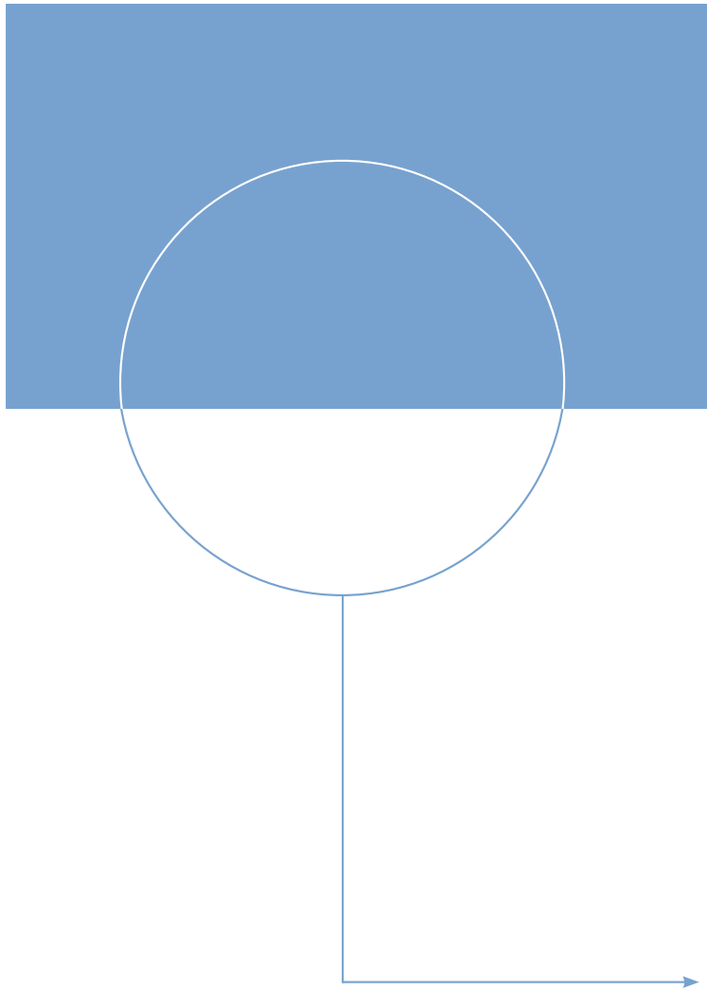


Master's thesis

2019

Tina Olivia Sørlie Oftedal

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Mathematical Sciences

Tina Olivia Sørlie Oftedal

Uncertainty Measures and Transfer Learning in Active Learning for Text Classification

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Uncertainty Measures and Transfer Learning in Active Learning for Text Classification

Tina Olivia Sørlie Oftedal

Applied Physics and Mathematics
Submission date: June 2019
Supervisor: Erlend Aune

Norwegian University of Science and Technology
Department of Mathematical Sciences

Summary

Deep learning has become a prominent and popular tool in a wide range of applications concerned with processing of complex data. However, in order to train a sufficient model for supervised tasks, deep learning relies on vast amounts of labelled data. Even when data itself is easily attainable, acquiring labels can be tedious, expensive, and in need of an expert annotator. *Active learning* (AL) aims to lower the data requirement in deep learning, and machine learning in general, and consequently reduce labelling cost. By letting the learner *actively* choose the data it wants to learn from, active learning aspires to label only the most *valuable* data, and to train a classifier with only a small labelled training set. The idea is that the model is able to single out examples of high *informativeness* from a pool of unlabelled data, i.e. instances from which the model will gain the most information, which often is linked to model uncertainty. Through this thesis, several aspects of pool-based active learning in text classification are explored, by combining ideas that have shown good results individually. To ensure diverse actively queried samples, both adding randomness to the active selection, and clustering of the unlabelled pool have been investigated. Further, seeing that deep models rarely represent models uncertainty, a Bayesian approximation is computed by sampling sub-models by applying dropout at test time, and averaging over their predictions. Lastly, active learning is studied in a transfer learning setting, combined with the previously explored ideas. The experiments clearly show how active learning depends on data and model, as the two different models and datasets showed quite dissimilar results. The models in question are a simple CNN for sentence classification, and an AWD LSTM with pre-training, both tested on the binary sentiment analysis IMDB movie review dataset, and the multi-class AG news corpus. While there were no effect from any AL strategy on AG, with or without advances, all variations showed improved results on IMDB with the CNN. Although clustering appeared as the preferred choice for the CNN, it had a negative effect when combined with transfer learning and the AWD LSTM. The combination of clustering and Bayesian approximations did not add anything more than raised computational cost, even though both boosted validation accuracy and loss individually with the CNN. All in all, no method was exceedingly better than random sampling, however, many results introduced interesting ideas for further work.

Sammendrag

Dyp læring har blitt et fremtredende og populært verktøy i et bredt spekter av applikasjoner som omhandler behandling av komplekse data. For å kunne trene en modell tilstrekkelig, er imidlertid dyp læring avhengig av store mengder annotert data. Selv når data i seg selv er lett tilgjengelig, kan annotering være tidkrevende, dyrt, og ofte avhengig av en ekspert. *Aktiv læring* (AL) tar sikte på å redusere datakravet i dyp læring, og maskinlæring generelt, og dermed redusere annoteringskostnadene. Ved å la modellen *aktivt* velge de dataene den ønsker å lære fra, ønsker aktiv læring å kun annotere de mest *verdifulle* dataene, og trene en modell med kun et lite annotert treningssett. Ideén er at modellen skal kunne identifisere informative eksempler fra en stor samling med uannotert data, hvor informativitet ofte knyttes til modellens usikkerhet. Gjennom denne oppgaven utforskes flere aspekter ved aktiv læring i tekstklassifisering, ved å kombinere idéer som har vist gode resultater individuelt. For å sikre mangfold i aktivt valgte data har to metoder for å utforske større deler av rommet blitt utforsket. Den ene blander inn noen tilfeldig valgte data i det aktive utvalget, mens den andre *grupperer* den store samlingen med uannortert data, og velger kun ett datapunkt i hver klynge. Videre har en bayesiansk tilnærming til modellusikkerhet blitt testet, i og med at dype modeller som regel ikke representerer modellusikkerhet. Til slutt utforskes også de ulike idéene sammen med transfer learning. Forsøkene viser tydelig hvordan aktiv læring avhenger av data og modell, da de to forskjellige modellene og datasettene viste tydelig ulike resultater. De to modellene er en CNN for setningsklassifisering, og an AWD LSTM med pre-trening, som begge er testet på et filmanmeldelsedatasett (IMDB) med to klasser, of et nyhetsartikkel-datasett (AG) med fire klasser. Selv om ingen metoder viste noen effekt på AG, forbedret alle variasjoner resultatene for IMDB med CNN. Mens grupperingsmetoden virket som det mest fordelsaktige valget for CNN, ga det kun negativ effekt med AWD LSTM. Kombinasjonen av gruppering og bayesianske tilnærminger ga ingen bedre sammenlagt effekt, selv om begge ga gode resultater individuelt. Alt i alt viste ingen metoder overdrevent bedre resultater enn tilfeldig utvalgt data, men mange av resultatene ga interessante idéer for videre arbeid.

Preface

This master thesis completes my final year and master's degree in Industrial Mathematics within the Applied Physics and Mathematics M.Sc. program at the Norwegian University of Science and Technology. The work has been accomplished during the spring semester of 2019 at the Department of Mathematical Sciences, as a continuation of my specialization project, which was finalized in January of 2019.

I would like to thank my supervisor, Erlend Aune, for guiding me with his ideas and knowledge through my master thesis and specialization project. His feedback has been valuable for my understanding, and has pushed my work forward. Further, I would like to thank the NTNU HPC group for providing GPU resources, making the experiments of this study possible. Lastly, I would also like to thank my family for supporting me through my studies, and my friends for making my time in Trondheim an incredible experience.

Trondheim, June 2019

Tina Olivia Sørlie Oftedal

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning has come to be an important field within artificial intelligence, where models learn patters from data, rather than having algorithms of explicit instructions. In other words, machine learning lets the data *speak for itself*. Even though shallow machine learning models excel at many tasks, reality is too complex for them to fully describe it. *Deep learning*, a sub-field of machine learning, has gained recognition for its ability to extract useful features from complex data, taking inspiration from how the human brain process information. It has today become a widely applicable tool, and has automated numerous tasks, such as translation, spam detection, image captioning, customer service to a degree, and much more.

However, deep models for supervised classification, and supervised machine learning models in general, are relying on vast amounts of labelled data for sufficient training. In some cases, labelled data is easily attainable, however, in other situations, gathering and labelling data can be extremely tedious and expensive. *Active learning* addresses this issue by letting the model *actively* select data it wants to learn from. Aiming to increase the *value* of the training data, the idea behind active learning is to train a model with a small labelled training set, then let the model use its knowledge to choose data from which it will gain the most information. The model can then *query* the labels of the most informative examples from an *oracle*, often a human annotator, and add them to the training set. With the updated labelled set, the model can be re-trained, and with its newly gained knowledge, query more informative examples. In this way, active learning intends to obtain higher accuracy with less annotated data, and substantially reduce the labelling cost.

A common way of measuring the *informativeness* of unlabelled data is to consider the model *uncertainty*. When classifying an instance, how *certain* is the model about the label? If there's high uncertainty, the model is likely to gain a lot of information from knowing that label. Often, the uncertainty is computed from the predictive class probabilities, since a large probability would suggest that the learner is certain in its classification. An issue arising when active learning is combined with deep learning models is that deep

models often lack a representation of model uncertainty. The predictive probability does not necessarily reflect upon the model's confidence towards a prediction. To overcome this problem, Gal et al. (2017) propose computing a Bayesian approximation by averaging over T randomly sampled sub-models. By applying dropout at test time, a new sub-model is sampled at each forward pass, and the average over T predictions can represent model uncertainty in deep models (Gal and Ghahramani, 2016). Gal et al. (2017) demonstrate that a Bayesian approximation increases classification accuracy for three different active learning query strategies in computer vision tasks, including handwritten digit recognition on MNIST, and skin cancer diagnosis from lesion images.

Computational cost is another challenge with active learning. This becomes especially apparent when employing large, complex models, as deep models often are. The model is usually trained from scratch every time a single instance is added to the labelled training set, a process that could lead to weeks of training, which is unsuitable for many practical applications today. A solution could be to select more than one informative instance at once, but then there's a chance that many similar examples are added, since similar examples would have similar informativeness. An instance considered as informative during one active selection might not be seen as informative at the next active selection, due to the new knowledge obtained from the previous query. Thus, querying more examples at once is a way of wasting the *labelling budget*. To overcome this problem, Zhdanov (2019) presents a method for clustering the unlabelled data, and selecting instances in a way that ensures diversity in the queried sample. The unlabelled data is clustered by the K-means algorithm into n clusters, then n instances are queried, one from each cluster. The informativeness is incorporated in the K-means objective function, and the instances closest to the cluster centroids are selected. The method guarantees dissimilarity among queried examples, while still taking informativeness into account. Note that the method is independent of the model, and can therefore yield different results in different situations.

There are other ways of dealing with the computational complexity of active learning. Shen et al. (2017) and Wang et al. (2017) both employ an incremental approach, where the model is not trained entirely from scratch when an instance is queried, but rather trained further with the updated labelled set. Another idea is to make use of transfer learning, which is done by Huang et al. (2018), among others. Transfer learning exploits model architecture and already learned general features from similar tasks, and can both speed up training, and lower the data requirement, since the task specific data can be used mostly for learning task specific features. In combination with active learning, Huang et al. (2018) also introduce a new criterion for querying, namely *distinctiveness*, to separate source task from target task. The proposed algorithm, named *active deep model adaptation* (ADMA), takes advantage of the perks of transfer learning in the active learning setup, and queries examples based on a trade-off between their informativeness and distinctiveness. ADMA performs well on four different computer vision datasets for both binary and multi-class classification tasks, and makes use of the pre-trained models AlexNet (Krizhevsky et al., 2012), VGG (Simonyan and Zisserman, 2015), and ResNet (He et al., 2015).

Transfer learning has for long been important in computer vision, utilizing pre-trained

models trained on large image databases for general computer vision tasks, then fine-tuning the model for the target task. However, the same kind of general approach has not been present for natural language processing (NLP) tasks before recently. Seeing that language modelling can in NLP serve as large scale image recognition in computer vision, Howard and Ruder (2018) propose a method for fine-tuning a pre-trained language model trained on a large corpus, then augmenting it for classification, or other NLP tasks. The method, known as *Universal Language Model Fine-tuning* (ULMFiT), promises transfer learning comparable to computer vision for any NLP task.

In this thesis, pool-based active learning in a text classification setting is explored by looking at several aspects. Several uncertainty measures are investigated, combined with various adaptations, and further explored in a transfer learning setting. The aim is to answer to questions on how active learning is influenced by these variations. Inspired by Zhdanov (2019), one of the adaptations is to cluster the unlabelled pool of data by K-means clustering, however, independently of *both* the model and query strategy. Furthermore, model uncertainty is computed as a Bayesian approximation, motivated by the findings of Gal and Ghahramani (2016) and Gal et al. (2017), to see the effect with the models at hand. The various methods are first applied with a convolutional neural network (CNN) for text classification (Kim, 2014), before the same aspects are studied in combination with transfer learning by introducing ULMFiT (Howard and Ruder, 2018) and a new model, an ASGD weight dropped long short-term memory network (AWD LSTM) (Merity et al., 2017). The aim is to answer the following questions:

- When clustering is independent of both model *and* query strategy, will it still have an effect over not clustering?
- If both clustering and Bayesian approximations can yield improvements to the active learning results individually, as seen for Zhdanov (2019) and Gal et al. (2017), will the combination of the two have a supplementary positive effect?
- When not accounting for distinctiveness, like Huang et al. (2018) do in computer vision, could transfer learning in active learning be successful in a text classification setting?

After conducting experiments on the binary sentiment analysis IMDB movie reviews dataset, and the multi-class text categorization AG news corpus, non of the active learning methods improved to random sampling on AG. However, there were improvements from all on IMDB. While both Bayesian and clustering boosted results, the combination of the two did not improve additionally. When applied with transfer learning, however, clustering had more of a negative effect, while Bayesian gave an advantage. All in all, there were not substantial improvements from random selection for any method or variation, only small differences in accuracy and loss. Although the results were insufficient for drawing any positive conclusion on the potential of the methods tested, the experiments gave many new ideas for future work. Additionally, it is important to address that most of the work behind this thesis has involved computationally expensive experiments, and as new discoveries have come to light, there has been little time left to explore further advances and new ideas.

The thesis is outlined as follows. Chapter 2 introduces the concepts of deep learning, transfer learning, and active learning, along with basic theory and recent advances. A description of the models, data, and experimental setup is presented in Chapter 3, and the results are presented and discussed in Chapter 4. Finally, the conclusion of this thesis is provided in Chapter 5. All implementations are made publicly available[1][2].

[1] https://github.com/tinaolivia/cnn_al
[2] https://github.com/tinaolivia/lstm_al

Chapter 2

Basic Theory

2.1 Deep Learning

With today's technology, more and more tasks are automated by computers and algorithms. Not only are computers replacing human labor in time consuming, necessary, straightforward jobs, they are also generally more effective. One such task is e-mail filtering, which automatically sends sketchy e-mails to your spam folder, preventing scams and computer viruses. This task could have easily been done by a human, and it is necessary to prevent scam and to not overlook e-mails of interest. However, e-mail filtering is an example of a task where constructing algorithms with explicit instructions is infeasible. A spam detection model should rather *learn* patters from data and perform inference, a field known as *machine learning* (ML). First formulated by Arthur Samuel (Samuel, 1959), machine learning has become a well known, and not least important, field within artificial intelligence (AI). *Supervised* ML algorithms build mathematical models from labelled *training data*, aiming to make predictions or decisions when faced with new, unseen data points. E.g. predicting whether an e-mail is spam or not, after seeing a training set of many examples of spam and non-spam e-mails. In general, the task of an ML algorithm is to approximate a function. In a regression setting, it would be to map the input to its response value, and in classification, it would be to map the input to its label. This thesis will be mostly concerned with supervised classification methods, that is, methods for classification where training data is available *with* labels.

So-called *shallow* ML algorithms, such as linear regression, logistic regression, and linear discriminant analysis, are great to extract the necessary patterns in many tasks. However, the real world is often more complex than what these types of methods are able to represent. *Deep learning* is a sub-field of machine learning, developed to handle more complex data. As a term, deep learning was first introduced in the 80's in the machine learning community (Dechter, 1986; Schmidhuber, 2017), and has been linked to *artificial neural networks* (ANN) for approximately the last two decades (Aizenberg et al., 2000). The intention behind ANNs was to make computers able to process data in a more *human like*

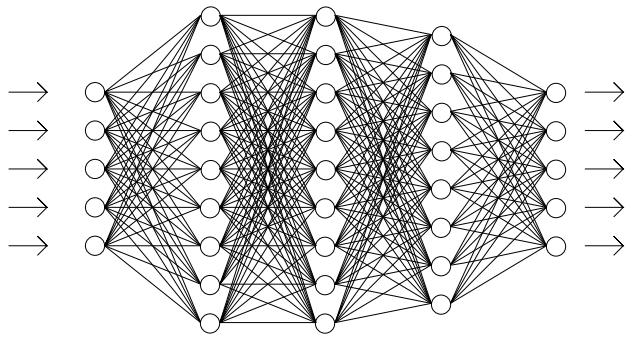


Figure 2.1: Illustration of a typical fully connected artificial neural network.

fashion, therefore, ANNs were originally inspired by the neural networks of the human brain. A typical fully connected feed forward ANN is illustrated in Figure 2.1, which will be further discussed in Section 2.1.1. An ANN is constructed of an input layer, an output layer, and some hidden layers in between, where *deep models* are considered as ANNs with more than one hidden layer. Input is passed through the input layer, then to approximate the target function, the hidden layers' task is to transform the information into something that is useful for the output layer. That is, in a classification task, it should be easy for the output layer to classify the input based on the information it receives from the hidden layers.

The layers of an ANN consist of *artificial neurons*, which are represented by circles in Figure 2.1. In a fully connected neural network, each neuron in a layer receives information from all of the neurons in the previous layer, computes a weighted sum of these inputs, adds a bias, and passes it to the next layer through an *activation function*. Mathematically, in an L layer neural network this means

$$x_j^{(\ell)} = \phi^{(\ell)}\Big(\sum_{i=1}^{K^{(\ell-1)}} w_{i,j}^{(\ell)} x_i^{(\ell-1)} + b_j^{(\ell)} \Big), \quad \ell = 1, ..., L, \tag{2.1}$$

where $x_j^{(\ell)}$ and $x_i^{(\ell-1)}$ are the outputs from the j^{th} neuron in layer ℓ, and i^{th} neuron in layer $\ell - 1$, respectively, $w_{i,j}^{(\ell)}$ is the weight from neuron i in layer $\ell - 1$ to neuron j in layer ℓ, and $b_j^{(\ell)}$ is the bias associated with neuron j in layer ℓ. Furhter, $K^{(\ell-1)}$ is the number of neurons in layer $\ell - 1$, and $\phi^{(\ell)}$ is the *activation function* of layer ℓ. The objective to learn is the weights and biases of the model, which are optimized by *training* the network with some gradient descent based optimization algorithm. Equation (2.1) demonstrate how ANNs essentially are functions of functions of functions, and so on, seeing as $x_j^{(\ell)}$ is a function of the weighted sum of the outputs from layer $\ell - 1$, which again are functions of the weighted sum of the outputs from layer $\ell - 2$, etc. It is this layering of functions that

enables deeper models to process more complex data, where non-linearity is introduced by the activation function. Due to these properties, deep learning has become a popular tool in a wide range of applications, such as machine translation, weather or stock market predictions, or recognition of speech, handwriting, or objects in images, to name a few. Some of the basic concepts of deep learning are presented in the following.

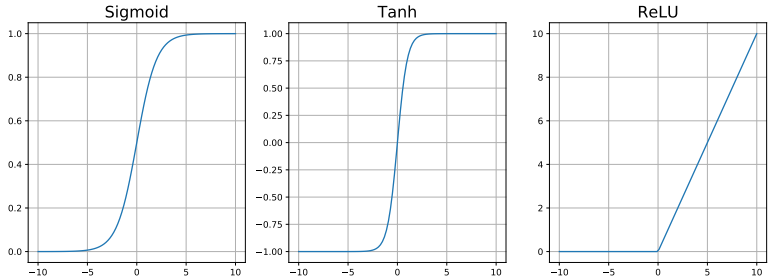


Figure 2.2: Typical activation functions in deep learning. Sigmoid (left), hyperbolic tangent ($\tanh$) (middle), and rectified linear unit (ReLU) (right).

As seen in Equation (2.1), the *activation function* returns the final output from a neuron, and it's main task is to introduce non-linearity to the approximation. Arguably, the most popular activation function is the *rectified linear unit* (ReLU) (Glorot et al., 2011), a function simply stripping away negative values $\phi(x) = \max\{0, x\}$. ReLU possesses many desired properties from linear functions, while not being linear. Looking at Figure 2.2, which displays ReLU (right), as well as two other activation functions, the sigmoid (left), and the hyperbolic tangent (middle), the key to ReLU's success is that it has gradient 1 for all positive values. Because ANNs normally are optimized by gradient descent based algorithms, the sigmoid and hyperbolic tangent introduce the *vanishing gradient* problem, as their gradients are mostly 0. Still, the sigmoid and hyperbolic tangent are useful in output layers in binary classification, where a value close to 1 indicates class 1, and a values close to 0 or -1 indicates class 0 or -1, respectively. In multi-class classification, the output is a vector, as opposed to a scalar, typically of size C, where C is the number of classes. Then the output is typically *softmaxed* in the output layer, i.e. normalized according to

$$\sigma(\mathbf{z})_j = \frac{\exp\{z_j\}}{\sum_{k=1}^{K} \exp\{z_k\}},$$

where $\mathbf{z}$ is the un-normalized output, z_j is the j^{th} element of $\mathbf{z}$, and $\sigma(\mathbf{z})_j$ is the j^{th} element of the normalized vector.

To train a neural network, the model parameters, i.e. the weights and biases, are typically initialized at random, then optimized by *backpropagation*. Backpropagation is an algorithm which *propagates* input data forward through the network, making predictions, before *backpropagating* information about how the predictions compare to the target values.

Backpropagation was already developed by various researchers in the 60's (Schmidhuber, 2014), but did not gain recognition until 1986 (Rumelhart et al., 1986a,b). The comparison of the predictions to the true values is done by a *loss function* or *cost function*, and should express how much is *lost* or how much it *costs* when predictions are wrong. Model parameters are optimized by minimizing the loss function by some gradient descent based optimization algorithm.

The loss function should reflect upon the task, and would normally be a function of the model's predictions and target values. In a classification setting, the targets are discrete classes, $\mathcal{C} = \{c_1, ..., c_C\}$, and the outputs are vectors containing class predictive probabilities, $\hat{\mathbf{y}} = (p_{\boldsymbol{\theta}}(y = c_1|\mathbf{x}), ..., p_{\boldsymbol{\theta}}(y = c_C|\mathbf{x}))^T$, where $\boldsymbol{\theta}$ is the model parameters, and $\mathbf{x}$ the input. For an instance $\mathbf{x}$, the true class probabilities are contained in a one-hot vector $\mathbf{y} = (0, ..., 1, ..., 0)^T \in \mathbb{R}^C$, where the placement of 1 represents which class $\mathbf{x}$ belongs to. In these situation, the goal would be to have $\hat{\mathbf{y}}$ as close to $\mathbf{y}$ as possible, that is, minimize the distance between the two probability distributions $\hat{\mathbf{y}}$ and $\mathbf{y}$. A way of doing this is to minimize the *cross-entropy*, which is defined by

$$\begin{aligned} H(p, q) &= \mathbb{E}_p[-\log(q)] = H(p) + KL(p\|q) \\ &= -p\log(p) - p\log\left(\frac{q}{p}\right) = -p\log(q), \end{aligned}$$

for probability distributions p and q, where $H(p) = -p\log(p)$ is the entropy (Shannon, 1948) of p, and $KL(p\|q) = -p\log(\frac{q}{p})$ is the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) from p to q. The KL divergence is often used as a measure of how similar two probability distributions are, even though it is not formally a distance, due to asymmetry, and that it does not satisfy the triangle inequality. Minimizing the cross-entropy is evidently the same as minimizing the KL divergence, i.e. minimizing the difference between p and q, as a KL divergence of 0 would imply that the two distributions in question are identical. In a C-class classification problem, the cross-entropy loss would be a sum over all classes and instances,

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N}\sum_{c=1}^{C} p(y = c|\mathbf{x}_i)\log(p_{\boldsymbol{\theta}}(y = c|\mathbf{x}_i)). \tag{2.2}$$

Minimizing the cross entropy is also equivalent to minimizing the *negative log likelihood*, thus maximizing the likelihood.

An important hyperparameter when optimizing the model parameters is the *learning rate*, i.e. the *step size* of the optimization algorithm. A too large learning rate would lead to overshooting on the minimum, while a too small learning rate would lead to slow convergence, as illustrated in Figure 2.3. It can be challenging to choose the optimal learning rate, especially if it's not adjusted while training. A common practice is the use of *annealing*, which decreases the learning rate during training, since larger values are often feasible to begin with, then smaller learning rates are more beneficial when getting closer to the minimum. Another is *cyclical learning rates* (CLR) (Smith, 2015), which like annealing adjusts the learning rate, but in a cyclical pattern, and eliminates the need to find

one optimal learning rate. The learning rate is gradually increased and decreased between a minimum and maximum value, and is based on the observation that increasing learning rates has a negative effect short-term, but a positive effect in the long run, as larger learning rates can help avoid saddle points. The *one cycle policy* (Smith, 2018) resembles CLR, but only one cycle is performed. Starting at the minimum, for approximately half of the total number of epochs, the learning rate is increased linearly to the maximum, before decreasing back to the minimum in the same amount of epochs. Then the model is trained for a few more epochs with the learning rate decreasing further below the minimum.

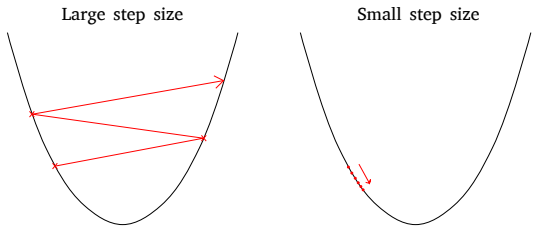


Figure 2.3: Illustration of gradient descent with large step size (left) and small step size (right).

ANNs are powerful machine learning tools, but there is a significant chance of overfitting due to the large number of model parameters. Learning with basic backpropagation can lead to co-adaption among neruons, which generalizes poorly to unseen data. *Dropout* (Hinton et al., 2012; Srivastava et al., 2014) is a deep learning regularization technique lowering dependence and co-adaption between neurons. The idea is simple, at each forward pass during training, activations are randomly set to zero with probability p. An illustration is presented in Figure 2.4. Since activations are randomly dropped, neurons can't rely on the presence of other neurons, thus it breaks up any potential co-adaption. The elementary idea behind dropout can be translated to other models. Essentially, when training a large model which easily overfits, sample and train sub-models, which combined will function as a more general version of the original model.

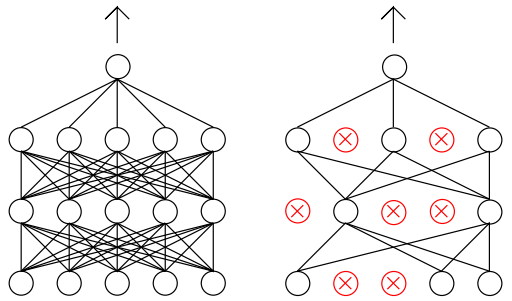


Figure 2.4: Illustration of how dropout affects a neural network when setting random activations to zero.

2.1.1 Feed Forward Neural Networks

Feed forward neural networks are maybe the simplest kind of ANNs, the easiest to comprehend and illustrate, and therefore often used as illustration when talking about ANNs and deep learning in general, as done in Figure 2.1. The name, *feed forward* network, comes from the information flow in the model, which is straight forward, i.e. there are no internal loops of information. One can say that each neuron is only visited once per forward pass. Feed forward networks are also known as *multilayer perceptrons* due to their connection to Rosenblatt's perceptron (Rosenblatt, 1958) developed in the 50's and 60's. The perceptron is a single neuron, taking as input several binary inputs, computing the weighted sum, and returning 1 if the sum is above some threshold value, and 0 otherwise. Figure 2.5 shows this operation, and notice how the first input is a 1, representing a bias term.

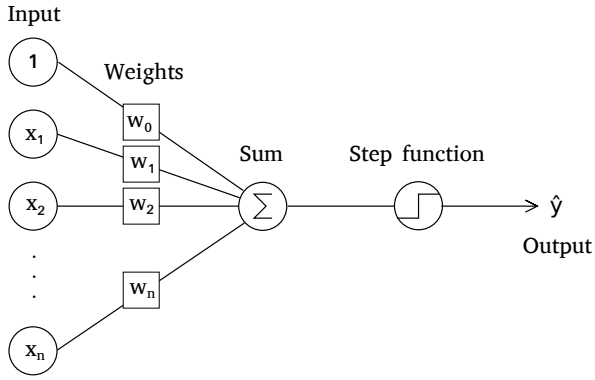


Figure 2.5: Illustration of Rosenblatt's percepton (Rosenblatt, 1958).

The perceptron showed a great ability to approximate functions by learning the weights, however, it was unable to solve non-linear problems, such as the exclusive-or (XOR) problem, illustrated in Figure 2.6 along with the AND and OR logic gates. There are two axis, x and y, and four points, one placed in the origin (0,0), one placed at the x-axis (1,0), one placed on the y-axis (0,1), and the last placed in (1,1). A white point represents the output 1 (true), and a black point represent the output 0 (false). The three logic gates are simply x AND y, x OR y, and exclusive x OR exclusive y. The perceptron could be trained to separate the linearly separable AND and OR problems, while the XOR is not linearly separable, creating problems for the perceptron. The solution would be more layers, however, there were not yet any known ways of training a multi-layer neural network. It lead to the near death of ANN research, until the backpropagation algorithm gained recognition (Rumelhart et al., 1986a,b). Still, with backpropagation available, the activation function, which in the perceptron's case is the step function, would be unsuited for training due to mostly zero gradients. Nonetheless, neurons in modern day neural networks are still

essentially the same perceptron developed in the late 50's.

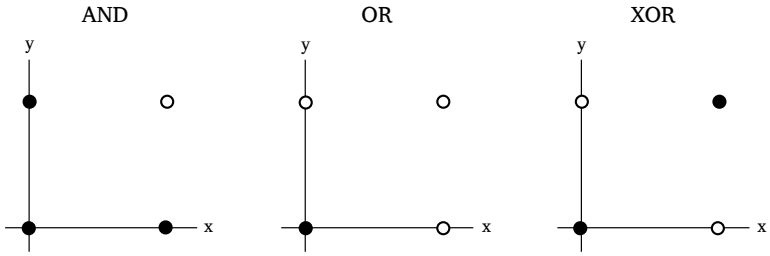


Figure 2.6: Illustration of the AND, OR and XOR problems.

Convolutional Neural Networks

Up to now, only fully connected feed forward neural networks have been considered. For instance, Figure 2.1 displays a fully connected network, and Equation (2.1) describes the output of a neuron in a fully connected ANN. *Fully connected* indicates that all neurons in one layer receives information from all neurons in the previous layer, and pass information to all neurons in the succeeding layer. Not only do fully connected networks have a substantial number of parameters to learn, they are also prone to overfitting due to the complete *connectedness*. Imagine a computer vision task, for instance classifying cats from dogs. Each input will have a height of h pixels, a width of w pixels, and in colored images, each pixel has three color channels, RGB. The input is thus a matrix of $h \times w \times 3$ elements, which means that in a fully connected neural network, each neuron in the first layer has $h \times w \times 3 + 1$ parameters to learn, and dependent on the size of the hidden layers, neurons in succeeding layers may have even more.

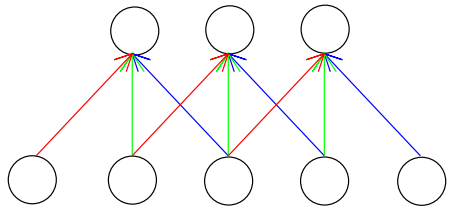


Figure 2.7: Illustration of local and shared weights in a CNN.

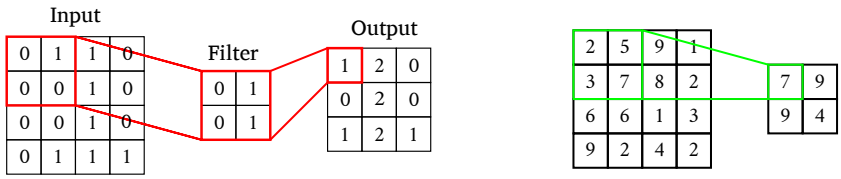
Convolutional neural networks (CNN) are feed forward neural networks that are *not* fully connected. They were originally developed for computer vision tasks, taking inspiration from a study of the visual cortex system of cats and monkeys (Hubel and Wiesel, 1968). What was discovered is that in the visual cortex, neurons are responding to a small part of the visual field, and neighboring cells have overlapping receptive fields. In a bid to

process image data similar to living creatures, CNNs adapt this finding by having local connections between layers, and shared weights. Figure 2.7 is an illustration of how both local connections and shared weigts work between two layers in a CNN. The arrows of the same color represent the same weights, and notice how a neuron in one layer is connected to only a few of the closest neurons in the other layer. These properties reduce the number of weights to learn considerably.

To give a better understanding of how this works in more dimensions, picture an input matrix in two dimensions, e.g. 4×4. The *convolution* between two $N \times M$ matrices, A and B, is defined by

$$A * B = \sum_{i=1}^{N} \sum_{j=1}^{M} a_{ij} b_{ij},$$

where a_{ij}, b_{ij} range over the elements of A, B, respectively. Now, let there be a *kernel* or *filter* smaller than the input, maybe 2×2, which contains weights. This filter slides over the input matrix, computing the convolution between the weights and the part of the input matrix it is covering, see Figure 2.8a. The *stride* is the number of cells the filter moves at the time, which is one in Figure 2.8a. The convolution is essentially a weighted sum of the values covered by the filter, and the weighted sum is computed with the same weights over the whole input. When the input is three dimensional, as in computer vision with colored images, the same principle is applied with three dimensional filters. This kind of layer is called a convolutional layer, and by definition, a CNN is a neural network with at least one convolutional layer.



(a) Convolution **(b)** Max pooling

Figure 2.8: Illustration of a convolutional layer (left) and max pool layer (right).

In general, after a convolutional layer, there's a *pooling layer*. Pooling is a way of reducing variance, and lower the number of parameters, while still keeping the most important information received from the convolutional layer. Pooling is used to detect invariant patterns in the input. Like in a convolutional layer, there's a window sliding over the

input, although in this case, the window does not contain any values, and it slides over non-overlapping parts of the input. What the pooling layer does, is to return a value based on the values within the window at each step. This could be the average of the values, called *average pooling*, or the maximum, called *max pooling*. A simple max pooling example is presented in Figure 2.8b. In a computer vision task, it is often beneficial choosing max pooling over average pooling. Average pooling tends to smooth out lines, and blur out the image, while the effect of max pooling is enhancement of lines and edges. Take for instance the example in Figure 2.8b. Imagine the values being shades of grey, where 0 is black and 9 is white. In the input matrix, there are fairly high values in the top left, top right, and bottom left 2×2 windows, while the bottom right contains fairly low values. It could be an edge between an object and the background in an image. This is enhanced when max pooling, as three of the cells in the output matrix contain large values (top left and right, bottom left), while the last contains a smaller value. The edge is *clearer*, or *enhanced*, after max pooling.

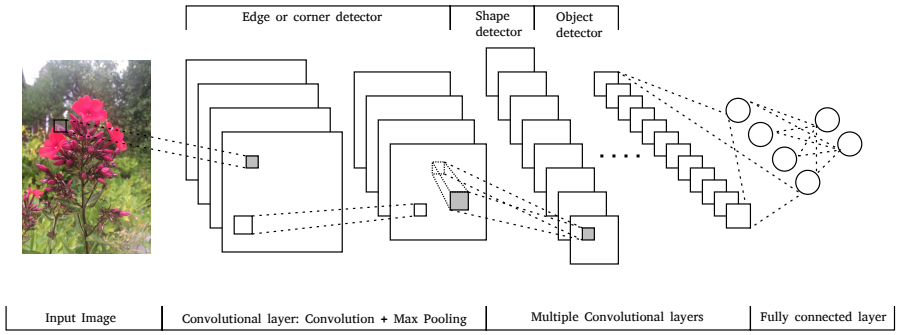


Figure 2.9: Typical CNN architecture.

There are usually multiple layers of convolutions and pooling detecting features in a hierarchical order. The different convolutional layers have different filters, constructed to detect different patterns. Basic features are generally detected in early layers, such as edges and corners, then later layers detect task specific features, like specific shapes or objects. An illustration of a typical CNN architecture for object detection is presented in Figure 2.9.

Although CNNs originally were developed for computer vision, they have shown great success in NLP tasks as well (Kim, 2014; Young et al., 2017). In NLP, the input is typically a sentence, a paragraph, or a longer text document, such as an article. Representing these as something a computer can understand, that is, as a matrix of values, is usually done by *word embeddings*. Words could be represented by one-hot vectors, however, this is not beneficial when faced with large vocabularies, due to the curse of dimensionality. Word embeddings are distributional vectors representing the meaning of a word, hence, words

that have similar meaning should have similar word embeddings. They are constructed based in the *context* of a word, rather than the word itself. Let $E(w)$ denote the embedding of a word w, then they should be created in such a way that e.g. $E(\text{king}) - E(\text{man}) + E(\text{woman}) \approx E(\text{queen})$. The size of word embeddings is not fixed, however, the larger the embedding, the more *accurate* they are. With word embeddings, the input to a neural network in an NLP task would be an $N \times M$ matrix, where N is the number of words, or tokens, in the input document, and M is the size of the word embeddings. For a CNN, the filter would be of size $K \times M$, where K typically is a number between 2 and 5, so that the filter covers 2-5 tokens at a time. Commonly, different layers have different size K to detect different features.

2.1.2 Recurrent Neural Networks

When reading a text document, humans don't start thinking from scratch at every word. The meaning of a word is not extracted from the word alone, we understand the meaning from context, a sequence of words stored in memory. In feed forward neural networks, information flows in one direction. Consequently, the prediction of one instance is completely independent of the previous, in the sense that one prediction does not influence another. Feed forward networks have no sense of *order in time*, they can not *remember* previous examples when presented with another, making them unsuited for tasks involving sequence data, such as time series prediction, or language modelling.

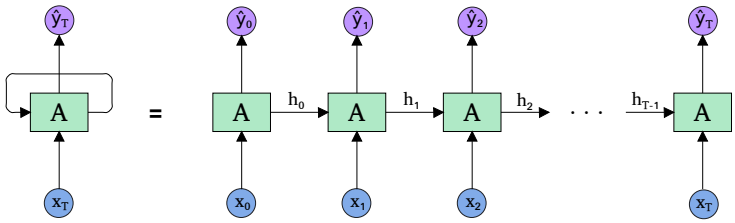


Figure 2.10: Illustration of a recurrent neural network (left), and an unrolled recurrent network (right).

A language model (LM) is a probabilistic model which goal is to predict the next word in a sequence of words, given the previous words. Denoting a word by w, a probability is placed over entire sequences, $P(w_1, .., w_n)$, in addition to a probability of the likelihood of a given word to follow. Given a sentence, e.g. "The trip has to be cancelled because of the weather.", feeding one and one word into a feed forward network making predictions on the next word in the sentence would give poor results. Since predictions are independent, the same word would be predicted after "the" both times it appears in the sentence, and it would probably be neither "trip" or "weather". It would be the most frequently occurring

word after "the" in the training data. Language modelling is important as it is the base for many other NLP tasks, such as machine translation, automatic question answering, and speech recognition. In order to handle these kind of tasks, as well as other sequence data problems, *recurrent neural networks* (RNN) were developed in the early 70's, a type of ANN with *memory*. The first RNN was described in 1972 (Little, 1974), and got the name Hopfield network after being popularized by John Hopfield in 1982 (Hopfield, 1982). Later, RNNs have become a popular tool whenever sequence data is involved.

The memory of an RNN is an internal, or a hidden, state in the network. At every time step, the network computes this state, and passes it back to itself, creating an inner loop in the architecture. Figure 2.10 is an illustration of the RNN architecture (left), and an *unrolled* version (right) for a better understanding of what is going on. RNNs can be viewed as several copies of a neural network in a sequence, passing information to each other. The hidden state can be described mathematically by

$$\mathbf{h}_t = \phi(W\mathbf{x}_t + U\mathbf{h}_{t-1}),$$

where, $\mathbf{h}_t$ is the current hidden state, $\mathbf{h}_{t-1}$ is the previous hidden state, $\mathbf{x}_t$ is the input, W is the weight matrix, and U is the hidden-to-hidden-state matrix. The function ϕ is some suitable activation function, often the hyperbolic tangent. The output $\hat{\mathbf{y}}_t$ could equal the hidden state, or be a *filtered* version of it, as a consequence, $\mathbf{h}_t$ is often referred to as the *exposed* hidden state. An RNN has thus two inputs at each time step, the input itself, $\mathbf{x}_t$, and the hidden state, $\mathbf{h}_t$, in other words, the present and the recent past. Both inputs are weighted by the matrices W and U, respectively. W and U are learned through gradient descent and *backpropagation through time* (BPTT), a backpropagation algorithm taking several time steps into account when updating the weights. Looking at the unrolled version of the RNN in Figure 2.10, each time step can be viewed as a copy of the network. Each copy has an input $\mathbf{x}_t$, an output $\hat{\mathbf{y}}_t$, and a loss $\mathcal{L}_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$, for $t = 1, ..., T$. Then the objective to minimize is the loss over several time steps $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{t=1}^{T} \mathcal{L}_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$. BPTT can be fairly computationally expensive, and is therefore often limited to a certain amount of time steps T. Commonly, the hidden state is initialized by zero, but can also be randomly initialized with e.g. a Gaussian distribution.

Recurrent networks are not only important in many NLP tasks, as long as the data is sequential, i.e. the order matters, RNNs are suited, and feed forward networks are probably not. Tasks include predicting time series, e.g. from financial data or weather data, image captioning, a cross between NLP and computer vision, and predicting the next chord in a music piece, opening the possibility of machine composed music. Furthermore, recurrent networks are not constrained to mapping one input to one output like for feed forward networks. RNNs can map one to many, many to one, or many to many as well.

Long Short-Term Memory

In general, the traditional RNN has a short-term memory, making it difficult to extract important information from time series with lags between important events. In language modelling, there are cases where the information from the recent past is enough, and there

are cases where information further back in the text is needed. E.g. in the sentence "The library is filled with *books*", the model can predict the word "books" from the word "library" in the recent past. In a longer text, e.g. "I lived three years in Spain with my family growing up. ... I speak fluent *Spanish*", predicting the word "Spanish" is linked to the word "Spain" much earlier in the text, making it hard for vanilla RNNs to predict correctly. In addition, RNNs often have problems with vanishing and exploding gradients.

Partially solving both these problems are *long short-term memory* (LSTM) networks. Initially developed to deal with vanishing and exploding gradients, LSTMs also proved to have longer memory than vanilla RNNs. The LSTM network was first introduced by Hochreiter and Schmidhuber (1997), and originally an LSTM contained cells with input and output gates, and later, a forget gate was added (Gers et al., 1999).

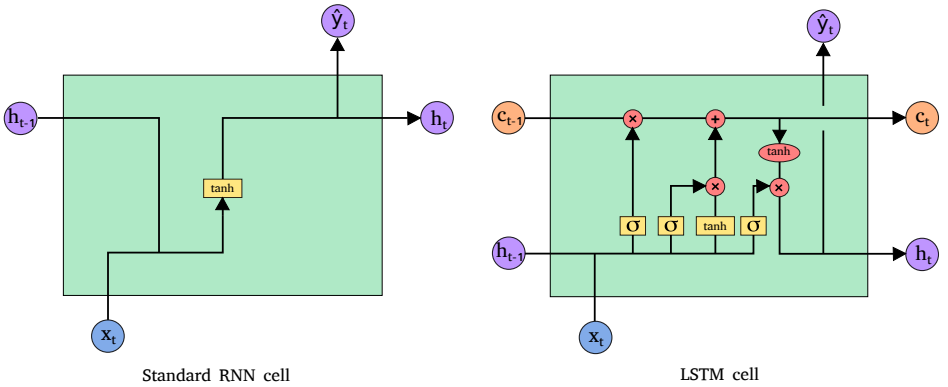


Figure 2.11: Illustration of a standard RNN cell (left), and a typical LSTM cell (right).

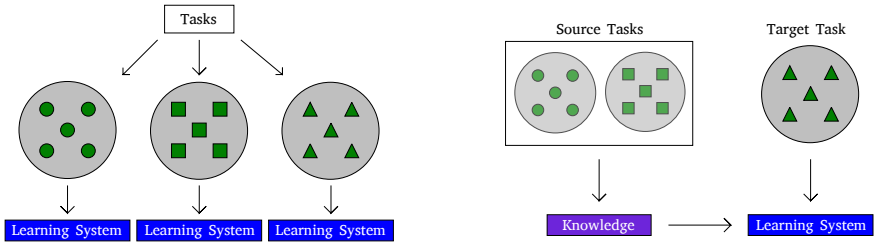
A typical LSTM cell is presented and compared to a vanilla RNN cell in Figure 2.11, but different versions have been proposed through the years. In the illustration, yellow boxes represent neural network layers, with σ the sigmoid and $\tanh$ the hyperbolic tangent, red circles are element-wise operations, with $+, \times$ addition and multiplication, respectively. The cells are represented by the A-block in the illustration of an RNN in Figure 2.10. While the standard RNN cell only has two input channels, the LSTM has three, as well as two values that are passed back to itself. In Figure 2.11, the bottom horizontal channel in the LSTM cell represents the exposed hidden state $\mathbf{h}_t$, while the one at the top represents the *memory cell state* $\mathbf{c}_t$. Further, the LSTM has four layers, where the sigmoid layers, from left to right, are the forget gate, input gate, and output gate, and the last layer is linked to the memory cell state. An LSTM can be described mathematically by

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(W^i \mathbf{x}_t + U^i \mathbf{h}_{t-1}), \\
\mathbf{f}_t &= \sigma(W^f \mathbf{x}_t + U^f \mathbf{h}_{t-1}), \\
\mathbf{o}_t &= \sigma(W^o \mathbf{x}_t + U^o \mathbf{h}_{t-1}), \\
\tilde{\mathbf{c}}_t &= \tanh(W^c \mathbf{x}_t + U^c \mathbf{h}_{t-1}), \\
\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + f_t \odot \tilde{\mathbf{c}}_{t-1}, \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}
\tag{2.3}
$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ are the input, forget, and output gates, respectively, at time step t, $\mathbf{c}_t$ is the memory cell state, $\mathbf{h}_t$ the exposed hidden state, and $\odot$ is element-wise multiplication. W^i, W^f, W^o, W^c are the non-reccurent weights associated with the input gate, forget gate, output gate, and cell state, respectively, and U^i, U^f, U^o, U^c, are the corresponding hidden-to-hidden weights. The sigmoid layers return values between 0 and 1, representing the amount of information to be let through, i.e. 0 means no information is passed on, while 1 means all information is kept. The forget gate determines what can be thrown away from the internal state, the input gate decides what new information should be stored in the internal state, and lastly, the output gate determines what to output, which is a filtered version of the cell state.

2.2 Transfer Learning

A traditional machine learning assumption is that the training data and test data have to belong to the same domain, have the same input feature space and come from the same distribution, as illustrated in Figure 2.12a. E.g. when building a classifier for distinguishing dogs from cats in images, the training data is images of dogs and cats, and the test data is images of dogs and cats. However, in many cases, data is limited, expensive or difficult to obtain, and there might not be resources for collecting enough to train a model sufficiently. Imagine you have built a classifier for sentiment analysis of reviews of a certain product, e.g. digital cameras, and you want to do the same for another completely different product, e.g. food (Pan and Yang, 2010; Weiss et al., 2016). However, there are considerably less reviews about the other product, far from enough to train a sufficient classifier. There are many similarities, as both are reviews, probably text, hopefully in the same language, but there's a difference in domain, which may lead to the first classifier being insufficient for the second product. *Transfer learning* is a method dealing with this by *transferring* knowledge from one domain to another, from a *source task* to a *target task*, see Figure 2.12b. The idea is that some of the information obtained when solving the source task also is useful when solving the target task, making it wasteful to learn the same information again, especially if the target task data is limited. Sentiment analysis of reviews are pretty similar regardless of product. By using the first classifier, and tweaking it with the small set of data about the second product, a high-performance classifier can be trained for this task as well.



(a) Traditional machine learning

(b) Transfer learning

Figure 2.12: Illustration of the difference between traditional machine learning and transfer learning.

Clearly, there must be some relationship between the source domain and the target domain. Picture two individuals trying to learn the piano, one has musical background from playing the guitar, while the other has experience from playing football. Evidently, the knowledge from football is not overly useful, while the one that has played the guitar can use previous knowledge to learn to play the piano faster (Pan and Yang, 2010; Weiss et al., 2016). The same applies for transfer learning. There is not much benefit from using a model trained to classify dogs and cats when trying to classify food reviews, while a classifier for camera

reviews might come in handy. There must be a correspondence in the form of the data, e.g. if the input is images or text, and the source and target domain should be sub-fields of the same domain, like guitar and piano both belong to the music domain, and food reviews and camera reviews both are reviews.

2.2.1 Definitions

In this section, transfer learning is formally defined in a classification setting. A domain $\mathcal{D}$ is defined by its feature space $\mathcal{X}$, and its marginal probability distribution $P(X)$, $X = (x_1, ..., x_n) \in \mathcal{X}$, i.e. $\mathcal{D} = \{\mathcal{X}, P(X)\}$. A task $\mathcal{T}$ is defined within a domain by a label space $\mathcal{Y}$, and a predictive function $f(\cdot)$. The predictive function is learned from observations, (x_i, y_i), and in a probabilistic classification setting, it can be written as $P(Y|X)$. Thus, a task is formally written $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. Further, a source domain and task are denoted $\mathcal{D}_S$ and $\mathcal{T}_S$, while a target domain and task are denoted $\mathcal{D}_T$ and $\mathcal{T}_T$.

In order to apply transfer learning, given a source domain $\mathcal{D}_S$ and task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ with a corresponding task $\mathcal{T}_T$, there must be some sort of relationship between source and target, and some differences. If the source and target domains differ, it means that either $\mathcal{X}_S \neq \mathcal{X}_T$ or $P(X_S) \neq P(X_T)$. If the tasks differ, it would imply that $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$. Generally, there are four different scenarios;

(1) $\mathcal{X}_S \neq \mathcal{X}_T$, the feature spaces are different. This is referred to as heterogeneous transfer learning, and for NLP tasks, it can translate to documents being written in different languages for different tasks. Or it could be that the source data is text, while the target data is images (Zhu et al., 2011). When $\mathcal{X}_S = \mathcal{X}_T$, it is called homogeneous transfer learning.

(2) $P(X_S) \neq P(X_T)$, i.e. the marginal probability distributions are different, which is referred to as domain adaptation. An NLP example is document classification where the documents in the different tasks are concerning different topics. For computer vision, it could be the difference in product images from a web shop and consumer images of the same products.

(3) $\mathcal{Y}_S \neq \mathcal{Y}_T$, the label spaces differ, i.e. the labels of the target task differ from the labels of the source task. This is a common scenario, but it very rarely occurs without also the conditional probability distributions being different.

(4) $P(Y_S|X_S) \neq P(Y_T|X_T)$, the conditional probability distributions differ. This scenario is common in practice, and means that the classes in the source and target data are unbalanced.

Definition 2.2.1. *(Transfer Learning)* Given a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and target task $\mathcal{T}_T$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$, *transfer learning* aims to help learning the predictive function of the target task, $f_T(\cdot)$, by using knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$. (Pan and Yang, 2010).

There are mainly three types of transfer learning, *inductive* transfer learning, *unsupervised* transfer learning, and *transductive* transfer learning.

Definition 2.2.2. *(Inductive transfer learning)* Given a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and target task $\mathcal{T}_T$, *inductive transfer learning* aims to help learning the target task predictive function $f_T(\cdot)$, using knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{T}_S \neq \mathcal{T}_T$. (Pan and Yang, 2010).

From Definition 2.2.2, *inductive transfer learning* is the case when the source and target tasks differ, that is, scenario (3) or (4), or both. The domains may differ as well in this setting, but the difference in tasks categorizes it as inductive transfer learning. This scenario also requires that there is labelled target data available, so that the target predictive function $f_T(\cdot)$ can be *induced* from this data, hence the name. Inductive transfer learning is also most relevant to this thesis, as one of the models utilized in the experiments employs transfer learning from language modelling to text classification.

The only difference between inductive transfer learning and unsupervised transfer learning is that the label spaces $\mathcal{Y}_S, \mathcal{Y}_T$ are not observed in unsupervised transfer learning. Consequently, for unsupervised transfer learning, the focus is on unsupervised methods, such as clustering and dimensionality reduction.

Definition 2.2.3. *(Transductive transfer learning)* Given a source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$, and target domain $\mathcal{D}_T$ and target task $\mathcal{T}_T$, *transductive transfer learning* aims to help learning the target task predictive function $f_T(\cdot)$, using knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. (Pan and Yang, 2010).

Also in the *transductive transfer learning* setting, see Definition 2.2.3, must some labelled data for the target task be available. This covers scenario (1) and (2), i.e. the feature spaces or the marginal distributions differ, whereas the latter is known as *domain adaptation*. Since the source and target tasks are the same, the source predictive function $f_S(\cdot)$ can be adapted to the target task with the available labelled target task data. The word *transductive* in this setting is used to accentuate that the tasks are the same and there's access to labelled target data in this type of transfer learning.

Through the rest of this chapter, when referring to transfer learning, it is in an inductive or transductive setting, without it being stated explicitly which of the two. The main assumptions are that there is some difference in source and target, labelled source data is available or easy to obtain, while labelled target data is sparse or laborious to gather.

2.2.2 Applications

The need for transfer learning appears when training data for a task is challenging or expensive to obtain, inaccessible, or there simply is not much of it. A necessary condition for transfer learning is that there's some related domain where this kind of data is plentiful, accessible, and maybe already used to train a model for a similar task.

One of the most common of the scenarios presented above in Section 2.2.1 is (2), domain adaptation, when the marginal probability distributions of the source and target task don't coincide. Sometimes, the training and test data might look the same to a human, while

for the model, there's a bias, causing overfitting to the training data. For example, when the training data is images of objects without background, testing the model on images of the same objects, but *with* background can induce such a bias. This can happen when the training data is product images from a web shop, since these often are annotated and easy to obtain, and the test data is consumer images of the same products. The product review example from earlier also falls within this category. Different words are used to describe cameras than to describe food, but can still express the same opinion, or sentiment. Moreover, a lot of pre-trained NLP models are usually trained on a large corpus made up by news data or other kind of articles, so in the meeting with less formal text documents, like reviews, or social media messages, the results will not be optimal without fine-tuning.

Relevant for the technological advances of today is speech recognition, as tools like Siri[1], Alexa[2], and Google home[3] infiltrate our everyday lives. These are trained to recognize *standard* accents, making it harder for immigrants, people with dialects or speech impediments, or any other not speaking *standard* to be understood. In addition, while they are getting quite good in English, other languages are behind. When a friend tried to convert Siri to Norwegian, she could not be understood and switched back to English, even though she *is* speaking *standard* Norwegian. Despite that, transfer learning can also be used in a bilingual setting, transferring knowledge across languages. A lot of NLP data is available in English, but more sparse in other languages. Being able to use some of the information obtained in the English language to train better models in other languages would be a great advantage (Czapla et al., 2018), and recent advances shows great improvements in this area (Johnson et al., 2016).

Sometimes, gathering real-world data can, in addition to being time consuming and expensive, even be dangerous, e.g. when training self driving cars. Gathering data from simulations is a great way of lowering risk, expense, and time. This is an example of scenario (2), and as simulations are getting closer to real world data, $P(Y_S|X_S)$ is getting closer to $P(Y_T|X_T)$. Still, simulations will most likely never be able to replicate the real world completely, since the interaction patterns between all physical objects are too intricate to be fully replicated. For self driving cars, simulations are essential[4]. As well as being an easy way of collecting data fast, training can also be accelerated, as learning can be parallelized. Simulated data could also be most practical in robotics and AI. Training models on real robots is too slow, as well as expensive (Rusu et al., 2016), while in AI, training an agent in the real world is not only expensive, but real-world data might be too complex for the agent to learn well (Mikolov et al., 2015). Sometimes, a learner benefits from simpler data.

[1] https://www.apple.com/siri/

[2] https://developer.amazon.com/alexa

[3] https://store.google.com/product/google_home

[4] https://techcrunch.com/2017/02/08/udacity-open-sources-its-self-driving-car-simulator-for-anyone-to-use/?guccounter=1guce_referrer_us=aHR0cDovL3J1ZGVyLmlvL3RyYW5zZmVyLWxlYXJuaW5nL2luZGV4Lmh0bWwguce_referrer_cs=SGJ2VvKWaew0-uu2-Y64Rg

2.2.3 Transfer Learning in Deep Learning

Another motivation for transfer learning arises when training a model is computationally expensive. It has thus become a popular tool in deep learning because of the vast amounts of resources required to train deep neural networks on large challenging datasets. Training a deep model adequately requires substantial amounts of data, it can take weeks, and the time increases for larger and more complex models. There are mainly three challenges to overcome in deep learning, (i) designing and creating a network architecture suited for the task at hand, (ii) tuning the hyperparameters to obtain the best possible result, and (iii) obtain considerable amounts of data for training. These three steps are challenging since (i) requires domain knowledge and experience to build a qualified model, (ii) must typically be done in a trial-and-error fashion, which is tough when the number of hyperparameters is high, and (iii) as mentioned, can be time consuming, demanding, expensive, and even dangerous. Somehow, by trying to lower the data requirement, transfer learning also eliminates (i) as a problem, since model architecture from the source task can be exploited in the target task. As long as one is able to determine if the source and task domains are sub-fields of the same domain, there is not need for much more domain knowledge or experience. Moreover, since many features are already learned, and less data is needed to obtain good results for the target task, training time is reduced considerably.

In deep neural networks, early layers capture general features, while later layers capture more task specific features in the data, as illustrated in Section 2.1.1 under *Convolutional Neural Networks*. A transfer learning approach in deep learning is to take a pre-trained model, *freeze* the early layers, and *fine-tune* the later layers to fit the target task, and in that manner avoid wasting data and time on learning general features that are already learned. Both in computer vision and NLP, one aspect is to look for general tasks that can serve as source task for many target tasks. With large publicly available datasets, like ImageNet[5] and WikiText-103[6], the opportunity for creating large pre-trained models is present, the next challenge is to train good models for suited source tasks, that can easily be generalized to different target tasks. In the following, transfer learning in a computer vision and NLP setting will be discussed, along with some known datasets and pre-trained models.

Computer Vision

Transfer learning has become an important tool in computer vision (Long et al., 2014; Weiss et al., 2016; Huang et al., 2018), and applied computer vision models are seldom trained from scratch. Pre-trained models are trained on large labeled, and well known, image datasets, such as ImageNet, COCO[7], or other.

ImageNet is a large image database organized in accordance to the WordNet[8] hierarchy.

[5] http://www.image-net.org/

[6] https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/

[7] http://cocodataset.org/#home

[8] https://wordnet.princeton.edu/

This hierarchy is an ordering that groups words and expressions in so called *synonym sets*, or *synsets* for short, where each synset represents a specific concept. In ImageNet, there are more than 100'000 synsets, whereas most are nouns, which makes sense since nouns are more intuitive to photograph. The ImageNet synsets have on average 500 images, and the goal is to double this, leaving plenty of image data for most categories. COCO is another large-scale image dataset for object detection, segmentation, and captioning. It contains more than 300'000 images, of which more than 200'000 are labeled, and combined there are more than 1.5 million object instances in over 80 object categories. There are large-scale image recognition contests based on both these datasets, where the best models often become publicly available pre-trained models used for transfer learning.

One such example is AlexNet (Krizhevsky et al., 2012), the winner of ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012[9]). AlexNet is a convolutional neural network developed by Alex Krizhevsky, with five convolutional layers, some followed by max pool layers, three fully connected layers, and combined 60 million parameters and 650'000 neurons. Krizhevsky et al. (2012) also applied the then new regularization technique dropout, and used GPUs for more efficient training of the large model. A second model is VGG (Simonyan and Zisserman, 2015), a very deep convolutional network for large-scale image recognition, which won the ILSVRC2014[10] challenge. Simonyan and Zisserman (2015) explored CNNs of increasing depth with very small 3×3 convolutional filters, and achieved state-of-the-art results with 16-19 layers. Another is ResNet (He et al., 2015), the winner of the same challenge in 2015, ILSRVC2015[11], and the COCO 2015 competition[12]. In contrast to AlexNet and VGG, ResNet is a residual network, a kind of network that eases the complexity, allowing substantially deeper networks than earlier, without impacting the training time excessively. Residual networks considers the *residual* function $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$, where $\mathcal{H}(\mathbf{x})$ is the underlying mapping fitted by a few stacked layers in a network, but not necessarily the entire model, and $\mathbf{x}$ is the input to the first of these layers. Note that $\mathcal{H}(\mathbf{x})$ must be of the same dimension as $\mathbf{x}$ when the residual function is $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The idea is then to let the stacked layers approximate $\mathcal{F}(\mathbf{x})$ instead of $\mathcal{H}(\mathbf{x})$. The model ends up having 152 layers, which is 8 times as deep as VGG, while still having lower complexity.

Natural Language Processing

While transfer learning has made a great impact on computer vision, equivalent general approaches have not been available for NLP tasks before recently (Howard and Ruder, 2018). The already existing methods would mostly work only for very similar tasks, and would often be in need of a large set of in-domain documents to obtain good results, which in a sense contradicts the motivation for transfer learning. However, Howard and Ruder (2018) have exploited that *language modelling* (LM) on a large corpus can be used equivalent to large-scale image recognition model trained on ImageNet or COCO in computer vision, and they claim that the resulting method, *Universal Language Model Fine-Tuning*

[9] http://image-net.org/challenges/LSVRC/2012/index
[10] http://image-net.org/challenges/LSVRC/2014/index
[11] http://image-net.org/challenges/LSVRC/2015/index
[12] http://cocodataset.org/#detection-2015

(ULMFiT), can be applied to any NLP task. Language modelling is briefly described in Section 2.1.2. Building a high-performance LM can be challenging, typically the model is a recurrent network, see Section 2.1.2, and a large corpus is necessary for training. LM can be used as is, to generate text, or as a base for other NLP tasks, and is crucial in problems such as machine translation, speech recognition, spelling correction, handwriting recognition, automatic summarization, and much more.

ULMFiT is based on the most general inductive transfer learning setting for NLP (Pan and Yang, 2010), see Section 2.2.1, and there are three steps for building and training a classifier. The first is to train a language model, or to take a pre-trained language model. Howard and Ruder (2018) have made a publicly available AWD LSTM based on the AWD LSTM by Merity et al. (2017), through the `fastai`[13] framework. It is trained on the WikiText-103 dataset, a dataset containing more than 100 million tokens, gathered from verified articles of Wikipedia. The next step is to fine-tune the later layers of the LM with data from the task at hand, before the last step, which is augmenting the LM to a classifier, and fine-tuning it by gradually unfreezing layers.

Since the results of ULMFiT were published, the method has been used in several NLP tasks, such as language modelling for Polish (Czapla et al., 2018), text classification for low resource tasks using backtranslation (Shleifer, 2019), and suggestion mining (Anand et al., 2019). Czapla et al. (2018) stress the importance of transfer learning in a bilingual setting, since most languages other than English do not have large labelled datasets available for training good language models. By adapting ULMFiT with sub-word tokenization to handle Polish, Czapla et al. (2018) came in first in Task 3 of the Polish LM competition PolEval18[14]. Also emphasizing the need for transfer learning in low resource tasks is Shleifer (2019). By augmenting data with backtranslation and utilizing ULMFiT, Shleifer (2019) significantly improves the results of low resource NLP tasks. Backtranslation is an augmentation method for NLP data, where a text is translated to another language, then *backtranslated* in order to obtain a slightly altered text. Suggestion mining from online text documents is the task of determining whether a review expresses a suggestion or not. Suggestions can be found in informal online text documents such as reviews, forums, blogs, or social media posts or comments. In industry, suggestion mining is advantageous when developing a product, especially when direct reviews are sparse, since very few actually leave reviews directly. Many would rather complain about or recommend products in social media or forums. Anand et al. (2019) build a model for suggestion mining based on ULMFiT, resulting in 10^{th} place for Sub Task A of SemEval 2019[15]. ULMFiT is also used in this thesis, to see how transfer learning works together with *active learning*, which will be discussed next, in Section 2.3.

Devlin et al. (2018) aim to improve fine-tuning based approaches to transfer learning in NLP (Howard and Ruder, 2018; Radford et al., 2018) by proposing *masked language modelling* (MLM) as source task. MLM is a language model that, rather than predicting the

[13] `https://docs.fast.ai/`
[14] `https://n-waves.com/poleval2018`
[15] `http://alt.qcri.org/semeval2019/`

next word in a sequence of words, masks random tokens, and intends to predict the masked tokens based on context. This approach opens for *bidirectional* training, as left and right context can be fused, as opposed to in left-to-right LMs. Devlin et al. (2018) develop a model based on transformers (Vaswani et al., 2017), and propose the language representation model Bidirectional Encoder Representations from Transformers (BERT), which outperforms the state-of-the-art in eleven NLP tasks, including named entity recognition, sentiment analysis, and question answering.

2.3 Active learning

As stated in Section 2.2.3, there are three leading challenges in deep learning. These are (i) designing an architecture efficient and convenient for the task at hand, (ii) tuning the hyperparameters of the model, and (iii) having enough available labelled data for training. *Active learning* (AL), like transfer learning, is a method developed to lower the data requirement in deep learning, and machine learning in general. Even in cases where data is available, labelling it can be extremely tedious, demanding without domain expertise, and thus often expensive. By letting the model *actively* choose examples to learn from, the aim of active learning is to gain better results with far less data, hence reducing the labelling cost significantly. Note that active learning will be discussed in a classification setting, seeing that it is most relevant for this thesis.

A simple active learning setup consists of a large *pool* of unlabelled data $\mathcal{U}$, a small labelled set $\mathcal{L}$, a model, an informativeness measure, also known as an *acquisition function*, an oracle, and a labelling budget. The model is first trained on $\mathcal{L}$, then, with the knowledge it has gained from $\mathcal{L}$, it selects one instance from $\mathcal{U}$ based on its *informativeness*. The selected instance is then labelled by an *oracle*, often a human annotator, and added to $\mathcal{L}$. The model is trained from scratch with the updated labelled set, and with its new knowledge, new instances are queried from $\mathcal{U}$. This loop is repeated until the *labelling budget* is empty, or a satisfactory classification accuracy is obtained. This scenario is appropriately called pool-based active learning, which will be described in more detail in Section 2.3.1, while ways of measuring the informativeness is presented in Section 2.3.2. Mainly, informative examples can be found near class borders, since these are areas the model would be interested in learning more about.

The need for active learning arises when data is available, but labels are expensive to obtain, either in the sense of money, time, or experience, or a combination of the three.

One such example is classification of magnetic resonance imaging (MRI) scans. To acquire labels, one needs an experienced radiologist to interpret the scans, and the process is immensely tedious and expensive. Getting an MRI scan in the U.S., according to Time Magazine based on Medicare data[16], can have a cost ranging from $474 to unbelievable $13'259, depending on location, hospital, and which MRI procedure is performed. E.g. an MRI of the brain is more expensive than an MRI of the shoulder. On average, the cost comes at $2'611. Assuming the professional fee, i.e. the fee related to the interpretation by a radiologist, is 20%, on average the cost will be $522 per label, *if* the MRIs themselves are already available. To achieve a sufficient amount of labels to train a classifier to acceptable accuracy level, labelled data will come at several $100'000, maybe even millions.

Another example is *speech recognition*. To obtain data, there are vast amounts of sound clips available online, or one can simply press record on a recording device. Labelling the

[16] `http://time.com/money/2995166/why-does-mri-cost-so-much/`

data, on the other hand, can be extremely time consuming, and requires a human expert in linguistics. According to Zhu (2005), transcription can take up to 10 times the length of the audio at word-level, and as much as 400 times the recording's duration at phonetic level. If one hour of audio takes 400 hours, i.e. more than 16 days or 48 working days, to interpret, imagine how much work lies behind the data used for training a network, or how much it will cost to pay the linguist expert for the job.

In many NLP tasks, such as document classification, and information extraction, acquiring labels has various cost for different documents (Settles and Craven, 2008a). Not only can document length differ excessively, the language complexity and domain may affect the labelling cost as well. For instance, some domains, like scientific documents, require experts for labelling, and come at a higher labelling cost. Nonetheless, even simple newswire documents can take up to half an hour to properly label. How long will it take to label a complex article within some expert acquiring domain?

Active learning is a great way to lower labelling cost, however, the computational cost can be extensive. In the "traditional" approach, only one instance is added to $\mathcal{L}$ at each round, and the model is trained from scratch every time $\mathcal{L}$ is updated. This can lead to days, or even weeks, of training, which is unacceptable in many of today's practical systems. There are several proposed solutions for this issue, e.g. adding a larger sample to $\mathcal{L}$ at each round would reduce the computational time considerably. Nevertheless, this could lead to sampling of similar examples, since similar examples have similar informativeness, thus, it could be a waste of labelling budget. Other solutions include incremental learning and transfer learning. While incremental learning might lead to a bias towards the initial labelled set, transfer learning is more general, as knowledge is transferred from similar tasks. Some results from applying these solutions are further discussed in Section 2.3.3, and additionally, adding larger samples to $\mathcal{L}$ and transfer learning will be investigated in this thesis.

2.3.1 Active Learning Scenarios

Active learning is possible when the model is able to query examples, which it is in several problem settings. According to literature (Settles, 2009), the main scenarios are *query synthesis* and *sampling*, whereas the latter is divided into *stream-based* and *pool-based*, emerging at the three central scenarios,

(i) Membership query synthesis,

(ii) Stream-based selective sampling,

(iii) Pool-based sampling.

The seemingly most common, and probably most intuitive scenario, is pool-based active learning, which was introduced as an introductory example earlier. All three are described in the following, even though only pool-based is relevant for the experiments of this exploration.

Membership Query Synthesis

Membership query synthesis is the combination of *membership querying* and *query synthesis*. Simply explained, membership querying is asking if a sample is member of some set (Angluin, 1988), and query synthesis is a scenario where the learner can generate samples based on the input space. The combined scenario is that the learner can generate a sample, ask whether it is part of the labelled set $\mathcal{L}$ or not, then proceed to ask for its label if the answer is not. Membership query synthesis enables the learner to ask for any instance in the input space, and is efficient in settings where the problem domain is finite (Angluin, 2001). To acquire informative examples, synthesized queries will typically be near class borders.

A problem with membership query synthesis was discovered when Lang and Baum (1992) tried to use active learning with a human oracle to classify handwritten characters. The learner generated examples with no recognizable symbols, making them impossible for the human oracle to label. From this, one could imagine that a range of tasks would encounter the same problem. E.g. in NLP tasks, the learner might generate sequences of words without any meaning, making it impossible for a human annotator to give it a label.

However, membership query synthesis can be useful in settings where the oracle is not human. In a bid to automate the scientific process, King et al. (2004) use active learning and a *robot scientist* to perform biological experiments on the yeast *Saccharomyces cerevisiae*. Each instance consists of a growth medium and a yeast mutant, and the label is whether or not the yeast thrives in said medium. The whole process is autonomous, from originating hypothesis in order to describe observations, to physically performing experiments, and analyzing the results. The composition of the growth medium is subject for synthesizing, resulting in a 3-fold and 100-fold decrease in cost in comparison to cheapest and random experiments, respectively.

More recent advances exhibit a query synthesis active learning approach in combination with *generative adversarial networks* (GAN) (Zhu and Bento, 2017). A GAN (Goodfellow et al., 2014) is a combination of two models, one generative, the *generator* G, and one discriminative, the *discriminator* D. While discriminative algorithms aim to classify input data, i.e. find the probability of the label y given the features x, $p(y|x)$, generative models seek the opposite, finding the probability of the features x given the label y, $p(x|y)$. The steps of GANs is to feed labels to G, which will generate features, then the generated input together with *real* input is fed to D, which job is to label the input as *real* or *fake*. G's objective is for D to fail, i.e. to generate as realistic data as possible, making GANs able to create data of uncanny resemblance to the real world. Zhu and Bento (2017) use GANs to synthesize queries, avoiding the problems of Lang and Baum (1992) since GANs are constructed to return seemingly authentic data. Although the approach produces some good results, it can not outperform pool-based approaches in all settings, probably since the GAN always will generate examples close to class boundaries, while in pool-based approaches, the model can explore more of the data space when all boundary examples are already in the training set.

Stream-Based Selective Sampling

The problem with query synthesizing is mainly that when data is generated from the whole input space, there are parts of the input space that doesn't make sense, and is impossible to interpret. *Stream-based selective sampling* (Cohn, 1994; Cohn et al., 1994) solves this by sampling directly from the underlying distribution. If the distribution is uniform, the behavior is similar to query synthesis, nonetheless, when the distribution is non-uniform, or unknown, there is a guarantee that the samples drawn are representative for real world data.

A crucial assumption for selective sampling to be advantageous is that sampling from the underlying distribution is free, or at low cost compared to the cost of labelling. When that being the case, instances can be sampled, then the learner decides whether to query their labels, or discard them. Commonly, examples are drawn one at the time, and the learner must determine whether the label is worth asking for before seeing the next instance, thus called *stream-based* or *sequential* active learning. There are several ways of making this decision, and it is often based on the informativeness according to some query strategy, see Section 2.3.2. One approach is to compute the informativeness, and query examples in a biased randomised matter, such that more informative examples are queried (Dagan and Engelson, 1995). Another is to define a region where the learner is uncertain, a *region of uncertainty*, and only query instances within this region (Cohn et al., 1994).

Stream-based active learning is useful in tasks such as part-of-speech tagging (Dagan and Engelson, 1995), i.e. marking words in speech as verb, noun, adjective etc. Another example is word sense disambiguation (Fujii et al., 1998), a task concerned with finding the meaning of words that can have several meanings. E.g. the word "lead" can be both a verb and a noun, and in both categories, it can have more than one meaning. Stream-based selective sampling can also be useful in the development and training of chatbots. Chatbots receive questions in a stream-based matter, and are often trained with a *human-in-the-loop* approach, where they *should* ask for assistance when confidence is low.

Pool-Based Sampling

Pool-based sampling is somewhat more intuitive than the other two active learning scenarios. As indicated by the name, a large *pool* of unlabelled data, denoted by $\mathcal{U}$, is available, along with a small labelled set $\mathcal{L}$. The labelled set is often initialized by annotating a random subset of $\mathcal{U}$. A model is trained on $\mathcal{L}$, then it scans through the instances in $\mathcal{U}$ computing the informativeness, and queries one or more of the most informative examples. The model is trained again on the updated labelled set, and with its new knowledge, new instances are queried from $\mathcal{U}$. An illustration is provided in Figure 2.13. Where stream-based sampling examines instances in a sequential fashion, considering each example individually, pool-based sampling compares the informativeness of all unlabelled data before making a decision. A stream-based approach can hence be preferable when there are memory restrictions when solving a problem.

The pool-based scenario might be the most described in literature (Settles, 2009), and

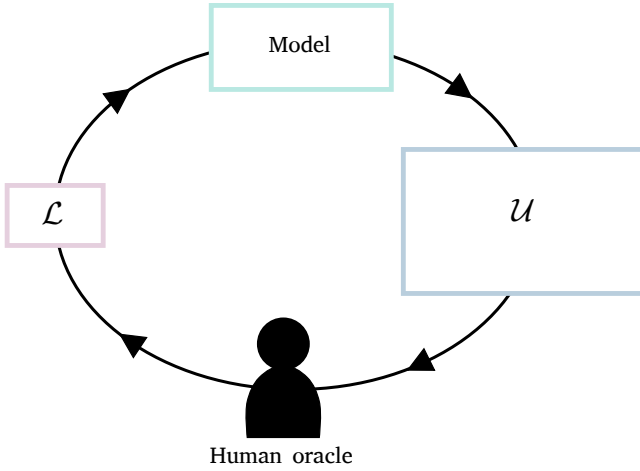


Figure 2.13: Illustration of a typical pool-based active learning loop.

applies to a wide range of real-world problems. Pool-based sampling is relevant when large quantities of data can be gathered at once, which applies to many tasks in modern society, as heaps of data is stored at every second, in the hopes of being useful in some way. As well as a great deal of content being published for information and enlightenment, or entertainment to the public, such as news articles, scientific papers, podcast clips, images or videos. However, when the application is not yet known, *if*, *how*, and *what* to label is not always obvious. In consequence, when a problem suited for machine learning emerges, the data is often available, but the labels are missing. In this setting, pool-based active learning is useful.

Given the large amounts of data available, examples of pool-based applications are plentiful. Various text documents can be used for categorical classification, while reviews, social media messages, or forum posts can be exploited in sentiment analysis. Videos can not only be used for video captioning or other video or image tasks, the sound can be used for e.g. speech recognition. Lewis and Gale (1994), McCallum and Nigam (1998), and Tong and Koller (2002), are just some of the researchers who successfully have studied pool-based active learning for text classification, while Thompson et al. (1999) and Settles and Craven (2008b) use text data for information extraction. There are also successful examples of image and video classification, speech recognition, and cancer diagnosis classification.

2.3.2 Query Strategy Frameworks

One of the most important aspects of active learning is the question of how to measure informativeness. There are a variety of query strategy frameworks, whereas *uncertainty sampling* probably is the most known. In the following, a number of ways to measure uncertainty is presented, in addition to two other frameworks, *query-by-committee* and *expected model change*, although uncertainty sampling is the only relevant framework in the

experiments. Let x^*_{AL} denote the most informative instance according to the query strategy AL. Moreover, $p_{\boldsymbol{\theta}}(y|x)$ will denote the predictive probability w.r.t. model parameters $\boldsymbol{\theta}$ of x belonging to class y, $p(y|x) \in \{0, 1\}$ the true probability of x belonging to class y, and $\hat{y}$ the prediction for instance x, i.e.

$$\hat{y} = \underset{y}{\operatorname{argmax}}\big\{p_{\boldsymbol{\theta}}(y|x)\big\}.$$

Further, when talking about the *most informative* instance, it would be in a pool-based setting, seeing that in membership query synthesis and sequential selective sampling, examples are considered individually.

Uncertainty Sampling

As already mentioned, uncertainty sampling might be the most well known of the query strategy frameworks of active learning. The idea is simple, query examples the model is most *uncertain* about to gain more certainty in uncertain areas, typically around class borders. For probabilistic models, uncertainty sampling is quite unambiguous, and combined with binary classification, one simply queries the instance for which the predictive probability is closest to 0.5. The next question to answer is how to compute the uncertainty of an instance in a multiclass classification setting. The perhaps simplest method is to query examples by the *least confidence* principle, selecting instances for which the learner is least confident in its prediction. This is done by maximizing the *variation ratio*, given by

$$\text{variation ratio}(x) = 1 - p_{\boldsymbol{\theta}}(\hat{y}|x).$$

The most informative example is thus given by

$$x^*_{LC} = \underset{x}{\operatorname{argmax}}\big\{1 - p_{\boldsymbol{\theta}}(\hat{y}|x)\big\}. \tag{2.4}$$

When the predictive probability of x belonging to $\hat{y}$ is small, i.e. the variation ratio is large, it means that the model is *uncertain* in the prediction, as it would suggest that the sum of the predictive probabilities of the other classes is large. It might indicate that the instance is near a class border, making it a good candidate for querying.

However, least confidence uncertainty takes only one probability into account. Even though it seems like the model is uncertain about the instance x, since the probability $p_{\boldsymbol{\theta}}(\hat{y}|x)$ is small, one can not know how much more certain it is about x belonging to class y than to e.g. class z. To incorporate this, *margin sampling* compares the probability of x belonging to the most probable class and the second most probable class by looking at the difference, or *margin*,

$$M_\theta(x) = p_{\boldsymbol{\theta}}(\hat{y}_1|x) - p_{\boldsymbol{\theta}}(\hat{y}_2|x),$$

where $\hat{y}_1$ and $\hat{y}_2$ are the most probable and second most probable labels for x, respectively, w.r.t. the model parameters $\boldsymbol{\theta}$. If the margin is large, the model is quite confident that x

belongs to $\hat{y}_1$ compared to other labels. If the margin is small, on the other hand, an instance is almost as likely to belong to the second most probable class as the most probable class, thus, informative instances minimize the margin,

$$x_M^* = \underset{x}{\operatorname{argmin}}\big\{M_{\boldsymbol{\theta}}(x)\}. \tag{2.5}$$

A small margin implies that an instance lies in the border area between the two classes considered, however, there is no information on remaining classes. Margin sampling takes into account some of the information of the distribution of the other labels, although, when the number of classes is large, a great part of the label distribution is still ignored.

An unceratainty sampling method considering *all* class probabilities, making a more general measure, takes the *entropy* (Shannon, 1948) as a measure of informativeness. The entropy, often referred to as *Shannon entropy*, is an information-theoretic measure, which for a stochastic process, measures the average rate information is gained or produced. It is given by

$$H(X) = -\sum_i p(x_i)\log p(x_i),$$

for a random variable X, and discrete probabilities $p(x_i)$ for the possible realizations of X. That is, unlikely events are more *informative* than likely events. In statistical mechanics, entropy is the measure of *chaos* within a substance, which is not a bad interpretation in information theory either. Interpreting *chaotic* as *uncertain*, the most informative instance maximizes the entropy

$$x_H^* = \underset{x}{\operatorname{argmax}}\Big\{-\sum_i p_{\boldsymbol{\theta}}(y_i|x)\log p_{\boldsymbol{\theta}}(y_i|x)\Big\}, \tag{2.6}$$

where y_i for $i = 1, .., C$ are the possible labels.

In a binary classification setting, all three presented methods reduce to the same, namely choosing the instance which predictive probability is closest to 0.5. Figure 2.14 displays plots of $-p\log p$ against p (left), and for all possible probabilities p, the entropy in a binary classification setting against p (right). The right plot shows how $p = 0.5$ maximizes the entropy. In a multi-class setting, the left plot can also give a good indication on how the entropy behaves. The entropy when one class probability is close to 1, and the rest are close to 0, would give a rather small entropy. On the other hand, picture a four-class problem where all probabilities are at approximately 0.25, then the entropy is evidently large.

Even though active learning has become important in machine learning, applying it to deep learning is not straightforward (Gal et al., 2017). One of the challenges is the combination of deep models relying on large amounts of data, while active learning tries to accomplish the opposite. Second is that uncertainty acquisition functions, as the one discussed, often are relying on model uncertainty, which is not always represented in deep learning models. The predictive probabilities, i.e. softmaxed outputs of deep neural networks, does not

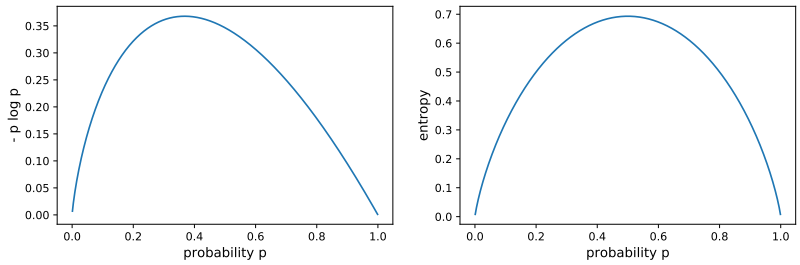


Figure 2.14: $-p \log p$ plotted against p (left), and the entropy in a binary classification problem, $-\big(p \log p + (1-p) \log(1-p)\big)$, plotted against all possible probabilities p.

necessarily represent the confidence the model has towards its prediction. To overcome the second challenge, Gal et al. (2017) introduce an active learning framework based on Bayesian approximations using dropout (Gal and Ghahramani, 2016) and Bayesian CNNs. The Bayesian CNN places a prior on the set of model parameters, $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$, and defines a likelihood as

$$p(y|x, \boldsymbol{\theta}) = \text{softmax}(f_{\boldsymbol{\theta}}(x)),$$

where $f_{\boldsymbol{\theta}}(x)$ denotes the model output. Further, to approximate the posterior, inference is made by applying dropout at test time, thus every forward pass is made by a sub-model of the entire network, resulting in slightly different probabilities at each forward pass. The approximation is done by taking an average over several forward passes, motivated by the approximation

$$\begin{aligned} p(y = c|x, \mathcal{L}) &= \int p(y = c|x, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{L}) d\boldsymbol{\theta} \\ &\approx \int p(y = c|x, \boldsymbol{\theta}) q_\gamma^*(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &\approx \frac{1}{T} \sum_{t=1}^{T} p(y = c|x, \hat{\boldsymbol{\theta}}_t), \end{aligned} \tag{2.7}$$

where $\hat{\boldsymbol{\theta}}_t \sim q_\gamma^*(\boldsymbol{\theta})$ is the activated parameters at forward pass t, $q_\gamma^*(\boldsymbol{\theta})$ is the dropout distribution, and T is the number of forward passes made. The first approximation can be made since q_γ^* minimizes the Kullback-Leibler (KL) divergence to the posterior $p(\boldsymbol{\theta}|\mathcal{L})$. This approximation for $p(y = c|x, \mathcal{L})$ is then used in Equation (2.4), (2.5), and (2.6), to maximize the variation ratio, minimize the margin, and maximize the entropy, respectively.

When using dropout at test time, another uncertainty measure could be to compute the

variability in the predictions. Still, an approximation to $p_{\boldsymbol{\theta}}(y = c|x, \mathcal{L})$ is computed according to Equation (2.7), then the variability is computed by

$$V(x) = \sum_{i,j} \big(p(y = i|x, \hat{\boldsymbol{\theta}}_j) - p(y = i|x, \mathcal{L})\big)^2,$$

and maximized to obtain the most informative instance,

$$x^*_{DV} = \underset{x}{\operatorname{argmax}}\Big\{ \sum_{i,j} \big(p(y = i|x, \hat{\boldsymbol{\theta}}_j) - p(y = i|x, \mathcal{L})\big)^2 \Big\}. \tag{2.8}$$

Here $i = 1, ..., C$ are the class labels, and $p(y = i|x, \hat{\boldsymbol{\theta}}_j), j = 1, ..., T$, is the probability of x belonging to class i according to sub-model j. High variability will then imply uncertainty in the model, seeing as it would indicate that different sub-models yield very dissimilar results.

Query-by-Committee

Another query strategy framework is known as *query-by-committee* (QBC) (Seung et al., 1992). A *committee* of models, $\mathcal{C} = \{\boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(T)}\}$, representing different hypotheses, are trained on $\mathcal{L}$, then all committee members have a *vote* when predicting the label of an instance x. x is considered informative if there's high *disagreement* on its label among committee members. The committee members should represent hypotheses within the *version space*, which is defined by the set of hypothesis separating $\mathcal{L}$ correctly according to classes, or, in other words, are consistent with $\mathcal{L}$. The underlying goal of QBC, and active learning in general, is to minimize the version space, constraining the search within the version space for the best model. The basic QBC setup is thus (i) a committee of models representing different regions of the version space, and (ii) a measure of disagreement.

One way of constructing a committee of models is by using dropout similarly to Gal et al. (2017) in uncertainty sampling, i.e. applying dropout at test time. Each prediction is then performed by a *sub-model* of the entire network, thus, distinct predictions can be seen as the prediction of different committee members. Another way to put it, committee members are sub-models sampled from the entire model with the dropout distribution. The next question is how many members a committee should have. Even small committees of two or three members have shown good results (Seung et al., 1992), however, there is not a common consensus on the optimal committee size in literature (Settles, 2009).

Measuring the disagreement is mainly done in two ways, *vote entropy* (Dagan and Engelson, 1995) and *average Kullback-Leibler divergence* (McCallum and Nigam, 1998). The simplest strategy is vote entropy, which computes the entropy with $\frac{V(y)}{T}$ as the probability that x belongs to class y, where T is the number of committee members, and $V(y)$ is the number of votes y receives for x. As for entropy uncertainty sampling, the vote entropy is maximized in order to find the most informative instance,

$$x^*_{VE} = \underset{x}{\operatorname{argmax}}\Big\{ -\sum_i \frac{V(y_i)}{T} \log \frac{V(y_i)}{T} \Big\},$$

where, y_i ranges over all classes.

The second approach aims to maximize the KL divergence between each committee member and the committee as a whole. If the KL divergence is large between a certain committee member and the *consensus*, the committee member *disagrees* with the common belief. For each committee member, the KL divergence between a member and committee is defined by

$$KL(p_{\boldsymbol{\theta}^{(t)}} \| p_{\mathcal{C}}) = \sum_i p_{\boldsymbol{\theta}^{(t)}}(y_i|x) \log \left(\frac{p_{\boldsymbol{\theta}^{(t)}}(y_i|x)}{p_{\mathcal{C}}(y_i|x)} \right),$$

where $y_i, i = 1, .., C$ are the possible labels, $p_{\boldsymbol{\theta}^{(t)}}$ represents a particular committee member, while $p_{\mathcal{C}}$ represents the whole committee, i.e. $p_{\mathcal{C}}(y_i|x) = \frac{1}{T} \sum_{t=1}^{T} p_{\boldsymbol{\theta}^{(t)}}(y_i|x)$. To select instances, the average KL divergence over all committee members is maximized,

$$x^*_{KL} = \underset{x}{\operatorname{argmax}} \Big\{ \frac{1}{T} \sum_{t=1}^{T} KL(p_{\boldsymbol{\theta}^{(t)}} \| p_{\mathcal{C}}) \Big\}.$$

In addition to these two disagreement measures, similar computations can be done with other uncertainty measures, such as margin and variation ratio. Moreover, instead of committee members giving hard votes on the label of an instance, the posterior label probabilities can be used, together with weights reflective upon the model uncertainties of the committee members.

Expected Model Change

The third and last active learning query strategy framework to be discussed here is *expected model change*, which queries instances that are expected to change the model the most if their labels were known. Most common in this framework is *expected gradient length* (EGL) (Settles and Craven, 2008b), which can be applied to any model using gradient-based training. It interprets *model change* as *gradient change*, and selects examples that are likely to change the training gradient the most given that their labels are known.

Let $\nabla \ell_{\boldsymbol{\theta}}(\mathcal{L})$ be the gradient of the objective function ℓ w.r.t. the model parameters $\boldsymbol{\theta}$, then $\nabla \ell_{\boldsymbol{\theta}}(\mathcal{L} \cup \langle x, y \rangle)$ would be the gradient if $\langle x, y \rangle$ is added to $\mathcal{L}$. Further, for computational efficiency, this can be approximated with $\nabla \ell_{\boldsymbol{\theta}}(\langle x, y \rangle)$ since $\nabla \ell_{\boldsymbol{\theta}}(\mathcal{L}) \approx 0$ when the model is trained to convergence. Since the label y for an instance x is not known, the gradient length must be calculated as an expectation over all possible labels, and this expectation should be maximized,

$$x^*_{EGL} = \underset{x}{\operatorname{argmax}} \Big\{ \sum_i p_{\boldsymbol{\theta}}(y_i|x) \| \nabla \ell_{\boldsymbol{\theta}}(\langle x, y_i \rangle) \| \Big\}.$$

Seeing as the informativeness is calculated over all possible labels, the queried examples are the ones that would have the greatest impact on the model parameters, independent of the actual label queried.

Similar to uncertainty sampling methods, EGL could also be performed in a Bayesian setting, approximating the posterior $p(y = c|x, \mathcal{L})$ as an average over several forward passes with dropout, see Equation (2.7). However, considering EGL could be computationally expensive if the feature space and label set is large, a Bayesian approach may not be beneficial, as it would increase the computational cost even more.

2.3.3 State-of-the-Art

Although active learning decreases the data requirement in machine learning, the computational cost, especially training time, can increase significantly. One reason for this is the slow increase of the labelled set $\mathcal{L}$, due to only one example being added at the time, and consequently the model is re-trained numerous of times. Zhdanov (2019) argues for adding a mini-batch of size B at each round, and uses the informativeness combined with diversity in the mini-batch to choose examples. By clustering the unlabelled pool $\mathcal{U}$ and choosing the examples closest to the cluster centres in each cluster, Zhdanov (2019) shows good results compared to random sampling and uncertainty without clustering. The clustering is done by K-means clustering, where the informativeness is *incorporated into the K-means objective function*. This is why the examples closest to cluster centres are of interest, and not the most informative example in each cluster, since informativeness already is accounted for. The approach is tested on both NLP and computer vision tasks, and performed on the Browse Nodes UK dataset[17], the 20 Newsgroup dataset[18], MNIST[19] and CIFAR-10[20].

In another bid to reduce computational cost in active learning, Shen et al. (2017) propose incremental deep active learning for named entity recognition (NER). Instead of retraining from scratch at each round, newly labelled instances are mixed in with the old, and the network's weights are updated by training on the updated labelled set for a few epochs. The computational cost is additionally reduced by the model architecture, consisting of a convolutional character-level encoder, a convolutional word-level encoder, and an LSTM tag decoder, a lightweight architecture for NER tasks. The model is used on the OntoNotes-5.0[21] English and Chinese datasets, and examples are actively queried by least confidence, maximum normalized log-probability, and Bayesian Active Learning by Disagrement (BALD). When testing, the proposed CNN-CNN-LSTM model turns out to be approximately twice as fast as a CNN-CNN-CRF model, and 44% faster than a CNN-LSTM-LSTM model on the same datasets, while still obtaining good results. The active learning algorithms all have similar results which outperform the baseline random sampling. With only 24.9% of the available English data and 30.1% of the Chinese data, they obtain 99% performance of the best model trained on the entire set.

Wang et al. (2017) also incorporate incremental learning in their novel active learning framework, *Cost-Effective Active Learning* (CEAL). The efficiency comes from assigning

[17] http://amazonnodes.com/
[18] http://qwone.com/~jason/20Newsgroups/
[19] http://yann.lecun.com/exdb/mnist/
[20] https://www.cs.toronto.edu/~kriz/cifar.html
[21] https://catalog.ldc.upenn.edu/LDC2013T19

pseudo-labels to high confidence examples, and not only focusing on examples of low uncertainty. As well as identifying and querying low confidence examples, the CEAL framework automatically assigns pseudo-labels to high confidence examples, without including a human annotator, or other expensive oracles. When updating the model, the labelled set, including the newly labelled uncertain examples, and the currently pseudo-labelled examples are utilized. The framework is combined with a CNN, and applied to image classification on the Cross-Age Celebrity Dataset[22] (CACD) and the object dataset Caltech-256[23]. The algorithm shows an improvement over random sampling and state-of-the-art active learning methods.

Since deep models rarely represent model uncertainty, Gal et al. (2017) use dropout at test time to do a Bayesian approximation, and use the approximated model uncertainty to compute informativeness in four different ways. The query strategies are maximizing the entropy, maximizing the variation ratio, maximizing the standard deviations, as well as maximizing the difference between the entropy and expected entropy, i.e. the information gained about the model parameters. The method is called *Bayesian Active Learning by Disagreement* (BALD) by Gal et al. (2017). The baseline is random selection. The methods are performed with a Bayesian CNN on the well known MNIST dataset, as well as on the skin cancer diagnosis dataset ISIC2016[24]. Starting with 20 images in the labelled set, ten images maximizing the acquisition functions are added to $\mathcal{L}$ at each loop, for 100 loops. After averaging over three runs, random sampling and maximizing the standard deviations gave similar results, while the other methods could obtain the same accuracy with approximately half of the training set on the MNIST data. Moreover, BALD and max variation ratio did significantly better than their deterministic counterparts, while max entropy did not improve notably.

To overcome the three main challenges of deep learning, which are stated in Section 2.2.3, Huang et al. (2018) combine active learning with transfer learning, exploiting the architecture, parameters, and knowledge from a pre-trained model while lowering labelling cost with active learning. When transfer learning is not incorporated in active learning, the labelling budget is often wasted on querying information held in pre-trained models. In addition to overcome the challenge of designing a neural network, transfer learning also lowers the training cost, since only the last layers are trained at each active learning loop. To select instances for querying, Huang et al. (2018) also introduce the criterion *distinctiveness*, which distinguishes the source task from the target task. There should be a trade-off between the uncertainty and distinctiveness. The resulting algorithm, *Active Deep Model Adaption* (ADMA), is applied with the pre-trained models AlexNet (Krizhevsky et al., 2012), VGG (Simonyan and Zisserman, 2015), and ResNet (He et al., 2015), which are all presented in Section 2.2.3, and further tested on the datasets PASCAL VOC2012[25], Indoor, DOGvsCAT, and INRIA Person Dataset. ADMA gives a clearly better result than

[22] https://bcsiriuschen.github.io/CARC/
[23] https://authors.library.caltech.edu/7694/
[24] https://challenge.kitware.com/#challenge/560d7856cad3a57cfde481ba
[25] http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

random sampling at PASCAL VOC2012 and Indoor[26], and also performs overall better than *Active Incremental Fine-Tuning* (AIFT) (Zhou et al., 2017) on DOGvsCAT[27] and INRIA Person dataset[28].

[26] http://web.mit.edu/torralba/www/indoor.html
[27] https://www.microsoft.com/en-us/download/details.aspx?id=54765
[28] http://pascal.inrialpes.fr/data/human/

Chapter 3

Experiment

This chapter will give a thorough description of the experiments conducted for this thesis, see Section 3.3, along with detailed descriptions of the employed models and data, in Section 3.1 and 3.2, respectively.

3.1 Models

Two models are explored in the experiments, a CNN for sentence classification based on Kim (2014), and an AWD LSTM inspired by Merity et al. (2017), and implemented in the `fastai` library[1]. Both are outlined and described mathematically in the following.

3.1.1 CNN for Sentence Classification

The first model is a CNN for sentence classification, based on Kim (2014). It's a relatively simple model with an embedding layer of dimension $d = 128$, followed by a convolutional layer with three filters of sizes $k = 3, 4, 5$, before a max pool layer and a dropout layer with dropout probability $p = 0.5$, and finally a fully connected linear layer. The output is then softmaxed to obtain class probabilities. An illustration of the architecture is provided in Figure 3.1. The three filters capture different features, before the max pool layer identifies the most salient information. The dropout probability $p = 0.5$ has been shown to give good train and test errors (Srivastava et al., 2014), and is thus set accordingly. In the original model, Kim (2014) makes use of pre-trained word vectors[2] in the embedding layer, but in this case, the embeddings are randomly initialized and learned from scratch. Pre-trained word embeddings would probably give better results in terms of accuracy, nonetheless, the interesting aspect is the effect of active learning. However, pre-training is an interesting aspect, which will be pursued with the next model, see Section 3.1.2. The implementation

[1] `https://docs.fast.ai/index.html`
[2] `https://code.google.com/archive/p/word2vec/`

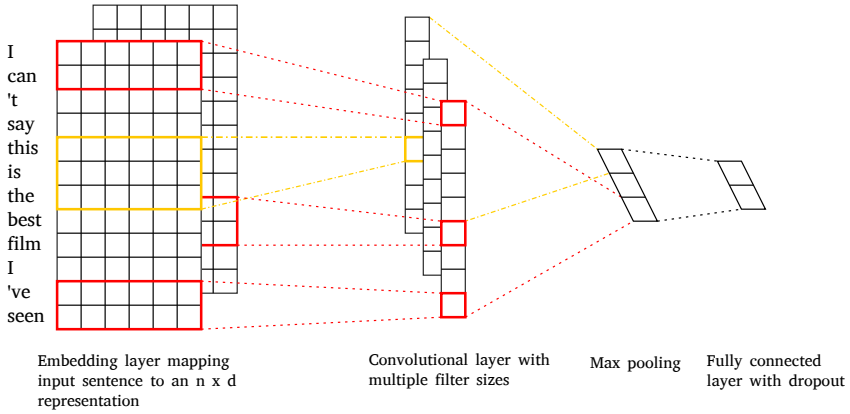


Figure 3.1: Illustration of the model architecture of the CNN for sentence classification.

of the CNN itself, and the training loop, is taken from a publicly available repository with an implementation of Kim's model[3].

Mathematically, the feature extraction can be described as follows. Let the input be a sentence of length n, denoted by $\mathbf{S} = (w_1, ..., w_n)^T$ where $w_i, i = 1, ..., n$ represent the distinct words or tokens in the sentence. Further, let $\mathbf{x}_i \in \mathbb{R}^d$ be the embedding of dimension d of the token w_i, and $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n)^T \in \mathbb{R}^{n \times d}$ the input matrix to the convolutional layer, i.e. the concatenation of the word vectors of the n tokens in $\mathbf{S}$. The concatenation of words, $\mathbf{x}_i, ..., \mathbf{x}_{i+j}$ will be denoted by $\mathbf{x}_{i:i+j}$, thus, $\mathbf{X} = \mathbf{x}_{1:n}$. A filter containing weights, $\mathbf{W}^{(k)} \in \mathbb{R}^{k \times d}$, detects features in the input sentence by taking the convolution between the filter weights and windows of k words. A feature $f_i^{(k)}$ is produced by the convolution between $\mathbf{W}^{(k)}$ and the window $\mathbf{x}_{i:i+k-1}$, i.e.

$$f_i^{(k)} = \phi(\mathbf{W}^{(k)} * \mathbf{x}_{i:i+k-1} + b), \tag{3.1}$$

where b is a bias term, and ϕ is the activation function, in this case ReLU. When applied to all possible windows of size $k \times d$, a *feature map* is constructed by concatenating all the generated features, $\mathbf{f}^{(k)} = (f_1^{(k)}, ..., f_{n+k-1}^{(k)})$. The max pool layer then extracts the most important feature, $\hat{f}^{(k)} = \max\{\mathbf{f^{(k)}}\}$, which will be the feature associated with the particular filter. That is, there are three main features reported after applying the three filters.

The model is trained by minimizing the cross entropy loss with Adam optimization (Kingma and Ba, 2014), and the learning rate is set to 10^{-3}. Further, the model is trained for 50

[3] https://github.com/Shawn1993/cnn-text-classification-pytorch

epochs with early stopping, which terminates training if the accuracy have not improved in the last 250 iterations. The number of epochs is not set higher since the model has a small amount of training data, and should not be trained for too long. The implementation, along with adjustments for the relevant data and active learning, is made publicly available[4].

3.1.2 AWD LSTM

The second model is somewhat more complex than the CNN described above. The *ASGD weight-dropped LSTM* (AWD LSTM) is based on the work of Merity et al. (2017), which propose an LSTM for word-level language modelling, applying DropConnect (Wan et al., 2013) to the hidden-to-hidden weights, and *averaged SGD* (ASGD) (Polyak and Juditsky, 1992) for optimization. DropConnect is a generalization of dropout, where random *weights* are set to zero, rather than random *activations*, such that every neuron receives information from a random subset of the units in the preceding layer. Regular dropout applied to RNNs might *disturb* the hidden state's ability of maintaining long-term dependencies, while DropConnect on the hidden-to-hidden weights does not affect the RNN in the same manner. Further, ASGD is a form of SGD, where weights are updated by considering the average over the last $I - T + 1$ iterations, where I is the total number of iterations, and T is some threshold value indicating when to start computing the average.

An LSTM can be described mathematically by Equation (2.3) where W^i, W^f, W^o are the non-recurrent weights, and U^i, U^f, U^o, U^c the hidden-to-hidden recurrent weights. In the AWD LSTM, DropConnect is applied to the recurrent weights, U^i, U^f, U^o, U^c, and the same weights are reused over a full sequence. That is, a DropConnect mask is drawn at random once per input sequence, so that weights that are set to zero stay zero for the length of the input.

In the `fastai` implementation, there are four kinds of dropout, embedding dropout, input dropout, weight dropout, and hidden dropout. The embedding dropout maps some tokens to a zero vector instead of their embedding, while the input dropout zeros some of the output from the embedding layer. The weight dropout is applied to the hidden-to-hidden weights, and the hidden dropout is applied to the output of the layers of the RNN before being used as input to the next layer. All four dropout varieties are constant for each input sequence, and randomly initialized at the beginning of each sequence. The default dropout probabilities are set to $(0.1, 0.6, 0.5, 0.2)$ for embedding, input, weight, and hidden, respectively. These values are used in the experiments.

Moreover, a pre-trained AWD LSTM language model, trained on WikiText-103 is made available through `fastai`, and a classifier is trained by applying ULMFiT, see Section 2.2.3. First, the last layer of the pre-trained LM is fine-tuned with the relevant data, running one cycle of one epoch of the one cycle policy (Smith, 2018), before the model is unfreezed, and the entire model is fine-tuned with one cycle of 10 epochs and early stopping monitoring the accuracy. The maximum learning rate is set to 10^{-2} and 10^{-3}, respectively. Next, the fine-tuned LM is used as encoder when training a classifier. The model is

[4] `https://github.com/tinaolivia/cnn_al`

gradually unfreezed and fine-tuned with one cycle of one epoch until the three last layers are fine-tuned. The maximum learning rates are set to $2 \cdot 10^{-2}, 10^{-2}, 5 \cdot 10^{-3}$, respectively. The the entire model is unfreezed and fine-tuned with one cycle of 100 epochs and early stopping, and a maximum learning rate of 10^{-3}. The implementation is made publicly available[5].

[5] https://github.com/tinaolivia/lstm_al

3.2 Data

The experiments are performed on two datasets, the IMDB movie reveiw dataset (Maas et al., 2011) for sentiment analysis, and the AG News corpus[6] (Zhang et al., 2015) for topic classification. The datasets are divided into train, validation, and test sets, where the test set represents the *pool* of unlabelled data. The train and test sets are updated by active learning, while the validation set is constant and used for model validation.

- **IMDB Movie Review data:** a dataset containing movie reviews along with the sentiment expressed by the writer. The labels are negative (0) and positive (1). The dataset contains in total 50'000 movie reviews, whereas 25'000 are negative, and 25'000 positive. There are no movie rated more than 30 times in the dataset, since reviews towards the same movie tends to have correlated ratings. Initial train, validation, and test sets are created by randomly sampling 90% of the data to be the test set, and from the remaining 10%, 4% is randomly sampled as the train set. The initial partition is thus 192 examples in the train set, 4842 examples in the validation set, and 44'966 examples in the test set.
- **AG News corpus:** a dataset consisting of web news articles, in total of 496'835 articles from more than 2000 sources divided into the four main classes. The categories are world, sport, business and sci/tech. A subset of 127'600 documents are used in this exploration, whereas 90% are randomly assigned to the test set, and 1.6% of the remaining 10% are randomly sampled as the train set. The initial split is 196 examples in the train set, 12'667 in the validation set, and 114'737 in the test set.

[6] `http://www.di.unipi.it/ gulli/AG_corpus_of_news_articles.html`

3.3 Experimental Setup

This thesis involves several experiments investigating different active learning query strategies in a pool-based setting. The experiments look into the effect of adding randomness to the active learning selection, clustering of the unlabelled pool of data, in addition to exploring a Bayesian approximation of model uncertainty in deep models, and how transfer learning and active learning interact. The experiments are divided into four parts,

(1) Active learning query strategies,

(2) Exploring the data space by adding randomness and clustering of $\mathcal{U}$,

(3) A Bayesian approach,

(4) Transfer learning and active learning combined.

Two models are taking part in the experiments, a simple CNN mainly used for the three first parts, and an AWD LSTM with pre-training for investigation of transfer learning. The models have been described in Section 3.1. There are four query strategies tested, least confidence sampling, entropy uncertainty sampling, margin uncertainty sampling, and prediction variability when applying dropout, all described in Section 2.3.2 under *Uncertainty Sampling*. Random sampling will be used as baseline. These will from now be referred to as variation ratio, entropy, margin, variability, and random, respectively. Note that variability only is relevant in part (3), since is relies on a Bayesian approximation, and can not be computed deterministically. The experiments are performed using two NLP datasets, the sentiment analysis IMDB movie review set, and the multi-class text categorization AG News corpus, see Section 3.2. These will be referred to as IMDB and AG in the following.

3.3.1 Active Learning Query Strategies

In the introductory experiments, the three active learning query strategies variation ratio, entropy, and margin, are investigated and compared against random sampling. The CNN is first trained on the initial labelled sets, which contains 192 ans 196 samples for IMDB and AG, respectively. Then for each active learning loop, $n = 1, 10, 100$ instances are selected from $\mathcal{U}$ and added to $\mathcal{L}$, before the model is re-trained, until 1000 samples are added in total. Some experiments might have more or less than 1000 examples annotated in total, and when that being the case, it will be stated. There are four methods explored in this part,

- **Random: (baseline)** n instances are sampled at random from $\mathcal{U}$, annotated and added to $\mathcal{L}$. The model is trained on the updated labelled set, before n new instances are sampled from $\mathcal{U}$. The process continues until 1000 instances have been added to $\mathcal{L}$.
- **Variation ratio:** With the initially trained model, n instances are selected from $\mathcal{U}$ according to Equation (2.4), annotated, and added to $\mathcal{L}$. The model is re-trained with the updated labelled set, before it selects n new instances from $\mathcal{U}$ in the same manner. The process continues until a total of 1000 examples have been added to $\mathcal{L}$.

- **Entropy:** With the initially trained model, n instances are selected from $\mathcal{U}$ according to Equation (2.6), annotated, and added to $\mathcal{L}$. The model is re-trained on the updated labelled set, before n new instances are selected. The process continues until a total of 1000 examples have been added to $\mathcal{L}$.
- **Margin:** With the initially trained model, n instances are selected according to Equation (2.5), annotated, and added to $\mathcal{L}$. With the updated labelled set, the model is re-trained, before selecting n new instances in the same manner. The process continues until 1000 instances have been added to $\mathcal{L}$ in total.

The four methods can all be performed with both datasets, IMDB and AG, and with three distinct values of n, namely $n = 1, 10, 100$, thus in total 24 experiments. However, all experiments will first be performed on the binary classification IMDB data, then the methods yielding the best results are performed with the AG data, to see if the methods still work in a multi-class classification setting and with another dataset. Thus, for this part, 12 experiments are performed to begin with. In addition to comparing the methods against each other, for each method, the results for different n are also investigated. The idea is to see the effect of n in active learning, since there is a concern that adding more than one example will waste the labelling budget. Accounting for this concern is discussed in the next part, Section 3.3.2.

3.3.2 Exploring the Data Space

A challenge with active learning when querying several instances at once is that there's a considerable chance of querying many similar examples. The informativeness of examples are computed w.r.t. the knowledge the model retain at the time, and two similar examples will have similar informativeness. However, if one of these are labelled, then at the next active selection, the other is no longer considered informative, since the model has gained new knowledge from the first example. In a setting where several examples are labelled at once, both these examples would have been selected, wasting the labelling budget on instances that are non-informative later in the process. In addition, when oversampling in some areas, a lot of the data space is consequently ignored, and it could lead to an undesired bias in the model. To avoid this, two methods aiming to explore larger parts of the data space are described in the following.

Adding Randomness

The first and simplest approach is to add some randomness to the active selection. With probability p, each of the n most informative examples is replaced by a randomly sampled instance from $\mathcal{U}$. Thus, some instances are queried based on their informativeness, and some are sampled at random. Note that the informative examples in the final sample not necessarily are the *most* informative examples in $\mathcal{U}$. Hopefully, the randomness will add some information about a larger part of the space when sampling more than one instance at the time.

The experiment is performed for various probabilities, $p = 0.05, 0.1, 0.25, 0.5$, as well as baselines $p = 0$ (no randomness) and $p = 1$ (random selection) for comparison. To start

with the approach is tested with entropy with CNN on IMDB, for $n = 10$ and $n = 100$, then more experiments are conducted if it seems feasible. Further, in both cases, $n = 10$ and $n = 100$, 100 active learning loops are performed. Thus, for $n = 100$, a total of 10'000 examples are annotated and added to $\mathcal{L}$ to see the long time effect for $n = 100$.

Clustering

The second approach is to cluster the unlabelled pool of data $\mathcal{U}$, and make sure that there's no oversampling within the clusters. In cases where n examples are to be queried, the data is clustered into n clusters, and only one example is queried from each cluster. That one example is the one with the highest informativeness within its cluster. The aim is that no parts of the space is oversampled, but at the same time, considering the informativeness. It's expected that this approach will do fairly better than adding randomness to the selection, since the exploration of the space is more targeted, and even in parts with low informativeness, the most informative examples are queried.

To cluster the unlabelled data, the raw text documents are first vectorized by tf-idf with English stop words removed, then clustered by K-means clustering. Notice that the clustering is independent of both the model and query strategy, unlike Zhdanov (2019), which created a K-means algorithms only independent of the model, see Section 2.3.3. Consequently, the method might behave differently in different situation. Each cluster is examined to find the most informative document, and these are added to the labelled training set $\mathcal{L}$. Since Zhdanov (2019) incorporated informativeness into the K-means objective function, he can query examples closest to the cluster means, ensuring highly informative examples and high diversity in the sample. This approach yields more diversity than simply querying the most informative example within a cluster, however, implementing an efficient K-means algorithm is out of the scope of this thesis.

The procedure is conducted with CNN on both datasets, for varition ratio, entropy, and margin, while random is used as a baseline. Further, the approach is tested for $n = 10$ and $n = 100$, and both random sampling and $n = 1$ are used as baselines. Nedless to say, clustering is not performed for $n = 1$, as it would be exactly the same as not clustering.

3.3.3 A Bayesian Approach

Since deep models often fail to express model uncertainty (Gal and Ghahramani, 2016), the next experiments focus on computing a Bayesian approximation, and compare to deterministic active learning, as seen so far. The experiments are conducted for all three methods, variation ratio, entropy, and margin, where the acquisition functions stay the same, namely Equation (2.4), (2.6), and (2.5). Nevertheless, the predictive probabilities are computed according to Equation (2.7), rather than directly employing the softmaxed model output. The number of forward passes performed to compute a Bayesian approximation is set to $T = 10$. In addition, a fourth method, which will be referred to as variability, will be investigated. Variability selects informative instances in accordance with Equation (2.8), and has not been considered so far, for the reason that it depends on a

Bayesian approximation.

All methods are tested for $n = 1, 10, 100$, and compared to the equivalent deterministic approaches, except for variability, which lacks one. Random is also added as baseline. In addition, the Bayesian approach is combined with clustering, and compared to deterministic methods with clustering when $n > 1$. Thus, there are five methods compared in the final plots when $n > 1$, and three when $n = 1$, since clustering has no effect in this case. For variability there are only three and two, respectively, since there are no deterministic counterparts. The non-Bayesian and Bayesian approaches to variation ratio, entropy, and margin, will be referred to as deterministic and Bayesian, respectively, and it will be clear which method is discussed. Variability, on the other hand, will only be known as variability.

3.3.4 Transfer Learning in Active Learning

In this section, a new model i considered, namely the AWD LSTM presented in Section 3.1.2. The model is an augmentation of a language model, pre-trained on the WikiText-103 corpus, and fine-tuned for the data and task at hand. The same datasets, IMDB and AG, are used. The aim is to see how transfer learning and active learning works in combination, and if it may speed up active learning. Since the AWD LSTM is considerably more complex than the CNN in the preceding experiments, training will be longer, and training time can thus not be compared for the two models. An idea is to rather compare training time for AWD LSTM with and without pre-training, however, training an efficient language model from scratch on the amount of data in the initial training set $\mathcal{L}$ would be impossible. In addition, training the AWD LSTM is too time consuming to repeat the experiments without pre-training as well.

In general, the idea is to perform the same experiments as in part (1), (2), and (3). However, after seeing the results with the CNN in the foregoing experiments, and considering the computational cost related to training the AWD LSTM, some experiments are eliminated. That is, not all of the preceding experiments are repeated. The initial set up is variation ratio, entropy, margin, variability, and random as baseline, with parameter $n = 1, 10, 100$, and on datasets IMDB and AG. First, the best methods and parameter values are chosen based on part (1), (2), and (3). These experiments are then performed in a binary classification setting on IMDB, and the most interesting experiments are also tested for multi-class classification on AG.

Chapter 4

Analysis

In the following, the results are reported and discussed in accordance to the four parts described in Section 3.3. The results are discussed while they are presented in Section 4.1, then a summary of the principal findings and further discussion is provided in Section 4.2.

4.1 Results

The results are presented in plots, reporting on the validation accuracy and corresponding loss, both plotted against the size of the training set $\mathcal{L}$. In order to dispose of noise in the results, and increase visibility of the effects of the methods, an average over up to ten identical experiments are reported when possible. The results are further smoothed by computing a moving average with window w, where w depends on the variations in the results. The window w is reported together with the number of instances n added at each active learning round.

In the first three experimental parts, Section 4.1.1, 4.1.2, and 4.1.3, all experiments are performed with the CNN model, while the experiments in Section 4.1.4 are carried out with the AWD LSTM. See Section 3.1 for a detailed description of the models, Section 3.2 for a presentation of the datasets, and Section 3.3 for an introduction to the experimental setup.

4.1.1 Active Learning Query Strategies

This section reports on the findings regarding experiments with the *vanilla* deterministic active learning query strategies, which are described in Section 3.3.1.

Figure 4.1 shows how the active learning query strategies compare to random sampling when a single instance is added to $\mathcal{L}$ at each round, i.e. $n = 1$. The plotted results are an average over three identical runs of the experiments, in addition, a moving average

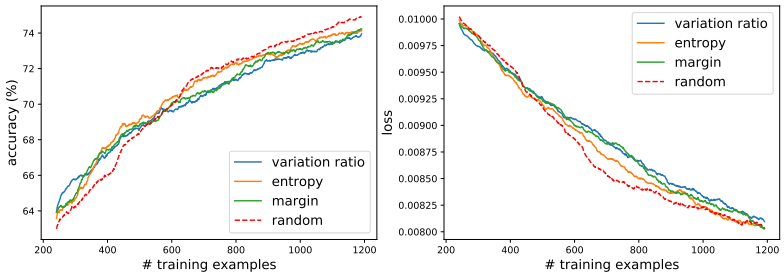


Figure 4.1: Deterministic variation ratio, entropy, and margin are compared to random sampling on the IMDB dataset. $n = 1$, $w = 50$.

is computed to lower the noise even more. The moving average window is set to $w = 50$. The active learning strategies give fairly similar results. All are doing better than random sampling in the beginning, up to $\mathcal{L}$ contains approximately 600 instances, then random sampling does better towards the end. This makes sense, as active learning often gives most effect in the beginning, since informative examples are more valuable when the training set is small. Later, on the other hand, random sampling has the advantage of exploring more of the data space, while the AL query strategies keep sampling in border areas, or outliers. Nonetheless, it seems like all three active learning methods stay quite stable, with entropy superior when the size of $\mathcal{L}$ is between 600 and 900. Variation ratio and margin have similar results throughout. One would expect to see slightly better results for AL compared to random, however, the model is a neural network, and there might be an improvement when looking into Bayesian approximations, which is done later, in Section 4.1.3.

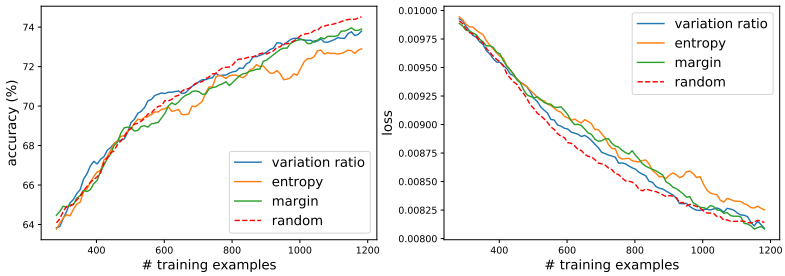


Figure 4.2: Deterministic variation ratio, entropy, and margin are compared to random on the IMDB dataset. $n = 10$, $w = 10$.

Next, n is set to 10, and the same experiments are conducted, with results presented in Figure 4.2. Three consecutive runs were performed and averaged, and a moving average

with window $w = 10$ computed. To begin with, all four methods perform similarly, while the results become more variable later on for the AL methods. There's no apparent advantage to active learning in the first rounds, as was seen for $n = 1$, supporting the preconception regarding adding a larger sample to $\mathcal{L}$ without facilitating a way of ensuring diversity in the sample. Random selection, on the other hand, will have more variety in each sample, hence a more stable progress.

The same effects are evident also for $n = 100$ in a larger scale, see Figure 4.3. Here, an average is taken over ten identical runs, while a moving average is computed with $w = 2$. Variation ratio, entropy, and margin behave very similar in this situation, which makes sense, since the three methods *should* be similar in a binary classification setting. However, random initialization of weights, and machine accuracy can lead to small differences when the informativeness is computed for different methods, and after one sample is added to $\mathcal{L}$, which are different for the three methods, the models will have different information, and act differently from there. This difference is less evident in this case for several reasons. First of all, when 100 examples are annotated in contrast to 1 or 10, machine accuracy does not affect the sample as much. Second, when averaging over ten experiments, the effects become less apparent. When $n = 100$, an active selection will oversample certain areas of the data space, primarily border areas, in a larger scale than for $n = 10$. The samples are thus not as useful as a random selection, since random adds more diversity, yielding a sample containing both uncertain and certain examples.

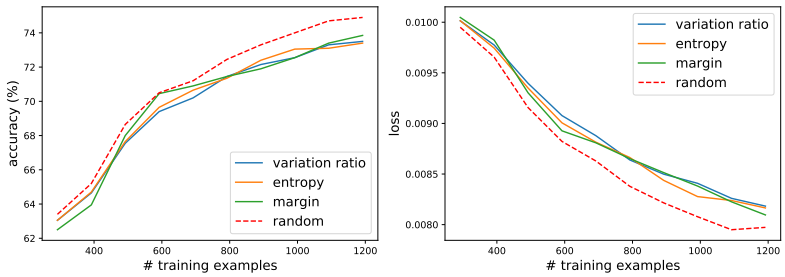


Figure 4.3: Deterministic variation ratio, entropy, and margin are compared to random on the IMDB dataset. $n = 100$, $w = 2$.

However, an interesting observation when running the same experiment for a longer period of time, is that the random validation loss starts increasing after passing approximately 2000 examples in the training set, see Figure 4.4. Here 100 examples are queried for 100 rounds, adding a total of 10'000 examples to $\mathcal{L}$. The results presented are an average over ten identical experiments, and a moving average is computed with $w = 10$. Still, while the loss increases, insinuating that the model is overfitted, surprisingly the accuracy stays quite high. The reported loss is cross entropy, which is computed according to Equation (2.2), and is large when many predictive probabilities are close to 0.5. Hence, the model might

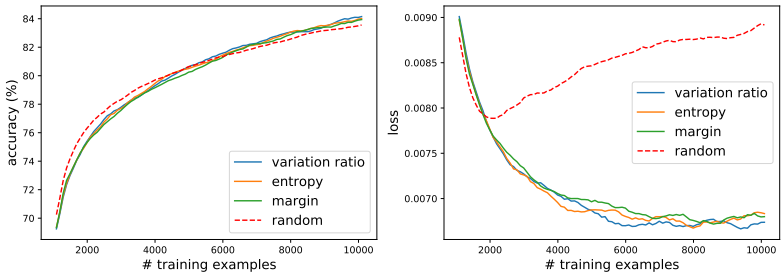
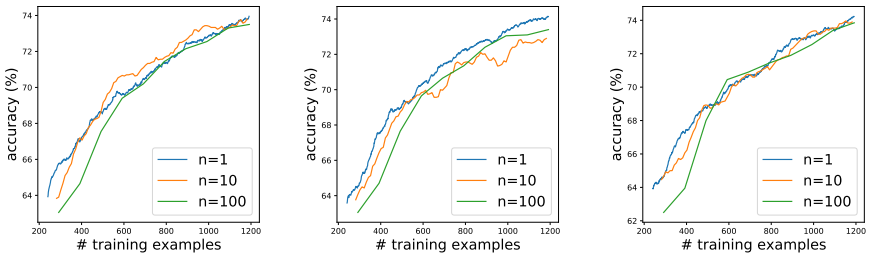


Figure 4.4: Deterministic variation ratio, entropy, and margin are compared to random on the IMDB dataset. 100 active learning rounds with $n = 100$, $w = 10$.

classify correctly, but with class probabilities close to 0.5. In one way, this would say that in the long run, when selecting the most informative examples, the model will have clearer class boundaries, which somewhat is the goal of active learning. Another unexpected effect is that active learning performs worse than random sampling at first, then better around half way through the experiment. One would on the contrary anticipate active learning to have the most effect in the first loops. Here it seems like active learning works well when random sampling doesn't have *more to add*. This is most likely dependent on the data, but could be an advantage, and motivation for exploring active learning with a *change of strategy* through the process. By starting out with a method performing well in early loops, then *changing strategy* to a method more effective in later loops, the overall performance could improve.



(a) Variation Ratio **(b)** Entropy **(c)** Margin

Figure 4.5: Comparison of $n = 1, 10, 100$ for variation ration (left), entropy (middle), and margin (right).

Now, the effect of n on the active learning validation accuracy is inspected by comparing the different n-values. Figure 4.5 displays the validation accuracy for $n = 1, 10, 100$ for variation ratio (4.5a), entropy (4.5b), and margin (4.5c). The moving average windows are set as in the previous experiments, that is, $w = 50$ for $n = 1$, $w = 10$ for $n = 10$, and

$w = 2$ for $n = 100$. Evidently, for all three methods, $n = 1$ does best in the beginning, and for entropy sampling, remains the best throughout. Moreover, also supporting the foregoing discussion, $n = 100$ starts out the worst, but becomes better as the labelled set grows. Hence, n as less to say when $\mathcal{L}$ increases. This also gives weight to the idea of strategy change, both to obtain good results, and to lower computational cost. All three values of n will be explored further in the other parts of the experiments.

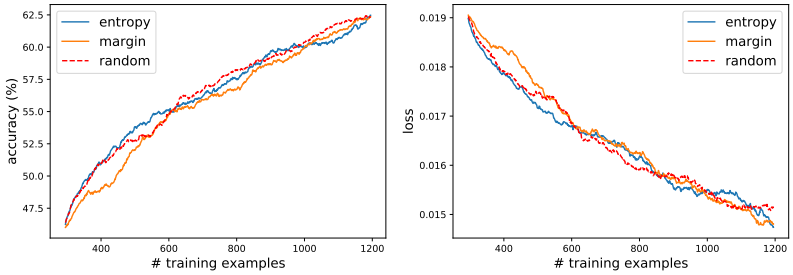


Figure 4.6: Deterministic entropy and margin compared to random on AG. $n = 1$, $w = 100$.

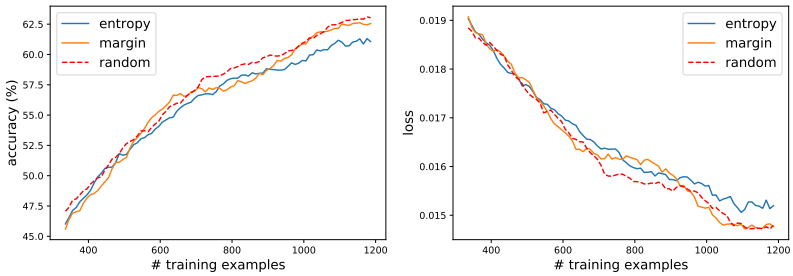


Figure 4.7: Deterministic entropy, and margin are compared to random sampling on AG. $n = 10$, $w = 15$.

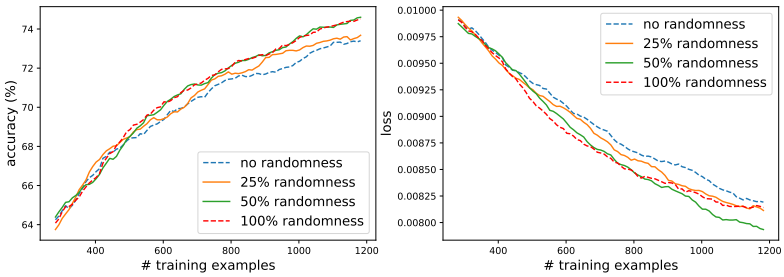
Lastly, the methods are also explored in a multi-class setting with another dataset, namely the AG news corpus. Since variation ratio and margin have produced similar results so far, only entropy and margin are presented here. The vanilla deterministic methods in a binary setting did not give groundbreaking results, and this is also reflected with the multi-class AG dataset. Figure 4.6 exhibits the validation accuracy and loss for $n = 1$, where a moving average of one single experiment is computed with $w = 100$. The methods are fairly similar, as also seen with IMDB, however, here there is no benefit for active learning in the first loops. Margin even performs worse than random sampling for the most part. The same is noticeable for $n = 10$, see Figure 4.7, where the methods also yield similar accuracy, nevertheless, in this case, entropy seems to perform the worst. The results for

$n = 10$ is an average over three identical experiments, as well as a moving average with $w = 15$. $n = 100$ is not included, since random selection was overall superior to the other methods for $n = 100$ on IMDB. Additionally, since both $n = 1$ and $n = 10$ showed little effect on AG, there is not reason to believe that $n = 100$ will be any better.

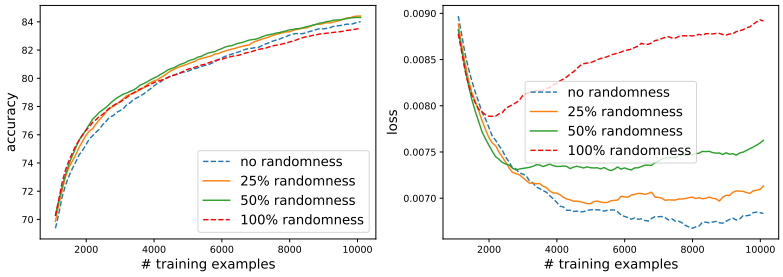
4.1.2 Exploring the Data Space

Next, two approaches for exploring more of the data space are examined, as an attempt to avoid oversampling in uncertain areas. The approaches are presented in Section 3.3.2.

Adding Randomness



(a) $n = 10$



(b) $n = 100$

Figure 4.8: Adding randomness to the active selection. Entropy uncertainty sampling on IMDB with $n = 10$ (top), $n = 100$ (bottom). $w = 10$ in both cases.

The experiments in this section have only been performed with one method, namely entropy, with $n = 10$ and $n = 100$. Different values for p have been investigated, $p = 0.05, 0.1, 0.25, 0.5$, in addition to baselines $p = 0$ (no randomness) and $p = 1$ (random selection). These experiments are not performed for $n = 1$, since then 0 or 1 examples would be replaced at each round, while seeing the effect when replacing a fraction

of each actively selected sample is more of interest. After conducting experiments for all p, it was observed that $p = 0.05$ and $p = 0.1$ gave fairly similar results to $p = 0$, and were thus omitted from the final plots.

Figure 4.8 exhibits the validation accuracy and loss for $n = 10$ (4.8a), and $n = 100$ (4.8b). In both cases, the final results are averaged over ten runs, and computed as a moving average with $w = 10$. A fascinating observation is that $p = 0.5$ gives similar results to random sampling short-term, but better results in the long run, and additionally it outperforms entropy with no randomness. Still, it's interesting to see that the loss for $n = 100$ is less for no randomness, while the accuracy is better for $p = 0.25$ and $p = 0.5$. Note also that the loss for higher levels of randomness has the same tendencies as completely random selection in the long run.

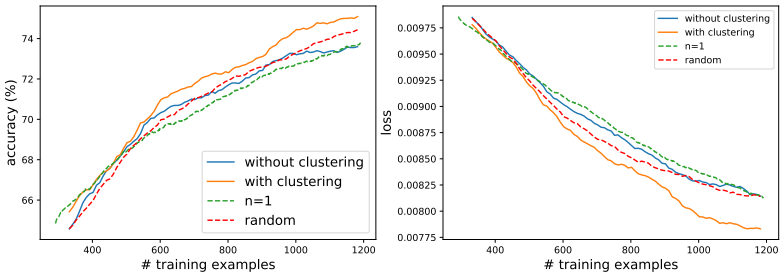
From these findings, it seems like some randomness can be an advantage, however, short-term it is not better than completely random sampling. A potential explanation is that when $p = 0.5$, half of the sample will be around class border, while the other half will be random, creating a *trade-off*, in a sense, between uncertain and certain examples. This method is not taken further in other experiments, even though it would be interesting to see the interaction between adding randomness and some of the other methods. Furthermore, this approach could be interesting to include in a change-of-strategy setup.

Clustering

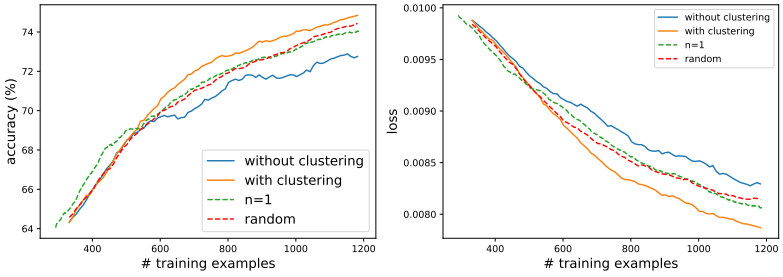
The next approach is more *targeted*, in a sense, to dealing with the problems arising when $n > 1$ in an active query. By rather clustering $\mathcal{U}$, and only choosing the most informative instance within each cluster, it's guaranteed that no areas in the data space is oversampled, while the informativeness of the whole sample is considered as well.

For $n = 10$, the pool of unlabelled data $\mathcal{U}$ is clustered into 10 clusters, before the most informative example in each cluster is labelled and added to $\mathcal{L}$. The results for not clustering and clustering, along with random and $n = 1$ as baselines, are displayed in Figure 4.9. For the active learning methods with and without clustering, the presented results are an average over three identical experiments, and a moving average is taken over a window $w = 15$. For random sampling, the average is computed over ten experiments, and the moving average window is as for the AL methods, namely $w = 15$. Lastly, for $n = 1$, an average is computed over three consecutive experiments, and a moving average is computed with $w = 100$. For all three methods, variation ratio (4.9a), entropy (4.9b), and margin (4.9c), the improvement from not clustering to clustering is obvious. Furthermore, clustering outperforms both baselines in all three cases. Apparently, the more targeted approach to obtaining diversity also ensures more valuable labelled data, at least short-term. An unexpected observation is that for variation ratio, see Figure 4.9a, $n = 10$ without clustering outperforms $n = 1$. This could be linked to the computation of the moving average. Until now, the moving averages of results in the same figures have been computed with the same window size w, so that the results *relative* to other methods are unaffected. An exception is Figure 4.5, which compares different values of n. However, when differ-

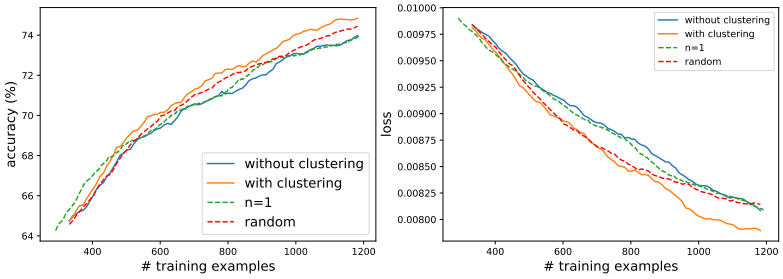
ent values of n are compared, the window is larger for smaller n, as results are reported more frequently, and are therefore noisier. If there are many *dips* in the results, this will influence the moving average negatively.



(a) Variation Ratio

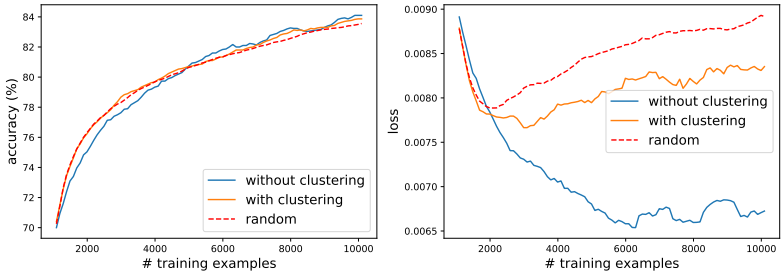


(b) Entropy

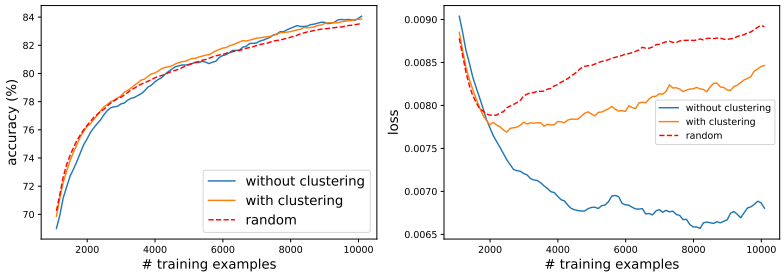


(c) Margin

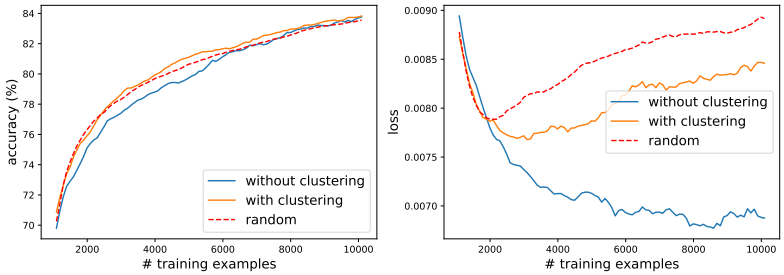
Figure 4.9: Deterministic variation ratio (top), entropy (middle), margin (bottom) in IMDB with and without clustering. $n = 10$, $w = 15$. Random and $n = 1$ are added as baselines.



(a) Variation Ratio



(b) Entropy



(c) Margin

Figure 4.10: Deterministic variation ratio (top), entropy (middle), margin (bottom) in IMDB with and without clustering. $n = 100$, $w = 10$. Random is added as baseline.

There are also improvements for $n = 100$, see Figure 4.10. In this case, 100 active learning loops are performed, and a moving average with window $w = 10$ is computed to get smoother results and better visibility. Random sampling is added as baseline, how-

ever, $n = 1$ is omitted in these plots, due to the computation time of 10'000 rounds with $n = 1$. In all three cases clustering improves from not clustering, although not compared to random sampling before late in the process. Random sampling and clustering yield quite close results, which could be explained by the size of n. When $n = 100$, there are many clusters, whereas some contains barely any informativeness. Combined, the actively selected samples do not hold considerably more information than the randomly selected samples. It's also interesting to see how the loss explodes in a similar manner to random.

After inspecting the results from clustering, it seems like entropy with $n = 10$ yields the best results on IMDB, and is also tested in a multi-class setting on AG. The validation accuracy and loss are presented in Figure 4.11. The experiments were performed three times, and the average results computed, in addition to taking the moving average with window $w = 15$. Unlike for IMDB, there's no evident advancement from not clustering to clustering, and there's only a short interval where clustering performs better than not clustering and random. The effect of active learning is dependent on the data and model, and in this case, this specific CNN with entropy uncertainty sampling might not be the best choice. It's also important to be aware that the model is fairly simple, and developed for *sentence* classification. In the original paper, Kim (2014) tests the model on, among others, a single sentence movie review dataset. Even though many of the movie reviews in IMDB are longer than one sentence, they are still considerably shorter than news articles, which may explain the different results with AG.

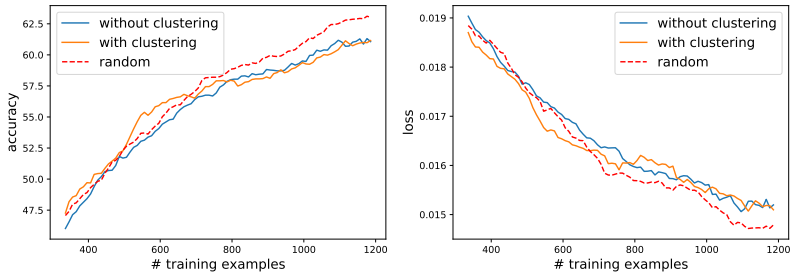


Figure 4.11: Deterministic entropy with and without clustering on AG. $n = 10$, $w = 15$. Random selection is added as baseline.

4.1.3 A Bayesian Approach

Next, a Bayesian approach to approximating model uncertainty is looked into. The experimental setup is presented in Section 3.3.3.

For all methods and $n = 1$, there's an improvement from deterministic to Bayesian, see

Figure 4.12, 4.13, and 4.14, for variation ratio, entropy, and margin, respectively. For variation ratio, Bayesian does not perform better than deterministic until approximately 400 examples are labelled, and $\mathcal{L}$ contains 600 instances. Unfortunately, the approach does not improve to random sampling either, besides in the first few iterations.

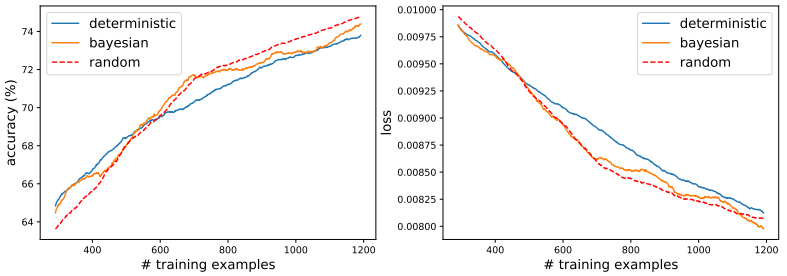


Figure 4.12: Deterministic and Bayesian variation ratio on IMDB. $n = 1$, $w = 100$. Random is added as baseline.

For entropy, however, deterministic and Bayesian behave almost the same, even though the same tendencies as for variation ratio are present when the size of $\mathcal{L}$ grows beyond 600. See Figure 4.13. However, Bayesian is overall better than deterministic. In addition, compared to variation ratio, both deterministic and Bayesian entropy achieves better results compared to random. In the last 200 loops, the accuracy of Bayesian and random coincides, but note that the loss is less for Bayesian entropy than random.

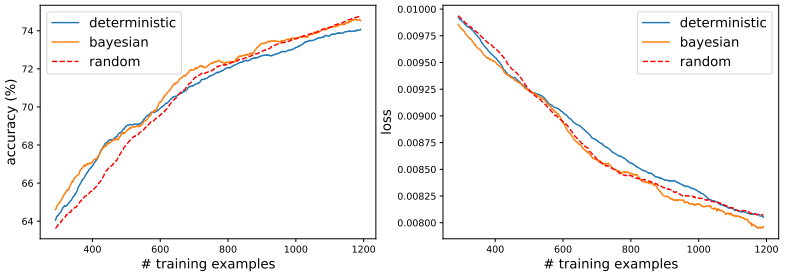


Figure 4.13: Deterministic and Bayesian entropy on IMDB. $n = 1$, $w = 100$. Random is added as baseline.

Maybe most interesting are the results for margin, see Figure 4.14. Bayesian has a clear and stable enhancement to deterministic, both in the sense of accuracy and loss, in addition to performing best compared to random. Also here does random catch up with Bayesian in terms of accuracy, while the loss for Bayesian stays lower.

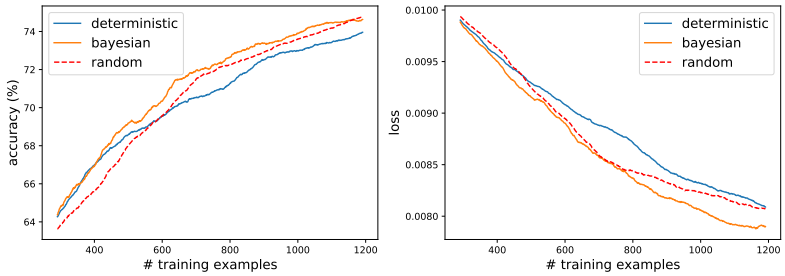


Figure 4.14: Deterministic and Bayesian margin on IMDB. $n = 1$, $w = 100$. Random is added as baseline.

The last results to be presented for $n = 1$ are for variability compared to random selection, and are shown in Figure 4.15. As expected in active learning, the active approach has largest impact in the first loops. Nonetheless, after a while, the progress slows down, again after passing approximately 600 instances in the labelled set. The variability method does not perform any better than any of the deterministic methods, while one would expect it to perform similarly to the Bayesian approaches, seeing as it includes a Bayesian approximation to the model uncertainty. However, this approach does not consider the *values* of the predictive probabilities or the *values* of the Bayesian approximations, only the distance between them, which could explain the poor results.

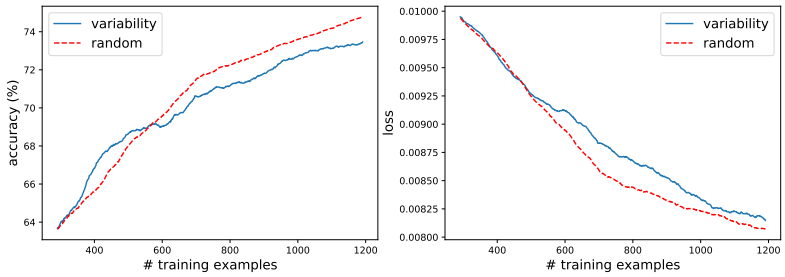
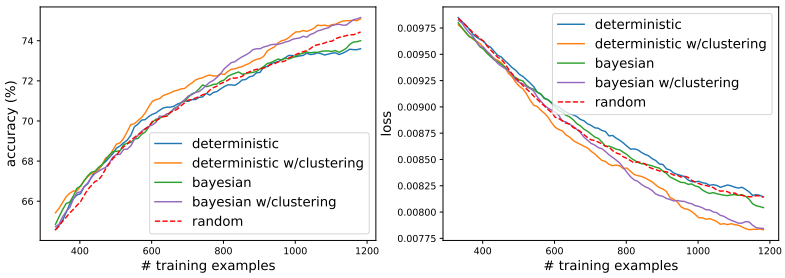


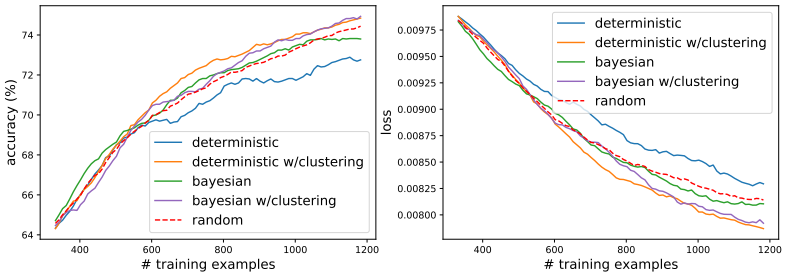
Figure 4.15: Variability on IMDB. $n = 1$, $w = 100$. Random is added as baseline.

When $n = 10$, the Bayesian approaches will, in addition to being compared to the deterministic approaches, also be combined with clustering, and compared to deterministic clustering. Variability will only be compared to variability with clustering.

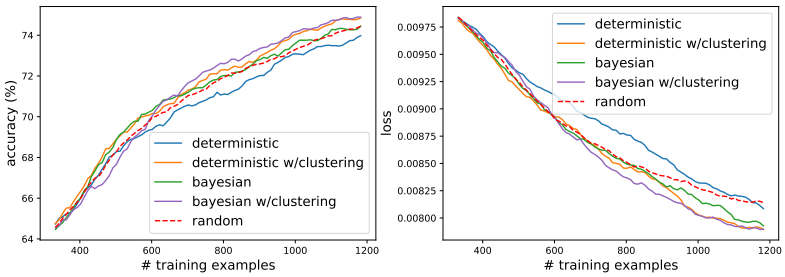
The outcomes for variation ratio, entropy, and margin are presented in Figure 4.16, and there are some of the same tendencies present for all three. For instance, the improvement



(a) Variation Ratio



(b) Entropy



(c) Margin

Figure 4.16: Deterministic and Bayesian, with and without clustering, on IMDB. $n = 10$, $w = 15$. Random is added as baseline.

from deterministic to clustered deterministic is notable for all. While there is little difference between deterministic and Bayesian for variation ratio, see Figure 4.16a, there is a clear boost for Bayesian when querying examples according to entropy and margin, see Figure 4.16b and 4.16c, respectively. However, even though both clustering and Bayesian improves results, the combination does not improve the results further for any of the three methods. A possible explanation is that diversity in the sample is more *important*, or gives more *effect*, than computing more accurate uncertainty. In addition, when only choosing one example within a cluster, the difference between deterministic and Bayesian uncertainty might be less in some clusters, leading to less differences among deterministic and Bayesian examples in the actively selected samples.

For variability and $n = 10$, the active learning advantage that was evident for $n = 1$ in the first loops is absent, as seen for the other methods as well. Still, there's a significant improvement when clustering the unlabelled data before starting the active selection, but it's still not significantly better than random sampling. This might imply that when clustering and actively choosing examples based on variability, the diversity obtained is also present in the random sample. Moreover, based on these results, variability does not seem like a feasible way of actively querying examples, and is therefore not considered in further experiments. An idea could be to see if adding randomness could boost results even more. Nevertheless, computing a Bayesian approximation is somewhat computationally expensive, as it requires T forward passes, where $T = 10$ in this case, and is thus not pursued further in this thesis.

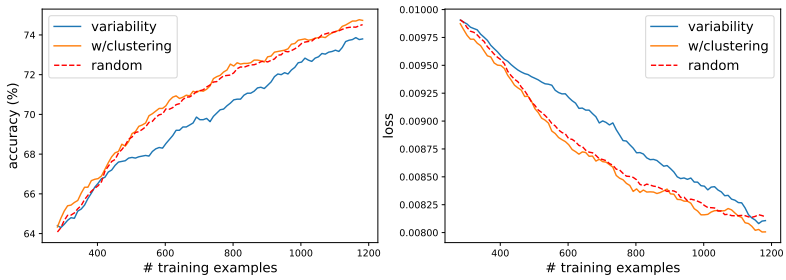


Figure 4.17: Variability with and without clustering on IMDB. $n = 10$, $w = 10$. Random is added as baseline.

None of the discussed methods gave evidently better results than random selection for $n = 100$, thus these results are omitted.

In winding up this part of the experiments, the methods have also been applied to the AG news corpus, for assessment in a multi-class setting. Figure 4.18 shows how deterministic and Bayesian entropy, with and without clustering, compares to random with $n = 10$. As

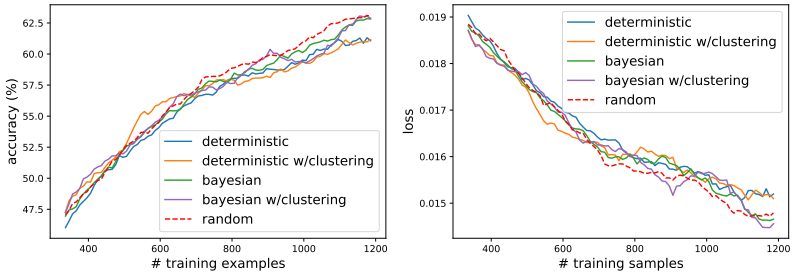


Figure 4.18: Deterministic and Bayesian entropy, with and without clustering, on AG. $n = 10$, $w = 15$. Random is added as baseline.

seen in Section 4.1.2, under *Clustering*, there is little to report, and only one method is tested for that reason. The reason for the poor results in multi-class classification could be due to the size of the validation set, combined with multi-class classification being a more difficult problem, and that the CNN is poorly suited for the AG data. The validation set was constructed to be almost three times as large as for IMDB, and the results might have been better with a smaller validation set, seeing as the model is trained on small amounts of data. However, repeating the 12 experiments already performed on AG would be too time consuming. Still, a smaller validation set could be considered in the remaining part of the experiments concerning transfer learning in Section 4.1.4. Another reason could be, as mentioned in Section 4.1.2, is that active learning is data dependent, and it can be challenging to generalize from one dataset to the next.

4.1.4 Transfer Learning in Active Learning

The last part of the experiments is concerning transfer learning in active learning, introducing a new model, an AWD LSTM, and ULMFiT, see Section 3.3.4. Most of the previously conducted experiments are repeated with the new model, but all experiments regarding $n = 100$ are eliminated, in addition to Bayesian with clustering, due to computational cost. After seeing the foregoing results, the methods tested in this section will be entropy and margin, with parameters $n = 1$ and $n = 10$. Instead of doing the experiments step by step, like for the CNN, where the plain AL methods were presented first, then clustering, and finally the Bayesian approach, all adaptations are directly compared, common to in the previous section, Section 4.1.3. In the same manner as in part (1), (2), and (3), binary classification on IMDB will be considered first, then some experiments are also conducted for multi-class classification on AG.

Starting with $n = 1$, it's important to point out that the process of training the LSTM 1000 times is extremely tedious. Since the label set is small, training the model once is not too time consuming, on average it took approximately 10 minutes with the IMDB data. However, that is 10'000 minutes for 1000 rounds, which is roughly 167 hours, and does

not include the time to actively query examples. Therefore, the experiments were rather run for a certain amount of time. After running for 6 days, random sampling managed 358 rounds, while entropy and margin accomplished 335 and 331, respectively. However, when adding a Bayesian approximation for model uncertainty, which is more computationally expensive, entropy finished 216 rounds, and margin 214. In the succeeding figures, the results are plotted up to the shortest amount of rounds for that experiment. E.g. for entropy, all variations are plotted up to a total of 216 examples are added to $\mathcal{L}$.

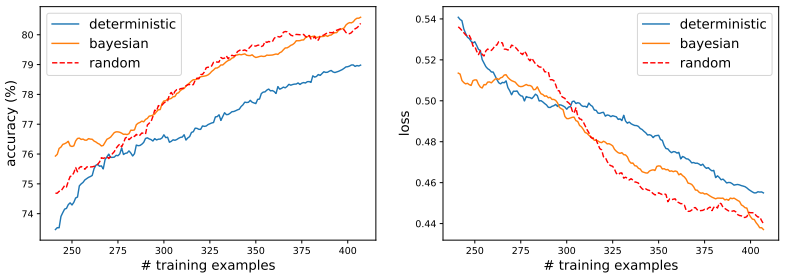


Figure 4.19: Deterministic and Bayesian entropy with AWD LSTM on IMDB. $n = 1$, $w = 50$. Random is added as baseline.

The validation accuracy and loss for deterministic and Bayesian entropy with $n = 1$ are exhibited in Figure 4.19, with random sampling as baseline. Due to computational cost, only one experiment is conducted, and a moving average is computed with $w = 50$. It is expected that the AWD LSTM will perform better than the CNN in general, seeing as it is a more complex model, in addition to being pre-trained. Nonetheless, it's interesting to see that in this interval, up to 400 labelled instances are in the training set, entropy with $n = 1$ and the CNN had better performance compared to random sampling. Here, the deterministic approach performs notably worse than random, while the Bayesian approach is better for only a few active learning rounds.

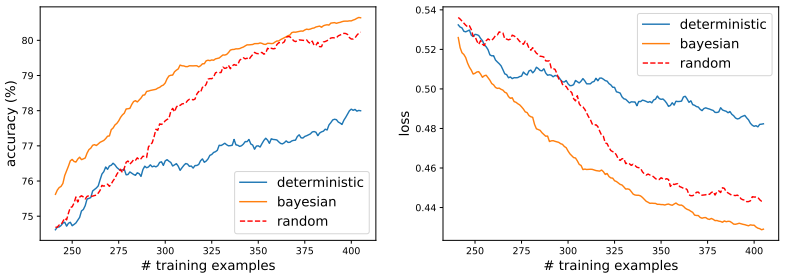


Figure 4.20: Deterministic and Bayesian margin with AWD LSTM on IMDB. $n = 1$, $w = 50$. Random is added as baseline.

Further, the same results for margin are shown in Figure 4.20. Interestingly, deterministic margin performs worse than deterministic entropy, while Bayesian margin is better than Bayesian entropy. Bayesian margin also exhibits better results compared to random, and attains some of the effects seen earlier in the first active learning loops. These results for entropy and margin with $n = 1$ show how active learning can act differently with different models. With more time and resources available, it would be interesting to see the effect in a longer experiment.

The computation time for $n = 10$ is substantially lower than for $n = 1$, since the model is re-trained 100 times, as opposed to 1000. Thus, the standard setup applies for $n = 10$, where 1000 examples are added to $\mathcal{L}$ in total. The final results are averaged over three identical experiments, and a moving average with $w = 10$ is computed. Further, deterministic, with and without clustering, Bayesian, and random as baseline, are all presented together.

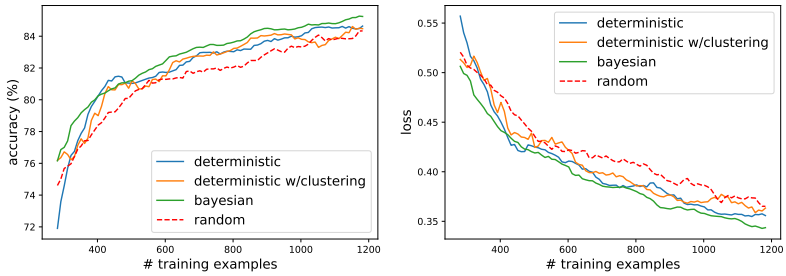


Figure 4.21: Deterministic and Bayesian entropy with AWD LSTM on IMDB. $n = 10$, $w = 10$. Random is added as baseline.

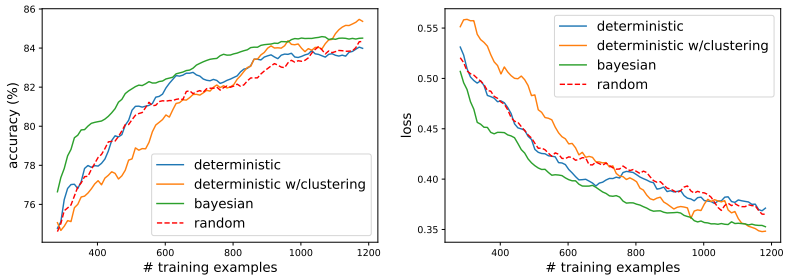


Figure 4.22: Deterministic and Bayesian margin with AWD LSTM on IMDB. $n = 10$, $w = 10$. Random is added as baseline.

For entropy, see Figure 4.21, it's interesting to see how both deterministic and Bayesian

gives better accuracy and loss than random sampling, while none of the deterministic methods did better than random for $n = 10$ with the CNN. Further, clustering does not give the same effect as seen with the CNN, it stays almost the same as deterministic without clustering for the duration of this experiment. The results for margin are fairly similar for deterministic and Bayesian, and are displayed in Figure 4.22. Moreover, clustering is worse than all other methods for half of the experiment, then it shows sign of improving towards the end. With more time, it would be interesting to observe a longer experiment, in order to see if clustering keeps progressing in the same manner. Seeing that the clustering algorithm not only is dependent of model, but also query strategy, here the latter is also evident, which is not seen so far. Again, it's illustrated how different models can respond differently to active learning, and that it's crucial to consider both model and data when deciding on an active learning strategy.

Finally, the AWD LSTM with active learning is also applied to the AG news corpus. An alteration from the CNN experiments is that the validation set is smaller, closer to the size of the IMDB validation set. The initial labelled set stays unchanged, however, the validation set now contains 4908 examples, and the test set 122'496 examples.

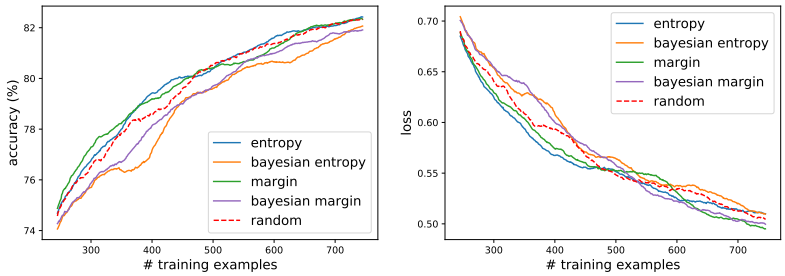


Figure 4.23: Deterministic and Bayesian entropy and margin with AWD LSTM on AG. $n = 1$, $w = 50$. Random is added as baseline.

Like for IMDB, re-training the model 1000 times is tedious, however, training time on AG was significantly lower than on IMDB. The same setup for $n = 1$ was applied, and after 6 days, deterministic managed all 1000 rounds, while the Bayesian approaches accomplished approximately half, 550 rounds. For $n = 1$, one figure displays both the deterministic and Bayesian approaches to both entropy and margin, namely Figure 4.23. Both deterministic methods, and random sampling, are very similar, which differs a great deal from the IMDB results, which displayed very poor results for the deterministic approaches. However, bear in mind that the same experiment on IMDB only completed a couple of hundred AL loops. Yet, it's still clear that up to $\mathcal{L}$ contains approximately 400 examples, the results of the deterministic approaches is better on AG than IMDB. Furthermore, the Bayesian approaches surprisingly perform worse than the deterministic, while for IMDB it showed a considerable improvement. Once more, it becomes transparent that

	Entropy	Margin	Random
IMDB	631.95	632.57	676.23
AG	117.81	117.27	107.86

Table 4.1: Average training time in seconds for AWD LSTM every time $n = 1$ examples are added to $\mathcal{L}$.

active learning is data dependent. Further, notice how the accuracy is comparable to the IMDB results, even though AG has twice as many classes. This was not the case for the CNN, where the accuracy for AG was lower than for IMDB. Nevertheless, the results now might be better due to the validation set being smaller, or it could be that this model is a better fit for AG than the CNN. As discussed in the previous sections, the CNN is constructed for sentence classification, and movie reviews are closer to sentences than what news articles are. Second, the AWD LSTM is pre-trained on the WikiText-103 corpus, which i closer to AG than IMDB. As discussed briefly in Section 2.2.2, pre-trained language models are mainly trained on *formal* text documents, like verified Wikipedia articles as in this case, and need more fine-tuning when encountering less formal text documents, such as e.g. reviews. This is reflected in the training time of the model, see Table 4.1, which reports the average training time when training the model between active selections, for both IMDB and AG. Training to convergence is five to six times as fast with AG, confirming that the model needs far less fine-tuning than for IMDB. Even with a smaller validation set when testing the CNN, it's not expected that the accuracy is as high for AG as IMDB, seeing that AG has more classes. It's rather more intuitive to presume that most of the improvement is because the AWD LSTM is a better fit for the AG news corpus.

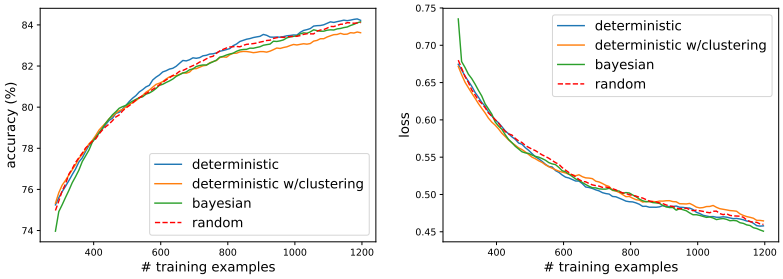


Figure 4.24: Deterministic and Bayesian entropy, with and without clustering, with AWD LSTM on AG. $n = 10$, $w = 10$. Random is added as baseline.

Lastly, for $n = 10$, entropy and margin are presented in distinct plots, where three consecutive experiments are carried out and averaged, and a moving average with $w = 10$ is computed. First looking at entropy, presented in Figure 4.24, deterministic, Bayesian, and clustering give very similar results. Surprisingly, there is even a small deterioration from

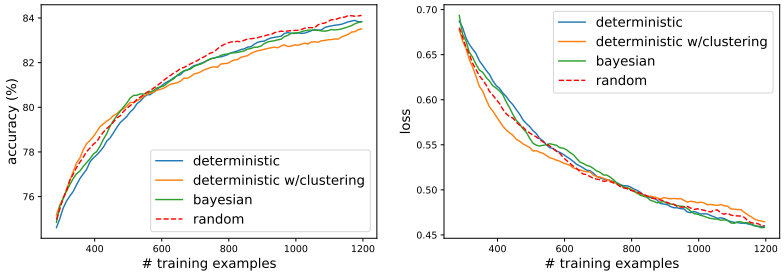


Figure 4.25: Deterministic and Bayesian margin, with and without clustering, with AWD LSTM on AG. $n = 10$, $w = 10$. Random is added as baseline.

deterministic to both Bayesian and clustering. The results for margin, see Figure 4.25 are very similar, there's not a particularly visible change from deterministic to Bayesian, but not either the deterioration seen for entropy. Like for entropy though, clustering has a small negative effect also for margin. Apparently, active learning does not have an effect on the AG news corpus, after observing the same with both the CNN and AWD LSTM. In Section 4.1.3, it was argued that the absence of any effect could be due to the size of the validation set, which was more than 12'000. However, the validation set was adjusted to just below 5000 examples for the experiments with the AWD LSTM, which is roughly the same size as the IMDB validation set. Still, there are no visible effects, raising a question whether all examples in AG attain the same informativeness, at least w.r.t. these two models.

4.2 Discussion

This section contains a summary of the findings from Section 4.1.1, 4.1.2, 4.1.3, and 4.1.4, along with some further discussion. While the first three parts are performed with a CNN, the last is carried out with an AWD LSTM employing ULMFiT and a pre-trained LM. Essentially, the last part introduces a new model and transfer learning, and the most interesting experiments from the preceding parts are performed. All methods were first tested in a binary classification setting on the IMDB movie review dataset, then some were tested in multi-class classification on the AG news corpus. Active learning is highly data dependent, hence, when implementing an AL strategy, it is important to investigate whether it shows promising effects on the data in question. In this exploration, the active learning strategies had negative or no effect on AG, while there were encouraging results for IMDB, thus, the findings discussed in the following are mostly concerning experiments on IMDB.

To sum up some of the main findings, Table 4.2 contains short comments on the discoveries related to the distinct methods on IMDB in this thesis. Take into considerations that these comments reflect on active learning with the two models of this exploration, and are not universal statements. The effect of active learning is dependent on the model and data, which is demonstrated by some of the findings of this thesis.

The first part of the experiments, part (1), which results are presented in Section 4.1.1, looked into three deterministic active learning query strategies without any particular advances, for different values of the parameter n. The methods are variation ratio, entropy, and margin, and n was set to 1, 10, and 100. Active learning with $n = 1$ performed better than random sampling for just below 400 active learning loops, however, for $n = 10$ and $n = 100$, random performed similar to, or better, than the active learning methods for the duration of the experiments. While all methods were best for $n = 1$ in the beginning, the choice of n had less to say in the long run. Nonetheless, none of the methods excelled, and random selection was overall better. This was reflected with both datasets, IMDB and AG. However, in a longer experiment with 100 rounds of $n = 100$, the AL methods gave slightly better results than random when $\mathcal{L}$ contained more than 6000 annotated examples.

The next section, Section 4.1.2, looked into ways of exploring more of the data space with active learning. First, some randomness was added to the active selection, and with 50% randomness in the actively selected sample, entropy performed as good as random short-term, and better in the long run. Still, this approach was only tested for entropy on IMDB, for $n = 10$ and $n = 100$, and was not explored further. Second, the unlabelled pool $\mathcal{U}$ was clustered into n clusters, then only the most informative example within each cluster was queried. This was tested for variation ratio, entropy, and margin, for $n = 10$ and $n = 100$, and clustering definitely improved the accuracy. For $n = 100$ the clustered approach gave almost the same results as random selection, but for $n = 10$, it did better for all three methods. The clustering approach was inspired by Zhdanov (2019), who showed better results compared to random than what has been seen here with $n = 100$. However, his approach includes the informativeness in the K-means algorithm,

Method	Comment
Plain AL	In general no better than random.
Clustering	Improves from not clustering. Better than random for $n = 10$, as good as random for $n = 100$, and better in the long run.
Bayesian	Mostly improves to non Bayesian. Not better than clustering.
Bayesian w/cluster	No notable improvement to deterministic cluster. Computationally expensive.
Transfer Learning	Works well for $n = 10$, better than without. Works poorly for $n = 1$, which is computationally expensive as well.
Transfer Learning and Bayesian	Improvement to deterministic.
Transfer Learning w/clustering	No benefit in these trials.

Table 4.2: Summary of the main findings on IMDB.

	Variation Ratio	Entropy	Margin
Deterministic	46.19	46.37	47.53
Deterministic w/clustering	59.22	55.35	54.46
Bayesian	390.07	406.96	392.32
Bayesian w/clustering	455.64	416.72	402.87

Table 4.3: Average time in seconds for one active learning query for $n = 10$ with CNN on IMDB.

which allows him to query the examples closest to the cluster centres. These examples will have high informativeness, as well as being diverse, as the centres are the points that are furthest apart in the data space. In addition, when clustering independent of model *and* query strategy, the clusters stay more or less the same through the whole process, while in Zhdanov's case, the clusters will change every time the model gains more information. Moreover, Zhdanov (2019) filters the data before clustering, and only considers a pool containing the $\beta k \geq k$ most informative examples, where $k \in \mathbb{N}$ is data dependent, and $\beta \in \mathbb{N}$ is a parameter explored in the paper. Even though expanding the K-means algorithm was out of the scope of this thesis, filtering could have been explored, as it gives a more informative pool to cluster, which changes after each time the model gains new knowledge.

In part (3), the last part featuring the CNN model, the uncertainty measures were computed with a Bayesian approximation to model uncertainty, see section 4.1.3. The same three uncertainty measures were used, with $n = 1$ and $n = 10$, in addition to a new method, called variability. Variability did not perform better than random sampling, only as good when combined with clustering for $n = 10$, and was thus not taken forward in other experiments. For entropy and margin, a Bayesian approach did improve results for both $n = 1$ and $n = 10$, however, for $n = 10$, clustering did better, and combining clustering with the Bayesian approximation did not improve significantly to deterministic clustering. An issue with a Bayesian approach is that it requires T forward passes per instance in $\mathcal{U}$, which significantly influence the computation time of an active selection. In this case $T = 10$, and while clustering a deterministic approach only increases the computation time of one active selection by approximately 30%, clustering a Bayesian approach takes almost ten times as long as deterministic without clustering, see Table 4.3. The table displays the average time in seconds for an active selection for the different methods and variations when $n = 10$. For the CNN on IMDB, clustering with a deterministic approach to model uncertainty would thus be preferred over a Bayesian approach. Nonetheless, for $n = 1$, clustering is not an option, as there is no difference to not clustering. In these cases, a Bayesian approach is clearly beneficial over deterministic for the CNN. The results with Bayesian and $n = 10$ are pretty similar to Gal et al. (2017) when comparing deterministic to Bayesian without clustering. However, Gal et al. (2017) obtain better accuracy when comparing the Bayesian approaches to random, while in this exploration, Bayesian with $n = 10$ is only as good as random for all four methods studied in this section. Moreover, for future work, it could be interesting to see if a Bayesian approximation to uncertainty would have an effect on Zhdanov's clustering approach, seeing as the computation of the uncertainty would affect the clustering.

In the last part, part (4), see Section 4.1.4, the AWD LSTM was introduced with pre-training and ULMFiT (Howard and Ruder, 2018). Entropy and margin were investigated, while variation ratio was not explored in this part. The experiments with $n = 100$ were eliminated, as well as Bayesian with clustering, seeing as it showed little effect and was rather computationally expensive. The experiments performed were thus deterministic and Bayesian with $n = 1$, and deterministic, with and without clustering, as well as Bayesian with $n = 10$. All experiments were performed on both IMDB and AG. Due to the computational cost of retraining the AWD LSTM 1000 times on IMDB, just about 215 rounds were reported for $n = 1$. However, on AG, the model needed less fine-tuning, and could be reported up to 550 AL rounds. Still, there were no considerable results on AG, hence, only IMDB is discussed from here. An unexpected observation was that deterministic with $n = 10$ performed better than deterministic with $n = 1$ compared to random, for both entropy and margin. In fact, deterministic for $n = 10$ performed better than random, which was not seen with the CNN for any method. Moreover, the Bayesian approaches showed improvements to $n = 1$ and $n = 10$ for both methods. Clustering, on the other hand, did not improve results, and for entropy, yielded close to the same accuracy as for deterministic without clustering. For margin, even though clustering did the worst to begin with, towards the end of the experiment it could look like clustering had faster progress than deterministic, Bayesian, and random. Clustering might be an advantage after the training set passes 1200 examples. This could have been investigated further with more time available. In future work, it could also be of interest to apply ADMA (Huang et al., 2018) in a text classification setting, and see if some of the studied advances, such as Bayesian and clustering, can have a positive effect on the results.

A returning observation throughout the experiments has been that one method can be better than another to begin with, then after a while, the roles reverse. For instance, for the plain AL methods and $n = 1$, see Figure 4.1, AL started out better than random, however stagnated before half way through. The opposite was evident in the long run for $n = 100$, see Figure 4.4, as random did best in the beginning, then AL got the upper hand when the number of labelled instances grew past 6000. Maybe even more impressive was when adding $25 - 50\%$ randomness to entropy sampling, see Figure 4.8b. Active learning with some randomness was first as good as random, then better in the long run experiment with $n = 100$. These kinds of observations could motivate a study on active learning with a change-of-strategy. By starting out with a method that works well in early loops, then changing strategy during the active learning process, in order to always obtain the best possible results. As seen in part (1), $n = 1$ is often a good choice to begin with, then a larger n has less to say as the training set grows, see Figure 4.5. In part (4) for margin and $n = 10$, a Bayesian approach was clearly superior until the end of the experiment, however, past that point, deterministic clustering appeared to be beneficial. This could be an interesting aspect in further work, where the benefits of several methods are exploited, as opposed to finding one optimal AL method for the model and data in question.

Chapter 5

Conclusion

Through this thesis, pool-based active learning in text classification has been investigated, with various adaptations, and combinations of these. Motivated by scientific papers exploring active learning combined with clustering in order to sample diverse mini-batches (Zhdanov, 2019), Bayesian approximations to model uncertainty in deep models (Gal et al., 2017), and active learning in combination with transfer learning (Huang et al., 2018), this study involves similar methods independently, and combined. Two models have taken part in the experiments, a CNN for sentence classification (Kim, 2014), and an AWD LSTM with pre-training (Merity et al., 2017). The different methods have first been tested in a binary classification setting on the IMDB movie review dataset, before also seeing the effect with another dataset in a multi-class setting with the AG news corpus. Moreover, the methods have been tested for $n = 1, 10, 100$, which controls the number of examples queried at each active learning loop.

Through all experiments regarding the AG news corpus, active learning had close to no effect on the results, leaving the question of whether all AG examples are of the same informativeness, at least for the two models of this thesis, and uncertainty sampling. It is peculiar to see that two such different models are showing the same effects when applying active learning to the same dataset, nonetheless, it might imply that uncertainty sampling is unsuited for this particular data. It is evident from the results of this thesis, however, that active learning is highly data dependent, which coincides with prior beliefs. In the following, remarks are hence mostly with regards to experiments on IMDB.

First looking at the CNN, plain active learning had little effect, yet, all adaptations improved results. Even though both a Bayesian approach and clustering boosted the accuracy independently, the combination of the two did not make further advancement. Clustering appears as the optimal choice for this model and data, seeing as it gives better results than Bayesian, as well as being far less computationally expensive. Still, in future work, it would be interesting to see the combination of clustering and Bayesian when informativeness is contained in the K-means algorithm, like when Zhdanov (2019) performs clustering

to obtain diverse mini-batches in active learning. In this thesis, the clustering has been independent of the query strategy, such that the clusters stays mainly the same after each query. For Zhdanov (2019), on the other hand, the clusters are affected every time a query is made, as it will change the informativeness of the unlabelled data. Thus, when clustering depends on the query strategy, there might be a larger consequence from a Bayesian approximation, since a Bayesian approach will influence the informativeness, and hence, also the clusters.

When the methods are introduced to a new setting with transfer learning, and a new model with the AWD LSTM, the same conclusions do not apply. Here, a Bayesian approach for both $n = 1$ and $n = 10$ is most beneficial. Clustering even deteriorate results, but leaves an impression of working better later in an active learning process for margin uncertainty sampling. In future studies, this is an objective to research further, as well as ADMA (Huang et al., 2018) in a text classification setting, and maybe together with some of the advances explored in this thesis.

After seeing the same active learning strategies with two different models and datasets, it is definitely clear that the strategy is dependent on these factors. When employing active learning, it is thus very important to reflect upon the task and data, before settling for a model and strategy. An observation through many of the performed experiments is that different methods excel at different points during the active learning process. Hence, there could be a possibility of trying to change method *during* the active learning process, proposing a *change-of-strategy* framework. E.g. for the CNN on IMDB, it could be a good idea to start out with querying one example at the time, maybe in a Bayesian manner, before increasing the size of the queried sample, and adding some randomness to the active selection. With this strategy, there is no need to settle on *one* method, but necessary to explore some of the advantages and disadvantages for several strategies for the task at hand. As further work, it is appealing to explore the effect of strategy change, and if the benefits from several methods can yield a better overall result.

Bibliography

Aizenberg, I. N., Aizenberg, N. N., Vandewalle, J. P., 2000. Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications. Kluwer Academic Publishers, Norwell, MA, USA.

Anand, S., Mahata, D., Aggarwal, K., Mehnaz, L., Shahid, S., Zhang, H., Kumar, Y., Shah, R. R., Uppal, K., 2019. Suggestion Mining from Online Reviews using ULMFiT. CoRR abs/1904.09076.
URL http://arxiv.org/abs/1904.09076

Angluin, D., Apr 1988. Queries and Concept Learning. Machine Learning 2 (4), 319–342.
URL https://doi.org/10.1023/A:1022821128753

Angluin, D., 2001. Queries Revisited. In: Abe, N., Khardon, R., Zeugmann, T. (Eds.), Algorithmic Learning Theory. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 12–31.

Cohn, D., Atlas, L., Ladner, R., May 1994. Improving Generalization with Active Learning. Machine Learning 15 (2), 201–221.
URL https://doi.org/10.1007/BF00993277

Cohn, D. A., 1994. Neural Network Exploration Using Optimal Experiment Design. In: Cowan, J. D., Tesauro, G., Alspector, J. (Eds.), Advances in Neural Information Processing Systems 6. Morgan-Kaufmann, pp. 679–686.
URL http://papers.nips.cc/paper/765-neural-network-exploration-using-optimal-experiment-design.pdf

Czapla, P., Howard, J., Kardas, M., 2018. Universal Language Model Fine-Tuning with Subword Tokenization for Polish. CoRR abs/1810.10222.
URL http://arxiv.org/abs/1810.10222

Dagan, I., Engelson, S. P., 1995. Committee-based Sampling for Training Probabilistic Classifiers. In: Proceedings of the Twelfth International Conference on International Conference on Machine Learning. ICML'95. Morgan Kaufmann Publishers Inc., San

Francisco, CA, USA, pp. 150–157.
URL http://dl.acm.org/citation.cfm?id=3091622.3091641

Dechter, R., 1986. Learning While Searching in Constraint-Satisfaction-Problems. In: Kehler, T. (Ed.), AAAI. Morgan Kaufmann, pp. 178–185.
URL http://dblp.uni-trier.de/db/conf/aaai/aaai86-1.html#Dechter86

Devlin, J., Chang, M., Lee, K., Toutanova, K., 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR abs/1810.04805.

Fujii, A., Tokunaga, T., Inui, K., Tanaka, H., Dec. 1998. Selective Sampling for Example-based Word Sense Disambiguation. Comput. Linguist. 24 (4), 573–597.
URL http://dl.acm.org/citation.cfm?id=972764.972766

Gal, Y., Ghahramani, Z., 20–22 Jun 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In: Balcan, M. F., Weinberger, K. Q. (Eds.), Proceedings of The 33rd International Conference on Machine Learning. Vol. 48 of Proceedings of Machine Learning Research. PMLR, New York, New York, USA, pp. 1050–1059.
URL http://proceedings.mlr.press/v48/gal16.html

Gal, Y., Islam, R., Ghahramani, Z., 2017. Deep Bayesian Active Learning with Image Data. CoRR abs/1703.02910.
URL http://arxiv.org/abs/1703.02910

Gers, F. A., Schmidhuber, J., Cummins, F., Sep. 1999. Learning to Forget: Continual Prediction with LSTM. In: 1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470). Vol. 2. pp. 850–855 vol.2.

Glorot, X., Bordes, A., Bengio, Y., 11–13 Apr 2011. Deep Sparse Rectifier Neural Networks. In: Gordon, G., Dunson, D., Dudík, M. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Vol. 15 of Proceedings of Machine Learning Research. PMLR, Fort Lauderdale, FL, USA, pp. 315–323.
URL http://proceedings.mlr.press/v15/glorot11a.html

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative Adversarial Nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., Weinberger, K. Q. (Eds.), Advances in Neural Information Processing Systems 27. Curran Associates, Inc., pp. 2672–2680.
URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep Residual Learning for Image Recognition. CoRR abs/1512.03385.
URL http://arxiv.org/abs/1512.03385

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2012. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. CoRR

abs/1207.0580.
URL http://arxiv.org/abs/1207.0580

Hochreiter, S., Schmidhuber, J., 12 1997. Long Short-term Memory. Neural Computation 9, 1735–80.

Hopfield, J. J., 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Sciences 79 (8), 2554–2558.
URL https://www.pnas.org/content/79/8/2554

Howard, J., Ruder, S., 2018. Fine-tuned Language Models for Text Classification. CoRR abs/1801.06146.

Huang, S., Zhao, J., Liu, Z., 2018. Cost-Effective Training of Deep CNNs with Active Model Adaptation. CoRR abs/1802.05394.

Hubel, D. H., Wiesel, T. N., 1968. Receptive Fields and Functional Architecture of Monkey Striate Cortex. Journal of Physiology (London) 195, 215–243.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F. B., Wattenberg, M., Corrado, G., Hughes, M., Dean, J., 2016. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. CoRR abs/1611.04558.
URL http://arxiv.org/abs/1611.04558

Kim, Y., 2014. Convolutional Neural Networks for Sentence Classification. CoRR abs/1408.5882.

King, R., Whelan, K., Jones, F., Reiser, P., Bryant, C., Muggleton, S., Kell, D., Oliver, S., 1 2004. Functional Genomic Hypothesis Generation and Experimentation by a Robot Scientist. Nature -London- 427 (6971), 247–252.

Kingma, D. P., Ba, J., 2014. Adam: A Method for Stochastic Optimization. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
URL http://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. NIPS'12. Curran Associates Inc., USA, pp. 1097–1105.
URL http://dl.acm.org/citation.cfm?id=2999134.2999257

Kullback, S., Leibler, R. A., 03 1951. On Information and Sufficiency. Ann. Math. Statist. 22 (1), 79–86.
URL https://doi.org/10.1214/aoms/1177729694

Lang, K. J., Baum, E. B., 1992. Query Learning Can Work Poorly when a Human Oracle is Used. Proceedings of the IEEE International Conference on Machine Learning (ICML).

Lewis, D. D., Gale, W. A., 1994. A Sequential Algorithm for Training Text Classifiers. CoRR abs/cmp-lg/9407020.
URL http://arxiv.org/abs/cmp-lg/9407020

Little, W., 01 1974. The Existence of Persistent States in the Brain 120, 101–120.

Long, J., Shelhamer, E., Darrell, T., 2014. Fully Convolutional Networks for Semantic Segmentation. CoRR abs/1411.4038.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., Potts, C., June 2011. Learning Word Vectors for Sentiment Analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, Portland, Oregon, USA, pp. 142–150.
URL http://www.aclweb.org/anthology/P11-1015

McCallum, A., Nigam, K., 1998. Employing EM and Pool-Based Active Learning for Text Classification. In: Proceedings of the Fifteenth International Conference on Machine Learning. ICML '98. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 350–358.
URL http://dl.acm.org/citation.cfm?id=645527.757765

Merity, S., Keskar, N. S., Socher, R., 2017. Regularizing and Optimizing LSTM Language Models. CoRR abs/1708.02182.

Mikolov, T., Joulin, A., Baroni, M., 2015. A Roadmap towards Machine Intelligence. CoRR abs/1511.08130.
URL http://arxiv.org/abs/1511.08130

Pan, S. J., Yang, Q., Oct. 2010. A Survey on Transfer Learning. IEEE Trans. on Knowl. and Data Eng. 22 (10), 1345–1359.
URL http://dx.doi.org/10.1109/TKDE.2009.191

Polyak, B. T., Juditsky, A. B., Jul. 1992. Acceleration of Stochastic Approximation by Averaging. SIAM J. Control Optim. 30 (4), 838–855.
URL http://dx.doi.org/10.1137/0330046

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., 2018. Improving Language Understanding by Generative Pre-Training. In: arxiv.

Rosenblatt, F., 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. Psychological Review, 65–386.

Rumelhart, E, D., Mcclelland, J., L, J., 01 1986a. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1. Foundations.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986b. Learning Representations by Back-propagating Errors. Nature 323 (6088), 533–536.
URL http://www.nature.com/articles/323533a0

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., Hadsell, R., 2016. Sim-to-Real Robot Learning from Pixels with Progressive Nets. CoRR abs/1610.04286.
URL http://arxiv.org/abs/1610.04286

Samuel, A. L., 1959. Some Studies in Machine Learning Using the Game of Checkers. IBM JOURNAL OF RESEARCH AND DEVELOPMENT, 71–105.

Schmidhuber, J., 2014. Deep Learning in Neural Networks: An Overview. CoRR abs/1404.7828.
URL http://arxiv.org/abs/1404.7828

Schmidhuber, J., 2017. Deep Learning. In: Encyclopedia of Machine Learning and Data Mining. Springer, pp. 338–348.

Settles, B., 2009. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.
URL http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf

Settles, B., Craven, M., 2008a. Active Learning with Real Annotation Costs.

Settles, B., Craven, M., 2008b. An Analysis of Active Learning Strategies for Sequence Labeling Tasks. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP '08. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 1070–1079.
URL http://dl.acm.org/citation.cfm?id=1613715.1613855

Seung, H. S., Opper, M., Sompolinsky, H., 1992. Query by Committee. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. COLT '92. ACM, New York, NY, USA, pp. 287–294.
URL http://doi.acm.org/10.1145/130385.130417

Shannon, C. E., 7 1948. A Mathematical Theory of Communication. The Bell System Technical Journal 27 (3), 379–423.
URL https://ieeexplore.ieee.org/document/6773024/

Shen, Y., Yun, H., Lipton, Z. C., Kronrod, Y., Anandkumar, A., 2017. Deep Active Learning for Named Entity Recognition. CoRR abs/1707.05928.
URL http://arxiv.org/abs/1707.05928

Shleifer, S., 2019. Low Resource Text Classification with ULMFit and Backtranslation. CoRR abs/1903.09244.
URL http://arxiv.org/abs/1903.09244

Simonyan, K., Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: International Conference on Learning Representations.

Smith, L. N., 2015. Cyclical Learning Rates for Training Neural Networks. CoRR abs/1506.01186.
URL http://arxiv.org/abs/1506.01186

Smith, L. N., 2018. A Disciplined Approach to Neural Network Hyper-parameters: Part 1 - Learning Rate, Batch Size, Momentum, and Weight Decay. CoRR abs/1803.09820.
URL http://arxiv.org/abs/1803.09820

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 1929–1958.
URL http://jmlr.org/papers/v15/srivastava14a.html

Thompson, C. A., Califf, M. E., Mooney, R. J., 1999. Active Learning for Natural Language Parsing and Information Extraction. In: Proceedings of the Sixteenth International Conference on Machine Learning. ICML '99. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 406–414.
URL http://dl.acm.org/citation.cfm?id=645528.657614

Tong, S., Koller, D., Mar. 2002. Support Vector Machine Active Learning with Applications to Text Classification. J. Mach. Learn. Res. 2, 45–66.
URL https://doi.org/10.1162/153244302760185243

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I., 2017. Attention Is All You Need. CoRR abs/1706.03762.
URL http://arxiv.org/abs/1706.03762

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., Fergus, R., 17–19 Jun 2013. Regularization of Neural Networks using DropConnect. In: Dasgupta, S., McAllester, D. (Eds.), Proceedings of the 30th International Conference on Machine Learning. Vol. 28 of Proceedings of Machine Learning Research. PMLR, Atlanta, Georgia, USA, pp. 1058–1066.
URL http://proceedings.mlr.press/v28/wan13.html

Wang, K., Zhang, D., Li, Y., Zhang, R., Lin, L., 2017. Cost-Effective Active Learning for Deep Image Classification. CoRR abs/1701.03551.
URL http://arxiv.org/abs/1701.03551

Weiss, K., Khoshgoftaar, T. M., Wang, D., May 2016. A Survey of Transfer Learning. Journal of Big Data 3 (1), 9.
URL https://doi.org/10.1186/s40537-016-0043-6

Young, T., Hazarika, D., Poria, S., Cambria, E., 2017. Recent Trends in Deep Learning Based Natural Language Processing. CoRR abs/1708.02709.
URL http://arxiv.org/abs/1708.02709

Zhang, X., Zhao, J. J., LeCun, Y., 2015. Character-level Convolutional Networks for Text Classification. CoRR abs/1509.01626.
URL http://arxiv.org/abs/1509.01626

Zhdanov, F., 2019. Diverse mini-batch Active Learning. CoRR abs/1901.05954.
URL http://arxiv.org/abs/1901.05954

Zhou, Z., Shin, J., Zhang, L., Gurudu, S., Gotway, M., Liang, J., July 2017. Fine-Tuning Convolutional Neural Networks for Biomedical Image Analysis: Actively and Incrementally. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4761–4772.

Zhu, J., Bento, J., 2017. Generative Adversarial Active Learning. CoRR abs/1702.07956. URL http://arxiv.org/abs/1702.07956

Zhu, X., 2005. Semi-Supervised Learning with Graphs. Ph.D. thesis, Carnegie Mellon University.

Zhu, Y., Chen, Y., Lu, Z., Pan, S. J., Xue, G.-R., Yu, Y., Yang, Q., 2011. Heterogeneous Transfer Learning for Image Classification. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence. AAAI'11. AAAI Press, pp. 1304–1309. URL http://dl.acm.org/citation.cfm?id=2900423.2900630