Jørgen Wilhelmsen and Bjørn Hoxmark

Active Deep Learning

TDT4501 Specialization Project, Fall 2017

Under the guidance of Massimiliano Ruocco

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology and Electrical Engineering

Abstract

In this paper, we investigate different state-of-the-art approaches to active learning for deep neural text classification. One of the major drawbacks of deep learning is the amount of labeled data required for acceptable performance. This labeled data can be difficult and expensive to obtain; numerous hours of manual labor must be spent to acquire and label data for a given task. The goal of active learning is to reduce the amount of labeled data needed for a model to achieve acceptable performance. This can bring benefits both in computational power in addition to reducing the cost associated with gathering enough labeled data. In this paper, we explore and implement uncertainty- and parameter-based active learning algorithms, and experiment on their effects on different deep neural network architectures. We see that the active learning algorithms perform better on a heterogeneous dataset where the unique information gained from each sample is significant, and perform worse on datasets which are homogeneous in nature. We explore the labeled dataset distribution selected by an active learner, and see how an active learner acts when applied to an unbalanced dataset. From this we draw the conclusion that active learners can act as equalizers, selecting a balanced dataset better suited for the model. The source code for the project is available at GitHub. [1]

[1] `https://github.com/hoxmark/TDT4501-Specialization-Project` (Accessed on 13/12/2017)

Preface

This paper is the result of an artificial intelligence specialization project done in the fifth year of a computer science master program. It is meant to give the authors a better understanding of deep learning in general, and to explore the state-of-the-art within deep active learning. The project will result in a foundation for a following master thesis. It has been conducted at the AI Lab at Norwegian University of Science and Technology (NTNU) in Trondheim, Norway.

We would first and foremost like to thank our supervisor, Dr. Massimiliano Ruocco for his excellent guidance throughout the entire semester. He has gone far beyond his required capacity as a supervisor at NTNU, and we would not have achieved the same without him. In addition we would like to thank Telenor for providing the AI Lab as a central hub for all AI students, both as a meeting place for theoretic discussions and as an opportunity to meet like-minded people; not only students, but also people from the industry.

Jørgen Wilhelmsen and Bjørn Hoxmark
Trondheim, December 14, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Deep learning has achieved amazing results in the last few years. In acoustic modeling [1], the error rate was improved from 24,4% to 20,7% by replacing Gaussian mixture models with deep neural networks. In sentiment analysis [2], accuracy was improved from 80% to 85.4% using a recurrent neural network. Question answering [3] saw an improvement with deep neural networks. Deep neural architectures consistently outperform other machine learning techniques, and it is no wonder why deep learning has obtained such attention.

One of the disadvantages with deep learning is the amount of data required to train the models. Active learning 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 supervised learning is a costly process. An example is ImageNet [4] which 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 by a small percentage on this scale would mean significant economical and computational benefits. Another is Deep Bayesian Active Learning with Image Data [5], where a 5% test error on the MNIST dataset [6] was achieved with active learning using 295 labeled images. On the contrary, 835 labeled images were required to achieve the same percentage using random sampling.

1.2 Goals and Research Questions

Our goal for this project was to explore and implement the state-of-the-art within active learning, and experiment with how active learning affects model performance when applied to different neural architectures, datasets and selection sizes. In addition to obtaining a deeper theoretic understanding of deep active learning in general, a central goal for our project was to gain practical experience with implementing and training deep neural networks using active learning.

This project will implement different deep learning architectures in combination with multiple active learning techniques. Furthermore, we will use the implementation on datasets different in nature. A heterogeneous dataset contains sentences written by different people, and each sentence will be written in a natural language quite different from another. On the contrary, a homogeneous dataset is a dataset with sentences written in the same natural language. Based on this, we will seek the answer to the following research questions:

- Will active learning algorithms be beneficial for different types of deep neural networks?
- How will active learning algorithms behave when applied to datasets of heterogeneous and homogeneous nature?
- How will active learning algorithms select samples when applied to unbalanced datasets?

1.3 Research Method

Our research method was a mix between a theoretical and experimental method. To achieve our goals of obtaining a deeper understanding within the field of active learning, we have studied recent research papers focusing on different aspects of our topic. After this introductory period where the authors gain knowledge about deep active learning, the research method transitioned into an experimental one, where we selected a basis for implementation as grounds for answering our research questions.

1.4 Contributions

This paper extends the work of [7]. The contributions in this paper are

- An implementation of a RNN for text classification and comparative study of active learning for CNN and RNN architectures

- A study of the effects active learners' selection size have on model performance
- A quantitative analysis of the effects of active learning on heterogeneous, homogeneous and datasets with high similarity
- A comparison of how different active learning algorithms select samples when applied to unbalanced datasets compared to the baseline

1.5 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 the models and hyperparameters we employed. Chapter 5 starts with our experimental plan and setup. Thereafter, we describe the datasets we made use of and some key characteristics of each dataset. The chapter is closed off with describing results and improvements we have done during the developmental process, before giving an overview of our final results. In the last chapter we discuss these results on a theoretical basis, before we clarify our conclusions and make suggestions for future work.

Chapter 2

Background Theory

This chapter involves basic background theory needed to follow the discussion and proposed theories for the rest of the paper.

2.1 Artificial Neural Networks

Artificial Neural Networks 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 dendrites of many other neurons, making the signal propagate forward through the network.

Artificial Neural Networks consists 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

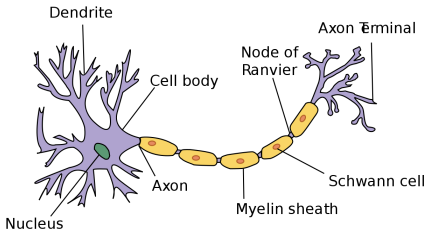

Figure 2.1: Neuron [1]

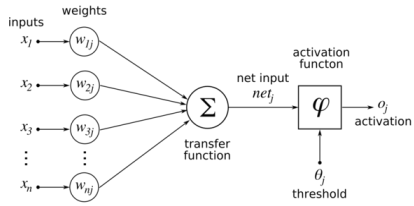

Figure 2.2: Artificial neuron [2]

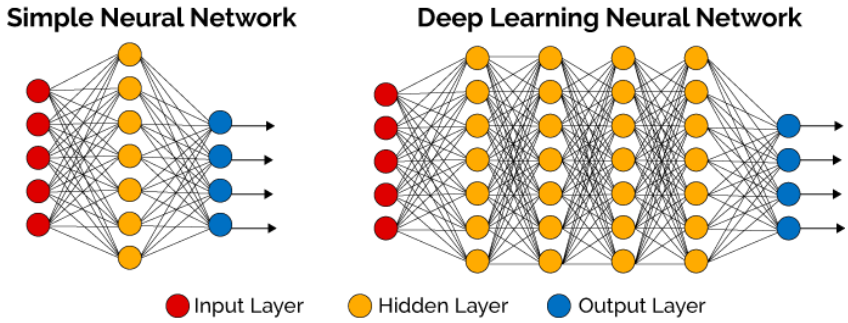

Figure 2.3: Example of a Deep Neural Network [3]

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

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 [8; 9; 10; 11; 12].

Artificial neural networks with multiple hidden layers are called deep neural networks. Traditionally, many hidden layers have made the networks too computationally difficult to work with, but in the later years, hardware development

[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)

has risen to a level where incredibly large networks are feasible. Deep neural networks may further be split into subcategories depending on what type of computation is done within the layers. Here we describe two particularly essential types of networks.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a very popular kind of neural networks, mostly used for image related tasks. CNNs had their breakthrough in image classification, and are also found to be the core of almost all computer vision systems to date. Essential in CNNs are the notion of *convolutions*. Simply put, convolutions can be thought of 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 (filter) extract different features from different regions, and repeated applications of convolutional- and max-pooling layers transform the low level pixel values to higher level information. E.g. one filter might detect edges in its sliding window, the next shapes from the detected edges, objects from the shapes after that, 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 slightly higher level. This method of computation does not take into account temporal information of previous data, which make CNNs poorly suited for tasks that are sequential in 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 comes from the ability to perform sequence modeling, and the ability to let previous time steps influence the present. This makes RNNs suited for tasks of sequential nature - e.g. natural language processing and text generation. Let's say we have a sentence x consisting of n words

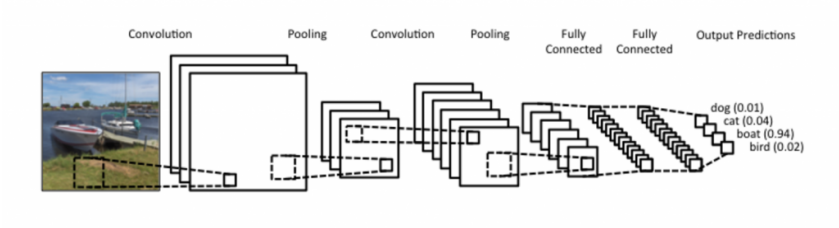

Figure 2.4: Example CNN architecture [4]

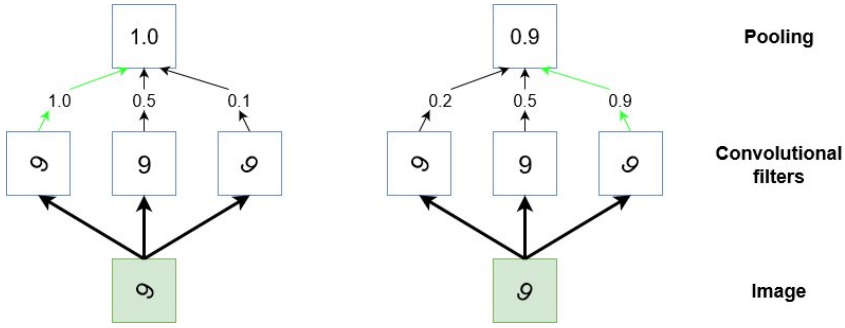

Figure 2.5: Purpose of CNN pooling [5]

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

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 can 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 which can both be used as an output at that time step, or used in the next time step $t + 1$.

There are several different implementations of recurrent neural networks. The general idea of outputting a hidden state to be used for the next time step is

[4] `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp` (Accessed on 09/11/2017)

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

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 time step increases - meaning that it suffers from long-term memory loss. Here we describe two principle kinds of implementations of a recurrent neural network.

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 4 layers and 3 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 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 behind this is that we want 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} should be combined with the new input x_t. The update gate controls how much

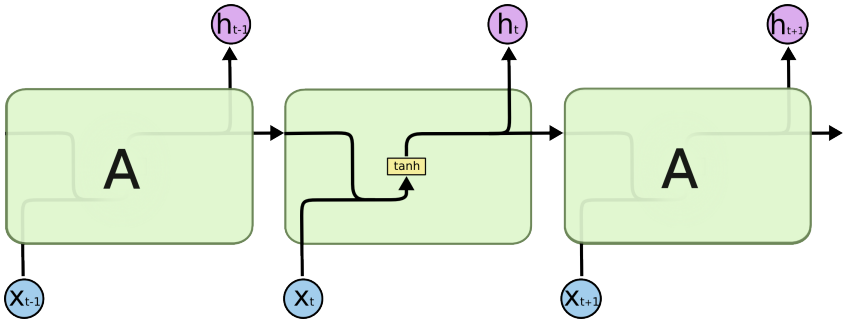

Figure 2.6: Traditional RNN [6]

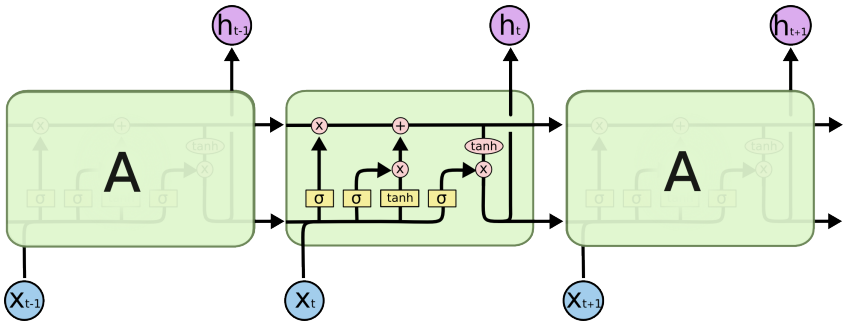

Figure 2.7: LSTM unit [7]

of that previous memory persists to the next time step. GRUs are also missing the internal memory c_t found in LSTMs.

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

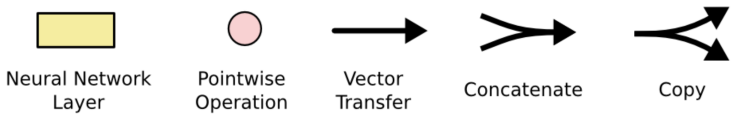

Figure 2.8: RNN Legend [8]

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

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

2.3.3 Bidirectional RNN

The performance of a RNN can 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 can be beneficial for the accuracy of a RNN. To make a RNN bidirectional, it will operate with 2 hidden states - one which works through the input sequentially, and another which works through the input backwards. The final output 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 Active 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 amount of labeled samples needed. To be able to do this we need to develop some sort of informativity measure which will be used to score all the unlabeled samples in a dataset, and then greedily select 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. A visualization of the process is shown in figure 2.11.

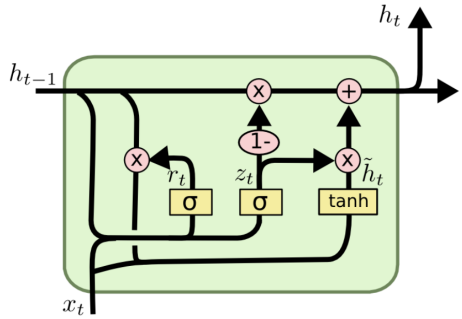

Figure 2.9: GRU [9]

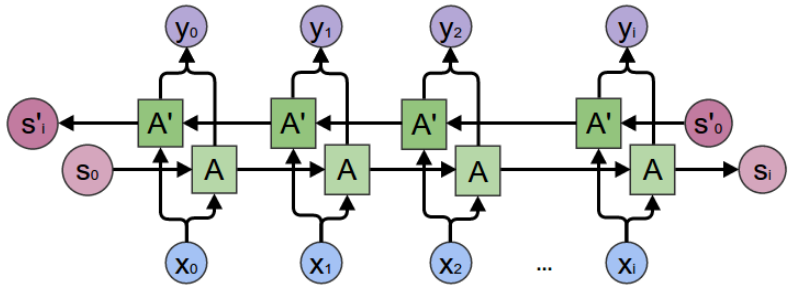

Figure 2.10: Bidirectional RNN [10]

The effectiveness of active learning depends solely on the informativity measure and how the individual samples in the unlabeled data pool are scored.

2.5 Pre-trained word vectors

A vital part of a deep neural nets performance is how to represent the input to the model. Although deep neural networks compute their own internal representations throughout the model, everything depends on the input representation of a given sample. The representation of the input should thus be one that sees to

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

[10] `http://colah.github.io/posts/2015-09-NN-Types-FP` (Accessed on 14/11/2017)

[11] `https://www.cs.cityu.edu.hk/images/research/ds2-f.png` (Accessed on 04/12/2017)

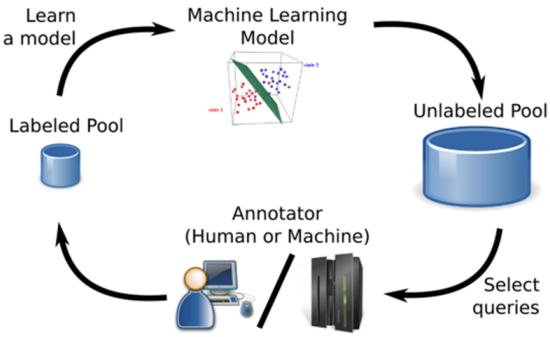

Figure 2.11: Active learning cycle [11]

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 vectors that each represents 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, pre-trained word-vectors almost always cause an increase in performance. The most commonly used are Word2Vec [12], GloVE [13] and fastText [13]. We have chosen word2vec in our implementation.

These vectors exhibit interesting behaviour. When mapped into a multidimensional space, we see that words with similar meaning are grouped together. See figure 2.12. Furthermore, the distance between similar pairs are the same. Vector addition and summation also shows very interesting properties - see figure 2.13.

$$vector('king') - vector('man') + vector('woman') \approx vector('queen') \quad (2.13)$$

[12] `https://code.google.com/archive/p/word2vec/` (Accessed on 07/12/2017)
[13] `https://fasttext.cc/` (Accessed on 01/12/2017)

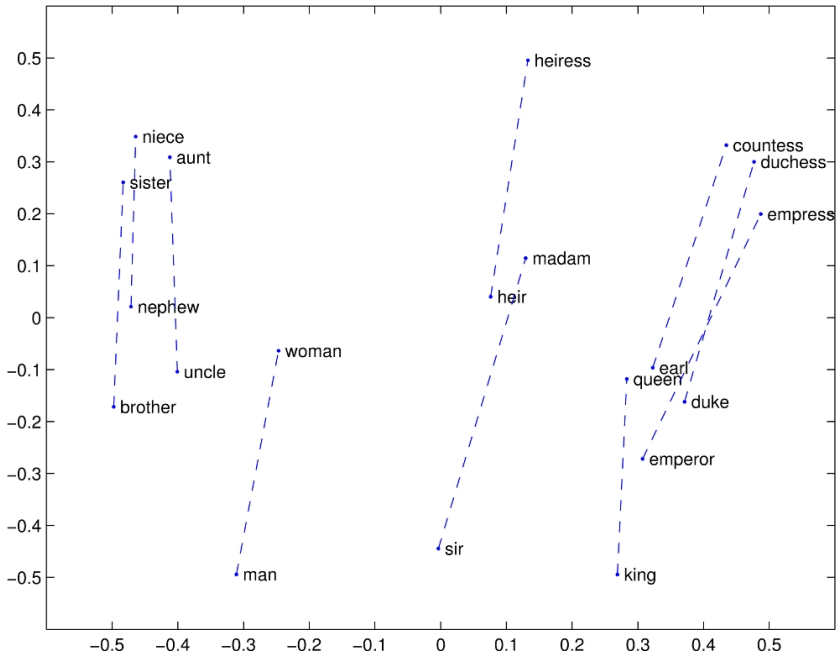

Figure 2.12: Word vectors exhibit interesting behaviour when mapped into a multidimensional space. [15]

[15] `https://nlp.stanford.edu/projects/glove/` (Accessed on 09/11/2017)

Chapter 3

State-of-the-Art

As mentioned in section 2.4, an active learner's performance relies heavily upon the selection algorithm chosen for adding unlabeled samples to the labeled data pool. There are several categories of active learning policies. Here we present some of the recent work within two of these categories.

3.1 Predefined scoring methods for active learning

One type of active learning is about designing scoring methods ahead of time. These scoring methods calculate a score for each unlabeled sample, and samples are then greedily selected at each acquisition stage.

3.1.1 Uncertainty sampling

One of the most popular scoring categories is *uncertainty sampling*. The idea behind uncertainty sampling is to select samples for which the model is least certain. A general variant implemented in [7] uses Shannon's entropy [14]. **Entropy** has traditionally performed well [7], and is defined as

$$-\sum_{k} P(y_i = k|x_i;\theta) \log P(y_i = k|x_i;\theta) \tag{3.1}$$

where k is all possible labels the sample may belong to.

[15] uses a Bayesian CNN (BCNN) for image classification, and explore active learning techniques in relation to this. They argue that a BCNN captures model

uncertainty in a better way than a deterministic CNN, and that because of this, scoring algorithms employing uncertainty-based measurements are better suited for this type of network. In addition to Shannon's entropy, [15] implements **Variation Ratios**. It is defined by

$$\text{variation-ratio[x]} := 1 - \max_y p(y|x, \mathcal{D}_{\text{train}})$$

3.1.2 Maximize parameter changes

[7] argue that because neural networks used for natural language tasks rely on word embeddings as representations, the active learning algorithms should focus on how the selected samples affect the embedding layer. A basis for scoring strategies described in [7] is the *Expected Gradient Length*. This method aims to select labels which would change the current model parameters as much as possible. The intuition behind this is to select samples that would contribute unique information to the model. EGL can be calculated with respect to different layers in the model, which makes several implementations of EGL possible. Here we describe different versions of EGL, as presented in [7].

EGL-word is used for sentence classification. EGL-word calculates for each sentence in $\mathcal{U}$ the expected gradient length with respect to the embeddings for the individual words contained in the sentence. The largest gradient with respect to the embeddings of the words contained in the sentence is for each possible category k multiplied with the probability of the sentence belonging to that category, and then summed a score for that sentence.

$$\max_{j \in x_i} \sum_k P(y_i = k|x_i; \theta) \left\| \nabla J_{E^{(J)}}(\langle x_i, y_i = k \rangle; \theta) \right\| \tag{3.2}$$

EGL-sm works on the opposite side of the model compared to EGL-word. EGL-word calculates the gradient with respect to the embedding layer, whereas EGL-sm calculates the gradient with respect to the final output layer:

$$\sum_k P(y_i = k|x_i; \theta) \left\| \nabla J_W(\langle x_i, y_i = k \rangle; \theta) \right\| \tag{3.3}$$

EGL-word-doc is used for longer text-classification. EGL-word is modified by first normalizing the gradient of each word by its frequency in the text. Then, instead of choosing the max-operator relying on one word, the gradients of the top-k frequency-normalized words are summed.

EGL-entropy-beta is another algorithm for longer text classification. This method considers both the entropy-based and the EGL-word scoring methods, and associates a composite score using these scoring methods. γ_t is a random variable with a temporal dependency. The intuition is that the EGL-word method should yield a more accurate score early in the active-learning process. As it continues, entropy should be a more correct estimate of the sentence score, and should thus affect the composite score in a larger manner.

$$\phi_t(i) = \gamma_t \cdot \mathcal{P}(\phi_{\text{Entropy}}(i), \{\phi_{\text{Entropy}}(j) : j \in \mathcal{U}\}) + (1 - \gamma_t) \cdot \mathcal{P}(\phi_{\text{EGL-word-doc}}(i), \{\phi_{\text{EGL-word-doc}}(j) : j \in \mathcal{U}\}) \tag{3.4}$$

[15] proposed **BALD**. It is similar to EGL[7], but the algorithm itself is quite different due to the model being a BCNN, as opposed to a traditional deterministic CNN:

$$\mathcal{I}[y, \omega | x, \mathcal{D}_{train}] = \mathcal{H}[y | x, \mathcal{D}_{train}] - \mathcal{E}_{p(\omega | D_{train})}[\mathcal{H}[y | x, \omega]] \tag{3.5}$$

where ω is the current model parameters, $\mathcal{H}[y|x, \omega]$ is the entropy of y given weights ω.

Mean STD calculates the mean standard deviation across all classes a sample can take:

$$\sigma_c = \sqrt{E_{q(\omega)}[p(y = c|x, \omega)^2] - E_{q(w)}[p(y = c|x, \omega)]^2} \tag{3.6}$$

$$\sigma(x) = \frac{1}{C} \sum_c \sigma_c \tag{3.7}$$

3.2 Learned scoring methods for active learning

When designing active learning strategies, heuristics for determining which samples we should request the true label for are needed. These heuristics must be designed ahead of time, and can often incur a human bias to which samples are selected. These heuristics are also specialized toward specific datasets. Furthermore, one can view the process of active learning as a sequential one; each acquired sample changes the model, and thus affects the scoring of the samples in the next acquisition stage. Because of this, several experiments have been carried out where the purpose is to learn an active learner. The sequential nature of active learning underpins the use of reinforcement learning for teaching an active learning policy.

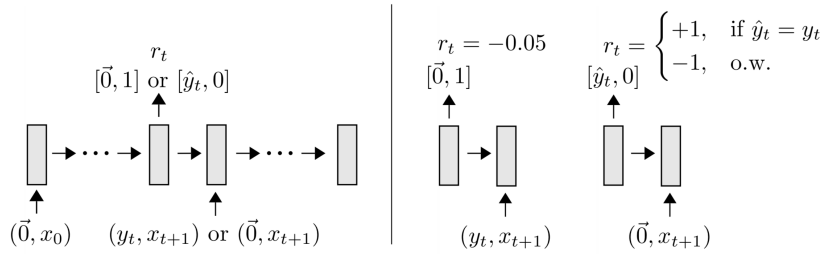

Figure 3.1: The left part of the figure shows the complete task structure. At time step t the model outputs an action corresponding to a request or a prediction. The right part show the possible scenarios for the reward r_t calculated at time step $t + 1$. Figure presented in [16].

[16] use deep reinforcement learning to learn a policy that determines whether or not a label for a given sample should be requested. The model receives a stream of images and must make a choice at each time step whether or not to predict a label for the current sample. The authors of [16] associate rewards with either requesting the label, predicting the correct label or predicting the wrong label:

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

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 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 for 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.

A weakness of learned active learning strategies is that the learned policy is 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,

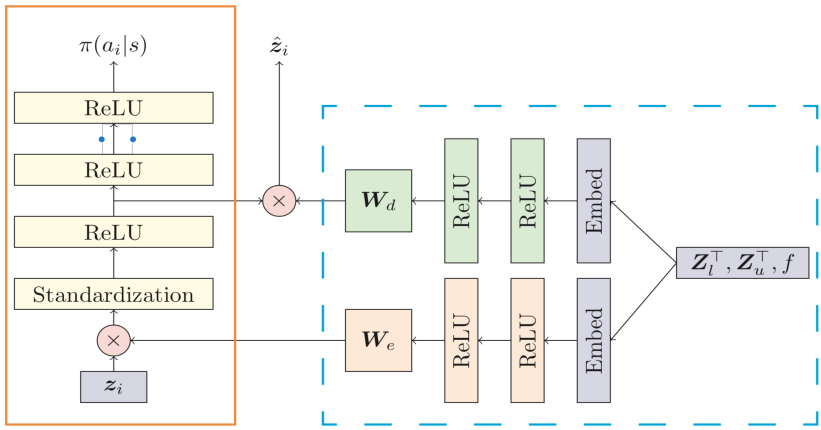

Figure 3.2: Policy and meta-learning network architecture from [17]. The meta-network on the right takes as input a featurization of the unlabeled and labeled samples Z_u^T and Z_l^T, and the current state of the classifier f, and outputs both an encoder W_e and a decoder W_d to be used for policy network to the left. The decoder is used for regularization purposes.

in a scenario where one has access to many labeled samples, an active learning approach would not be needed in the first place.

[17] also takes a reinforcement learning approach to active learning, and they investigate how to train an active learning policy that can 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 which sample to request a label for. The proposed architecture is shown in figure 3.2.

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

3.3 Convolutional Neural Networks for Sentence Classification

Convolutional Neural Networks have normally been used in tasks related to images and image processing. In later years, state-of-the-art performance has been achieved using these types of networks for text-related tasks - most notably for the task of text classification. One of the first and most cited works on CNN for text classification is [18], which proposes a way to utilize the power of CNN to achieve state-of-the-art text classification performance.

To be able to utilize CNNs on text, the method visualized in figure 3.3 is used. An input of n words consisting of

$$x = x_1 \oplus x_2 \oplus x_3 ... \oplus x_n \tag{3.8}$$

, where each x_n is the embedding of that respective word, is passed to the first convolutional layer. Each filter in the layer is applied to a window of h words, resulting in a feature vector c of k values for each filter applied to the sequence.

$$\vec{c} = [c_1, c_2, c_3, ..., c_k] \tag{3.9}$$

A max-pooling layer is then applied over each feature vector, and the resulting values are concatenated to a vector containing the max of each feature map. These max-values are then passed to a fully-connected layer which decodes the max-values to an output probability.

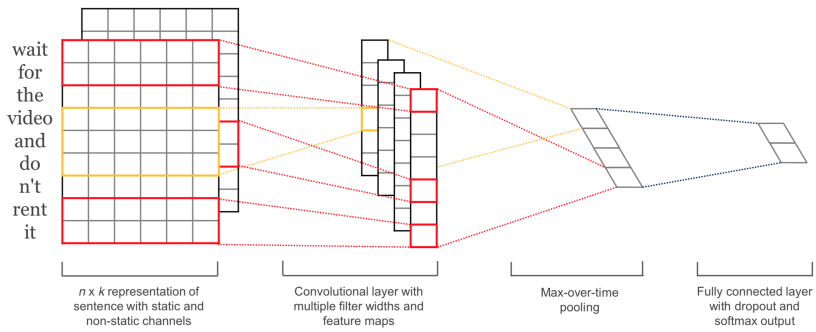

Figure 3.3: CNN architecture used for text classification, as proposed in [18]

The properties of a CNN used for text classification is the same as when used for image classification and other image-related uses. Because of the convolutional operations and max-pooling, each filter extracts the most important feature for each size over the sentence. CNNs do not take into account historical data, but rather try to capture the essence of each sentence, as opposed to attempting to understand each word from the historical context in which it is contained.

Chapter 4

Architecture/Model

This chapter describes the developmental process towards a final implementation, what design we implemented, what hyperparameters we chose and why we chose them.

4.1 CNN

The first phase of the project was implementing the CNN model from [18] and some of the active learning algorithms used in [7]. This paper was chosen as a basis because it was the first work addressing active learning for neural models for text classification.

Table 4.1: CNN hyperparameters

Learning rate	Filters	Filter number	Epoch
0.1	3,4,5	[100, 100, 100]	100

Batch size	Model dropout	Optimizer	Embedding dimensions
25	0.3	AdaDelta	300

The second phase phase revolved around the active learning techniques described [7]. Our first active learning results did not show the same classification accuracy as the paper. After further experimenting with a range of different optimizers, learning rate and regularization techniques, like early stopping, we installed Tensorboard [1]to visualize the performance of the different selection algorithms. We then had a central observation: Our active learning algorithms learned faster than the random baseline, like they were expected to - but the

overall performance was reduced by around 20%. We concluded that our active learning algorithms worked as intended, and that there had to be something inherently wrong with the underlying CNN used for classification. Further experimenting and testing shed light on the issue: We were not utilizing the power behind pre-trained word vectors in a proper way.

Our final hyperparameters shown in table 4.1.

4.2 RNN

The RNN implementation did not see the same gains from active learning compared to the CNN. We experimented with different hyperparameters and plotted the classification accuracy and test loss at each iteration. As the model was trained, the training loss was quickly reduced to a value close to 0, but the evaluation loss was reduced for some iterations before it increased in value. This is a clear sign of overfitting, and we added weight decay, dropout and early stopping.

After researching RNNs in general, we discovered we had overlooked a major hyperparameter: Model size. We based our RNN implementation on an existing RNN designed for a large dataset. Our first implementation had 2 hidden layers with 1200 hidden nodes in each layer, which is a large model. As explained in section 5.1, the active learning cycle starts with an empty pool of labeled data samples, and gradually extends the pool with labeled samples selected by our selection algorithm. Our model is thereby trained on an increasing, but very small dataset compared to traditional deep learning dataset sizes. When training a deep neural model on such a small dataset, overfitting is more likely to occur earlier. This, in addition to a large model, lead to overfitting.

When reducing the model size to 1 layer with 128 hidden nodes the results confirmed what we expected. The final RNN hyperparameters are listed in table 4.2.

[1] `https://www.tensorflow.org/get_started/summaries_and_tensorboard` (Accessed on 07/12/2017)

Table 4.2: RNN hyperparameters

Learning rate	Hidden layers	Hidden size	Epoch	Batch size
0.1	1	128	100	25

Weight decay	Model dropout	Embedding dropout	Embedding dim	Optimizer
$1 \cdot 10^{-5}$	0.4	0.2	300	AdaDelta

Chapter 5

Experiments and Results

This chapter described our experimental plan and setup, before showing the results from our experiments.

5.1 Experimental Setup

At the beginning of each experiment, we started with a pool $\mathcal{U}$ containing all the unlabeled samples in our dataset, and an empty pool $\mathcal{L}$ of labeled samples. We performed 20 rounds of mini-batch active learning. Each round consisted of:

1. Select *batch size* samples from $\mathcal{U}$ using the scoring function $\mathcal{F}$
2. Add the selected samples to $\mathcal{L}$
3. Retrain the model using the labeled samples in $\mathcal{L}$

The active learning algorithms used in experiments are entropy, described in equation 3.1, and egl-word, described in equation 3.2. We have run our experiments on a server with 48 cores, 64GB RAM and two NVIDIA Tesla P100 GPUs gifted to NTNU by Telenor. The server is running Ubuntu 16.04.2 LTS (64bit), Python 3.5.2, pyTorch 0.3.0.post4 and CUDA 8.0.61

5.2 Dataset

We worked with multiple dierent datasets in our project. They were chosen after having studied core literature in the field of deep learning. We found them to be suitable for our purpose.

5.2.1 Movie Review (MR)

The MR dataset from CS Cornell[1]is a perfectly balanced dataset containing 5331 positive and negative movie reviews in English. The sentences have different lengths, with a maximum of 59 words. All the sentences are written in lowercase. The dataset was first used in [19].

5.2.2 Text REtrieval Conference (TREC)

This dataset is provided by the TREC Conference series[2]. It was made for question classification, and it consists of 11403 questions samples across 6 different labels. The different classes are: Abbreviation(2%), Entity(23%), Description(21%), Human(22%), Location(15%), Numeric(16%). The dataset was first used by [20].

5.2.3 UMICH SI650 - Sentiment Classification

This dataset has been provided by University of Michigan SI650 class in Information Retrieval on Kaggl [3]. This perfectly balanced dataset consists of 7086 sentences extracted from social media. It stands out as a very simple dataset which uses a simple language with a lot of similar words and repetition. We chose to include this dataset as well to test out our system on an simpler language.

5.3 Experimental Results

In this section we describe our results.

5.3.1 CNN

Figure 5.1 and figure 5.2 shows the final results achieved by a CNN model on both the MR and the TREC dataset.

For the MR dataset we saw that the EGL algorithm performed the best, followed by entropy and random, in that order. The ordering of the different algorithms are consistent with what is achieved in [16], but our overall percentage is higher for all acquisition functions.

For the TREC dataset, we observed a different phenomena. The relative performance of the active learning algorithms compared to the random baseline

[1] `http://www.cs.cornell.edu/people/pabo/movie-review-data` (Accessed on 14/11/2017)
[2] `http://cogcomp.org/Data/QA/QC/` (Accessed on 14/11/2017)
[3] `https://www.kaggle.com/c/si650winter11` (Accessed on 07/12/2017)

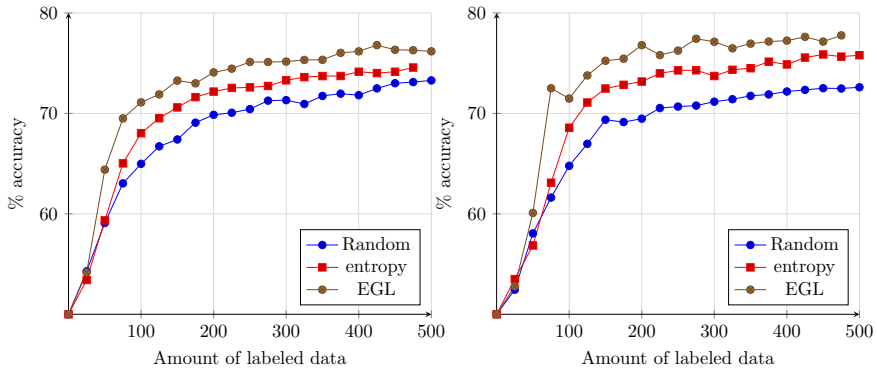

Figure 5.1: MR results. Left: CNN, right: RNN

has vanished. Moreover, we saw that EGL had a more constant learning slope compared to the other algorithms.

5.3.2 RNN

The final results from the RNN are displayed in figure 5.1 and figure 5.2. Like the CNN results, the RNN results showed that EGL performed the best on the MR dataset, followed by entropy and random. EGL achieved a high percentage quite fast, and at step 125 it has an accuracy of 73.7%, whereas the highest accuracy every achieved by the random algorithm is at step 500 at 72.6%. EGL beat the highest accuracy achieved by the random baseline using roughly 20% of the labeled samples, which really brings out the strength of active learning.

For the TREC dataset, we saw a similar decrease in the effectiveness of the active learning algorithms.

5.3.3 Different selection sizes

Table 5.1 shows the best classification accuracy for all algorithms on both datasets. As expected, the random acquisition function reached approximately the same classification accuracy regardless of the selection size at each step. Moreover, our active learners performed better as the selection size decreases.

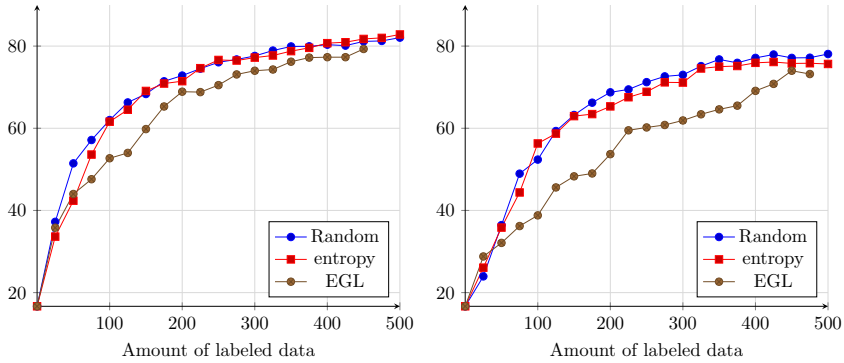

Figure 5.2: TREC results. Left: CNN, right: RNN

Table 5.1: Best classification accuracy achieved in the first 500 steps for different selection sizes on the MR dataset

	CNN			RNN		
Batch size	Random	Entropy	EGL	Random	Entropy	EGL
25	74.08	74.59	76.80	72.60	75.87	77.77
30	75.01	74.26	75.51	74.84	74.21	76.50
40	74.06	73.82	75.37	74.47	73.29	74.11

Chapter 6

Discussion and Conclusion

This section discusses the details presented in the previous chapter, and explain the results. The chapter ends with a conclusion that summarizes our findings, and answers our research questions.

6.1 Discussion

In this section we discuss the results from the previous chapter, and try to reason out and explain central observations.

6.1.1 CNN

First we look at the results the CNN results for the MR and TREC datasets, shown in figure 5.1 and figure 5.2.

The MR dataset consists of movie reviews written by different people, which means that each review is written in a natural language quite different from another. This makes it harder for the model to quickly generalize across different samples, which explains the decreased slope of the MR accuracy plot.

The differences of the datasets can also be quantified by looking at the uncertainty scores calculated by the entropy algorithm for the MR and TREC datasets, shown in figure 6.2. The number of classes in the MR dataset is 2 which makes the maximum possible entropy ϕ_{max} for any given sample:

$$\begin{aligned} \phi_{max} &= -\log \frac{1}{2} \\ \phi_{max} &\approx 0.6931 \end{aligned} \tag{6.1}$$

Figure 6.2 shows an average top MR uncertainty score very close to ϕ_{max} at each iteration. This means that the uncertainty of the selected samples is close to the theoretical maximum, which means that the model is very uncertain as to which label should belong to the samples. As we add more and more samples to the labeled pool, the average top score stays close to the theoretical maximum. The reason for this is that the MR dataset contains such a vast majority of different wordings, which makes it hard for a model to generalize. The unlabeled pool will for a very long time contain samples that do not resemble anything the model has seen before, which results in an average top score close to ϕ_{max}. This uncertainty is also quantified in the average uncertainty for the entire unlabeled pool. The average uncertainty score fluctuates as the active learning framework adds new samples to the labeled pool. This can be attributed to the difficulty of generalization across the dataset. One iteration might add a set of samples that contains a language significantly different from what the model has seen before, which may make the average uncertainty rise.

The classification accuracy on the TREC dataset starts at at 16.67% and rises to above 80% for all algorithms. Despite the increased amount of classes, the rate of accuracy change is much higher than what is achieved for the MR dataset. This can be explained using the same reasoning as in the previous paragraph. The language used in this dataset is much more consistent across samples, which makes it much easier to generalize. ϕ_{max} for this dataset is

$$\begin{aligned} \phi_{max} &= -\log \frac{1}{6} \\ \phi_{max} &\approx 1.7918 \end{aligned} \tag{6.2}$$

and we see that the average top score calculated by entropy descends below ϕ_{max} very quickly. Furthermore, the average uncertainty score for the entire pool shows a much more stable descent than what is displayed for the MR dataset. This means that the model rapidly reduces the average uncertainty across all unlabeled samples, and learns to generalize faster than compared to the MR dataset.

An interesting observation is that the active learning algorithms fail to increase the model performance after training on 500 samples. Another interesting observation is that the random algorithm learns the fastest in the early iterations, but is later matched by the entropy algorithm. The EGL algorithm initially has the slowest accuracy gain, but shows a more constant slope than the other two, and performs on par with them after 500 samples have been selected. The early accuracy gain of the random algorithm can be explained by looking at the samples selected by the active learners. All our models are randomly initiated, which will make the active learners select too many samples belonging to a few classes

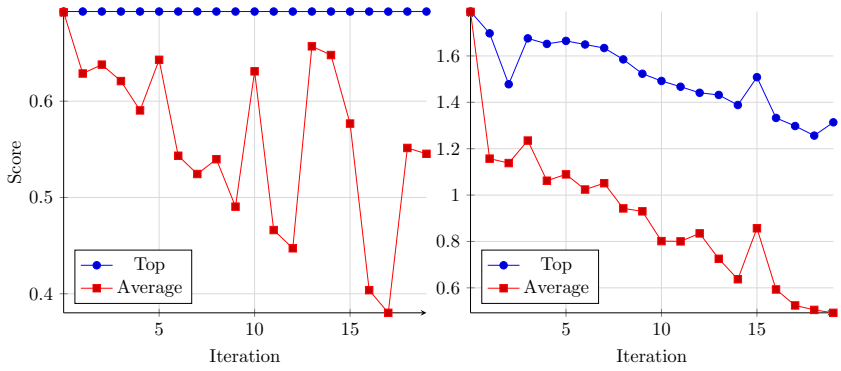

Figure 6.1: CNN entropy average total and top 25 scores. Left: MR dataset, right: TREC dataset

in the early iterations. After some time, especially displayed for the entropy algorithm, the active learners request samples that better suit their underlying model's needs. Furthermore, random learns rapidly does because the dataset is very easily learned.

6.1.2 RNN

The RNN model performs similarly for the MR dataset when compared to the CNN model. We see a clear separation of the different selection algorithms, which proves that the active learning algorithms also work when applied to a RNN.

When we compare the results for the MR and TREC dataset achieved by the RNN, we can explain the differences between them using the same reasoning as described in section 6.1.1.

6.1.3 Comparing CNN and RNN

We will now compare the results for the CNN and RNN models, displayed in figure 5.1 and 5.2.

Both models achieve roughly the same performance on the MR dataset. More importantly, they display the same difference between the EGL, entropy and random algorithms. The RNN model achieves a higher classification accuracy for all acquisition functions, which can be explained by analyzing the nature of the dataset. As explained in section 6.1.1, the MR dataset contains much more heterogeneous language. Because our CNN extracts certain features across word windows of lengths 3, 4 and 5, it will not be able to capture the real sentiment of the sentence as fast as a sequence-based deep neural network. The RNN takes

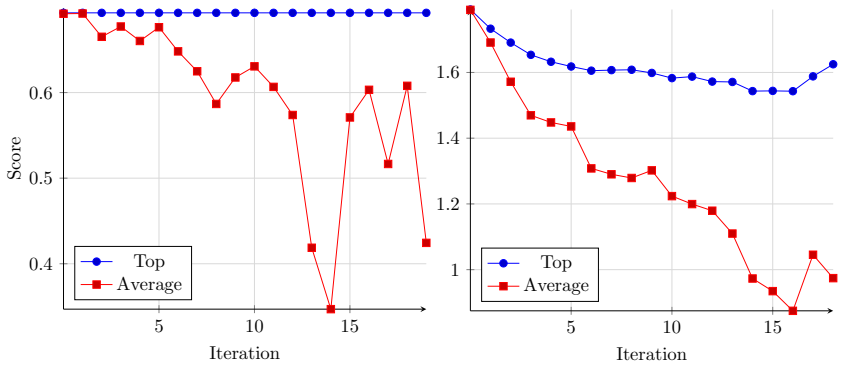

Figure 6.2: RNN entropy average total and top 25 scores. Left: MR dataset, right: TREC dataset

into account historical data further in the past, which makes it able to infer the sentiment for a larger fraction of the samples.

When comparing both models' results for the TREC dataset, we see that the CNN performance is better than what is achieved by the RNN. TREC is a dataset which contains questions, and the labels are what noun category an answer to that question would belong to. A question "Who has written this report?" would have a corresponding label of "Human". This dataset shows a more homogeneous nature, where the label of the sentence almost exclusively can be determined by looking at the question word it contains. For the previous example, we would only need to know that the question word is "Who" to be able to infer that the sentence was asking for a human. Because CNNs are well suited for extracting features over small windows of data, a dataset like this better fits the nature of CNNs, which explains the increased performance.

6.1.4 Different selection sizes

How many samples selected at each acquisition stage is a significant factor for the resulting model's classification accuracy. As displayed in table 5.1, we can see that a smaller selection size is beneficial for both our models' classification accuracy. The reason for this is that a larger selection size can be thought of as analogous to having a too large learning rate. By selecting a large number of samples for labeling at each acquisition stage, the labeled data pool will not reflect an optimal distribution, which will hurt the model's classification performance. Reducing the amount of samples added at each stage will make the model more frequently select samples that better fit its parameters. Based on this, one can expect active learning algorithms to perform better when a smaller selection size

is applied. We have not experimented with a selection size less than 25 due to runtime constraints.

6.1.5 Similar sentences in datasets

Using entropy or EGL as a scoring function on a dataset containing similar sentences will hurt its performance. If entropy assigns a high score to a sentence, similar sentences will also receive a high score and be selected for labeling. The unique information gained by adding similar sentences to the labeled pool is minimal, which will affect the model's classification performance. At an early stage in our development we experienced active learning algorithms performing worse than expected. The issue became evident when we looked at the actual sentences chosen for labeling each time. The active learner would choose duplicates with high scores and provide the model with less unique data. We later introduced functionality to remove all duplicates from the dataset, and experienced a drastic increase in accuracy. Table 6.1 shows one example of similar sentences our active learner chose.

6.1.6 Automatic dataset balancing

An interesting question in active learning is how the query strategy behaves when operating on an unbalanced dataset. As a baseline, the random query strategy is expected to result in a labeled pool which exhibits the same class distribution as the unlabeled pool. To analyze the effects of active learning on the resulting class distribution, we recorded the number of classes at each step in our active learning algorithm and the random baseline. The results are shown in figure 6.3.

As expected, the distribution when using the random query strategy reflects the original distribution. When we look at the graph for entropy, we can observe that entropy results in a different distribution. Because entropy is a form of uncertainty sampling, it is natural to expect that the active learner will request more labels for classes that are hard to generalize than for classes that are easier to generalize.

To explain the distribution selected by the entropy algorithm, we calculated the average sentence difference between each class using the following procedure: First, we calculated the average embedding vector for each sentence in the entire dataset. This was done by averaging the embedding vector for each word contained in the sentence. Using the average sentence vectors associated with a class, we calculate the average class vector. Lastly we calculate the cosine distance between each class average vector to quantify the average difference between sentences of different classes.

The calculated class difference is shown in table 6.2. When we compare the difference scores and the resulting class distribution obtained by the entropy

Table 6.1: Example of entropy selected samples in the UMICH dataset 5.2.3.

Sentence	**Uncertainty**	**Label**
I miss Harry Potter and stuff, man.....	0.6931	positive
Harry Potter and Titanic suck.	0.6931	negative
dudeee i LOVED brokeback mountain!!!!	0.6931	positive
I love Harry Potter!.	0.6931	positive
who LOVED brokeback mountain:.	0.6931	positive
I like Harry Potter.	0.6931	positive
I like Harry Potter..	0.6931	positive
I love Harry Potter!..	0.6931	positive
And I loved Harry Potter.	0.6931	positive
I LOVE BROKEBACK MOUNTAIN!!!	0.6931	positive
Harry Potter SUCKS!!	0.6931	negative
I love Harry Potter...	0.6931	positive
I love Harry Potter.	0.6931	positive
I LOVE BROKEBACK MOUNTAIN!!!!..	0.6931	positive
I LOVE BROKEBACK MOUNTAIN!....	0.6931	positive
. Harry Potter sucks..	0.6931	negative
Harry Potter Sucks.	0.6931	negative
I LOVE BROKEBACK MOUNTAIN..	0.6931	positive
I LOVE BROKEBACK MOUNTAIN!	0.6931	positive
Harry Potter Rocks!!!!!!!!!	0.6931	positive
Harry Potter sucks.	0.6931	negative
Harry Potter sucks...	0.6931	negative
yeah, I love Harry Potter to death, lol!..	0.6931	positive
i hated brokeback mountain.	0.6931	negative
i LOVE the da vinci code!	0.6931	positive

-	ABBR	HUM	LOC	NUM	DESC	ENTY
ABBR	0	0.206	0.159	0.171	0.073	0.114
HUM	0.206	0	0.099	0.104	0.123	0.072
LOC	0.159	0.099	0	0.066	0.081	0.054
NUM	0.171	0.104	0.066	0	0.061	0.053
DESC	0.073	0.123	0.081	0.061	0	0.032
ENTY	0.114	0.072	0.054	0.053	0.032	0
Average	0.145	0.121	0.092	0.091	0.074	0.065

Table 6.2: Cosine difference between average feature vectors of TREC classes. Classes with a larger difference shows a brigher color.

query strategy, it becomes apparent that the algorithm queries less labels for categories that are more different from the other classes. The ABBR class, for example, has an average class difference of 0.14. This is also the class the active learning framework queries the least labels for. It is also the class with the least amount of samples in the original data pool, but we see that it entropy queries even less because of the large difference towards other classes. Another example is the HUM class, which accounts for $22,5\%$ of the entire dataset. The average difference for that class is the second largest, and the HUM percentage in the pool selected by the entropy is reduced to 17 as a result of this.

We also did some experiments on the MR dataset. The MR dataset is originally perfectly balanced, but for this experiment we altered it so it contained 75% positive and 25% negative samples, and observed both the models performance and its sample distribution. The resulting dataset distribution in figure 6.4 shows a near equal dataset distribution for the classes, even though our unlabeled pool is largely unbalanced. The sentences belonging to the the positive and negative class has a low difference, which is why the amount of samples for each class is roughly equal each other. The acquisition functions can be described as automatic balancers, and selects a sample distribution that better suits the model's needs. An interesting thought is that even though this dataset balancing could have been done manually beforehand, the active learners will choose the samples based on performance criteria, resulting in a sort of optimal balanced dataset.

6.2 Conclusion

We have showed how active learning algorithms work on different models, datasets and selection sizes. Extending the work of [7], we find that the active learners bring an increase in accuracy when applied to both the CNN and the RNN. The RNN outperforms a CNN on longer, heterogeneous text classification. Moreover,

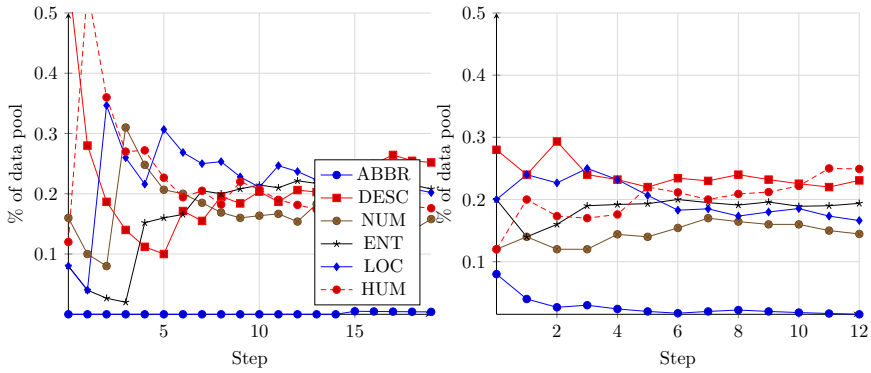

Figure 6.3: Category distribution from entropy and random acquisition functions on a CNN, used on the TREC dataset. Left: entropy, right: random.

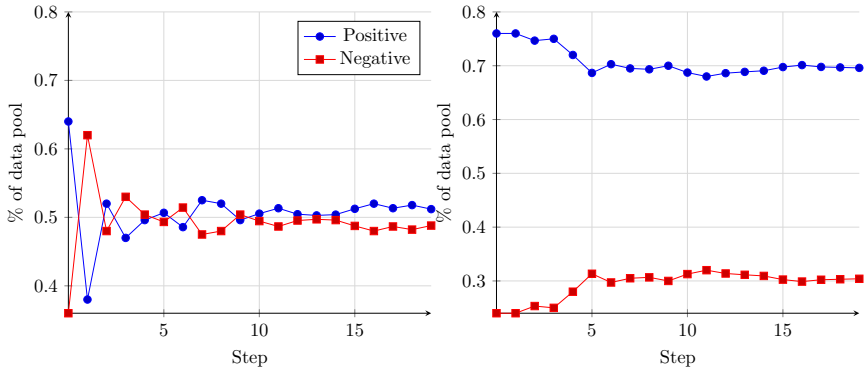

Figure 6.4: Category distribution from entropy and random acquisition functions on a CNN, used on the MR dataset. Left: entropy, right: random.

we explored the effects of active learning when applied to a homogeneous dataset, and observed that the same active learners fail to achieve an increase in performance. In addition to this, reducing the selection size is beneficial for the active learning algorithms. The active learners also fail to improve learning rates when the dataset contains similar sentences. An interesting observation was how the active learners balance their labeled sample pool, and we saw that the active learners tune the class distribution to better fit the model's needs.

6.3 Future Work

This project has made the authors get a deeper understanding of deep active learning. As described in chapter 3, there are two main approaches to active learning: Designing active learning policies beforehand, or learning the policies. When learning policies, the resulting policy is often a domain specific one. We would like to continue the work on general active learning, where a reinforcement learning approach can be used on generalizations of many different types of datasets.

We also think that the observations made in relation to automatic balancing of the selected data pool are very interesting, and are grounds for further research.

Bibliography

[1] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[2] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

[3] M. Iyyer, J. L. Boyd-Graber, L. M. B. Claudino, R. Socher, and H. Daumé III, "A neural network for factoid question answering over paragraphs.," in *EMNLP*, pp. 633–644, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[5] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," *arXiv preprint arXiv:1703.02910*, 2017.

[6] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[7] Y. Zhang and B. C. Wallace, "Active discriminative word embedding learning," *CoRR*, vol. abs/1606.04212, 2016.

[8] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, no. 1, pp. 109 – 118, 1990.

[9] R. J. Williams, "The logic of activation functions," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, pp. 423–443, 1986.

[10] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991.

[11] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, pp. 911–917, Jul 1995.

[12] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[14] C. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[15] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," *CoRR*, vol. abs/1703.02910, 2017.

[16] M. Woodward and C. Finn, "Active one-shot learning," *CoRR*, vol. abs/1702.06559, 2017.

[17] Anonymous, "Meta-learning transferable active learning policies by deep reinforcement learning," *International Conference on Learning Representations*, 2018.

[18] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[19] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of ACL*, pp. 115–124, 2005.

[20] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, (Stroudsburg, PA, USA), pp. 1–7, Association for Computational Linguistics, 2002.