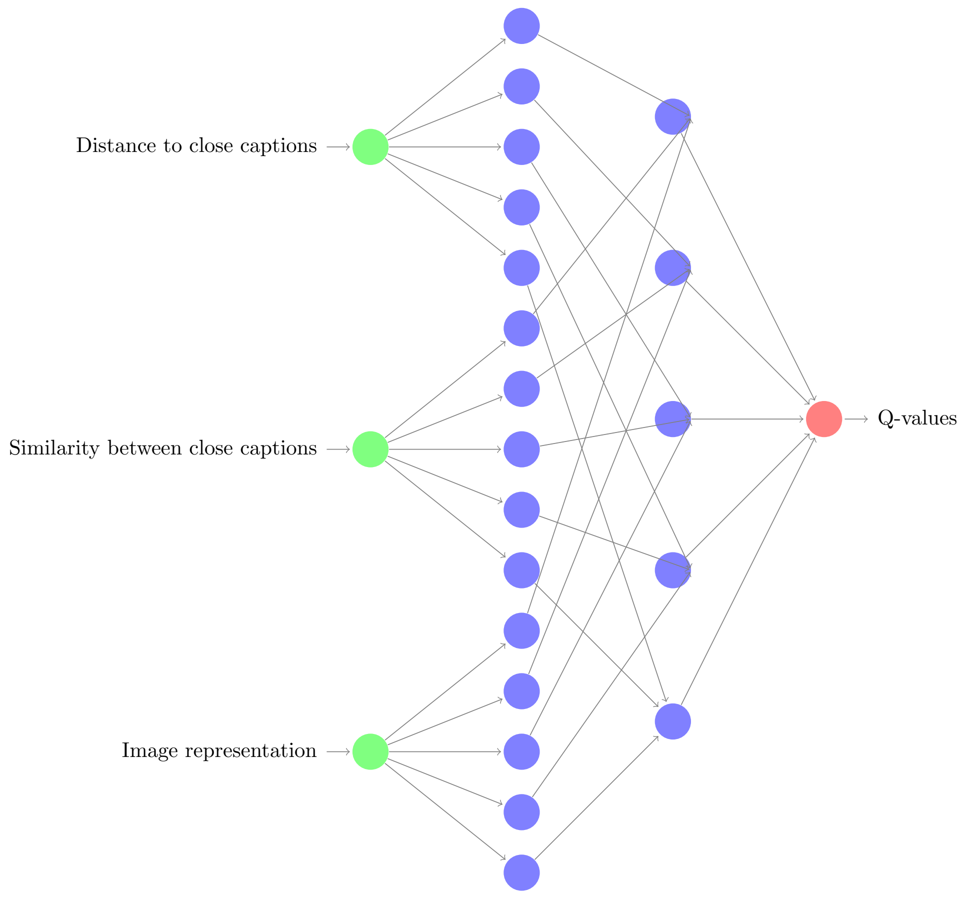


Figure 4.7: Illustration of our reinforcement learning network. Each element in the state is connected to a linear layer of equal size. The features from those linear layers are added together and input to a separate fully connected layer that downsamples the data to two values - one for a label action, and one for a discard action.

labels for samples, in order to force the agent to only request labels for samples that are of high quality. This penalty is determined empirically.

The change in performance may have different magnitudes for different datasets or applications. To increase the agent's ability to learn a general active learning query strategy, it is beneficial to make the reward equal in magnitude across datasets. By employing the same idea as in [17], a possibility is to limit the reward to the range $[-1, 1]$. Label queries that result in an increase in performance get a reward of 1, whereas a decrease in performance achieves a reward of -1. The reward for not labeling the sample still receives a reward of 0.

4.3.3 Selection radius

Each row in D is calculated with respect to a single image. However, the computational requirement of going through all images in the unlabeled pool is far too large. The state-generation procedure described in section 4.3.1 has to be done every time the agent is to output an action. Furthermore, because of the fact that the state should be representative of the most recent model, embeddings of both images and captions need to be re-computed at every label-action. To reduce the total runtime, we select a group of images at each label-action, as opposed to only including the image on which the current state was generated. The selection of images is determined by a similarity measure between the current image and all other images in the unlabeled data pool. How many additional similar images that are selected at each acquisition stage is from now on referred to as the *selection radius*, and a selection radius of 1 only constitutes the current image itself.

To calculate the similarity between images, we construct a state as explained in section 4.3.1 for all images in the unlabeled set, and calculate pairwise cosine similarity between the current state s_i and all other states. In addition to adding image i to the labeled data pool, we include the k images that had their resulting state most similar to s_i. If the model decides that state s_i is the result of an uncertain image, including images with the same uncertainty will probably lead to an increase in performance.

Furthermore, by increasing the selection radius, the resulting tuple (reward, action, next state) is one that is more representative for the current state in the stream. The model is reset before every reward calculation, and the state of the initial weights may influence the observed reward in subtle ways. Including additional samples with high similarity to the current one makes the perceived reward more representative of the current state, and reduces the chance that these minor fluctuations perturb the reward in a prominent way. Moreover, selecting a subset of the unlabeled pool at each acquisition stage makes the uncertainty of the next state change more than only selecting a single sample. The result is an agent that better captures model uncertainty.

4.3.4 Budget

The agent continues to output actions until it has reached a predefined budget. The budget sets a limit on how many labels the agent may request. When this number is reached, the environment is reset, and the agent plays the game again. Increasing the budget will cause an increase in the total time needed to train a model, but will naturally result in an increase in performance, because the agent sees more of the unlabeled data in each episode. In many real-life applications such as medical imaging, this budget is determined by economic factors.

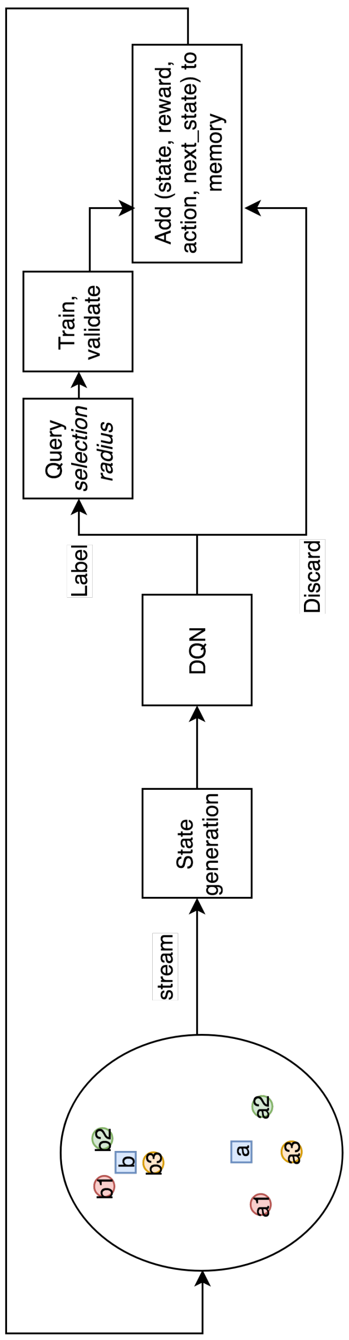


Figure 4.8: Stream-based reinforcement learning architecture

Chapter 5

Experiments and Results

In this chapter, we first describe the datasets applied in our experiments and the experiments themselves. Thereafter the results are displayed.

5.1 Hardware and frameworks

We have run our experiments on an Ubuntu 16.04.2 server with 48 cores, 64GB RAM and two NVIDIA Tesla P100 GPUs with CUDA 8.0.61. All the code and experiments have been written in Python, using the PyTorch[1]deep learning framework.

5.2 Datasets

This section describes the different datasets we chose to use in our experiments.

5.2.1 Movie Reviews

The Movie Review (MR) dataset is a perfectly balanced dataset containing 5331 positive and negative movie reviews written in English. The reviews are of variable length, and the longest sentence includes 59 words. The reviews are written by different people, and thus contains varying examples of natural language. The dataset is provided by Cornell Computer Science [2]. Examples from the dataset are shown in figure 5.1.

[1]`https://pytorch.org/` (Accessed 17.07.2018)
[2]`http://www.cs.cornell.edu/people/pabo/movie-review-data`

Sentence	Sentiment
an engaging overview of johnson's eccentric career .	Positive
if you sometimes like to go to the movies to have fun , wasabi is a good place to start .	Positive
a disturbing and frighteningly evocative assembly of imagery and hypnotic music composed by philip glass .	Positive
a visually flashy but narratively opaque and emotionally vapid exercise in style and mystification	Negative
here , common sense flies out the window , along with the hail of bullets , none of which ever seem to hit sascha .	Negative
unfortunately the story and the actors are served with a hack script .	Negative

Figure 5.1: Examples of sentences with sentiment from the MR dataset.

5.2.2 UMICH

Another dataset is provided by the University of Michigan (UMICH) class SI650. This is a perfectly balanced dataset that consists of 7086 sentences from social media. It is used for training models in sentiment analysis and contains samples with similar language and repetition, which makes it an easy-to-learn dataset. [3]

5.2.3 Flickr

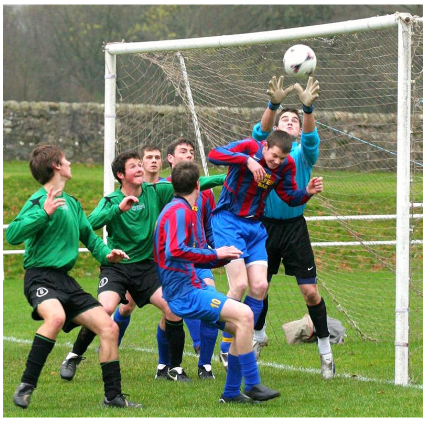
The next dataset consists of images from flickr.com, with five associated captions describing each image. The images contain multiple objects, and the captions usually describe the main and possibly one or two secondary subjects for each image. The dataset aims to train models to perform multi-label classification for either images or captions. It has been released in two sizes, Flickr8k and Flickr30k, and it is provided by the University of Illinois at Urbana, Champaign[4]. The Flickr datasets support different areas within computer vision, e.g. image segmentation and image captioning. An example image with captions is shown in figure 5.2.

5.3 Experimental Setup

This section describes the details and hyperparameters of each experiment we have performed.

[3]`https://www.kaggle.com/c/si650winter11` (Accessed on 02/7/2018)

[4]`http://shannon.cs.illinois.edu/DenotationGraph/` (Accessed on 01/6/2018)



- A goalie tries to catch a ball during a soccer game.
- A group of guys playing soccer on a field.
- Footballers are scrambling around the goal as the goal keeper reaches for the ball.
- Four men in green at a soccer goal post trying to score, while 3 men in blue try to prevent it.
- Two soccer teams converge at the goal and the goalie reaches for the ball.

Figure 5.2: Example of an image with corresponding captions from the Flickr dataset.

5.3.1 Similarity in traditional active learning

The experiment for exploiting similarity between samples uses the architecture described in section 4.1.

The labeled data pool starts empty. We perform several rounds of mini-batch active learning. In each round the model tries to select labels for 32 samples ranked by entropy scoring. This continues until we have queried labels for 250 samples, or no more labels can be queried due to high similarity threshold. The model is re-trained after each round. Similar samples in the selected subset are discarded and replaced with subsequent items further down the scoring list. This lasts until we either have a subset of 32 samples with sufficiently low similarity scores, or the entire dataset has been discarded. Samples in each selected subset are discarded if they have a similarity below a certain threshold, and we will experiment with several different thresholds, listed by *similarity threshold* in table 5.2 and 5.3. The similarity thresholds have been chosen empirically.

We will test the effects of two different ways of representing the sentence for similarity calculation, specifically *w2v* and *CNN*, as described in section 4.1.

The classifier is a CNN for text classification, based on [11]. Each sentence is embedded using w2v word-vectors, where each word-vector is of size 300. The CNN makes use of one-dimensional convolutions of size 3, 4 and 5, with 100 applications for each size. After the embedding layer, we apply a dropout of 0.2, before a ReLU activation function and a max-pooling layer is applied. These max-pooled results are concatenated to a vector of size 300 subject to a dropout of 0.4. To downsample the data to our output size we use a linear layer of size (300, 2) that produces the classification probability of each class.

The model is trained using the Adadelta optimizer, described in [35], and cross entropy loss function with learning rate 0.1 over 100 epochs. We validate the model's performance over a held-out part of the dataset after each training epoch. If the validation performance decreases while the training performance increases, we stop the training early as to not overfit the data in the training set.

An overview of the classifier hyperparameters is shown in table 5.1.

Hyperparameter	Value
batch size	32
budget	250
dropout embed layer	0.2
dropout fc layer	0.4
Filter sizes	[3,4,5]
Filter numbers	[100,100,100]
Learning rate	0.1
Loss function	Cross entropy
Optimizer	Adadelta
Train epochs	100
Word dim	300

Table 5.1: Hyperparameters for the CNN used for experiments on similarity in traditional active learning.

Hyperparameter	value
Similarity threshold MR	[0.00, 0.05, 0.08, 0.12]
Similarity threshold UMICH	[0.00, 0.08, 0.12, 0.14]

Table 5.2: Different similarity thresholds used in the experiments when using CNN as basis for inter-sample similarity

Hyperparameter	Value
Similarity threshold MR	[0.0, 0.54, 0.56, 0.58]
Similarity threshold UMICH	[0.0, 0.37, 0.39, 0.42]

Table 5.3: Different similarity thresholds used in the experiments when using average w2v-vector as basis for inter-sample similarity

5.3.2 Visual Semantic Embedding

In our experiments with active reinforcement learning, we employ a model for computing joint embeddings as described in section 4.2. The base model for computing joint embeddings is based on [4], with an implementation in Python available on GitHub[5].

The image representations are the output of the penultimate fully connected layer of VGG19, of size 4096. The caption representations are calculated using a Gated Recurrent Unit, with 1 hidden layer of size 1024. Each word in the input sentence is substituted with a w2v-vector of length 300.

The projections to the common embedding space are performed using linear layers of size 4096 x 1024 for images, and 1024 x 1024 for captions. Both the image- and the caption embeddings are normalized.

We use the Adam optimizer, described in [12], with learning rate 0.0002, and a contrastive loss as explained in section 4.2.2. The parameters included in the optimizer are the parameters of both linear projection layers and the GRU caption encoder.

An overview of the hyperparameters is shown in table 5.4.

Hyperparameter	Value
Caption encoder	GRU
Caption projection	Linear(1024, 1024)
CNN type	VGG19
Embedding size	1024
GRU hidden layer size	1024
GRU layers	1
Image encoder	VGG19
Image projection	Linear(4096, 1024)
Learning rate	0.0002
Loss function	Contrastive loss
Optimizer	Adam
Train epochs	15
word dim	300

Table 5.4: Hyperparameters used for the baseline VSE model.

5.3.3 Training RL agent

To train a reinforcement agent to make intelligent labeling decisions, we use the Flickr8k dataset described in section 5.2.3. The VSE- and RL-model are as

[5]`https://github.com/fartashf/vsepp`

described in section 4.2 and 4.3, with the VSE model using hyperparameters from section 5.3.2. The following paragraphs describe in detail the training of the RL agent in our experiment.

At the start of each episode, the VSE model is pre-trained with 224 samples in order to force the model to output a certain degree of uncertainty from the beginning. The agent used in the experiment is a traditional DQN agent that collects data and trains over 10000 episodes. Each episode lasts until the agent has queried labels for 1120 samples, or discarded the entire dataset. Selecting multiple samples at each stage (*selection radius*), makes the remaining budget decrease with *selection_radius* samples, and we make use of a selection radius of 32. The determination of the selection radius is a trade-off between performance and runtime, and 32 is the lowest threshold we were able to use while still maintaining an acceptable runtime. States for images are constructed using the 10 closest captions for each image. The reinforcement agent is trained using a regression objective, with the RMSProp optimizer with learning rate 0.01 and the Huber loss function. The agent's reward is the sum of both image- and caption-related recalls, R1, R1i, R5, R5i, R10, and R10i. The agent utilizes an epsilon-greedy exploration strategy, where a random action is selected with probability 10%. The probability of a random action selection decreases over time.

To validate the performance of the agent, we measure R1, R5, and R10 for both images and captions at the end of each episode.

The size and depth of the agent's Q-network depend on the state used in the experiment. The state in the experiment consists of three elements: The distance to the closest captions, the average distance between the closest captions and the representation of the image in the common embedding space, as explained in section 4.3.1. Each of these elements is connected to a linear layer of output size 512. These outputs are added together before connected to a final linear layer of size (512, 2), which outputs the Q-values for a label- and discard action. The DQN structure is shown in figure 4.7. The networks for the state elements are of size (1024, 512), (10, 512) and (10, 512), and the layer that outputs Q-values is of size (512, 2).

The agent has a replay memory of size 10000, and uses a separate target network to predict Q-values of the next state to determine the regression loss. The target network is updated to the policy network every 10 episodes.

An overview of the hyperparameters is shown in table 5.5.

5.3.4 Validating RL agent

If the training of the RL agent is successful in improving its query strategy, we will perform an experiment in which we measure how well the agent performs when operating on previously unseen data. After an agent is trained using the

Hyperparameter	Value
Agent	DQN with target network
Budget	1120
Episodes	10000
Epsilon	10%
Init samples	224
Learning rate	0.01
Loss function	Huber loss
Optimizer	RMSProp
Replay memory size	10000
Reward threshold	0
Selection radius	32
State linear layers	[(1024, 512), (10, 512), (10, 512)]
Target network update	10
Topk	10
Q linear layer	(512, 2)

Table 5.5: Hyperparameters used to train an RL agent to make labeling decisions on the Flickr8k dataset.

procedure described in section 5.3.3, we will test the generality of the learned policy by applying it to previously unseen data. The agent is used in a validation setting. Here it only outputs its proposed actions based on provided states, and does not learn from the environment transitions. The agent queries labels for *selection_radius* size at each acquisition stage, and the model is re-trained by using the expanded labeled data pool. When the size of the labeled data pool is equal to the budget, the performance of the model is validated over a held-out dataset. This process is repeated several times to calculate an average.

5.4 Experimental Results

In this section, we describe our results.

5.4.1 Similarity in traditional active learning

The results from the experiments are shown in figure 5.3 and 5.4. The experiment is run 10 times, and the figures show the average across all runs. The experiments have been performed using the MR and UMICH datasets.

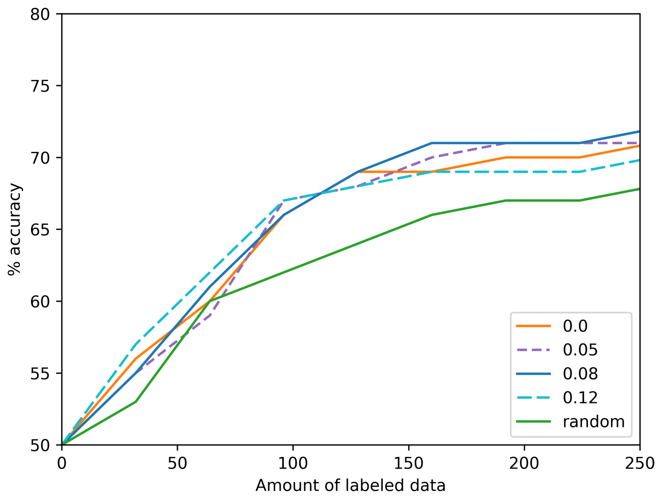


Figure 5.3: Discarding similar items on the MR dataset, using the model CNN as basis for similarity and different similarity thresholds. The line labeled 0.0 has a similarity threshold of 0.0, which means it is equivalent to traditional entropy sampling.

CNN similarity

Figure 5.3 shows that discarding similar items in the selected subset ultimately leads to an increase in performance. Moreover, we see that in the beginning, the different similarity thresholds perform comparably. However, after a few iterations of active learning, the runs with similarity threshold outperform the one without and also the random baseline.

Figure 5.4 shows similar features. The performance improvement is not prominent in the early iterations of the active learning experiment, but picks up after some time. The algorithm with similarity threshold outperforms traditional entropy and the random baseline.

w2v similarity

As for CNN similarity, the experiments using the average w2v-vector as basis for similarity calculation also outperforms both standard entropy and the random baseline. It is worth noting that the similarity thresholds with the best performance are significantly higher for the w2v experiments than those of the CNN

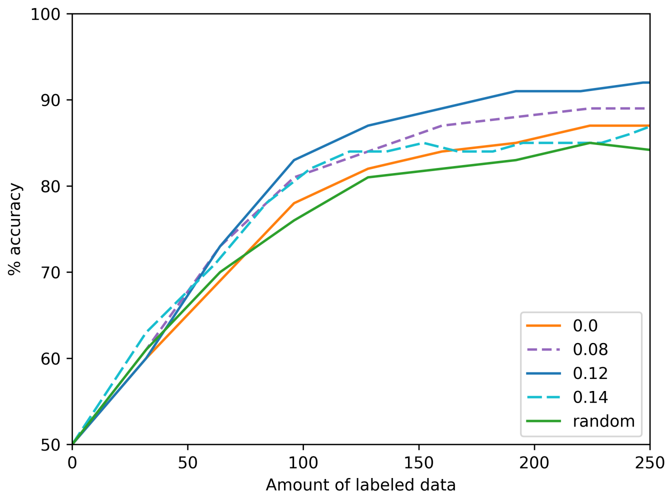


Figure 5.4: Discarding similar items on the UMICH dataset, using the model CNN as basis for similarity.

experiments.

Figure 5.5 shows the results of using w2v as basis for similarity on the MR dataset. However, as opposed to figure 5.3, we have an increase in performance from the beginning of the experiment. Moreover, there is a larger increase in performance when compared to figure 5.3.

Figure 5.6 shows the same experiment performed on the UMICH dataset. Again there is an increase in performance when using a similarity threshold, but the gain seems less significant than the others.

A summary of the results from inter-sample similarity is shown in table 5.6.

5.4.2 Training RL agent

The results from the experiments are shown in figures 5.7 - 5.11, performed using the Flickr8k dataset. It is apparent that the agent fails in learning a useful query strategy. The performance is fluctuating, but not increasing. This is also strengthened by figure 5.9, which shows that the agent does not discard a meaningful amount of samples in the stream.

Due to the fact that the agent failed in improving its query strategy in the training domain, we did not construct an experiment to validate its performance

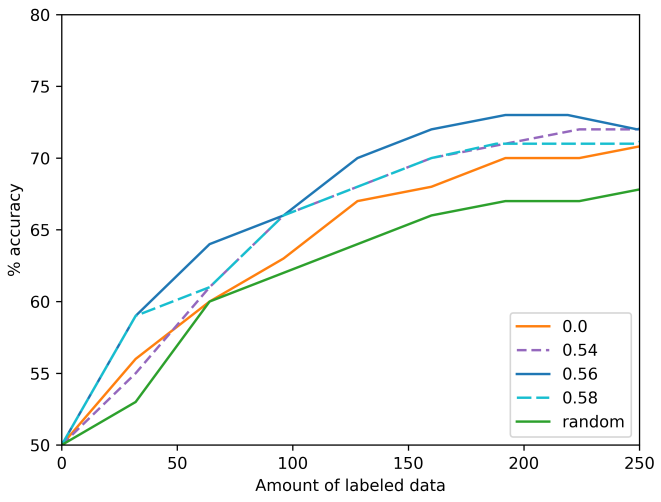


Figure 5.5: Discarding similar items on the MR dataset, using the average w2v-vector as basis for similarity.

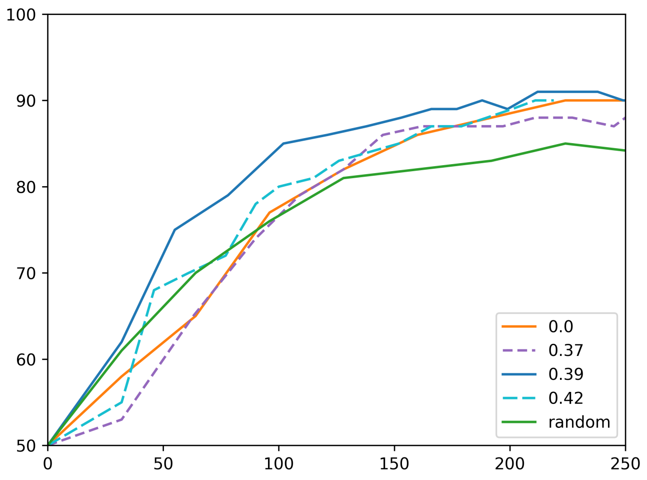


Figure 5.6: Discarding similar items on the UMICH dataset, using the average w2v-vector as basis for similarity.

Dataset-algorithm	Labeled samples	Accuracy	Relative amount of labeled samples
MR-w2v	160	72%	**62.5%**
MR-CNN	160	71%	62.5%
MR-entropy	256	71%	100%
UMICH-w2v	188	90%	**83.9%**
UMICH-CNN	192	91%	85.7%
UMICH-entropy	224	90%	100%

Table 5.6: Summary of the results from the experiments on inter-sample similarity. The result chosen for entropy is the best performing one, resulting in a worst-case calculation of the relative amount of data the architecture needs. Bold text shows the algorithm needing the least amount of data to reach or exceed the same performance as the entropy baseline.

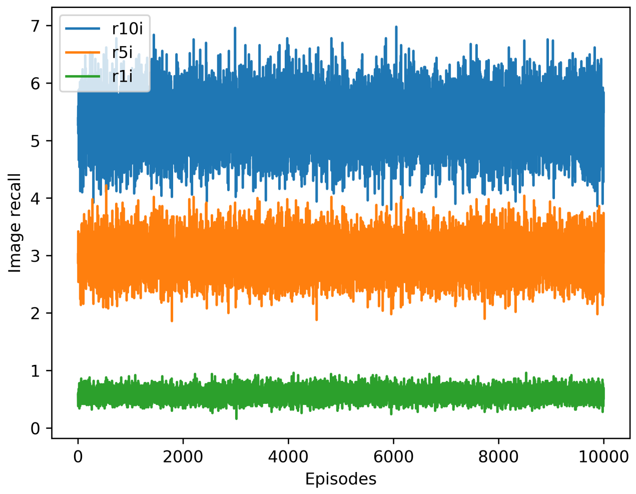


Figure 5.7: Different image recall measures at each episode end, measured over 10 000 episodes.

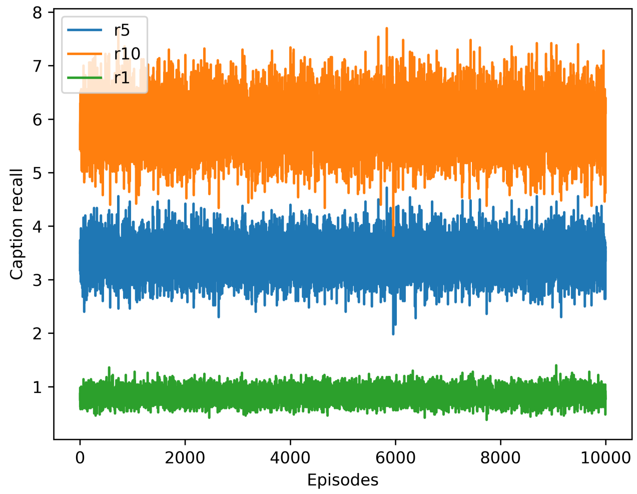


Figure 5.8: Different caption recall measures at each episode end, measured over 10 000 episodes.

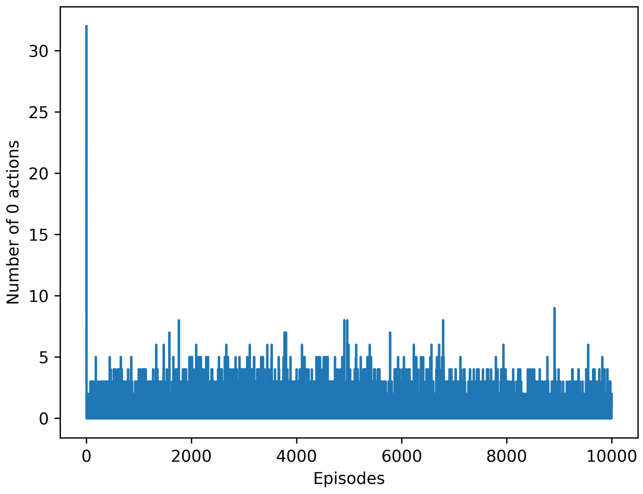


Figure 5.9: The number of discard actions output by the agent in each episode.

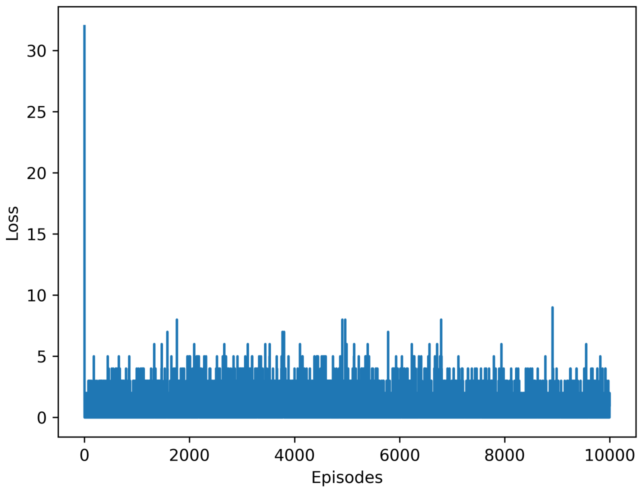


Figure 5.10: The loss over a held-out dataset at the end of each episode, measured over 10 000 episodes.

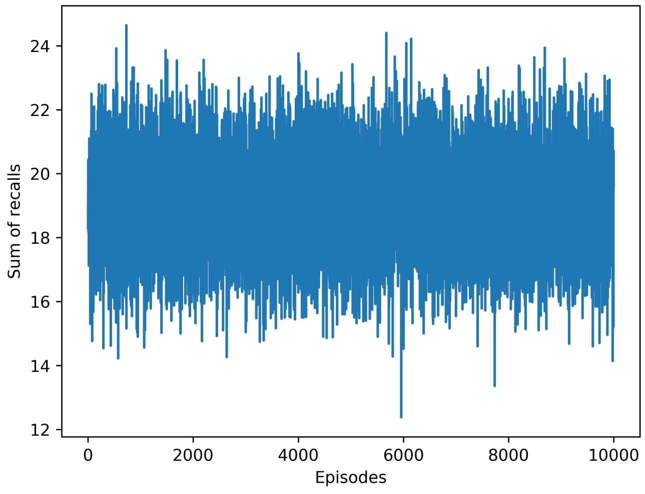


Figure 5.11: Sum of all image- and caption-related recalls at the end of each episode, measured over 10 000 episodes.

over an unseen dataset.

Chapter 6

Discussion and Conclusion

In this chapter, we discuss our results before we end with a conclusion that summarizes our findings and answers our research questions.

6.1 Discussion

In this section, we discuss the experimental results presented in the previous chapter.

6.1.1 Similarity in traditional active learning

As seen in section 5.4.1, not selecting samples with pairwise similarity above a given threshold improves the rate of accuracy of the model.

Delay in performance increase

In section 5.4.1 we described how the gains of discarding similar items did not reveal itself before after a couple of iterations. The basis for calculating similarity in the experiment was the output of the classifier used in the experiment, and is the reason for this delay in performance increase. The model starts with no labeled training data, and its weights are initialized randomly. As a consequence, the representation output of the model does not constitute a representative vector for the input sample. With no labeled training data, the model is not able to capture the differences that are of importance, and the resulting representation is not one that captures the meaning of the sentence in a sufficient way. However, after a couple of iterations, the performance increase of discarding similar items

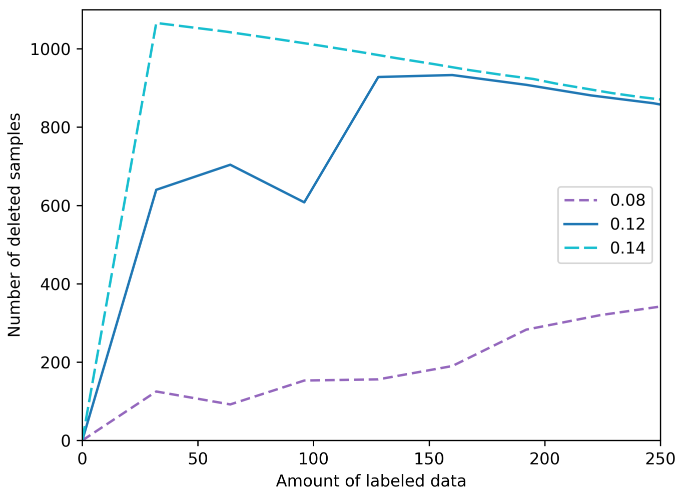


Figure 6.1: Number of discarded samples on the UMICH dataset when using CNN as basis for similarity.

is more prominent because the model now has been trained on labeled data and is able to capture the meaning of a sentence in a more satisfying way.

This delay in performance is not present when using the average w2v-vector as basis for sample similarity, shown in figure 5.5 and 5.6. Section 2.4 explains how the word vectors are pre-trained to capture the meaning of words, and this is why we see the immediate benefit of using intra-sample similarity to discard similar samples when using w2v as basis for similarity.

Effect of similarity threshold

Figure 5.3 shows the results when using several different similarity thresholds, using a CNN representation on the UMICH dataset. Intuitively, a higher threshold should yield a larger increase in performance. Selecting a more diverse subset at each acquisition stage is expected to increase the performance, and to further analyze the effects of the similarity threshold, we recorded the amount of discarded samples at each acquisition stage. The numbers are shown in figure 6.1.

As expected, using a higher similarity threshold makes the model discard a larger amount of samples due to fewer samples having sufficient distance from each other. However, the experiment with the largest similarity threshold does not achieve the best performance when compared to other similarity thresholds.

Moreover, the similarity threshold with the best performance changes as we add more labeled data. In the beginning, higher similarity thresholds perform better. This is because, as previously mentioned, the basis for similarity is the CNN output itself. In the beginning, the representations of sentences are more alike, and a larger similarity threshold makes the model select a more diverse subset in the selection stages.

As time progresses, the benefits of the higher similarity threshold experiments diminish. Similarity threshold 0.14 moves from being the top performer to performing the worst after 140 labeled data samples have been added. After 165 it is even outperformed by traditional entropy sampling. Recall that the samples are first scored using entropy, and then selected and possibly discarded due to pairwise similarity. If discarded, the lowest-scoring sample of the pair is replaced with the next sample according to its entropy score. If the similarity threshold is too high, many high-entropy samples will be discarded due to pairwise similarity, and they will be replaced with samples with lower entropy score, but with sufficient pairwise similarity. The result is that at the end of the cleaning process, many of the remaining samples in the selected subset have low entropy, which in turn is detrimental to the model's performance. The problem lies in the fact that the representations, and thus the best-performing similarity threshold, changes as the model receives more labeled training data, and motivates a time-dependent similarity threshold.

The same is not the case when using w2v representations for calculating inter-sample similarity. The pre-trained word vectors do not change as we add labeled data to the model. This is why there is more consistency in which similarity thresholds perform the best across the entire timeline.

The determination of similarity threshold in the experiments was done empirically, and has remained fixed throughout the experiment. The previous discussion describes how this may lead to sub-optimal performance, and motivates further research on an automatically determined similarity threshold that is re-evaluated at each time step.

Dataset comparison

When comparing the results for the MR dataset, figures 5.3 and 5.5, and the results for the UMICH dataset, figures 5.4 and 5.6, we see some differences. First of all, the similarity threshold is different for the two datasets and both basis for similarity calculation. As explained in section 5.2, UMICH is a dataset that contains sentences of very similar language and wording. As a consequence, the representation of the sentences in that dataset is more similar, and it is therefore necessary to increase the similarity threshold.

Furthermore, there is a difference in the relative performance increase for the two datasets in our experiments. This may also be attributed to the difference in the datasets themselves. Due to the simpler nature of the UMICH dataset, the model manages to generalize to the different classes in a shorter amount of time. If one instance of one class is very similar to another, the model will only need a few samples of that class to be able to generalize. As for the MR dataset, the model needs several samples for each class to be able to generalize for that class. Hence, the possible performance gain for using similarity-based active learning is larger for heterogeneous datasets.

6.1.2 Reinforcement Learning for active VSE

As seen in figures 5.7 and 5.8, the agent fails in learning which states lead to high rewards. Here we discuss possible reasons for the agent's inability to improve its query strategy.

In section 4.3.1 we described the different elements included in the state generation procedure. The reward used in the experiment was the increase in the sum of recalls measured over a held-out dataset. There must be some correlation between what the agent receives as a reward and the state upon which it calculated its action. This is to make it possible for the agent to learn the expected long-term future reward for various states. We will now explore possible reasons to why these may not be as correlated as initially theorized.

Distance to closest captions

One element of the state generation procedure was the numerical distances to the closest captions, on which a softmin function was applied to make the internal differences between the distances more prominent. The softmin function turns the distance distribution over the closest captions for a given image into a probability distribution. When framing the problem as an n-way classification problem, this probability distribution may be analyzed using common uncertainty sampling techniques, and was our initial intuition as to why this should be included in a state generation procedure. When defining the reward function as the increase of the sum of recalls, the de-correlation becomes more clear. When presented with a probability distribution over the distances to the closest captions, the agent has no ability to reason about anything but R1. Intuitively, a state

$$s_0 = [0.9, 0.025, 0.025, 0.025, 0.025]$$

will probably have its corresponding caption as the closest one among the closest captions because of the high relative proximity to the closest caption. On the other hand, a state

$$s_1 = [0.35, 0.35, 0.1, 0.1, 0.1]$$

will probably not have its caption as the closest one, due to the small difference in distance to the closest and the second-closest caption. This pattern is easily learnable, and is known as *margin sampling.* Margin sampling is simply the difference between the two most probable labels for a given sample, and is inversely proportional to model uncertainty.

Reasoning about anything but recall at one, however, is a more difficult task. If it turns out that in fact neither s_0 nor s_1 have had its correct caption as the closest one, which one would be more likely to have it in the closest 10? There is no intuitive combination of numbers that with a higher probability results in a positive recall measure for anything but R@1. By looking at the distances to the closest captions, the agent has no way of reasoning about the probable value of labeling the corresponding sample.

Using embedding space as image representation

Another part of the state generation procedure is the representation of the sample. This is done to increase the reinforcement agent's ability to reason about the quality of the current data sample. There may be parts of the joint embedding space that are more difficult to learn than others, and since our goal is to reach a high accuracy as fast as possible, these parts of the space should be avoided. Including a representation of the image in the common embedding space, should thus assist the agent in learning this.

The results in section 5.4.2 show the agent's performance after 10 000 episodes. During each episode, the agent selects a subset of the unlabeled data given its selection radius. This subset of data is added to the labeled pool and the model is re-trained. To make the state generation at the next acquisition stage accurate, the states for all remaining samples in the unlabeled pool have to be re-computed. The state generation procedure uses characteristics from the joint embedding space, and when the model is trained on new data, the mappings to the embedding space changes. This necessitates re-computing states for all unlabeled samples after each acquisition.

Imagine the state generated for one image at one specific acquisition stage in one specific episode. This state depends on the state of the linear projection layers, and the state of the projection layers depends in turn on what data has been previously labeled. If the agent chooses to label the image responsible for that state, it will receive a reward. Imagine then, in a different episode, at a different acquisition stage, the state generated for the same image. Due to the fact that the generated state, is strongly coupled to the previously added labeled data, the resulting state for the same image may be completely different.

Following the same argument, the reward may also be described as tightly coupled to the labeled data already selected. Imagine at one point in time, the agent saw a representation of an image in the joint embedding space, selected the sample, and received a positive reward. Due to the selection radius, the agent picks several images with a state closely resembling the current one, and add them to the labeled pool. The model now has access to several samples that lie in close proximity to one another in the same part of the embedding space. In the future, if the agent sees a state similar to the one previously described, it will probably select the same action as before, due to the previously received positive reward. However, now that the model already has several samples in that part of the space, the reward is probably different than before, and may confuse the agent.

Distance between closest captions

We initially believed that inter-caption distance was the most important factor for defining salient active learning features for the reinforcement agent. To test this, we redefined the VSE problem to be a traditional active learning scoring function problem, where the scoring function is the average inter-caption distance, as described in section 4.3.1. At each acquisition, we added the 32 images with corresponding captions that have the largest average inter-caption distance between their closest 10 captions. After each acquisition, the model is retrained and performance is measured before a new acquisition begins. This repeats until we reach a budget of 640 images, which makes 640 images $*$ 5 captions for each image $=$ 3200 total samples. The results are shown in figures 6.2, 6.3 and 6.4.

The results show that average intra-caption similarity is not as important as we first assumed. The active learning scoring function based on inter-caption similarity is consistently outperformed by random sampling. This is a possible reason why the reinforcement agent fails in improving its query strategy over the episodes.

Despite the fact that the naive scoring method approach previously described does not yield positive results, the notion of including inter-caption distance in the state provided to a reinforcement agent may still be a rational and beneficial choice. When used as a scoring function, the score is merely the average inter-caption distance between the closest captions. However, there may be a function that more accurately captures the salient relationships between the captions. Intuitively, the closest captions and the distances between them should shed some light on the expected performance gain of labeling that particular image, and we are not absolutely disregarding this idea.

Effect of selection radius

Including several samples at each selection stage substantially reduces the total run-time of the algorithm. However, there is a problem with the formulation as it is now. The reasoning behind using reinforcement learning as an active learner is

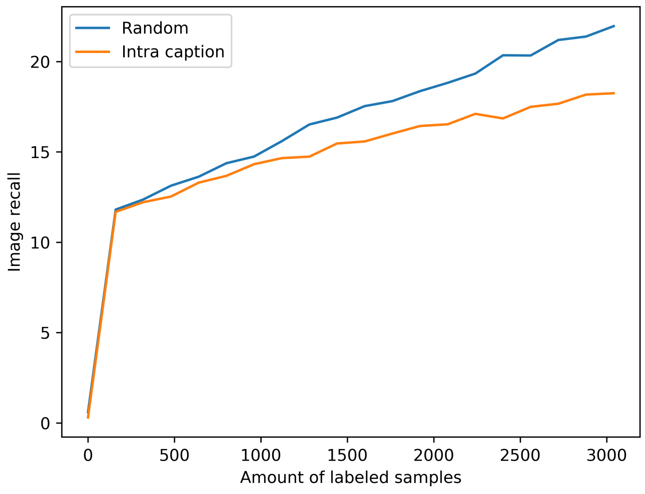


Figure 6.2: Sum of image recalls when using average inter-caption distance over closest captions as a scoring function, measured after each acquisition.

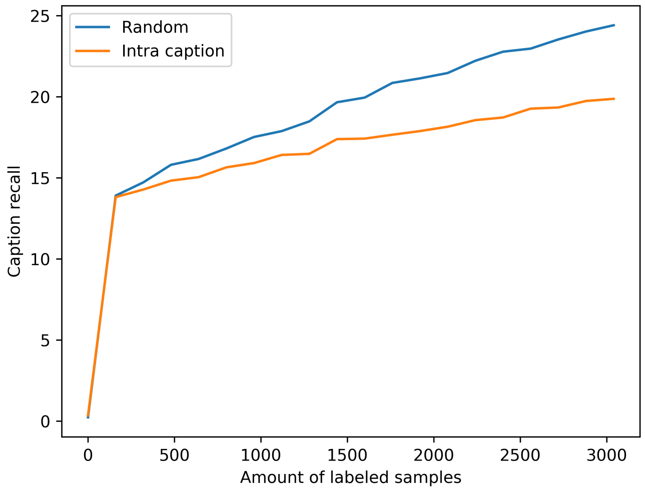


Figure 6.3: Sum of caption recalls when using average inter-caption distance over closest captions as a scoring function, measured after each acquisition.

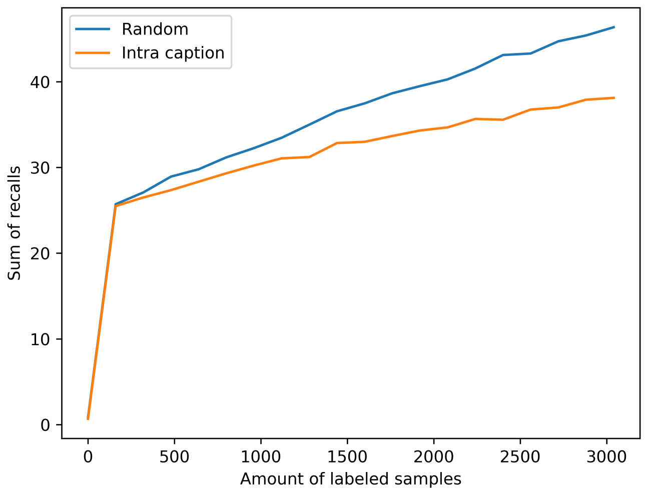


Figure 6.4: Sum of all recalls when using average inter-caption distance over closest captions as a scoring function, measured after each acquisition.

that the agent is better at finding intricate relationships between the input state and the reward, and also at establishing connections that are incomprehensible for human beings.

In our experiments, in addition to adding the current image, we also add the images that have their resulting state most similar to the one constructed for the current image. This similarity has been calculated using cosine similarity. In doing this, we are essentially incurring a bias as to what samples are similar to the current one. Including samples based on cosine similarity may be detrimental to the agent's performance, because it is not guaranteed that cosine similarity is a good measure of similarity when having the agent's performance in mind. Even though the agent has learned the complex relationships between the features in the input state, including other samples based on a predefined similarity measure could de-correlate the relationship between the input state and the received reward. This may be a possible reason to why the agent fails in improving its selection strategy over numerous episodes.

Choice of reward function

Section 4.3.2 describes the reward function used in our architecture. Here we discuss the possible consequences of choosing the total sum of recalls as the reward presented to the reinforcement agent.

As described in section 4, we need to select one domain as a primary domain, and compute characteristics in the common embedding space based on that. This may be framed as an n-way classification problem, where the classes are the possible corresponding items in the secondary domain. This simplifies the process of state generation and makes it easier to compute features that probably encapsulates model uncertainty.

Nonetheless, defining features when selecting one domain as a primary domain may have consequences as to how the reward function should be defined. The goal of active learning is to push decision boundaries apart using as few samples as possible, and the choice of reward function should be correlated with which decision boundaries are possible to separate, given the current state. Our theory is that the state constructed by using images as the primary domain is only correlated with the decision boundaries between different captions and not between different images. We have defined different performance measures with basis in retrieving the correct caption for a given image (caption recall), and retrieving the correct image for a given caption (image recall). We assume that a possible explanation of the results is that it is erroneous to include image recall related performance measures in the reward function. This motivates to work on alternative reward functions and is a possible direction for future research.

6.2 Conclusion

In this master thesis, we have experimented with active learning and the ultimate goal of general active learning. Based on the results and discussion in the previous chapters we summarize our findings in relation to our research questions.

Research question 1 *How may similarity between samples be utilized in order to improve on existing active learning techniques?*

Traditional active learning techniques are naive in that they neither perform any calculations on the relationships between the items of the selected subset, nor between the selected subset and the previously selected items, at each acquisition stage. We have proposed an architecture that leverages this information, in which the model discards samples that are too similar to one another, given a pre-defined threshold. The architecture has been tested on datasets with different characteristics, and proven to yield results outperforming traditional and commonly used active learning strategies. These results underpin the relevance of using similarity-based information in active learning. Similarity is moreover information that is inherently indifferent of domain and application, and reinforces its relevance for future work towards a general active learning strategy.

We therefore conclude that a simple, yet effective, approach of discarding samples with similarity above a certain threshold, leads to a substantial increase in performance when compared to traditional active learning strategies.

Research question 2 *How may uncertainty be captured in Visual Semantic Embedding?*

Section 4.3.1 explains in different ways how uncertainty may be defined in VSE and joint embeddings in general. The proposed architecture leverages, among other things, distances to and between the closest assignments for a given image. We provide an intuitive explanation of why we believe these features should be representative of model uncertainty in the joint embedding space. To the best of our knowledge, there is no other work on defining active learning techniques applicable to VSE, and we provide insight into how such algorithms may be designed.

Research question 3 *By applying factors that encapsulate model uncertainty in Visual Semantic Embedding, will a reinforcement agent be able to learn which states lead to high rewards?*

Our experimental results show, contrary to our initial hypothesis, that the reinforcement agent fails in learning anything useful with respect to model performance, uncertainty, and reward. In section 6.1 we discuss the results and possible explanations of why this is the case. Even though the features described in section 4.3.1 make intuitive sense, reinforcement learning is very sensitive to its input state. We provide insight into why the agent's query strategy fails to improve.

6.3 Contributions

This thesis proposes an architecture that makes use of similarity information with the aim of improving traditional active learning scoring strategies. The experimental results show that the elimination of similar samples yields an increase in performance. This is, to the best of our knowledge, the first work on using similarity information in this way.

Furthermore, active learning and uncertainty in VSE both represent areas which are complicated to define. Nonetheless, in section 4.3.1 we provide an intuitive proposition to how this may be done. We also propose an architecture that may leverage such information to make informed and intelligent active learning query decisions.

Finally, we present an analysis of the effectiveness of the proposed RL architecture. The results show that our approach does not succeed in learning a usable query strategy, and we provide a qualitative analysis to each component of the state generation procedure to shed light on why the agent is unsuccessful. Even we did not succeed in learning a general active learning strategy for joint embeddings, we believe the approach has potential and may yield results in the future.

Our main contributions are as follows:

1. An effective approach for exploiting similarity between samples in order to improve on existing active learning techniques

2. Insight to how to encapsulate model uncertainty in visual semantic embedding

3. An attempt to utilize model uncertainty to train a reinforcement agent to make labeling decisions in visual semantic embedding

4. A qualitative analysis of the effectiveness of the proposed approach

6.4 Future Work

During the work on this thesis, we encountered several alternative approaches that warrant further exploration. This section describes interesting directions for future work.

6.4.1 More advanced representations for similarity calculation

Section 4.1 described two possible ways of representing samples for calculating cosine similarity. A third option is to apply the representational power of autoencoders. An autoencoder is a type of neural network that has the objective of reproducing its input. It is then possible to extract a different, more condensed representation of the input at either of the hidden layers of its network that may represent the input sample in a better way. Being able to reconstruct the sentence as its output, is proof that the representation from the hidden layers actually contains useful information. Instead of applying the average word vectors or CNN representations directly, it is possible to choose a down-sampling autoencoder to calculate a more compact representation, and use this as basis for similarity. This, as with other representations, is grounds for future work on similarity in active learning.

6.4.2 Automatic determination of similarity threshold

Section 6.1.1 describes the effect of the similarity threshold, and how there is a trade-off between high performance and discarding all unlabeled data. Furthermore, when using a representation of samples that change as we add more samples to the labeled data pool, the threshold should be reevaluated at each stage. A natural question is then how to determine the threshold. In our experiments, the threshold has been determined empirically through a trial-and-error approach. However, to attain our goal of moving towards a general active learning strategy, the threshold should be automatically determined. One simple idea is to define the threshold as percentage-based, where the model may discard a percentage of the selected subset according to their similarity. Other ideas for automatic determination of similarity threshold are valid directions for future work.

6.4.3 More advanced similarity measures

This thesis has proven that using similarity information in active learning improves traditional active learning techniques. This information may also be used in a reinforcement active learning for VSE setting, where a similarity measure can be calculated between the sample to-be-added, and the already added samples in the labeled data pool. However, as previously mentioned, by using cosine similarity or other pre-defined similarity measures, we are defining how a similarity function should operate. These definitions of similarity may not coincide with what the reinforcement agent deems important, and the notion of using more advanced similarity measures is an interesting direction for future work.

6.4.4 Agents with memory

The previous subsection elaborates more advanced measures of similarity. We believe that the relationships between the samples selected at each acquisition stage are interesting, and furthermore, the relationships between the samples selected and the existing labeled data pool should be explored as well. If constructing a state that has the intention to capture model uncertainty, our hope is that the predicted model uncertainty for samples similar to what the model has seen before is lower than for unseen samples, and thus results in the sample being discarded. However, as explained above, the state may be insufficient in capturing the significant factors for this, and needs help in capturing the relationships between current and previously seen data. Moreover, simple similarity measures such as cosine similarity may not be an accurate measure when the goal is to calculate similarity with the intention of maximizing reward.

An interesting approach would be to have a separate reinforcement agent with memory that also predicts the long-term values of labeling the current sample.

The state provided to that agent may be the same state as explained in section 4.3.1, or simply the representation of the sample itself. Adding samples similar to what has already been added should be less beneficial than adding samples that better represent the data distribution. A reinforcement agent with memory should be able to remember what samples have already been added, and adjust its predicted values accordingly.

The decision of whether or not to label the current sample in the stream may then be a joint formulation of predicted values from the two reinforcement agents. A possibility for future work is to utilize a separate reinforcement agent to reason about label decision history.

6.4.5 Alternative grouping mechanisms

Section 4.3.3 described the notion of selection radius, and how it is required in order to reduce the runtime to an acceptable length. To summarize, whenever the agent outputs a label-action, we also add the images and their corresponding captions that have a state most similar to the current one. Similarity is calculated using common similarity measures, e.g. cosine similarity.

By adding additional images based on cosine similarity between states, we are essentially stating that all features in the state-vector are equally important. This is due to the fact that in cosine similarity, the different elements in the vectors are weighted the same. On the other hand, when a reinforcement agent is presented with a state, it performs complex and intricate calculations in its neural network and may give more attention to certain factors and relationships that are not effectively captured by cosine similarity. By including images based on cosine similarity between states, we may in reality be adding a bias to the reward seen by the agent, and may counteract the calculation done in its network. This may lead to an unstable approximate reward function.

Thus, an alternative grouping strategy makes for an interesting take on future work. Chapter 3 describes [33], in which such a grouping mechanism is employed. Contrary to our architecture, where the agent sees one image at the time and multiple are added by cosine similarity, the authors of [33] first split the unlabeled dataset into k subsets, where k is a hyperparameter. Then the reinforcement agent then makes decisions on which subset to label at each time step. This is one example of alternative grouping mechanisms and should be further explored in the future.

6.4.6 The Hungarian Algorithm

The Hungarian algorithm is an algorithm that solves the bipartite matching problem, more commonly known as the assignment problem, in polynomial time. The assignment problem consists of agents and tasks. Each agent can perform any task with a specific cost, and the problem may be written as an assignment matrix where $x_{i,j}$ is the cost of assigning agent i to task j.

A solution to the problem is an assignment that assigns one agent to one task. The total assignment cost is the least possible cost for that problem.

In section 4.3.1 we described the process of state generation for the current sample in the stream. The state was constructed using the similarity matrix D consisting of pairwise distances between images and captions, repeated here for readability:

$$D = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,n} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & \cdots & \cdots & \cdots & x_{n,n} \end{bmatrix} \tag{6.1}$$

D is identical to the assignment matrix used in the Hungarian Algorithm, and thus a solution to this may be found. Our datasets have a one to one relationship, where each point in the dataset consists of one image and one caption. An optimal assignment of images to captions captures the model's predictions in an appropriate manner and exploits the fact that a label cannot belong to multiple captions simultaneously.

Instead of using each row in D as a basis for state-construction and stream-based active learning, an alternative approach is to construct an optimal assignment using the Hungarian algorithm over D. This assignment can be used to make the agent reason about which entries in D should be fixed (queried).

However, to use the solution of the Hungarian algorithm in a deep learning pipeline, the algorithm itself needs to be differentiable. In its natural state, this is not the case. Several projects have tried to use neural networks to approximate an affordable, but non-optimal, solution of the Hungarian algorithm [3], [16]. The approximate solution may then be used in the way described in the previous paragraph. This may be a feasible approach, despite the fact that the approximate solution is not an optimal one. If the approximate solution yields a fairly low-cost assignment, the total performance of the approach should still be beneficial.

In the early stages of this thesis, we experimented with an approximate, differentiable solution to the Hungarian algorithm to utilize the approach described above, but we were not successful in achieving acceptable results. An alternative approach could incorporate a differentiable solution to the Hungarian algorithm to make a reinforcement agent query labels for samples and fix entries in the assignment matrix.

Bibliography

[1] Anonymous. "Meta-Learning Transferable Active Learning Policies by Deep Reinforcement Learning". In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=HJ4IhxZAb.

[2] Tianping Chen and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems". In: *IEEE Transactions on Neural Networks* 6.4 (July 1995), pp. 911–917. ISSN: 1045-9227. DOI: 10.1109/72.392253.

[3] Patrick Emami et al. "Machine Learning Methods for Solving Assignment Problems in Multi-Target Tracking". In: *CoRR* abs/1802.06897 (2018). arXiv: 1802.06897. URL: http://arxiv.org/abs/1802.06897.

[4] Fartash Faghri et al. "VSE++: Improved Visual-Semantic Embeddings". In: *CoRR* abs/1707.05612 (2017). arXiv: 1707.05612. URL: http://arxiv.org/abs/1707.05612.

[5] Meng Fang, Yuan Li, and Trevor Cohn. "Learning how to Active Learn: A Deep Reinforcement Learning Approach". In: *CoRR* abs/1708.02383 (2017). arXiv: 1708.02383. URL: http://arxiv.org/abs/1708.02383.

[6] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. "Deep Bayesian Active Learning with Image Data". In: *arXiv preprint arXiv:1703.02910* (2017).

[7] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: ttps://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[8] Mohit Iyyer et al. "A Neural Network for Factoid Question Answering over Paragraphs." In: *EMNLP*. 2014, pp. 633–644.

[9] Bekir Karlik and A Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.

[10] Andrej Karpathy and Li Fei-Fei. “Deep visual-semantic alignments for generating image descriptions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137.

[11] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *CoRR* abs/1408.5882 (2014). arXiv: 1408.5882. URL: http://arxiv.org/abs/1408.5882.

[12] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[13] Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. “Multimodal neural language models”. In: *International Conference on Machine Learning*. 2014, pp. 595–603.

[14] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. “Unifying visual-semantic embeddings with multimodal neural language models”. In: *arXiv preprint arXiv:1411.2539* (2014).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[16] Said Medjkouh, Bowen Xue, and Ghouthi Boukli Hacene. “Sparse Clustered Neural Networks for the Assignment Problem”. In: *COGNITIVE 2017 : The Ninth International Conference on Advanced Cognitive Technologies and Applications*. Athènes, Greece, Feb. 2017, pp. 69–75. URL: https://hal.archives-ouvertes.fr/hal-01513228.

[17] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[18] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. “Acoustic modeling using deep belief networks”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22.

[19] Scott Drew Pendleton et al. “Perception, planning, control, and coordination for autonomous vehicles”. In: *Machines* 5.1 (2017), p. 6.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[21] Tom Schaul et al. “Prioritized Experience Replay”. In: *CoRR* abs/1511.05952 (2015). arXiv: 1511.05952. URL: http://arxiv.org/abs/1511.05952.

[22] Burr Settles. “Active Learning Literature Survey. 2010”. In: *Computer Sciences Technical Report* 1648 (2010).

[23] C. Shannon. “A mathematical theory of communication”. In: *Bell system technical journal* 27 (1948).

[24] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[25] Richard Socher et al. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing.* 2013, pp. 1631–1642.

[26] Donald F. Specht. “Probabilistic neural networks”. In: *Neural Networks* 3.1 (1990), pp. 109–118. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(90)90049-Q. URL: http://www.sciencedirect.com/science/article/pii/089360809090049Q.

[27] Liwei Wang et al. “Learning two-branch neural networks for image-text matching tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[28] Jørgen Wilhelmsen and Bjørn Hoxmark. *Active Deep Learning.* https://github.com/hoxmark/Deep_reinforcement_active_learning/blob/master/selection_strategies/prethesis.pdf. [Online]. 2017.

[29] Ronald J Williams. “The logic of activation functions”. In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1986), pp. 423–443.

[30] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn.* 8.3-4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[31] Travis Williams and Robert Li. “An Ensemble of Convolutional Neural Networks Using Wavelets for Image Classification”. In: *Journal of Software Engineering and Applications* 11.02 (2018), p. 69.

[32] Mark Woodward and Chelsea Finn. “Active One-shot Learning”. In: *CoRR* abs/1702.06559 (2017). arXiv: 1702.06559. URL: http://arxiv.org/abs/1702.06559.

[33] J. Wu, L. Li, and W. Y. Wang. “Reinforced Co-Training”. In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.06035 [cs.CL].

[34] Y. Yang and M. Loog. “Active Learning Using Uncertainty Information”. In: *ArXiv e-prints* (Feb. 2017). arXiv: 1702.08540 [stat.ML].

[35] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[36] Rong Zhang and A. I. Rudnicky. “A New Data Selection Principle for Semi-Supervised Incremental Learning”. In: *18th International Conference on Pattern Recognition (ICPR’06).* Vol. 2. 2006, pp. 780–783. DOI: 10.1109/ICPR.2006.115.