Deep Active Learning using Monte Carlo Dropout

Lucas Albuquerque Medeiros de Moura

Text submitted
to
Institute of Mathematics and Statistics
of
University of São Paulo
for the
Master Degree In
Computer Science

Advisor: Professor Marcelo Finger

This research is supported by Cnpq, Brazil

São Paulo, November, 2018

Acknowledgements

"Não sou nada.
Nunca serei nada.
Não posso querer ser nada.
À parte isso, tenho em mim
todos os sonhos do mundo."

Tabacaria - Fernando Pessoa

Leaving my home town to pursue this Master's degree was not an easy task. I have left both my parents and friends behind to chase this dream. Although I passed for some hard times during this journey, I feel extremely lucky to have the opportunity of living and meeting such a wonderful group of people while I was living in São Paulo. I can honestly say that I wouldn't be able to finish this research without the aid and contribution of many of those people.

First, I would like to thank my parents, Antônio Cândido de Moura and Maria Helena de Medeiros Moura. Your words and support were essential for building my courage on moving to another city and starting chasing this new objective. You were my fortress and I knew I could always count on both of you, no matter how difficult the situation may seem.

I would also like to thank professor Paulo Roberto Miranda Meirelles, for helping and recommending me into pursuing this Master's Degree. Not only that, I would like to thank you for being the key professor on allowing me to become the Software Engineer I am today. You have not only opened my mind on the advantages of open source solutions, but also gave me an opportunity to work with people that have thought me so much about software development. I can honestly say that you have changed my life and I am certain that more students you have oriented feel the same way.

Furthermore, I also need to acknowledge my advisor, professor Marcelo Finger, for giving me the opportunity to do this Master's research. I am grateful for all the discussions we had about this research and the confidence you had on me when we discussed possible solutions for the problems we faced. Additionally, you have provided several resources that made this research possible while also helping me on further gathering addiational resources with other professors as well. One of these professors that also need to be acknowledge is Alfredo Goldman, which allowed me to use one of his machines to perform this research experiments.

Finally, I cannot thank enough for the friends that I had the pleasure to live with. First, I would

like to thank my friends in Lappis Society. I have been learning from you guys ever since I had the opportunity to work with you at UnB and this have not changed a thing in São Paulo, but only increased. I have always struggled to keep up with your competence and skills. I believe I always will, but I am happy to have lived with people that always pushed me to improve, not only on Software Engineering, but in my personal life as well.

I was also blessed on meeting amazing people on the laboratory I have worked on during this Master's research. I would like to personally thank Felipe Salvatore, Thiago Bueno and Paula Kintschev for helping me during this research. You guys have opened my eyes on the importance of sharing my work and showing to the world what I have done and what I can still do. This was one of the most valuable lessons I have learned in this journey. Not only that, but you guys have been an amazing company to have during this research. I can say with confidence that without you, the years I have spent would have been much harsher than they were.

Resumo

Modelos de Aprendizado Profundo necessitam de uma vasta quantidade de dados anotados para serem criados. Entretanto, existem muitas áreas onde obter dados anotados é uma tarefa custosa. Neste cenário, o uso de Aprendizado Profundo se torna bastante difícil. Uma maneira de lidar com essa situação é usando a técnica de Aprendizado Ativo. Inicialmente, essa técnica cria um modelo com os dados anotados disponíveis. Depois disso, ela incrementalmente escolhe dados não anotados que irão, potencialmente, melhorar à acurácia do modelo, se adicionados aos dados de treinamento. Para selecionar quais dados serão anotados, essa técnica necessita de uma medida de incerteza sobre as predições geradas pelo modelo. Entretanto, tal medida não é usualmente realizada em modelos de Aprendizado Profundo. Uma nova técnica foi proposta para lidar com a problemática de medir a incerteza desses modelos, chamada de *Monte Carlo Dropout*. Essa técnica permitiu o uso de Aprendizado Ativo junto com Aprendizado Profundo para tarefa de classificação de imagens. Essa pesquisa visa averiguar se ao modelarmos a incerteza em modelos de Aprendizado Profundo com a técnica de *Monte Carlo Dropout*, será possível usar a técnica de Aprendizado Ativo para tarefa de análise de sentimento, uma área com uma vasta quantidade de dados, mas poucos deles anotados.

Abstract

Deep Learning models rely on a huge amount of labeled data to be created. However, there are a number of areas where labeling data is a costly process, making Deep Learning approaches unfeasible. One way to handle that situation is by using the Active Learning technique. Initially, it creates a model with the available labeled data. After that, it incrementally chooses new unlabeled data that will potentially increase the model accuracy, if added to the training data. To select which data will be labelled next, this technique requires a measurement of uncertainty from the model prediction, which is usually not computed for Deep Learning methods. A new approach has been proposed to measure uncertainty in those models, called *Monte Carlo Dropout*. This technique allowed Active Learning to be used together with Deep Learning for image classification. This research will evaluate if modeling uncertainty on Deep Learning models with *Monte Carlo Dropout* will make the use of Active Learning feasible for the task of sentiment analysis, an area with huge amount of data, but few of them labeled.

Keywords: Deep Learning, Active Learning, Sentiment Analysis

Contents

Abbreviations

DL	Deep Learning
AL	Active Learning
SVM	Support Vector Machines
KL	Kullback-Leibler divergence
ELBO	Evidence Lower Bound
LC	Least Confident
H	Entropy
I	Mutual Information
LSTM	Long Short Term Memory
ALU	Active Learning with Monte Carlo Dropout
ALS	Active Learning with Softmax
SVM	Support Vector Machine
LMRD	Large Movie Review Dataset
SD	Subjective Dataset
CEAL	Cost-Effective Active Learning

Nomenclature

$\mathbf{A}$	Matrix
$\mathbf{a}$	Vector
a	Scalar
$\mathbf{X}$	Dataset inputs
$\mathbf{Y}$	Dataset outputs
$\mathbf{x_n}$	Input data point
$\mathbf{y_n}$	Output data point
f^θ	Model parametrized by θ
$\mathbf{x}^*$	Input data outside of $\mathbf{X}$
$\mathbf{y}^*$	Model output for $\mathbf{x}^*$

Chapter 1

Introduction

Deep Learning(DL) methods have achieved state-of-the-art results in a varied number of domains, ranging from Machine Translation [SVL14] to Object Classification and Image Detection [KSH12]. However, to provide such results, DL models rely on a large amount of labeled data. For example, the task of object classification for 1000 different classes required more than 1 million annotated images to train the *ImageNet* model[KSH12]. This scenario creates a challenge on applying DL methods on areas which only unlabeled data is available, but labeling is a costly process. For instance, using Deep Learning to diagnose Alzheimer on Magnetic Resonance Image(MRI) scans would require domain experts on that area to label the data [MFC+10].

One approach to handle the problem of few labeled data is the Active Learning(AL) technique. This technique divides the available data into two distinct groups, labeled and unlabeled data. The labeled group is used to create an initial model which is evaluated and used to select the most informative samples from the unlabeled group. These samples are annotated by an oracle, i.e. a domain expert, and added to the labeled group. With the new samples added to the labeled group, a new model is created and the AL cycle restarts. Therefore, this technique grants the model the option to select which data will be used for training. By adding this flexibility to the model, it is expected that learning occurs without the necessity of a large quantity of annotated data [TK02]. This approach was proven successful on a range of tasks, such as object classification [VG14] and text classification [SC00].

To use the AL technique it is required to define which examples in the unlabeled data group are the most informative, in other words, it is necessary to define a selection policy for unlabeled data. Most policies are based on the confidence score of the model for the unlabeled samples, or the uncertainty of the model on the samples it classify [Set09]. Although an important concept, this is not commonly measured in DL, making it difficult to apply AL techniques on it.

One approach to measure uncertainty on DL methods is the *Monte Carlo Dropout* technique, which is based on Variational Inference and Bayesian Neural Networks [GG15b]. This technique was further explored on the context of Bayesian Convolutional Neural Networks together with AL to classify handwritten digits, which achieved a test error bellow 1.43% using around 900 labeled images [GIG17]. Therefore, these new results indicate that it is possible to perform *Deep Active Learning*, AL together with DL. Following this idea, we believe that this can be used on other classification tasks, such as sentiment analysis. This task involves classifying a text as having a positive or negative feeling. This type of text is produced daily on websites such as *Facebook* and *New York Times*. However, most of the data produced is unlabeled, meaning that the amount of unlabeled data is always surpassing the amount of labeled data. By creating a model that can select which text an user should classify, we would enable a dynamic model, that could be incrementally developed without the need of a vast training set of labeled texts.

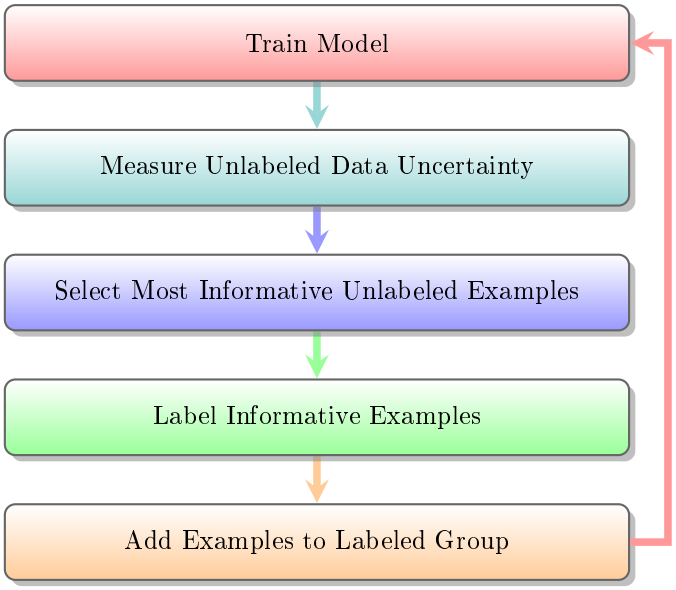

Figure 1.1: *Simplified diagram of the Active Learning cycle*

This research will focus on applying the modelling of uncertainty for DL methods and the use of AL for the task of sentiment analysis. To perform such task, the Large Movie Review Database [MDP+11] and Subjectivity Dataset [PL04] will be used, because of their use on well stablished researches on the area of sentiment analysis. We believe that by measuring uncertainty using the *Monte Carlo Dropout* technique, it will be possible to achieve higher accuracy values than by using only the conventional uncertainty measurements from Deep Learning models.

1.1 Motivation

To bring the advantages of DL models to different areas, we need to deal with tasks with few labeled data. AL is one possible framework that addressed the problem with few labeled data and machine learning. To use AL, we need to measure the model's uncertainty over unlabeled data and select the samples which will benefit our model the most, if they were labeled. To perform such selection, we need to select the samples which the model is more uncertain about its classification, we need the uncertainty measurement of the samples, as can be seen in Figure 1.1. This is not usually computed for DL models, because of the complexity to perform such measurements. Recently, the *Monte Carlo Dropout* technique has been proposed to allow for an easier measurement of the model uncertainty. This approach has been tested in the context of Image Classification together with Active Learning, achieving the best results in comparison with other AL approaches [GIG17]. Based on these results, this research aims at verifying if the *Monte Carlo Dropout* technique can be applied to different task using the AL framework. Therefore, we aim at evaluating how the *Monte Carlo Dropout* coupled with the AL framework behaves for a different problem, sentiment analysis, and a different architecture than the one used for Image Classification.

We strongly believe that by verifying the behavior of this technique for a different task context, we can verify if the technique is task or architecture dependent or if it can generalize to new tasks and architectures.

1.2 Objective

The main objective of this research is to investigate the use of AL together with DL for the task of sentiment analysis, using the *Monte Carlo Dropout* as an uncertainty measurement. This will require the understanding on how uncertainty can be measured in a DL model, ranging from

the classical approach of using the *softmax* function as an uncertainty measurement of the model to the rationale behind the *Monte Carlo Dropout* for uncertainty measurement.

To evaluate this goal, the following research questions were created:

- **Q1**: On the task of sentiment analysis, does modelling the uncertainty measurement of the model using the *Monte Carlo Dropout* technique help us achieve a better accuracy value in the Active Learning context ?
- **Q2**: Does *Monte Carlo Dropout* provides best uncertainty measurements then using the softmax output as a uncertainty measurement, the classical approach used in DL models ?

Chapter 2

Related Work

The combination of DL methods and sentiment analysis tasks is not new. Many researchers have already explored different DL architectures and approaches to handle sentiment analysis. For document level classification (verify if a document has a positive or negative opinion), the common approaches use a combination of Long-Short Term Memories (LSTM) models and Convolutional Neural Networks (CNN) to handle text data. Basically, both LSTM and CNN are used to encode the document into an input array. This array is then fed into another DL model, such as a Neural Network, and a classification is generated. This approach has been used in the works of [TQL15]. In that work, the authors encoded each text paragraph using a LSTM. Following that, each encoded paragraph is used as an input to another Recurrent Neural Network, creating the document encoding, which is used to perform the sentiment classification. Another work that uses LSTM to encode the document text for classification is [XCQH16]. This works modify the LSTM architecture, adding it a cache mechanism. This is achieved by splitting the memory of the LSTM into different groups, where each has it own rate for the forgetting gate. The authors argue that allowing some memories to have higher and lower forgetting gates, allows the network to learn about both global and local semantic features. Finally, one work that uses a CNN to encode the document instead of a LSTM is the one of [Kim14]. In this work, the author uses a CNN to create different n-gram filters for analysing the text. The document is then represented as the composition of these n-grams filters. The authors show that this network achieves similar results than using LSTM models, but with the facilities of using a CNN model, such as faster model training.

From these works, we have perveived that Deep Learnign models achieved state-of-the art results on the the task of sentiment analysis. Additionally, most works on the area rely on encoding the document data into an continuous array, using a LSTM or a CNN to achieve that task. Because of these results and the abundance of text data that can be gathered online, we believed that research on combining Active Learning (AL) and DL models would be abundant. However, this was not the case. We have found few works addressing a AL together with DL.

One work that addresses the question of using DL together with AL is the *CEAL* framework [WZL+17]. Different from other Active Learning frameworks, this model exploits unlabeled data to train the model. The unlabeled data which the model is most certain about are assigned a label and used as training data. After this data is used, their labels are erased and they are added again to the unlabeled data pool. Therefore, this framework guarantees that more data will be used for training on each Active Learning cycle. This strategy achieved better results when compared with other Active Learning techniques for image classification on the CACD dataset [CCH14] and Caltech-256 dataset [GHP07]. Although the results are expressive enough, there is a limitation on how the framework handles model uncertainty. Uncertainty measurement is given by the output of the softmax layer in the model, a classical approach to handle uncertainty in DL models. Although the softmax function produces a valid probability distribution, the result it produces should not be trustworthy when dealing with uncertainty. It is possible to produce unrecognizable images

that achieve a high accuracy prediction by networks which use the softmax layer as the prediction layer [NYC14]. One of the main reasons behind that is the use of Maximum Likelihood Estimation (MLE) for estimating the model's parameter. This technique finds parameters that maximise the likelihood of the model to produce the observed data, or the training data. This means that the probability distribution generated by the softmax function is based solely on a point estimate of the model's parameters. When feeding the softmax with a single point estimate, it tends to provide overly confident predictions, even for data far away from the observed data. If we could pass the entire distribution of the model's parameters, our softmax output would provide better predictions. [Gal16]. Another reason for not interpreting the softmax output as true uncertainty estimates is that since most DL methods work with high dimensional data, the classification space allocated to a given class may be far bigger than the area which is inhabited by the training examples. Therefore, images that are deep into a classification region may generate high confidence prediction, even if the images are very different from the natural images for that class [NYC14]. Therefore, we can understand that the probability distribution provided by the softmax output is not a sound measurement for uncertainty in the model's predictions.

Another work that deals with Deep Learning and Active Learning is the creation of a Bayesian Convolutional Neural Network, that extracts the model uncertainty by using the *Monte Carlo Dropout* technique [GIG17]. This model has achieved an error of only 1.43% for the MNIST dataset by using fewer than 900 labeled images. However, this strategy does not use the unlabeled data to further increase the training set, which is an approach that could be used.

When considering text data, the *Monte Carlo Dropout* technique was also used in task of Named Entity Recogniton (NER) in the work of [SYL+17]. In this work they have used AL to identify if they could achieve state-of-the-art results in the NER task using less annotated data. The *Monte Carlo Dropout* technique was one of the techniques used to provide uncertainty measurements for selecting new samples for labeling in the experiment. For the NER task, the *Monte Carlo Dropout* was slightly outperformed by a metric devised by the researchers. However, the designed metric used to measure the uncertainty cannot be easily generalized to other contexts, which is not the case for the *Monte Carlo Dropout* technique. The researchers found that both uncertainty measurements have achieved state-of-the-art results using 25% of the labeled data.

Another work that address text data and the AL technique is the work of [ZW16]. In this work, the researchers do not use the classification uncertainty of the model to select samples from the unlabeled dataset. Instead, they select unlabeled samples that, if added to the training data, would most affect the word embedding representation of the words. A word embedding is a vectorial representation of a word, mostly used to give a dense input to a neural network when dealing with text data [MCCD13]. In their research, they have observed that their technique has outperformed other techniques that use uncertainty measurements on sentence and document classification tasks. Their technique was compared with uncertainty measurements that use the softmax output as an uncertainty measurement. As we have discussed, using the softmax output is not a sound uncertainty measurement. Therefore, further comparisons could be made if the Al model that used uncertainty measurement used the *Monte Carlo Dropout* to better estimate the uncertainty.

There are also other researches on NLP tasks using AL, but using Machine Learning models instead of DL methods. For example, in [KSP+15], the authors first train a baseline linear SVM classifier on 1,600,000 tweets (50% positive and 50% negative), achieving an accuracy of 83.01%. After that, they create a dataset with 11,389 tweets (43% positive, 16% negative and 41% neutral) and separate it into a series of batches with 1000 tweets each. Each batch of data is fed into the baseline algorithm, where two scenarios are tested: The algorithm can use Active Learning and choose 100 tweets from each batch to be manually labeled and added to training set, or not use Active Learning and not retrain the model. In the first setting, the average accuracy for a batch

of tweets is 41.8%, while for the second, 34.9%. Although these accuracies are not very high, it is necessary to understand that the baseline algorithm was trained for binary classification (positive and negative tweets), and the test data contains 41% of neutral tweets, which may cause the low accuracy over the batches. However, even the baseline algorithm does not possess a high accuracy for the binary classification of tweets, which only have 140 characters. It is possible that the linear SVM model used may not be robust enough to handle tweets text information in order to perform a better classification.

Finally, to the best of our knowledge, we could not find any work that combines a pure LSTM DL model with AL. Although [SYL+17] uses a LSTM in their model, a CNN is responsible for encoding the text data. Therefore, the LSTM is used for a significant smaller input than if it was responsible for encoding the whole text unto a vectorial representation. Therefore, we believe this is be one of the first works that tries to combine a pure LSTM model with the AL technique.

Chapter 3

Background

This chapter will introduce the necessary information used on the rest of this work. Initially, we will address the mathematical notation used to explain the necessary concepts. After that, from section 3.2 through 3.3.3 we will provide the background notions for the creation and understanding of the *Monte Carlo Dropout* technique. On section 3.4, we will explain how a *Recurrent Neural Network* works and on section 3.5, how the active learning technique works. Finally, the final section of this chapter, 3.6 explains how we can combine *Monte Carlo Dropout*, *Recurrent Neural Networks* and *Active Learning*. Therefore, this sections will be an overview of the approach used in this research, allowing the reader to skip sections it already has knowledge about and focus only on how the techniques are going to be combined together.

3.1 Mathematical Notation

Through this work, we will use the following mathematical notation. Bold upper case letters $\mathbf{X}$ will denote matrices while bold lowercase letters $\mathbf{x}$ will denote vectors. Scalars will be defined by standard letters, such as x. Subscripts are used to denote different variables, such as $\mathbf{W_1}$ and $\mathbf{W_2}$. Also, bold lowercase letters will denote a row vector. If the variable also has a subscript, it will denote a certain row from a matrix. For example, $\mathbf{x_1}$ will denote the first row of matrix $\mathbf{X}$. To represent a scalar inside a matrix variable we will use three number, one to identify the variable, other to identify the row of the matrix and the last one, to identify the column of the matrix. For example, to identify the first row and column of the matrix $\mathbf{X_2}$, we will use the variable $x_{2,1,1}$.

When dealing with machine learning models, their parameters will be described by the θ variable and we will use superscripts such as f^θ to denote a model parametrised by the variable θ. Additionally, λ will describe an array of variables, i.e. $\lambda = \mu, \sigma$.

Finally, a common operation in machine learning models is the Hadamard product, where we multiply two vectors element-wise. To diferentiate this operation from the dot product between vectors, we will represent the Hadamard product with the symbol $\odot$.

3.2 Uncertainty and Deep Learning

Deep Learning(DL) is a subfield of machine learning interested in the training and use of large Neural Networks (left part of Figure 3.1). One of the most common tasks associated with DL is supervised learning. In this setting, the Deep Learning model is presented with a set of training inputs, $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}]$ and an associated set of outputs, $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_n}]$. The task of the model is to learn a function $\mathbf{Y} = f^\theta(\mathbf{X})$ capable of transforming the inputs $\mathbf{X}$ into the outputs $\mathbf{Y}$ using the parameters θ. Therefore, in order to use DL methods, we need to choose a model f^θ and find the best θ parameters for this model, an activity called *training*.

In order to train a DL model we need to define an *objective function*. This function is responsible for measuring how well the outputs of our network are matching the real outputs in our training set, Y. Most of the approaches used to train these models tries to minimize this objective function. However, we can also view the training step of these models through a probability perspective.

3.3 Bayesian Neural Network

One approach that can be used to estimate the parameters of a model is the Bayesian method, where each parameter of our network will be defined by a probability distribution, as can be seen on the right part of Figure 3.1.

A Bayesian Neural Network uses Baye's theorem to find a probability distribution for the parameters. The theorem can be defined by Equation 3.1

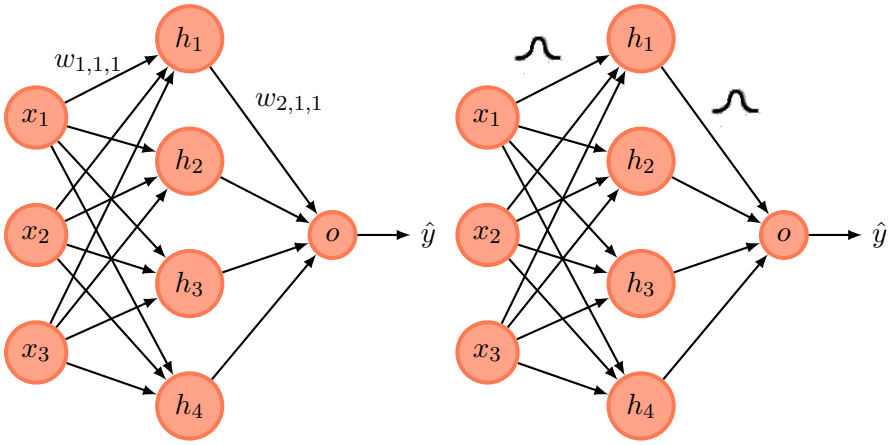

Figure 3.1: *Examples of two distinct approaches to Neural Networks. (Left) A classical Neural Network, where each parameter is a scalar value. (Right) A Bayesian Neural Network, where each parameter is defined by a probability distribution.*

$$p(\theta|\mathbf{X},\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X},\theta)p(\theta)}{p(\mathbf{Y}|\mathbf{X})} \tag{3.1}$$

In order to understand Equation 3.1 for the context of a DL model, each individual piece of the equation must be analyzed. The first piece is the term $p(\theta)$, our prior term. This term indicates an initial guess of the behaviour of our model's parameters. The choice of the probability distribution over our parameters happens before our model see any data, meaning that this choice is made when the model is created. The second term of our equation is $p(\mathbf{Y}|\mathbf{X},\theta)$, which is called the *likelihood function*. This likelihood function is a probability distribution that measures how probable the predictions $\mathbf{Y}$ are, given the model's input data, $\mathbf{X}$, and parameters, θ. Finally, the term $p(\mathbf{Y}|\mathbf{X})$ is the regularization term, used to guarantee that our posterior distribution $p(\theta|\mathbf{X},\mathbf{Y})$ sums to 1, meaning that it is a valid probability distribution. Once all these values are computed and we apply these values in the equation 3.1, we get a posterior distribution of our parameters, $p(\theta|\mathbf{X},\mathbf{Y})$. This posterior distribution captures the most probable parameters given our observed data.

The Bayesian approach gives a probability distribution over our model's parameters. Therefore, in order to give a prediction, $\mathbf{y}^*$ for an unseen input $\mathbf{x}^*$ we can use the following equation:

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \theta) p(\theta|\mathbf{X}, \mathbf{Y}) d\theta \tag{3.2}$$

As Equation 3.2 shows, we calculate the probability of a model parameter to generate a prediction $\mathbf{y}^*$ scaled by the posterior probability of that parameter $p(\theta|\mathbf{X}, \mathbf{Y})$. This procedure is applied to all θ parameters.

This approach is a costly operation, since integrating over the entire parameter space is a computationally costly procedure. A different approach for prediction is instead to generate t output samples, by sampling a new parameter configuration from our posterior distribution for each new output, as can bee seen on Equation 3.7 and best visualized on Figure 3.2

$$\hat{y}_1 = p(y^*|x^*, \hat{\theta}_1); \quad \hat{\theta}_1 \sim p(\theta|X, Y) \tag{3.3}$$
$$\hat{y}_2 = p(y^*|x^*, \hat{\theta}_2); \quad \hat{\theta}_2 \sim p(\theta|X, Y) \tag{3.4}$$
$$\ldots \tag{3.5}$$
$$\hat{y}_t = p(y^*|x^*, \hat{\theta}_t); \quad \hat{\theta}_t \sim p(\theta|X, Y) \tag{3.6}$$
$$p(y^*|x^*) = \frac{1}{t} \sum_{k=1}^{t} p(y^*|x^*, \hat{\theta}_k) \tag{3.7}$$

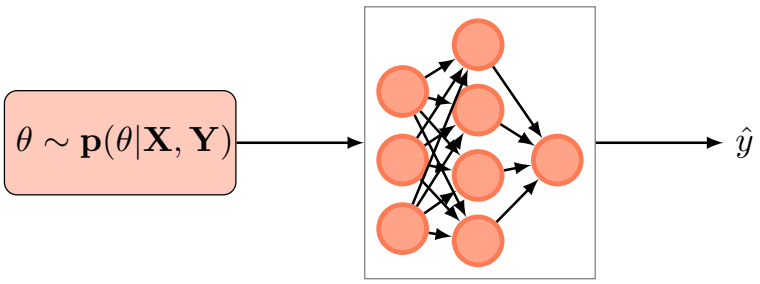

Figure 3.2: *Example of a Bayesian Neural Network used for classification. The Network needs to make a prediction for a given input. There will be t predictions for the input and for every prediction, we sample new parameterts from the posterior distribution and use these parameters to run our Network, generating a new prediction. With these predictions, we can measure the model uncertainty.*

Besides obtaining the expected value of our prediction, we can also use the t output predictions to calculate the prediction variance, quantifying how *uncertain* the model is about its prediction. If the predictions have a low variance, we can understand that our model is confident on its prediction. Uncertainty measurements can be specially useful when our model needs to make a prediction on *out of distribution* data [Gal16]. On this scenario, the model has been trained on a set of data, but receives an example completely different from the training set to make a prediction. In that setting, the model should not be confident on the prediction it delivers, and it must also return an uncertainty measurement to make that clear to anyone that will use the prediction.

Although using a Bayesian Neural Networks allows the measurement of uncertainty, using the full Bayesian approach is infeasible for most models in practice. This is because of the constant $p(\mathbf{Y}|\mathbf{X})$ that can be seen on equation 3.1. In order to calculate this constant, it is necessary to integrate over all possible θ values, as can be seen on Equation 3.8

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)d\theta \tag{3.8}$$

If we do not calculate the constant on Equation 3.8, we cannot calculate the full posterior distribution $p(\theta|\mathbf{X}, \mathbf{Y})$. Therefore, most deep learning models are trained with different methods than the full Bayesian approach.

3.3.1 Classical Neural Network

The main idea behind the Bayesian approach is to find the full posterior distribution over all possible parameters settings. Since doing this is infeasible for most models, instead of evaluating all possible parameter settings, we can find a point estimate of the parameter settings that would provide the best approximation for $p(\theta|\mathbf{X}, \mathbf{Y})$. We can adapt the Bayesian approach to perform such a task as can be seen on 3.9.

$$\theta = argmax_\theta \frac{p(\mathbf{Y}|f^\theta(\mathbf{X}))p(\theta)}{p(\mathbf{Y}|\mathbf{X})}\,[1] \tag{3.9}$$

Since we are calculating the *argmax*, the regularization term $p(\mathbf{Y}|\mathbf{X})$ will not affect the *argmax* computation. Therefore, we can update the equation:

$$\theta = argmax_\theta p(\mathbf{Y}|f^\theta(\mathbf{X}))p(\theta) \tag{3.10}$$

Assuming that our training data is independent and identically distributed (i.i.d), we can turn our 3.10 into 3.11.

$$\theta = argmax_\theta \prod_{i=0}^{N} p(\mathbf{y_i}|f^\theta(\mathbf{x_i}))p(\theta) \tag{3.11}$$

In order to avoid numeric overflow, we can calculate the log of the probabilities, turning a product into a summation 3.13.

[1] We can use the *softmax* function to turn $f^\theta(X)$ into a probability distribution. The softmax function is defined by: $softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

$$\theta = argmax_\theta \sum_{i=0}^{N} log(p(\mathbf{y_i}|f^\theta(\mathbf{x_i})))p(\theta)) \tag{3.12}$$

$$\theta = argmax_\theta \sum_{i=0}^{N} log(p(\mathbf{y_i}|f^\theta(\mathbf{x_i}))) + log(p(\theta)) \tag{3.13}$$

This technique is called *Maximum A Posteriori estimation* and we can use it to find a point estimate of our parameters. On most DL models, this is achieved by maximizing an objective function, $log(p(\mathbf{y_i}|f^\theta(\mathbf{x_i})))$ or minimizing the negative of this same objective function, which turns the objective function into the *cross-entropy* error. Also, it is common to add to the objective function a regularization term for our weights, $log(p(\theta))$. One common regularizer is the L2 regularizer, which assumes that our weights are drawn from a Normal distribution [GBC16] [2]. To maximize/minimize these functions we can use techniques such as *backprogation* [RHW88] and *Gradient Descent* as can be seen on Algorithm 1. In the algorithm we update the model's parameters until they do not change much from one iteration to the next, this means that the parameters have converged to a certain value [3]. By training a model in this manner produces a network similar to the one displayed on the left part of Figure 3.1, since every parameter of our model will be a scalar value.

Algorithm 1 Training of a Neural Network with scalar parameters

1: Given dataset $\mathbf{X}$, $\mathbf{Y}$,
2: Set learning rate η,
3: Randomly initialise parameters θ,
4: **repeat**
5: Choose S as random subset of $\mathbf{X}$ of size M.
6: Calculate the gradient of an objective function with respect to θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{1}{M}\sum_{i=1}^{M} logp(\mathbf{y_i}|f^\theta(\mathbf{x_i})) - log(p(\theta))$$

7: Update θ:

$$\theta \leftarrow \theta + \eta\widehat{\mathbf{\Delta}\theta}$$

8: **until** θ has converged.

By choosing only a single parameter setting for our model, we guarantee that our model will behave deterministically, meaning that for every input it receives it will always display the same output. Although this makes training our model feasible, we cannot extract proper uncertainty measurements from it. Therefore, it can be complicated to use such models on tasks where we must assess the confidence of our models, i.e classifying medical images. Also, it is not possible to differentiate if our model is receiving an *out of distribution* example or not when performing its prediction.

3.3.2 Variational Inference

Both Bayesian and Maximum Likelihood approaches have major drawbacks. While the full Bayesian approach is intractable, the use of a point estimate of our parameters does not allow a precise measurement of uncertainty for our model. However, it is possible to look back to the full Bayesian approach and observe our posterior distribution. It is understandable that computing the

[2] Regularization techniques will be discussed in section 3.3.3
[3] This approach can lead to overfitting problems, but it is a theoretical approach to cover the basics of the process.

posterior distribution for most models is unpractical. Instead, we define another distribution, $q_\lambda(\theta)$ that is as close as possible to the true posterior distribution of our Bayesian approach, as can be seen on Figure 3.3. This method is called *Variational Inference* [JGJS99] and $q_\lambda(\theta)$ is a *variational distribution* parametrized by λ.

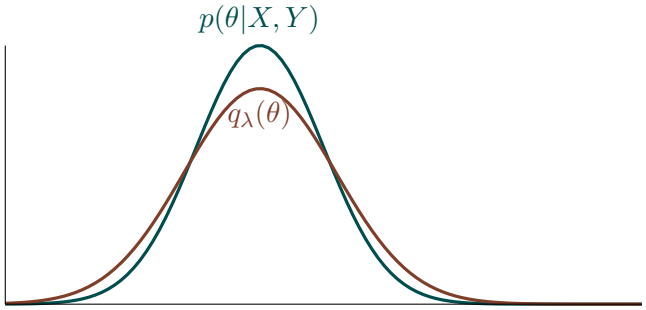

Figure 3.3: *Example of the variational inference technique.*

One way to measure the similarity between distributions is by evaluating the Kullback-Leibeler (KL) divergence between the distributions [Kul59]. This measurement can be seen on equation 3.14

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta|\mathbf{X},\mathbf{Y})} d\theta \tag{3.14}$$

From Equation 3.14, we can observe that we still need the posterior distribution $p(\theta|\mathbf{X},\mathbf{Y})$, which still does not allow the computation of $q_\lambda(\theta)$. However, we can change the original definition of the KL divergence into 3.15:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) - ELBO(\lambda) \leq logP(\mathbf{Y}|\mathbf{X}) \tag{3.15}$$

On Equation 3.15, the *Evidence Lower Bound (ELBO)* term is defined as: [4]

$$ELBO(\lambda) = \int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X},\theta) d\theta - KL(q_\lambda(\theta)||p(\theta)) \tag{3.16}$$

Since $p(\mathbf{Y}|\mathbf{X})$ is a constant term, we can understand that by maximizing the *ELBO* term, we will implicitly minimize the KL divergence. Also, Equation 3.16 does not depend on $p(\theta|\mathbf{X},\mathbf{Y})$, meaning that we can calculate its value. Therefore, in Variational Inference, we will maximize the *ELBO* term to approximate $q_\lambda(\theta)$ to $p(\theta|\mathbf{X},\mathbf{Y})$.

Using the machine learning perspective, we can observe that the *ELBO* is our objective function, and we can optimize it using backpropagation in order to find the best values for our λ values. But, in practice, we need to make some changes on the original *ELBO* function. The first change is that instead of calculating the *ELBO* for our full training set, we will calculate it for only a batch of our training set, turning 3.16 to 3.17.

[4] The full process on turning the KL diverge into Equation 3.15 is defined on appendix A

$$ELBO(\lambda) = \frac{N}{M}\sum_{i \in B}\int q_\lambda(\theta) logp(\mathbf{y_i}|\mathbf{x_i}, \theta)d\theta - KL(q_\lambda(\theta)||p(\theta)) \tag{3.17}$$

Where **B** represents the batch of examples and **M** is the size of the batch. The second problem with using *ELBO* in practice is to calculate the log likelihood term $\int q_\lambda(\theta) logp(\mathbf{y_i}|\mathbf{x_i}, \theta)d\theta$, since this is a costly operation. In order to deal with this problem, we can use a Monte Carlo integration approach, called the *re-parametrization trick* [KW13].

Re-parametrization trick

Before explaining the necessity of the re-parametrization trick, it is necessary to understand what happens in our network when we are using the *ELBO* function as our optimization function. First, recall that our neural network has the format presented on the left part of Figure 3.1.

When using Variational Inference, the weights of our network are the θ parameters of our model, and these values are defined by our variational distribution $q_\lambda(\theta)$, meaning that $\theta \sim q_\lambda(\theta)$. Therefore we can see that Equation 3.17 turns into 3.18.

$$ELBO(\lambda) = \frac{N}{M}\sum_{i \in B}\int q_\lambda(\theta) logp(\mathbf{y_i}|f^\theta(\mathbf{x_i})d\theta - KL(q_\lambda(\theta)||p(\theta)) \tag{3.18}$$

Therefore, when we are calculating our gradients in respect to λ, we need to calculate the following derivative 3.19

$$\frac{N}{M}\sum_{i \in B}\frac{\partial}{\partial\lambda}\int q_\lambda(\theta) logp(\mathbf{y_i}|f^\theta(\mathbf{x_i})d\theta - \frac{\partial}{\partial\lambda}KL(q_\lambda(\theta)||p(\theta)) \tag{3.19}$$

In order to visualize this gradient calculation, we can see it as a computational graph on Figure 3.4.

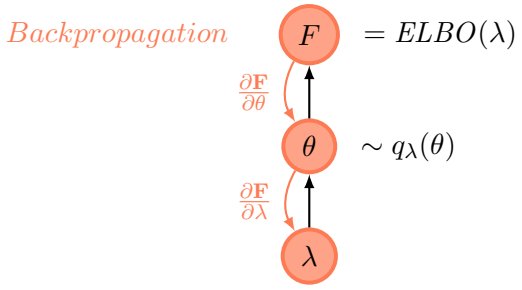

Figure 3.4: *Example of backpropagation for Variational Inference*

As can be seen on Figure 3.4, we must calculate the gradient over a random node θ, which is not a possible operation using backpropagation. The re-parametrization trick is an alternative approach to remove the stochastic variable from our computation graph. In order to achieve that, we will assume that our $q_\lambda(\theta)$ can be re-parametrized as a parameter free distribution $p(\epsilon)$, turning $\theta = g(\lambda, \epsilon)$ where $g(.,.)$ is a deterministic differentiable function. For a practical example of a re-parametrization, consider that our variational distribution $q_\lambda(\theta)$ is defined by a Gaussian distribu-

tion $\mathcal{N}(\theta; \mu, \sigma^2)$, where $\lambda = (\mu, \sigma^2)$. In that case, our variational distribution can be re-parametrized by $p(\epsilon) = \mathcal{N}(\epsilon; 0, 1)$ and $g(\lambda, \epsilon) = \mu + \sigma^2\epsilon$. Based on this example, we turn the *ELBO* equation from 3.18 to 3.20 and the computational graph from Figure 3.4 to 3.5

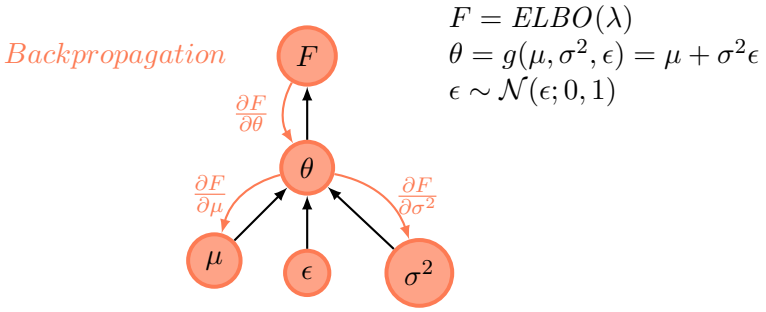

Figure 3.5: *Example of backpropagation using reparametrization trick for Variational Inference*

$$ELBO(\lambda) = \frac{N}{M}\sum_{i \in B}\int p_\epsilon logp(\mathbf{y_i}|f^{g(\lambda,\epsilon)}(\mathbf{x_i})d\epsilon - KL(q_\lambda(\theta)||p(\theta)) \tag{3.20}$$

We can see in Figure 3.5 that the node representing θ has turned from an stochastic node into a deterministic node. Allowing the flow of derivatives to reach the λ variables. This is not the only advantage of using the re-parametrization trick. We can now estimate the log likelihood term in Equation 3.20 using a Monte Carlo Integration method. First, let's call the derivative of our likelihood function as LL and define it by Equation 3.21.

$$LL(\lambda) = \frac{\partial}{\partial \lambda}\int p_\epsilon logp(\mathbf{y_i}|f^{g(\lambda,\epsilon)}(\mathbf{x_i}))d\epsilon \tag{3.21}$$

Now, our estimator of LL, $\hat{LL}$, can be defined by 3.22.

$$\hat{LL}(\lambda) = f'(g(\lambda, \epsilon))\frac{\partial}{\partial \lambda}g(\lambda, \epsilon) \tag{3.22}$$

Now, we can see that the expected value of our estimator will be the true derivative value, $\mathbb{E}_{p(\epsilon)}[\hat{LL}(\lambda)] = LL(\lambda)$. Therefore, we can turn 3.20 into 3.23 using our stochastic estimator:

$$\hat{ELBO}(\lambda) = \frac{N}{M}\sum_{i \in B} logp(\mathbf{y_i}|f^{g(\lambda,\epsilon)}(\mathbf{x_i})) - KL(q_\lambda(\theta)||p(\theta)) \tag{3.23}$$

Where $\mathbb{E}_{p(\epsilon)}(\hat{ELBO}(\lambda)) = ELBO(\lambda)$

Therefore, using the re-parametrization trick not only allows us to flow gradients through the network, but also to estimate the log likelihood function derivative using Monte Carlo Integration. With that knowledge in hand, we can use 3.23 as our final objective function and optimize it to find the best λ values for our variational distribution $q_\lambda(\theta)$.

Practical Algorithm

Together with the re-parametrization trick and Monte Carlo Integration, we can now derive a practical algorithm to train our model using Variational Inference. Recall that our objective function is 3.23 and we are going to minimize this function in order to apply backprogation to the λ parameters of our variational distribution $q_\lambda(\theta)$.

However, before presenting the algorithm, we must define how we will factorize our variational distribution over the weights. One of the early approaches of using Variational Inference factorize $q_\lambda(\theta)$ for each weight scalar [HvC93]:

$$q_\lambda(\theta) = \prod_{i=1}^{L} q_\lambda(W_i) = \prod_{i=1}^{L}\prod_{j=1}^{K_i}\prod_{k=1}^{K_{i+1}} q_{\mu_{i,j,k},\sigma_{i,j,k}}(w_{i,j,k}) = \prod_{i,j,k} \mathcal{N}(w_{i,j,k}; \mu_{i,j,k}, \sigma^2_{i,j,k}) \tag{3.24}$$

This means that we must sample each weight scalar from its respective variational distribution. Since the original work used a Gaussian distribution, we must hold for each weight scalar, two new variables, μ, σ, meaning that our number of parameters will also increase. Furthermore, by factorizing our distribution in that manner, we will loose any weight correlations in our network. In order to avoid such problems, we can instead factorize our distribution for each weight row $\mathbf{w_{l,i}}$ in each weight matrix $\mathbf{W_l}$. Turning our factorization into:

$$q_\lambda(\theta) = \prod_{l=1}^{L}\prod_{i=1}^{Rows} q_\lambda(w_{l,i}) = \prod_{l,i} \mathcal{N}(w_{l,i}; \mu_{l,i}, \sigma^2_{l,i}) \tag{3.25}$$

Since we are using the re-parametrization trick, our factorization of the weights will turn from $q_{\lambda_{l,i}}(w_{l,i})$ into $w_{l,i} = g(\lambda, \epsilon_{l,i})$. Also, we need to specify a parameter free distribution, $p(\epsilon_{l,i})$, which depends on the distribution $q_\lambda(\theta)$. Once that function is chosen, we can now use Algorithm 2 to calculate our λ values [Gal16]. In order to make the notation simple, on Algorithm 2, $p(\epsilon) = \prod_{l,i} p(\epsilon_{l,i})$ and $\theta = g(\lambda, \epsilon)$.

Algorithm 2 Minimize divergence between $q_\lambda(\theta)$ and $p(\theta|X, Y)$[Gal16]

1: Given dataset $\mathbf{X}$, $\mathbf{Y}$,
2: Set learning rate η,
3: Randomly initialise parameters λ,
4: **repeat**
5: Sample M random variables $\hat{\epsilon_i} \sim p(\epsilon)$, S a random subset of $\mathbf{X}$ of size M.
6: Calculate the Monte Carlo estimator of the $ELBO$ w.r.t to λ:

$$\widehat{\Delta\lambda} \leftarrow -\frac{N}{M}\sum_{i\in B} \frac{\partial}{\partial\lambda} logp(\mathbf{y_i}|f^{g(\lambda,\hat{\epsilon_i})}(\mathbf{x_i}) + \frac{\partial}{\partial\lambda} KL(q_\lambda(\theta)||p(\theta))$$

7: Update λ:

$$\lambda \leftarrow \lambda + \eta\widehat{\mathbf{\Delta}\lambda}$$

8: **until** λ has converged.

With this algorithm, we can make our predictions following a similar approach to Equation 3.7:

$$\tilde{q}_\lambda(y^*|x^*) := \frac{1}{t}\sum_{k=1}^{t} p(y^*|x^*, \widehat{\theta_k}) \tag{3.26}$$

with $\widehat{\theta_t} \sim q_\lambda(\theta)$.

Although a practical approach, we can see that this approach is complicated to explain, not only our weights are now probabilities distributions, but our objective function has also changed. However, there is still a third approach which is similar to Variational Inference and does not change the network architecture, called *Monte Carlo Dropout*.

3.3.3 Monte Carlo Dropout

The technique of *Monte Carlo Dropout* is an approach proposed on [GG16]. This approach aims at extracting uncertainty measurements from DL models trained with *Dropout*, an stochastic regularization technique. Before delving into the rationale behind this approach, it is necessary to explain what is *Dropout*.

Dropout

One of the most common problems that affects DL models is overfitting. This problem arises when the model adapts too well to the training data, but cannot generalize to new data. One regularization technique used to handle such issue is named *Dropout* [HSK+12]. In order to understand how *Dropout* works, let's consider a neural network with two layers. This network will have two distinct weight matrices, $\mathbf{W_1}$ and $\mathbf{W_2}$. In a common neural network setting, to train our network we would generate a prediction as such:

$$\mathbf{z_1} = \mathbf{x}\mathbf{W_1} \tag{3.27}$$
$$\mathbf{a_1} = \sigma(\mathbf{z_1}) \tag{3.28}$$
$$\mathbf{z_2} = \mathbf{a_1}\mathbf{W_2} \tag{3.29}$$
$$\hat{\mathbf{y}} = softmax(\mathbf{z_2}) \tag{3.30}$$

After that, we would plug our $\hat{y}$ prediction into an objective function and use backpropagation to update $\mathbf{W_1}$ and $\mathbf{W_2}$. This would happen for every $\mathbf{x} \in \mathbf{X}$. However, *dropout* modifies our neural by dropping some hidden units along the forward pass. For every forward pass, we sample two distinct vectors $\hat{\boldsymbol{\epsilon}_1}, \hat{\boldsymbol{\epsilon}_2}$ with dimensions corresponding to the x dimension and a_1 dimension, respectively. The element of each vector $\hat{\epsilon_i}$ are sampled from a Bernoulli distribution with probability $1 - p_i$, where $0 \leq p_i \leq 1$, for $i = 1, 2$. Therefore, our forward pass turns into:

$$\hat{\mathbf{x}} = \mathbf{x} \odot \hat{\boldsymbol{\epsilon}_1} \tag{3.31}$$
$$\mathbf{z_1} = \hat{\mathbf{x}}\mathbf{W_1} \tag{3.32}$$
$$\mathbf{a_1} = \sigma(\mathbf{z_1}) \tag{3.33}$$
$$\hat{\mathbf{a_1}} = \mathbf{a_1} \odot \hat{\boldsymbol{\epsilon}_2} \tag{3.34}$$
$$\mathbf{z_2} = \hat{\mathbf{a_1}}\mathbf{W_2} \tag{3.35}$$
$$\hat{\mathbf{y}} = softmax(\mathbf{z_2}) \tag{3.36}$$

Using that approach, after each forward pass we use just a portion of our hidden units to generate our output. Also, during backpropagation we just update the hidden units responsible for generating our output. This means that we maintain the same $\hat{\boldsymbol{\epsilon}_1}, \hat{\boldsymbol{\epsilon}_2}$ during a forward and backward pass in our model. After training our model, we scale both x and a_1 by $\frac{1}{1-p_i}$ to generate a prediction on unseen data.

This technique was further studied in [SHK+14], where some assumptions about *Dropout* were made. One of them is that training a network with *Dropout* is similar to training an assemble of different networks and using the expected value of their prediction as the prediction for a new data example. In that work, *Dropout* was also used to train different networks for tasks such as image and text classification, and the networks trained achieved better accuracy for the test data than models that did not use *Dropout* as a regularization technique.

Dropout as Monte Carlo Dropout

In order to derive the *Monte Carlo Dropout* technique from our network, we first need to understand that we want our network trained with *dropout* to behave in a similar manner than a Bayesian Neural Network. The first main difference between *Dropout* and a Bayesian Neural Network is that *Dropout* add stochastic noise into the input of each layer of our network, $\mathbf{x}, \mathbf{a_1}$. However, in a Bayesian Neural Network, our stochasticity is generated by the uncertainty we have over the model's parameters. This difference can be removed, if we adapt the original *Dropout* technique as follows [Gal16]:

$$\begin{aligned}
\hat{\mathbf{y}} &= softmax(\mathbf{z_2}) \\
&= softmax(\hat{\mathbf{a_1}}\mathbf{W_2}) \\
&= softmax((\mathbf{a_1} \odot \hat{\boldsymbol{\epsilon}_2})\mathbf{W_2}) \\
&= softmax((\mathbf{a_1} diag(\hat{\boldsymbol{\epsilon}_2}))\mathbf{W_2}) \\
&= softmax(\mathbf{a_1}(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2})) \\
&= softmax(\sigma(\mathbf{z_1})(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2})) \\
&= softmax(\sigma(\hat{\mathbf{x}}\mathbf{W_1})(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2})) \\
&= softmax(\sigma((\mathbf{x} \odot \hat{\boldsymbol{\epsilon}_1})\mathbf{W_1})(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2})) \\
&= softmax(\sigma((\mathbf{x}\mathbf{diag}(\hat{\boldsymbol{\epsilon}_1}))\mathbf{W_1})(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2})) \\
&= softmax(\sigma(\mathbf{x}(\mathbf{diag}(\hat{\boldsymbol{\epsilon}_1})\mathbf{W_1}))(diag(\hat{\boldsymbol{\epsilon}_2})\mathbf{W_2}))
\end{aligned}$$

We can see now that we have added stochasticity over our weights using *Dropout*. Let's call S as the *random variable* defined over the set of the real weight matrices, W and $\widehat{S}$ as the realization of S. With that notation, we can write $\widehat{\mathbf{S_1}} = diag(\boldsymbol{\epsilon_1})\mathbf{W_1}$ and $\widehat{\mathbf{S_2}} = diag(\boldsymbol{\epsilon_2})\mathbf{W_2}$ turning our $\hat{\mathbf{y}}$ prediction into:

$$\hat{\mathbf{y}} = softmax(\sigma(\mathbf{x}\widehat{\mathbf{S_1}})\widehat{\mathbf{S_2}}) = f^{\widehat{\mathbf{S_1}},\widehat{\mathbf{S_2}}}(x) \tag{3.37}$$

Now, let's plug 3.37 into a objective function typically associated with classification tasks:

$$E^{\mathbf{S_1},\mathbf{S_2}}(x,y) = -logp(y|f^{\widehat{\mathbf{S_1}},\widehat{\mathbf{S_2}}}(x)) + \alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2 \tag{3.38}$$

Where $\alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2$ is the the L2 regularization term and $||\mathbf{W}||^2 = \mathbf{W^T W}$.

If we calculate the objective function over a batch with size M. we turn 3.38 into:

$$E^{\mathbf{S_1},\mathbf{S_2}} = -\frac{1}{M}\sum_{i\in B} logp(\mathbf{y_i}|f^{\widehat{\mathbf{S_1}},\widehat{\mathbf{S_2}}}(\mathbf{x_i})) + \alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2 \tag{3.39}$$

Instead of calculating the full posterior distribution over our *random variables* S, we can instead approximate them by a posterior distribution $q_\lambda(\theta), \lambda = \{\mathbf{W_1}, \mathbf{W_2}\}$ and $\theta = \{\mathbf{S_1}, \mathbf{S_2}\}$. We can apply the re-parametrization trick here, by using a Bernoulli distribution $p(\boldsymbol{\epsilon})$, generating the following re-parametrization:

$$\begin{aligned} g(\lambda, \widehat{\boldsymbol{\epsilon}_i}) &= \{diag(\widehat{\boldsymbol{\epsilon}_{1,i}})\mathbf{W_1}, diag(\widehat{\boldsymbol{\epsilon}_{2,i}})\mathbf{W_2}\} \\ p(\boldsymbol{\epsilon}) &= \prod_l Bernoulli(1-p_l) \end{aligned}$$

With the re-parametrization trick, we can turn our objective function 3.39 into:

$$E^\lambda = -\frac{1}{M}\sum_{i\in B} logp(\mathbf{y_i}|f^{g(\lambda,\widehat{\boldsymbol{\epsilon}_i})}(\mathbf{x_i})) + \alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2 \tag{3.40}$$

Which will have the following derivative calculated when we apply backpropagation:

$$\frac{\partial}{\partial\lambda}E^\lambda = -\frac{1}{M}\sum_{i\in B}\frac{\partial}{\partial\lambda} logp(\mathbf{y_i}|f^{g(\lambda,\widehat{\boldsymbol{\epsilon}_i})}(\mathbf{x_i})) + \frac{\partial}{\partial\lambda}(\alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2) \tag{3.41}$$

We can understand than when we compute the derivative of λ, we are actually computing the derivate of our error function in relation to $\mathbf{W_1}$ and $\mathbf{W_2}$. For a more compact notation, we instead of writing the same formula twice, changing the derivative variable, we will summarize the derivative using $\frac{\partial}{\partial\lambda}$.

Now, with our objective function defined, we can now describe the training of such a network using Algorithm 3 [Gal16].

Algorithm 3 Optimisation of a Neural Network using dropout

1: Given dataset $\mathbf{X}$, $\mathbf{Y}$,
2: Set learning rate η,
3: Randomly initialise parameters λ,
4: **repeat**
5: Sample M random variables $\hat{\boldsymbol{\epsilon}_i} \sim p(\boldsymbol{\epsilon})$, S a random subset of $\mathbf{X}$ of size M.
6: Calculate the monte carlo estimator of the $ELBO$ w.r.t to λ:

$$\widehat{\Delta\lambda} \leftarrow -\frac{N}{M}\sum_{i\in B}\frac{\partial}{\partial\lambda} logp(\mathbf{y_i}|f^{g(\lambda,\hat{\epsilon_i})}(\mathbf{x_i}) + \frac{\partial}{\partial\lambda}(\alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2)$$

7: Update λ:
$\lambda \leftarrow \lambda + \eta\widehat{\Delta\lambda}$
8: **until** λ has converged.

We can see that Algorithm 2 and 3 are very similar algorithms, with just a couple of differences:

- The regularization term in Algorithm 2 is defined by $KL(q_\lambda(\theta)||p(\theta))$ and on algorithm 3 as $\alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2$
- The scale of $\widehat{\Delta\lambda}$

However, it is possible to deal with both of these differences. For the first difference, depending on the prior distribution $p(\theta)$ we have defined over our parameters, we have the following equation:

$$\frac{\partial}{\partial\lambda}KL(q_\lambda(\theta)||p(\theta)) = \frac{\partial}{\partial\lambda}N(\alpha_1||\mathbf{W_1}||^2 + \alpha_2||\mathbf{W_2}||^2) \tag{3.42}$$

This approximation is called *KL condition* [Gal16] and assuming it, we have that:

$$\frac{\partial}{\partial\lambda}E_{dropout}(\lambda) = \frac{\partial}{\partial\lambda}\frac{1}{N}ELBO(\lambda) \tag{3.43}$$

Where $\mathbf{E}$ is the objective function of our Neural Network trained with *Dropout*.

Therefore, we can see that the optimization of a neural network using *Dropout* is the same as the optimization of a neural network using Variational Inference for specific distributions $\mathbf{q}_\lambda(\theta)$ dictated by the re-parametrization $\mathbf{g}(\lambda, \epsilon)$. Therefore, any neural network trained with *Dropout* is a Bayesian Neural Network and we can assess all the properties of a Bayesian Neural Network in that setting [Gal16].

With that setting, a prediction with our neural network will be a similar procedure as the one seen in Variational Inference:

$$\tilde{q}_\lambda(y^*|x^*) := \frac{1}{t}\sum_{k=1}^{t} p(y^*|x^*, \widehat{\theta}_k, \widehat{\boldsymbol{\epsilon}}_k) \tag{3.44}$$

with $\widehat{\boldsymbol{\epsilon}}_t \sim p(\boldsymbol{\epsilon})$ and $\widehat{\theta}_t = g(\lambda, \boldsymbol{\epsilon}_t)$. This means that we will make t passes over our network and collect t predictions, sampling different *Dropout* parameters ϵ for each new prediction. These t samples will allow our network to measure uncertainty in a similar manner than using the Variational Inference and the Bayesian approach.

3.4 Recurrent Neural Network

Most machine learning models are created with the assumption that every input it receives is independent from each other input in the set. In that manner, a single input must contain all the necessary information to be classified into a classification label. However, this assumption does not hold true for all input types, for example, text data. If we want to predict the next word in the sentence "A dog belongs to the family ", we cannot consider only the word "family" as input, we should look at the whole *sequence* of text to perform a better classification. Although conventional neural networks cannot handle sequential data, there is a modification of neural networks that can, called *Recurrent Neural Network*.

These models have been widely used in tasks such as machine translation [SVL14] and speech recognition [GJ14]. We can see a Recurrent Neural Network as a neural network that take as input not only the input it should classify, but also the past hidden state it computed, as can be seen on Figure 3.6.

On Figure 3.6, we can see that x_t is an input at time step t and $\mathbf{h_t}$ is the computation of the hidden state for the same time step t. Since every step receives the past hidden step as another input, we can see that this hidden state symbolizes the "memory" of our network, since it captures information about the previous time steps in our network. In order to compute such state, Equation 3.45 can be used. Once the hidden state is computed, we can use it to compute the output $\hat{y}_t$ of our network at time step t using Equation 3.46. Furthermore, the parameters $\mathbf{U}, \mathbf{W}, \mathbf{V}$ are shared through the network. For example, the weight matrix $\mathbf{W}$ used in step $t-1$ is the same as the one used in step t.

$$\mathbf{h_t} = \sigma(\mathbf{U}x_t + \mathbf{W}\mathbf{h_{t-1}}) \tag{3.45}$$

$$\hat{\mathbf{y}}_\mathbf{t} = softmax(\mathbf{V}\mathbf{h_t}) \tag{3.46}$$

Although this architecture was designed to handle dependency between inputs, they present issues when they need to represent unbounded distance dependencies in an input [Hoc98]. For example, suppose we have a sequence of 10 different words. In order to calculate the gradients for an objective function at time step 10, we would need to apply the chain rule through the 10 steps of the Recurrent Neural Network. By flowing the gradient through the network through ten steps, we can have two distinct problems. If the norm of the gradients are ≤ 1, than when we reach early time steps of our networks, the gradient can already be zero through all these multiplications, creating the *vanishing gradient problem*. On the other hand, if the norm of our gradients are ≥ 1, than our gradient can explode when flowing through the network. Although we can deal better with exploding gradients [PMB12], dealing with the vanishing problem is harder, since we are essentially losing information. However, there exists a model architecture created with this problem in mind, called *Long Short Term Memory* (LSTM).

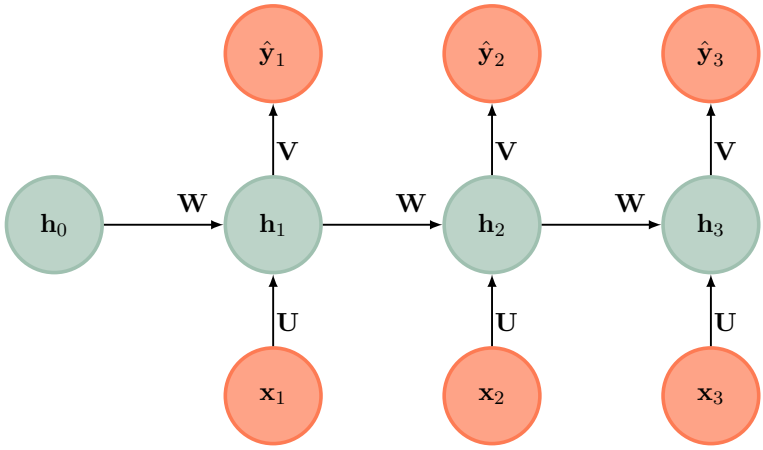

Figure 3.6: *Example of a Recurrent Neural Network. We can see that each step of the network receives the past state* $\mathbf{h_i}$ *to perform its computation.*

3.4.1 Long Short Term Memory

Although we can consider an LSTM a new architecture, this model does not drastically changes the Recurrent Neural Network setting. Instead, it updates how the hidden state $\mathbf{h_t}$ is computed [HS97]. We can see how the hidden state is computed on Image 3.7

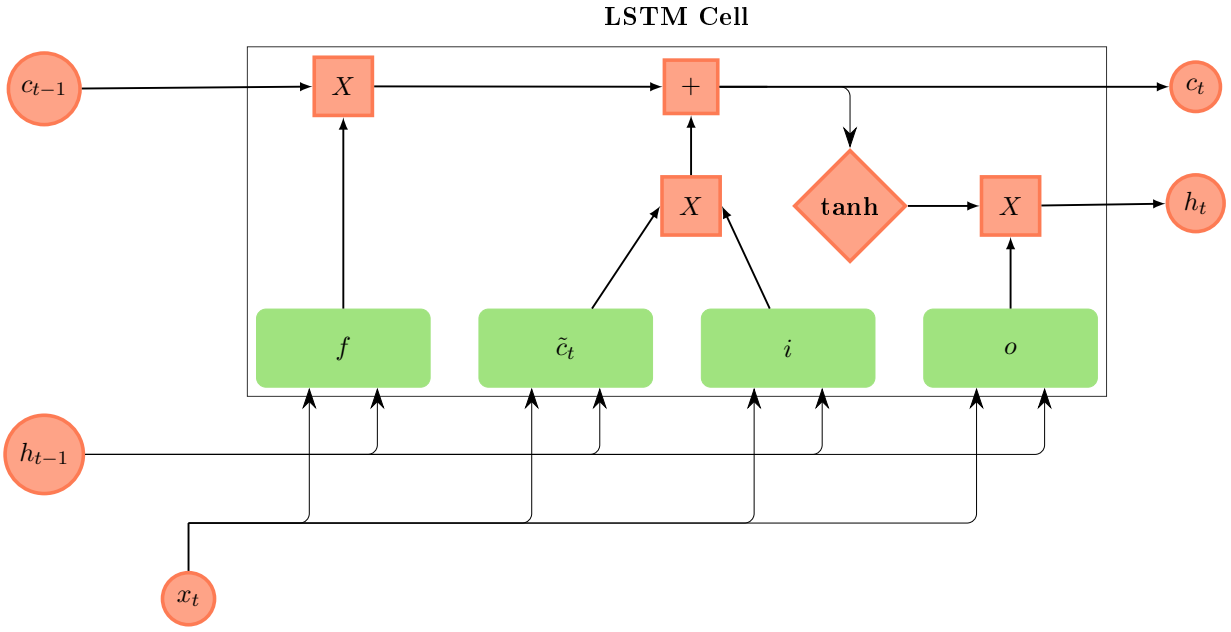

Figure 3.7: *Representation of an LSTM cell. There are three type of nodes in the image. The green rectangles represent gate operations. The square nodes represent mathematical operations, where the "X" nodes represent element-wise multiplication and the "+" represent addition. Finaly, the losangle represents that the input is passed thorough a function, in our case, a the hyperbolic tangent function (tanh). The other nodes represent input and output of the LSTM cell.*

We can also translate Figure 3.7 using mathematical equations, as can be seen in Equation 3.52.

$$f = \sigma(\mathbf{U_f x_t} + \mathbf{W_f h_{t-1}}) \tag{3.47}$$
$$i = \sigma(\mathbf{U_i x_t} + \mathbf{W_i h_{t-1}}) \tag{3.48}$$
$$o = \sigma(\mathbf{U_o x_t} + \mathbf{W_o h_{t-1}}) \tag{3.49}$$
$$\tilde{\mathbf{c}}_\mathbf{t} = tanh(\mathbf{U_g x_t} + \mathbf{W_g h_{t-1}}) \tag{3.50}$$
$$\mathbf{c_t} = \mathbf{f} \odot \mathbf{c_{t-1}} + \mathbf{i} \odot \tilde{\mathbf{c}}_\mathbf{t} \tag{3.51}$$
$$\mathbf{h_t} = \mathbf{o} \odot tanh(\mathbf{c_t}) \tag{3.52}$$

We can see that now in order to compute the hidden state $\mathbf{h_t}$, we need three inputs, $\mathbf{h_{t-1}}, \mathbf{c_{t-1}}$ and $\mathbf{x_t}$. Also, it can be seen that it outputs two values as well, the original hidden state $\mathbf{h_t}$ and $\mathbf{c_t}$. This new input/output $\mathbf{c_t}$, called the **cell** state, will actually be responsible for keeping memory of past information on our network. In order to understand how the cell state does that, it is necessary to understand the **gate** functions, $\mathbf{i}, \mathbf{f}, \mathbf{o}$. These functions are the ones used to modify the cell state. The first gate, $\mathbf{f}$ is called the **forget** gate. It is used to select which information from $\mathbf{c_{t-1}}$ should be forgotten. For example, if we have a sentence such as "The girl was doing her tasks and the boy was doing ...", in order to select the right pronoun we would not need the beginning of the sentence, and the network could forget about it when choosing the next word.

The second gate that needs to be addressed is the *input gate*, $\mathbf{i}$. This gate is used to select which values of the candidate cell state $\tilde{\mathbf{c}}_\mathbf{t}$ should be added to the new cell state c_t. Therefore, looking at how the new cell state is computed, $\mathbf{c_t} = \mathbf{f} \odot \mathbf{c_{t-1}} + \mathbf{i} \odot \tilde{\mathbf{c}}_\mathbf{t}$, we can see that first we decide what to forget from the past cell state, $\mathbf{f} \odot \mathbf{c_{t-1}}$. After that we compute a new candidate cell state and select which information from it should be used, $\mathbf{i} \odot \tilde{\mathbf{c}}_\mathbf{t}$. Finally, we add together both parts to compute our new cell state c_t.

With our cell state computed, we can compute our hidden state h_t using the cell state and the *output gate*, $\mathbf{o}$. This gate is responsible for looking into the cell state and selecting what we should output from it. For example, maybe at time step t we do not need our full memory cell c_t to perform a classification, but just the initial half of it. In that situation, the output gate is responsible for selecting the proper information from $\mathbf{c_t}$.

Using this structure to calculate the hidden state, this model can handle the problem of the vanishing gradient for longer sequences of data. Because of this behavior, this model has been used on a wide range of natural language processing tasks, such as image captioning [KL15] and question answering [XZS16].

3.4.2 Recurrent Neural Networks and Dropout

Recurrent Neural Networks can suffer from overfitting such as any conventional neural network. However, applying dropout to a Recurrent Neural Network is not as straightforward as on other neural networks. By applying *Dropout* over all connections of the network, even the recurrent ones, that compute h_t, we amplify the noise generated by *Dropout* over the recurrent steps, hurting the learning process [BOK+13]. However, using *Dropout* on the non-recurrent connections 3.8 has proven to reduce overfitting on several task [ZSV14].

However, there is another approach when using *Dropout* on a Recurrent Neural Network, which is derived if we use the Variational Inference optimisation objective for this case, called the *Variational Recurrent Neural Network* [GG15a]. Recall that the optimisation function of Variational Inference is the *ELBO* function 3.16. To adapt this function to our Recurrent Neural Network, we

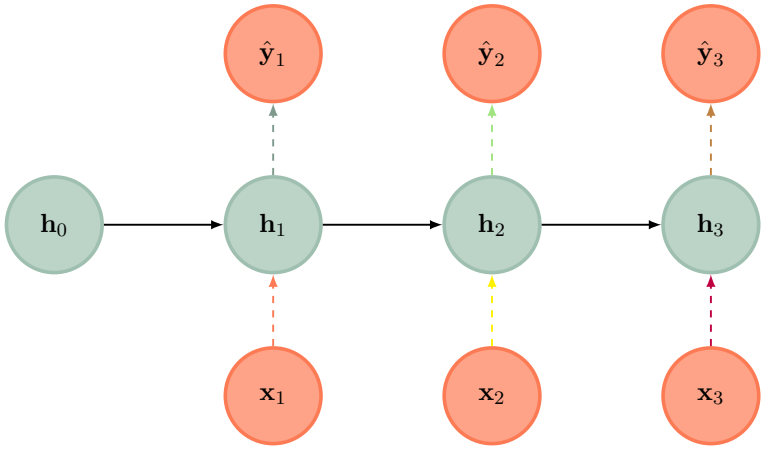

Figure 3.8: *Example of dropout applied to the non-recurrent connections of a Recurrent Neural Network. The dashed lines represent the dropout happening and the color change on each line show that for each time step t, we will sample a different dropout mask.*

can start by evaluating our log-likelihood function:

$$\int q_\lambda(\theta) log(p(y|f_y^\theta(\mathbf{h_T})))d\theta = \int q_\lambda(\theta) log(p(y|f_y^\theta(\mathbf{x_T}, f_h^\theta(\mathbf{x_T}, \mathbf{h_{T-1}}))))d\theta \tag{3.53}$$

$$= \int q_\lambda(\theta) log(p(y|f_y^\theta(f_h^\theta(x_T, f_h^\theta(...f_h^\theta(\mathbf{x_1}, \mathbf{h_0})))))))d\theta \tag{3.54}$$

With a sequence of data $\mathbf{X} = [\mathbf{x_1}, ..., \mathbf{x_T}]$, $\theta = \{\mathbf{U}, \mathbf{W}, \mathbf{V}\}$, f_h^θ the hidden state function 3.45, f_y^θ the output function 3.46 and $\mathbf{h_0} = 0$.

In that setting, we can use Monte Carlo Integration and turn 3.54 into 3.55:

$$\int q_\lambda(\theta) log(p(y|f_y^\theta(\mathbf{h_T})))d\theta \approx log(p(y|f_y^{\widehat{\theta}}(f_h^{\widehat{\theta}}(\mathbf{x_T}, f_h^{\widehat{\theta}}(...f_h^{\widehat{\theta}}(\mathbf{x_1}, \mathbf{h_0})))))) \tag{3.55}$$

With $\widehat{\theta} \sim \mathbf{q}_\lambda(\theta)$. Using Monte Carlo Integration, we can plug 3.55 into the full *ELBO* function, we will arrive at the following equation:

$$ELBO(\lambda) = -\sum_{i=1}^{N} log(p(y|f_y^{\widehat{\theta}_i}(f_h^{\widehat{\theta}_i}(\mathbf{x_{i,T}}, f_h^{\widehat{\theta}_i}(...f_h^{\widehat{\theta}_i}(\mathbf{x_{i,1}}, \mathbf{h_0})))))) + KL(q_\lambda(\theta)||p(\theta)) \tag{3.56}$$

We can observe that for every new sequence $\mathbf{x_i}$, we will sample new samples from our variational distribution, meaning that $\theta_i \sim q_\lambda(\theta)$. However, this parameters will be used at *every* time step in the sequence $\mathbf{x_i}$, as depicted in Figure 3.9.

If we now use *Dropout* to re-parametrize our variational distribution $q_\lambda(\theta)$, we will arrive in a similar objective function demonstrated in 3.3.3. This means that our sampling from $q_\lambda(\theta)$ results

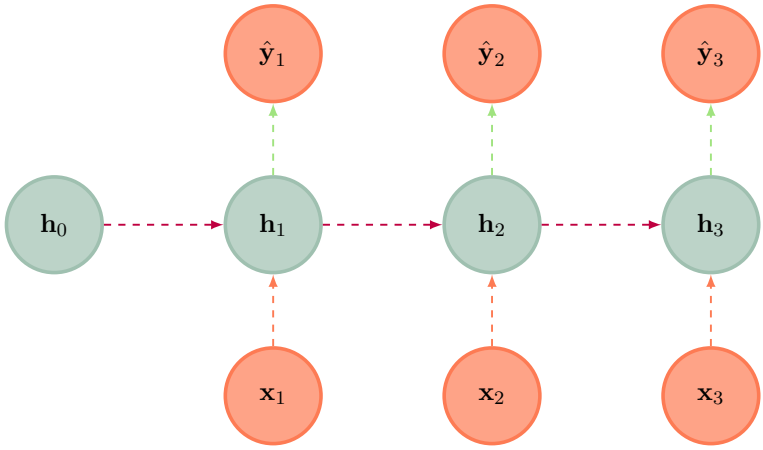

Figure 3.9: *Example of Variational Recurrent Neural Network using dropout. The dashed lines represent the dropout happening and the colored lines show that the dropout mask used for the connections are maintained for every time step of a sequence input*

to applying a *Dropout* mask (randomly zeroing rows) for every θ parameter, $\mathbf{U}, \mathbf{W}$ and $\mathbf{V}$ for every time step $t \leq T$ [GG15a]. This does not change if we are using an LSTM model, the only difference will be that our θ parameters will be $\{\mathbf{U_f}, \mathbf{U_i}, \mathbf{U_o}, \mathbf{U_g}, \mathbf{W_f}, \mathbf{W_i}, \mathbf{W_o}, \mathbf{W_g}, \mathbf{V}\}$.

The *Variational Recurrent Neural Network* is more robust when dealing with over-fitting issues while also achieving better generalization error than using *Dropout* only on the non-recurrent connections on a language generation task [GG15a]. Furthermore, using this technique also allows the measurement of uncertainty of a prediction from our network using the same idea presented in section 3.3.3.

3.5 Active Learning

Active learning(AL) is an iterative technique used to train models when few labeled data is available. It starts by dividing the available data in two distinct groups, labeled and unlabeled data. After that, it trains a model with the labeled data and after evaluating the trained model, it asks an **oracle**, i.e. a domain expert, to annotate the most informative data in the unlabeled data group. After the **oracle** performs this task, the new data is added to the labeled group and a new model is generated. Figure 3.10 illustrates the process of AL.

One of the most researched areas related to AL is how to select the most informative samples to be labeled by the oracle. There are two distinct approaches to select these samples, called *agnostic* and *non-agnostic* AL [SPdC17]. In *agnostic* AL, the samples are selected without using the model information. For example, when we can select the sample randomly, or cluster based strategies to sample the data [Das11]. This is approach is the complete opposite of the *non-agnostic* AL. In that setting, we select the most informative samples by using assessing model information about the unlabeled data points. One of the most used techniques in that setting is by measuring the model uncertainty over the samples. In the case of machine learning, the uncertainty of an input data can be defined as how certain the model is on classifying a given input data. For example, a model can assert that an image belongs to class X with 90% certain or that the image belongs to class Y with 20% of certain. It can be perceived that the model is uncertain about the classification it provided. Therefore, the most informative data for a model should be the ones on which it has higher uncertainty about its prediction.

With that uncertainty measured on the unlabeled data, it is possible to use *Uncertainty Selection Policies* to select the most informative data. One of the most simple ones is the *Least Confident*

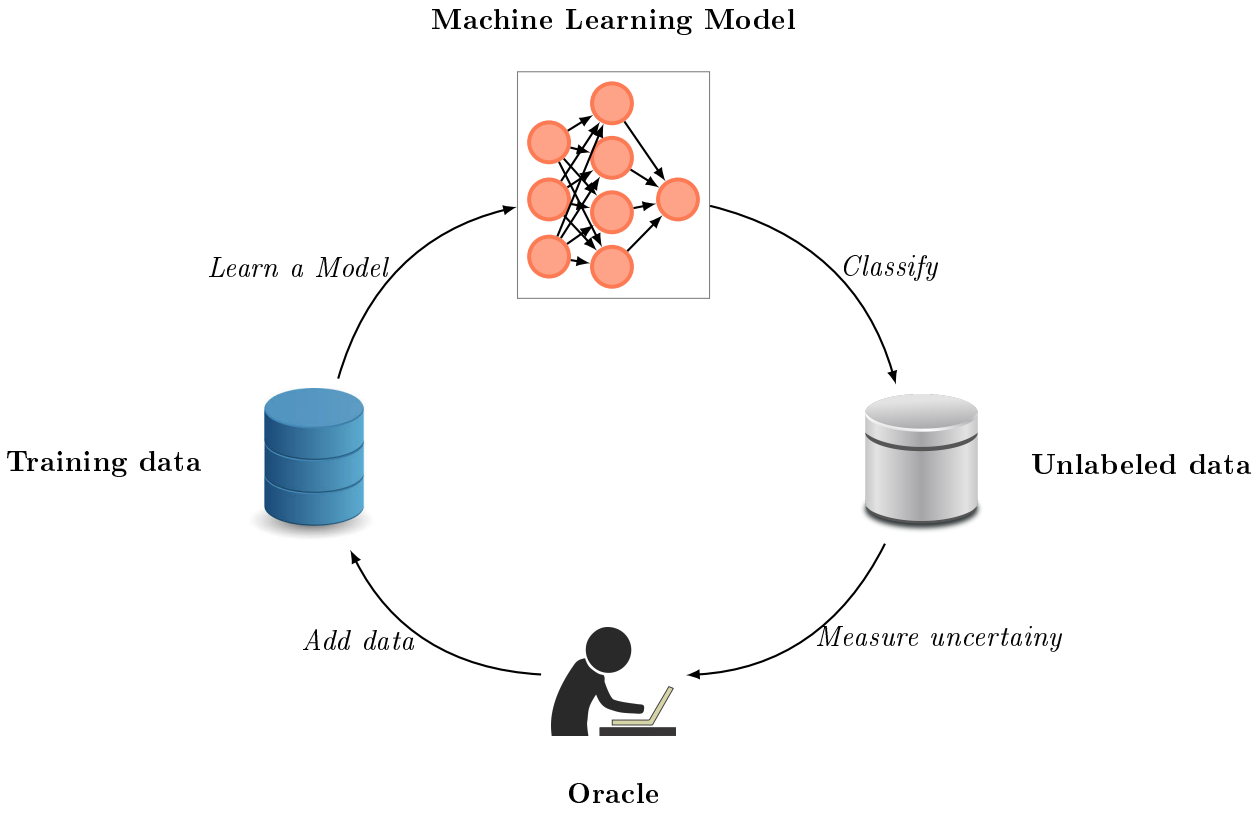

Figure 3.10: *Active Learning cycle. We start with few labeled examples on our training set. We use this training set to train an initial model. Once the model is trained, we use it evaluate the unlabeled set and measure the uncertainty of the model over the example on this set. We select the ones the model is more uncertain about and pass it to an oracle. The oracle labels these examples, which are now added to the training set, allowing the cycle to restart.*

strategy (LC). This policy selects the unlabeled data the model is least confident about [LG94]. This policy can be expressed by the following formula:

$$x^*_{LC} = argmax_x(1 - P_\theta(\hat{y}|\mathbf{x})) \tag{3.57}$$

Where $\hat{y}$ is the class with highest prior probability for the input data x corresponding to the trained model P_θ with θ parameters. In the context of models such as Variational Inference 3.3.2 and *Monte Carlo Dropout* 3.3.3, $P_\theta(\hat{y}|\mathbf{x}))$ would be the number of time the classification $\hat{y}$ was generated by the number of forward passes thorough the network, T, as can be seen in the following equation:

$$prediction_x = argmax_{c=0,1} \sum_{t=1}^{T} 1[y^t = c]$$
$$x^*_{LC} = argmax_x(1 - \frac{prediction_x}{T})$$

In the binary setting, the maximum value achieved by LC is 0.5 and minimum of 0.

A different approach to compute the model's uncertainty about a sample is using *Entropy* (H), which measures the amount of information found on the model's predictive distribution:

$$H(y|\mathbf{x}) = -\sum_{c}^{C} P_\theta(y = c|\mathbf{x}) log P_\theta(y = c|\mathbf{x}) \tag{3.58}$$

By selecting the samples with highest amount of entropy, we would select the sample which the model is "guessing" the prediction.

$$x^*_H = argmax_x(H(y|\mathbf{x})) \tag{3.59}$$

This idea can be easily adapted to Variational models by taking the mean probability for each class in the task, as can be seen in the following equation:

$$\hat{H}(y|\mathbf{x}) = -\sum_{c}^{C}((\frac{1}{T}\sum_{t}^{T} P_{\theta_t}(y = c|\mathbf{x}))log(\frac{1}{T}\sum_{t}^{T} P_{\theta_t}(y = c|\mathbf{x}))) \tag{3.60}$$

Similar to *LC*, the *Entropy* metric has a minimum of 0 when the model make the same prediction for every forward pass.

A third approach on measuring the model's uncertainty is through the use of the *mutual information* metric, which not only consider the predictive distribution $P_\theta(y|\mathbf{x})$, but also the posterior distribution $p(\theta|\mathbf{X})$ over the model's parameters θ:

$$I(y|\mathbf{x}) = H(y|\mathbf{x}) - \mathbb{E}_{p(\theta|\mathbf{X})}[H(y|\mathbf{x}, \theta)] \tag{3.61}$$

To allow the use of multiple forward passes used in Variational methods, we need to turn 3.61 into:

$$\hat{I}(y|\mathbf{x}) = -\sum_{c}^{C}((\frac{1}{T}\sum_{t}^{T} P_{\theta_t}(y = c|\mathbf{x}))log(\frac{1}{T}\sum_{t}^{T} P_{\theta_t}(y = c|\mathbf{x}))) + \frac{1}{T}\sum_{c,t} p(y = c|\mathbf{x}, \theta_t) log p(y = c|\mathbf{x}, \theta_t)$$

$$\hat{I}(y|\mathbf{x}) = \hat{H}(y|\mathbf{x}) - \mathbb{E}_{q_\lambda(\theta)}(\hat{H}(y|\mathbf{x}, \theta))$$

Although all three metrics measure uncertainty, the type of uncertainty they measure can be different. For example, if our model's binary prediction for an input x is $\{1, 0, 1, 0\}$, we have *predictive uncertainty*, where the model is uncertain about the prediction it is generating. However, if our model outputs $\{0.5, 0.5, 0.5, 0.5\}$, we can see that the prediction itself is constant, but the model itself is not confident in the prediction it is generating. In that case we have *model uncertainty*. The

mutual information metric is useful for capturing *model uncertainty*, while the other two metrics are not. However, all three metrics can capture *predictive uncertainty* [Gal16].

Furthermore, we can see that all three metrics address uncertainty in a different manner. Therefore, there is no overall better metric for every task. Comparisons between active learning metrics have already been made [SC08] and mixed results were achieved. This suggests that the metric choice should be based on the application being developed.

3.5.1 Cost-Effective Active Learning

A different approach to AL is the *Cost-Effective Active Learning* (CEAL) technique [WZL+17]. In a standard AL cycle, we use one of our uncertainty measurements to identify the most useful samples from our unlabeled group. The most informative samples are the ones the model has least confidence in its classification. These samples are then passed to an **oracle**, which labels the samples and add them to them labeled group, allowing the AL cycle to restart.

However the CEAL technique adds a new intuition about this idea. Instead of selecting only samples which the model is least confident about, the CEAL also looks at samples where the model is most confident about. For example, suppose we are using the Entropy metric and we are evaluating examples on our unlabeled group. If the Entropy value for a sample is 0.5, we can assume that the model has the highest amount of uncertainty about that example and it should be a useful example to be labeled by the oracle. However, what happens if one of our samples has an Entropy value of 0 ? We can understand that our model is highly confident in the classification of that example. What the CEAL technique proposes is that in each round of our AL cycle, we not only select the sample with higher uncertainty, but we exploit the examples with lowest uncertainty score as well.

To explore the examples with lowest uncertainty scores, the CEAL techniques proposes that we temporally label the examples with lowest uncertainty scores with the labels the model has predicted, which we will call CEAL examples. After that, we add both the CEAL examples and the oracle labeled examples to the labeled group and we train a new model. Once we have finished training the new model, we remove the CEAL examples from our labeled group and we restart the CEAL cycle. This can be better illustrated on Figure 3.11.

Furthermore, in order to implement the CEAL cycle we need to define a threshold for selecting the unlabeled examples the model is more certain about. We consider as the most certain unlabeled examples the ones which have an uncertainty measurement below a given threshold δ.

However, as we iterate over the CEAL cycle, we need to refine our threshold value as our model become more robust in order to guarantee that our auto-labeling process stay reliable [WZL+17]. We can do that with Equation 3.62.

$$\delta = \begin{cases} \delta_0, & \text{if } t = 0. \\ \delta - dr * t, & \text{if } t > 0 \end{cases} \tag{3.62}$$

Where δ_0 is our initial threshold value, t is our current AL iteration and dr is the variable controlling the threshold decay rate.

By using the CEAL mechanism, it is believed that we can achieve some benefits over the common AL cycle. The first one is by allowing better gradient estimates when training our model, since we will be training each AL cycle with at least more data than common AL. The second benefit is that we do not allow outliers to affect our model as much as in the common AL cycle, since our the gradient of our CEAL examples can counterbalance the selected outliers [WZL+17].

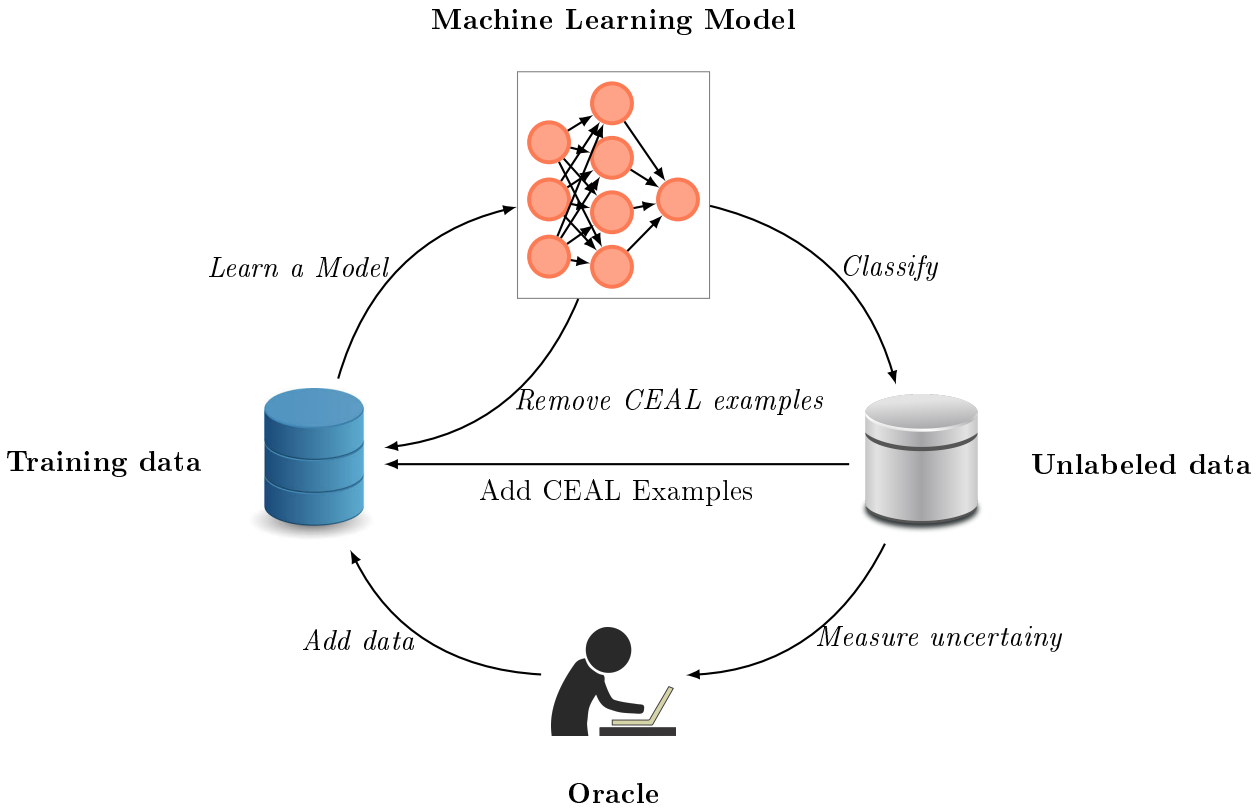

Figure 3.11: *The CEAL cycle. We can see that is almost the same as the AL cycle. However we select the examples that the model are more uncertain and uncertain from the unlabeled group. The examples the model are more uncertain are fed to the oracle, as happens on the AL cycle. However, the unlabeled examples the model is more certain about (The CEAL examples) are assigned the predicted label by the model add added directly to labeled group. Once we train a new model using the data labeled by the oracle and CEAL examples, we remove the CEAL example from the labeled group and we restart the CEAL cycle.*

3.6 Active Learning with Monte Carlo Dropout

With all the necessary information provided in the past sections, we can now introduce how we are going to combine the technique of Active Learning with Deep Learning and Monte Carlo Dropout. First let's remember the AL cycle presented in Figure 3.10. It has four distinct steps:

- **Train model:** We train the model with the current labeled data we have.
- **Select Unlabeled data:** We used the trained model to identify potentially useful unlabeled data points to the model. This is achieved by selecting the unlabeled data which the model is more uncertain on its classification.
- **Label the data**: The selected data is then labeled by an **oracle**. This new data is added into the labeled data group and a new model can now be trained, restarting the AL cycle.

By detailing these steps, we can now explain how we are going to approach each of these tasks in the AL cycle

3.6.1 Training the model

In this step we train a classical DL model with the labeled data we have in the labeled group. This model will be a *LSTM* network trained with both *Dropout* and *L2* regularization on it's

weights, $\{\mathbf{U_f}, \mathbf{U_i}, \mathbf{U_o}, \mathbf{U_g}, \mathbf{W_f}, \mathbf{W_i}, \mathbf{W_o}, \mathbf{W_g}, \mathbf{V}\}$, meaning that it will be a *Variational LSTM Network*.

Here, we will use the classical approach of training a DL model. This means that optimization will be performed using *Stochastic Gradient Descent* and the model will be trained by a fixed number of epoches.

Once this model is trained, we will use it to extract the uncertainty measures from our unlabeled data.

3.6.2 Selecting most informative unlabeled data

In order to select the most informative unlabeled data, we need to perform two distinct steps for **each** unlabeled data point in our unlabeled data group:

- **Generate Monte Carlo Dropout predictions:** For each data point in our unlabeled data group, we need to generate t predictions for it using the *Monte Carlo Dropout* technique.
- **Measure model uncertainty:** For each data point in our unlabeled group, we will use the t associated predictions to it and calculate an uncertainty metric for that data point, such as *Entropy* or *Mutual Information*. Once we have this value for each data point, we can rank these points and select the ones with higher uncertainty estimates. These will be the data points that will be passed to the **oracle**.

These are the main crucial steps we need to select the most informative points from our unlabeled group. However, let's perform them step by step.

First, let's assume that we have 10 unlabeled points on our unlabeled group, where $\mathbf{x_i}$ represent the ith data point in our unlabeled group:

$$\textbf{unlabeled group} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{bmatrix}$$

Now, for each $\mathbf{x_i}$ the unlabeled group we will generate t predictions for it. For example, let's assume we are doing binary classification and in our first round of predictions, we get the following predictions for our unlabeled points:

$$\textbf{unlabeled group} = \begin{bmatrix} x_1 : [0] \\ x_2 : [1] \\ \vdots \\ x_{10} : [1] \end{bmatrix}$$

This predictions were generated by the following procedure: We sample a *Dropout* mask from our Bernoulli distribution and apply it to our trained model. With this mask applied, we will them generate our predictions. In the above example, we can see that our $\mathbf{x_1}$ data point was classified as being of class 0 and our $\mathbf{x_2}$ point was classified as being from class 1. Now we will perform this predictions step again, but we will sample a **new** *Dropout* mask from our Bernoulli distribution.

Therefore, we can assume that we are using a different network to predict our data points than the one used in the first prediction.

With that said, let's assume that our second step generated the following predictions:

$$\textbf{unlabeled group} = \begin{bmatrix} x_1 : [0, 0] \\ x_2 : [1, 0] \\ \vdots \\ x_{10} : [1, 0,] \end{bmatrix}$$

We can now see that the second step has generated different predictions from our first step for both $\mathbf{x_2}$ and $\mathbf{x_{10}}$. We perform this cycle until we have reached the total number of t predictions.

Now let's assume that $t = 6$ and we have obtained the following result in the end:

$$\textbf{unlabeled group} = \begin{bmatrix} x_1 : [0, 0, 0, 0, 0, 0] \\ x_2 : [1, 0, 1, 0, 1, 0] \\ \vdots \\ x_{10} : [1, 0, 1, 1, 1, 1] \end{bmatrix}$$

We can see that for $\mathbf{x_1}$ all our t steps have generated the same prediction. Therefore, we can understand that our network is certain on classifying this data point as being 0. However, that is not the case for $\mathbf{x_2}$. For that data point, 50% of our predictions were 0 and the other 50% were of class 1. Therefore, we now that our model is highly uncertain about classifying this data point. The same occurs for $\mathbf{x_{10}}$, but with a smaller uncertainty that the one seen on $\mathbf{x_2}$, since only one prediction was not classified as 1.

We can use the uncertainty metrics provided in 3.5 to measure this uncertainty for each data point. For each data point, let's use all it's t predictions to calculate the *Least Confident* uncertainty metric for them, as can be seen on Figure 3.12.

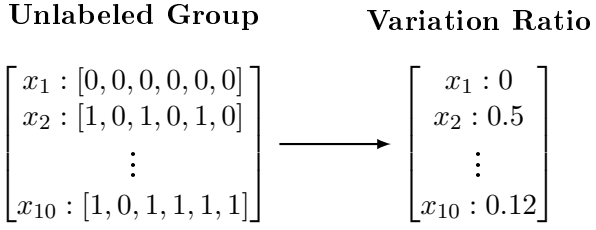

Figure 3.12: *Turning the t predictions generated by the use of Monte Carlo Dropout into a single value using the Variation Ratio uncertainty metric.*

We can now rank all the data points inside the unlabeled group based on the *Least Confident* values, as shown in Figure 3.13.

Now, our most informative samples will be the ones with higher value for the selected uncertainty metric. For example, we could choose the top 2 two data points and we would them select both $\mathbf{x_2}$ and $\mathbf{x_{10}}$ to be labeled by the **oracle**. Additionally, we must emphasize that this procedure **won't change** if we choose to use *Entropy* or *Mutual Information* instead of *Least Confident*.

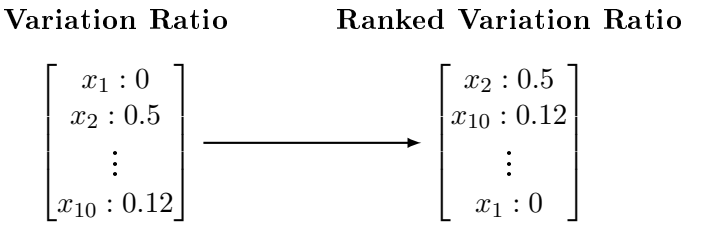

Figure 3.13: *Displaying how we rank our unlabeled data points by their highests values of the corresponding uncertainty metric, in this example, the Variation Ratio metric was used.*

3.6.3 Oracle

After selecting the most informative data points in the previous step, we can now ask for the **Oracle** to label each data point and add them to the labeled group, allowing the AL cycle to restart. The **Oracle** can be either a human expert that can appropriately label the data points or a computer program, which already knows beforehand the appropriate labels for the data points. The computer approach is a simple and cheap approach to perform experiments in AL.

Once the **Oracle** finishes labeling our example and the data is added to labeled group, our AL cycle can restart.

3.6.4 Cost-Effective Active Learning and Monte Carlo Dropout

The process of using CEAL and Monte Carlo Dropout together is the same process presented in Section 3.6.2. This means that no modifications to the process of using the *Monte Carlo Dropout* technique are necessary, we just need to ensure that our AL cycle is modified to attend the CEAL cycle, explained in Section 3.5.1.

Chapter 4

Experiments Design

The primary goal of this research is to evaluate if measuring uncertainty in DL models using the *Monte Carlo Dropout* technique could allow the creation of more accurate models using the AL framework. With this objective in mind, we raised two research questions1.2 and designed a set of experiment to exploit the use of the *Monte Carlo technique*; we had to consider several aspects, such as the datasets used in our models and the approach used to perform the AL experiments.

4.1 Dataset

To train a supervised learning model, it is necessary to have a labeled dataset corresponding to the task that the model aims to solve. This section details the two datasets used in this research.

4.1.1 Large Movie Review Dataset

The first dataset employed in this research is the *Large Movie Review Dataset* (LMRD). It is composed of 50000 labeled movie reviews. 25000 reviews are already separated for training purposes. The remaining 25000 reviews are used for tests. This separation is done primarily to test if our model is generalizing for unseen data, which is way the test the model in the training data. Furthermore, both training and tests sets are balanced as can be seen in Figure 4.1.

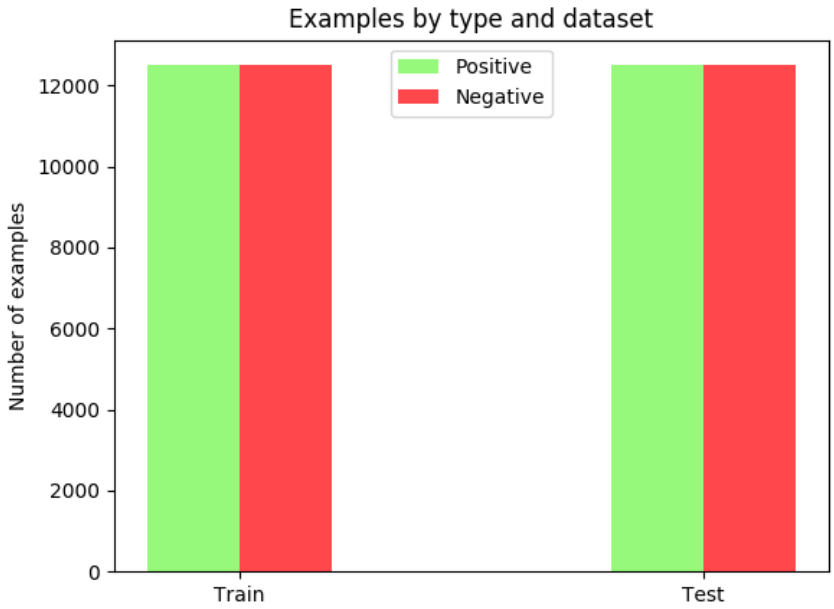

Figure 4.1: *Number of positive and negative reviews for training and test set.*

The number of words per review in the dataset represents another characteristic that should be noticed. Figure 4.2 shows the histogram of the number of words per movie review. We can see that most of the reviews have a size of 250 words. However, we can see that the distribution doesn't follow a Gaussian distribution, meaning that the average will not be a significant characteristic of the data. In order to have a more realistic approach when building our DL model, we have decided to use at most 600 words per movie review. By using this maximum amount of words, we cover more than 90% of the movie reviews in the positive dataset. We can see the similar behavior on the negative movie reviews, as can be seen in Figure 4.3.

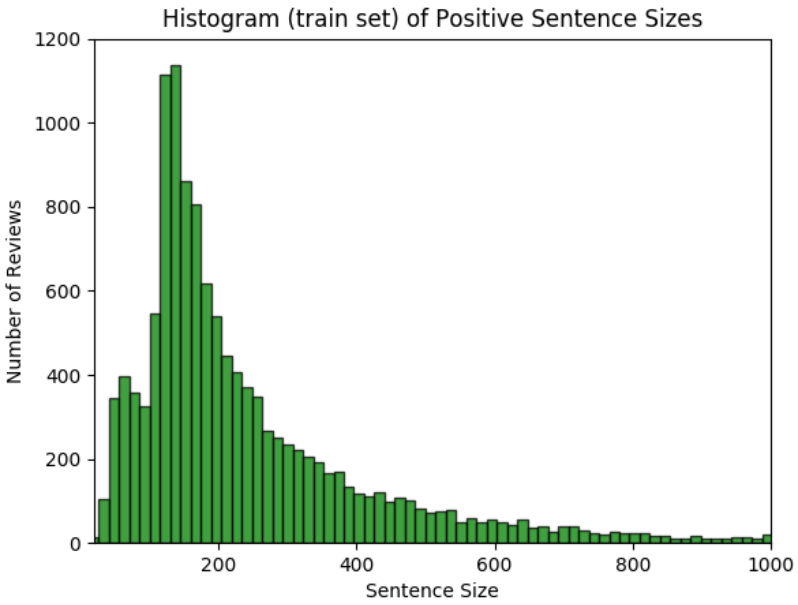

Figure 4.2: *Histogram for the size of the positive movies reviews in the Large Movie Review Dataset for the training set.*

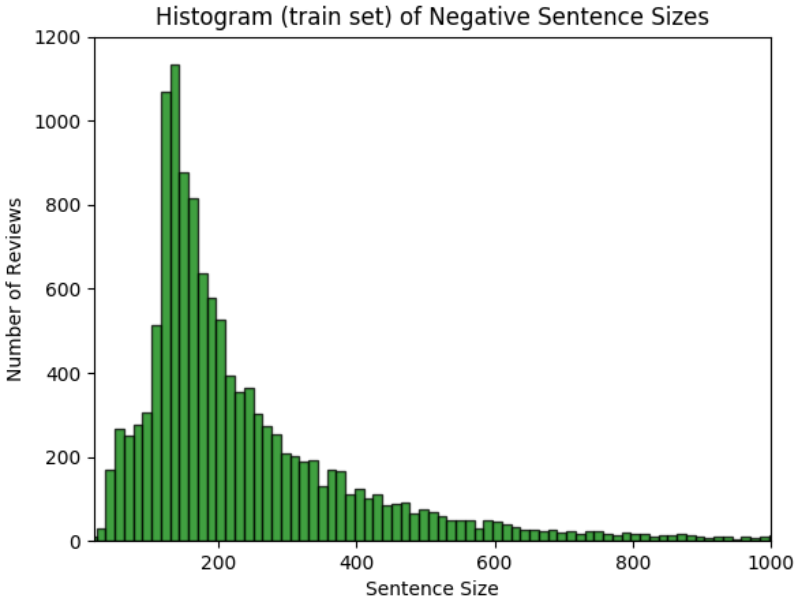

Figure 4.3: *Histogram for the size of the negative movies reviews in the Large Movie Review Dataset for the training set.*

Finally, we observed the number of words in the movie reviews on the test dataset. As can be seen in Figure 4.4 and 4.5, they display a similar behavior with the training dataset. Therefore, we can conclude that by using at most 600 words in the movie reviews, we will cover more than 90% of the reviews on both datasets.

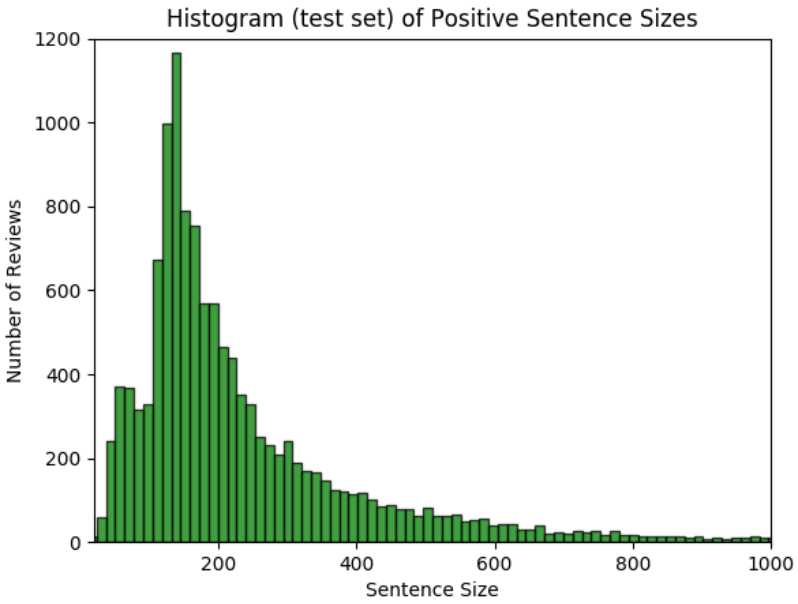

Figure 4.4: *Histogram for the size of the positive movies reviews in the Large Movie Review Dataset for the test set.*

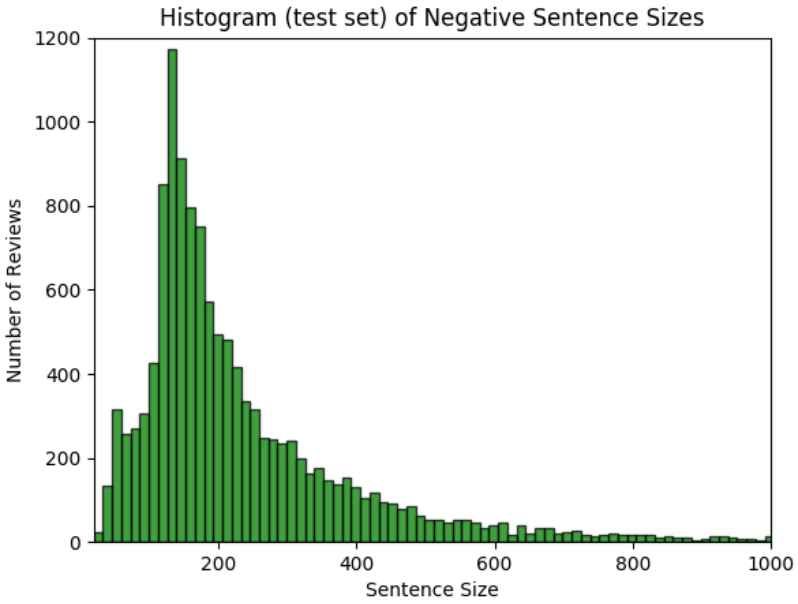

Figure 4.5: *Histrogram for the size of the negative movies reviews in the Large Movie Review Dataset for the test set.*

Furthermore, every review is stored in an ASCII file with raw text data. Since the task is to perform sentiment analysis, no stemming or stop words removal technique will be used, to avoid removing intentional words that describe an "emotion". However, some reviews still contain HTML

markup on the raw data, and a pre-processing stage will be performed to remove such items.

4.1.2 Subjectivity Dataset

The second dataset used in the research is the *Subjectivity Dataset* (SD), first introduced in [PL04]. This dataset contains 10000 sentences divided into two different classes: subjective and objective sentences. The former are sentences extracted from the *rotten tomatoes* movie reviews and the latter are sentences removed from movie plot summaries from the *IMDB* webpage. The identification of subjective and objective sentences can be an important subtask for sentiment analysis [ZWL18].

Similar to the LMRD, this dataset is perfectly balanced as can be seen in Figure 4.6. However, different than the LMRD, it does not have a defined test set, meaning that the test set must be created manually.

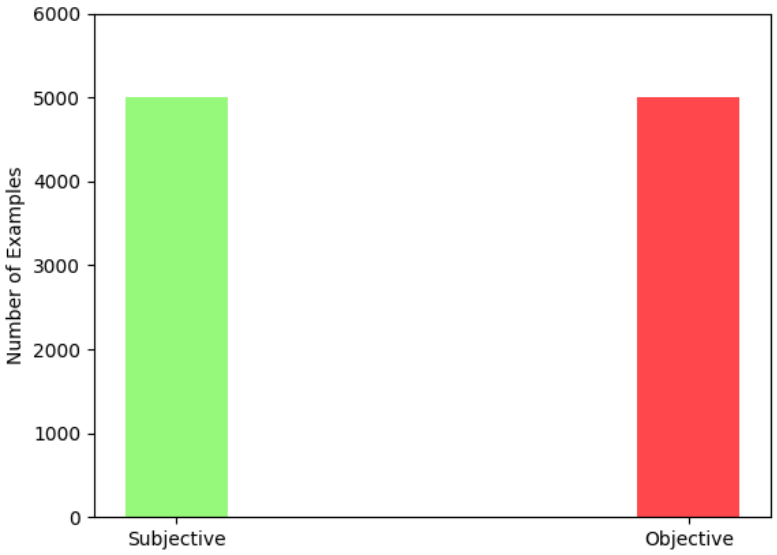

Figure 4.6: *Number of subjective and objective sentences.*

While the LMDR is mainly used to evaluate our model and access the AL cycle on a large model, the SD is used for evaluating our approach on a smaller task, leading also to faster experimentation. One of the main rationales behind this decisions can be seen on the histogram of the Subjective and Objective sentence sizes, as can be seen in Figures 4.7 and 4.8.

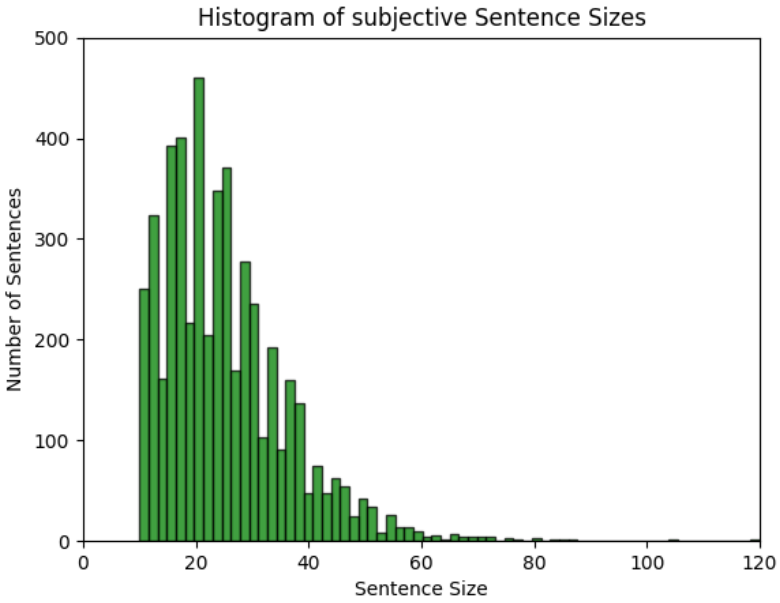

Figure 4.7: *Histogram of word counts for the Subjective sentences.*

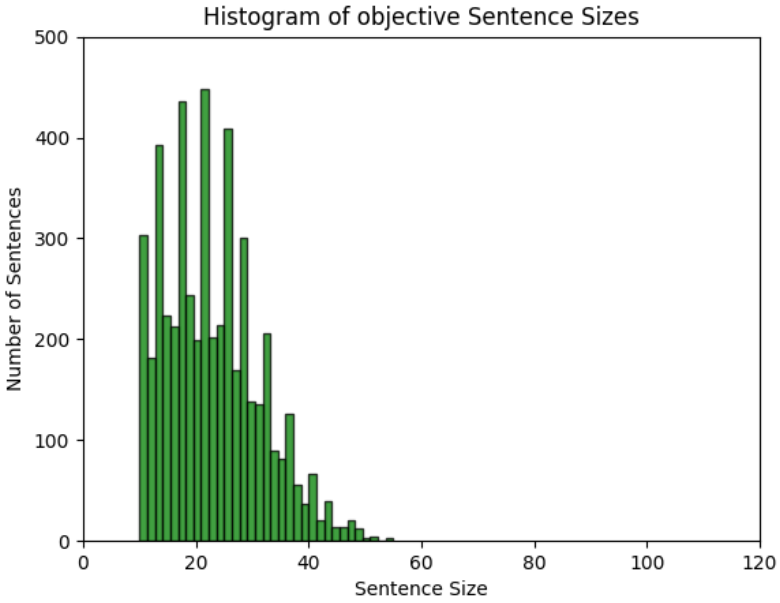

Figure 4.8: *Histogram of word counts for the Objective sentences.*

We can see from Figures 4.7 and 4.8 that the number of words in each sentence is smaller than the ones in the LMRD. Also, for faster experimentation, we have used, at maximum, 20 words per sentence when running the model.

4.1.3 Validation and Test Splits

To perform activities such as hyperparameter search or evaluating our model in the AL cycle, we will need a **validation** and a **test** dataset. LMRD already provides a test dataset with 25000 perfectly balanced examples, as shown in Figure 4.1. For the validation dataset, we have decided

Dataset	Train Set	Validation set	Test set
Large Movie Review Dataset	22500	2500	25000
Subjectivity Dataset	8100	900	1000

Table 4.1: *Number of examples in each set Large Movie Review Dataset and Subjectivity Dataset.*

to use 10% out of the 25000 training examples provided for the LMRD [1].

As for the SD, a test set is not provided beforehand, meaning that we must create it manually. Since the dataset offers 10000 perfectly balanced examples, as shown in Figure 4.6, we have chosen to use 10% of the total data as our test dataset. Furthermore, for the remaining 90% of the data, we managed to use 10% of it for the SD validation and the remaining data as the SD training set. The number of data in each dataset split can be better summarized in Table 4.1.

4.2 Model Definition

After defining our datasets, we can now describe the DL model that will be used to perform our experiments. This section will define the base model that we will use in the AL experiments and how we will validate it using the datasets defined in Section 4.1.

4.2.1 Baseline

In order to validate our model, it is necessary to define a baseline and verify if our model can achieve a similar or superior result than this baseline. For this type of comparison, we have decided to use as our baseline the model proposed on [MDP+11]. This paper not only introduces the LMRD 4.1, but also creates a model that achieves 88.89% accuracy on the dataset. The model used to obtain this accuracy uses word vectors representations and a linear SVM model for classification. We expect that our DL model will achieve a similar or superior accuracy result on the LMRD.

We will not use a baseline for the SD, since this dataset is mainly used for faster experiments. Also, we strongly believe that if our model reaches the baseline accuracy for the LMRD, it will be able to perform on the SD adequately.

4.2.2 Proposed Model

Our proposed DL model is composed of four distinct layers: **Word Embedding**, **LSTM**, **Fully Connected** and a **Softmax** layer; When we receive a sentence, we transform each word in an embedding representation, or a continuous vector. After that, we pass the **entire** sentence through the **LSTM** layer. This layer is a **dynamic** layer, meaning that its size will be generated in accordance to the sentence size. For example, if we have a sentence with 10 words, we will have an LSTM module with 10 layers of computation. After the *LSTM* generates the final vectorial representation, we pass this vector into the **Fully Connected** layer. The final layer, the **Softmax** transform the output from the **Fully Connected** layer into a probability distribution over the two available classes of both of our datasets. We can better visualize this architecture in Figure 4.9

Word Embeddings

The word embeddings layer in our network is a $V \times D$ matrix, where V is the maximum number of words in the vocabulary and D is the size of the word embedding or the dimension of the vector.

[1] The 10% rule was an empirical number we choose. We used a number that would be higher enough to validate our model and not big enough to affect training our model.

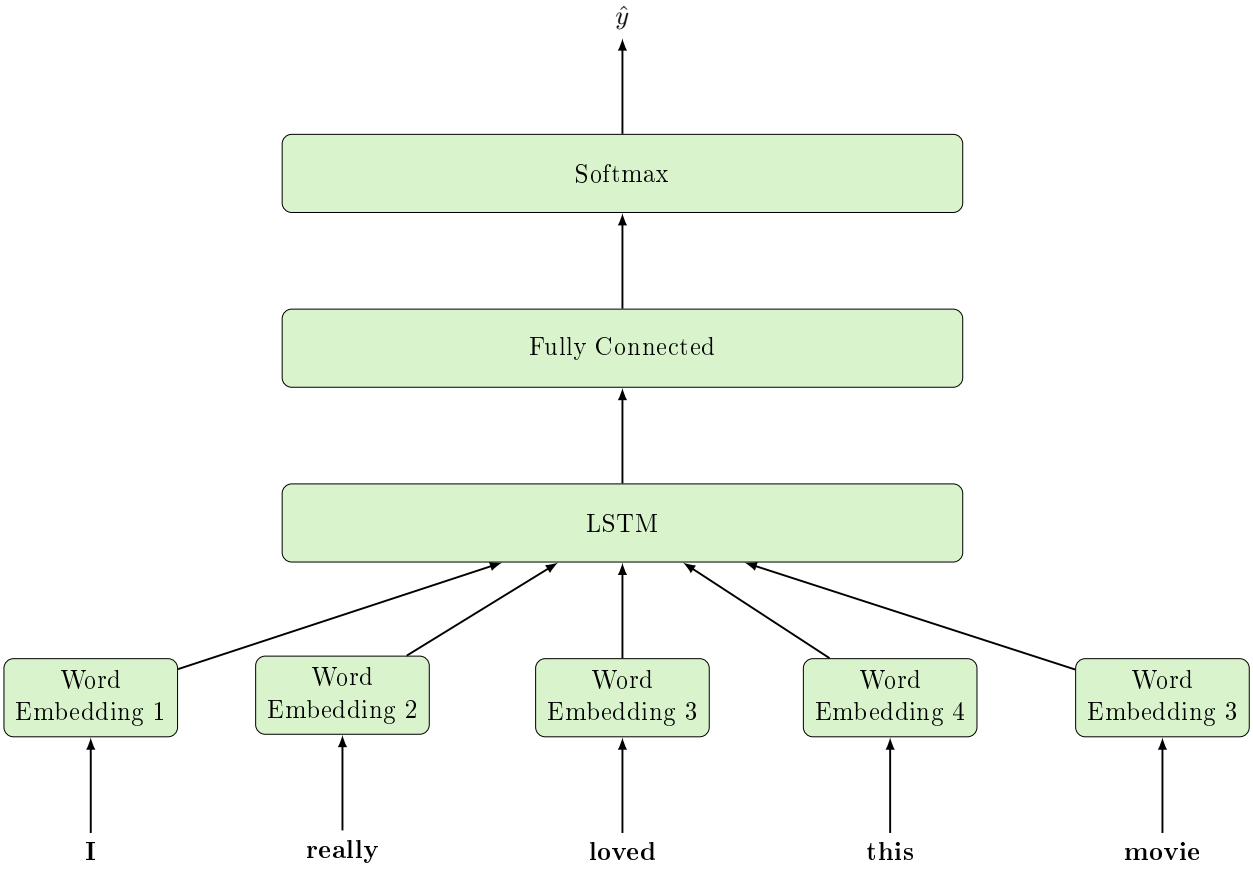

Figure 4.9: *Proposed model for our research. Every word is transformed into an word embedding and feed sequentially to a LSTM layer. Once every word is procesed in the LSTM layer, the final output of this layer is passed through a Fully Connected layer. Finally, the output of the Fully Connected layer is transformed into the predictions of the network using a the Softmax layer.*

This means that each word is mapped into a $1 \times D$ vector before being fed into the LSTM layer. There are several reasons to make this transformation. The first advantage is the embedding representation is memory efficient and allow an easier manipulation by the neural network in contrast with the common sparse representation used in classical Natural Language Processing (NLP) tasks. Secondly, this vectorial representations often carry semantic information, meaning that correlated words tend to have a similar vectorial representation. This can be an useful feature for many NLP tasks [MCCD13].

Usually, there are three distinct approaches when dealing with word vectors. The first one is to randomly initiate this matrix and allow the backpropagation process to further improve this vectors during training. The second one is to use techniques such as the *Skipgram* or the *GloVe* method to create this word vectors, as seen on [MCCD13] and [PSM14]. Normally, these vectors are trained on corpus such as the entire Wikipedia. However, this can be a costly operation, since training these word vectors on an enormous corpus can take a lot of time and computational resources. Because of such problems, the third approach is to use pre-trained word embeddings. These pre-trained word embeddings are typically trained on a large corpora by the institutions that developed the methodology to create these word embeddings, such as the *GloVe* pre-trained word embedding made available by *Stanford* University [2].

In this research, we have chosen to use the third approach to deal with our word embeddings. We adopted the *GloVe* pre-trained word embeddings, which were trained on the Wikipedia corpus. One

[2] https://nlp.stanford.edu/projects/glove/

Hyperparameter	Description
Batch Size	The number of examples that our model will process on a single step of the Stochastic Gradient Descent algorithm.
Number of Epoches	The number of times we will pass through our entire training dataset.
Embedding Matrix Dropout	Probability to drop rows from the embedding matrix.
Word Embedding Dropout	Probability to drop elements from the word embedding vector before feeding it into the LSTM layer.
Variational Dropout	Probability to drop weights from the matrices in the LSTM layer.
LSTM Output Dropout	Probability to drop elements from the LSTM output vector.
L2 Weight Decay	Weight decay regularizer.
LSTM Layer Size	Number of neurons in the LSTM layer.
Fully Connected Layer Size	Number of neurons on the Fully Connected layer.
Learning Rate	Controls how fast we adjust the weights in our model.

Table 4.2: *Hyperparameters used in the model presented in 4.2.2*

Hyperparameter	Value Set	Distribution
Batch Size	$\{x : x \in [32, 64, 128]\}$	Uniform
Number of Epoches	$\{x : x \in [4, 8, 10, 12, 14, 16]\}$	Uniform
Embedding Matrix Dropout	$\{x : 0.2 \leq x < 1\}$	Uniform
Word Embedding Dropout	$\{x : 0.2 \leq x < 1\}$	Uniform
Variational Dropout	$\{x : 0.2 \leq x < 1\}$	Uniform
LSTM Output Dropout	$\{x : 0.2 \leq x < 1\}$	Uniform
L2 Weight Decay	$\{x : 3.1e-5 < x < 1e-3\}$	Exponential
LSTM Layer Size	$\{x : 128 \leq x \geq 1024\}$	Exponential
Fully Connected Layer Size	$\{x : 128 \leq x \geq 1024\}$	Exponential
Learning Rate	$\{x : -4 < x < -0.6\}$	Exponential

Table 4.3: *Hyperparameters value's sets and distributions used for Random Search.*

of the main reasons for this decision is that *Stanford* made available word embedding with different dimension sizes. Because of that, we have selected a dimension size with a trade-off between the robustness of the word embeddings and the training time required to use such embeddings. With that trade-off in mind, we have chosen to use the *GloVe* embeddings with 100-dimensional size, meaning our embedding matrix will have a dimension of $V \times 100$.

Hyperparameter Tuning

An essential task on every DL model consist in selecting the appropriate values for each hyperparameter for the model. Table 4.2 display all the hyperparameters used in our model.

To select the best hyperparameter configurations, we will use the LMRD validation dataset. Each sample hyperparameter configuration will be evaluated on the LMRD dataset.

To generate different hyperparameters configuration, we have chosen to use the *Random Search* technique. This technique works based on defining a distribution for each of our parameters. To create a hyperparameter configuration, we sample each hyperparameter value from its corresponding distribution. Table 4.3 described the values and distributions used for each hyperparameter presented in Table 4.2.

Based on the definitions in Table 4.3 defined, we can them sample each hyperparameter distribution to create a hyperparameter configuration. We will test 60 hyperparameters configurations on our **validation** dataset and select the hyperparameter configuration that provided the highest accuracy value on the **validation** dataset. After that, we will use the selected hyperparameter configuration and use it to evaluate our model in the **test** dataset of the LMRD and verify if our model achieves an accuracy similar or better then the one defined in our baseline Section 4.2.1.

Furthermore, the best hyperparameter configuration found will also be used for our model when we are using it in the Active Learning (AL) experiments.

4.3 Active Learning Experiments

This section describes all the necessary details that we defined to perform our AL experiments. We will explain how we will answer our research questions with our AL models and some fine details of how the experiment will work. Finally, we present an overall visualization of how the whole process will occur.

4.3.1 Active Learning Models

To answer our research questions, different models need to be created. To address the research question **Q1**, it will be necessary to create an AL model with the *Monte Carlo Dropout* technique and other with a random measurement of uncertainty. To address **Q2**, it will be necessary to create another AL models, where the uncertainty measurement is coming from the softmax result from the output layer of our DL model. Therefore, to perform our research, 3 different models will be created:

- **ALU:** Model created using the combination of Active Learning and Uncertainty handling. The uncertainty will be extracted using the *Monte Carlo Dropout* technique.
- **ALS:** Model created by using AL and the softmax output as the uncertainty handling mechanism.
- **LSTM_RANDOM:** Model created by using AL, but selects a random unlabeled data to be annotated by the **oracle**. Therefore, this method does not use any information about the uncertainty handling of the unlabeled data and it is expected to be the lower bound on the experiment (Worst performance the model can reach using AL).

All of the models will be based on the proposed model presented in Section 4.2.2.

It is also worth mentioning that **ALU** will be created with three different selection policies, the Least Confident, Entropy and Mutual information metrics, named **ALU_LC**, **ALU_H** and **ALU_I**.

With the models created, the following comparisons will be performed:

- **Accuracy:** The number of correctly classified examples divided by the total amount of examples. We will measure if any of our **ALU** models outperform our **LSTM_RANDOM** model. The accuracy will be collected at each AL iteration using the **test** set of the LMRD or the SD.
- **Selection policies:** The accuracy for each of the different selection policies tested and verify if the selection strategies can present significant accuracy differences.

- **Uncertainty measurement:** Observe if using the softmax layer as an uncertainty measurement will underperform under a proper uncertainty measurement in a DL environment.

To display the results of such comparisons, a graph containing the measurements will be produced similarly to the approach used on [WZL+17]. This can be seen on Figure 4.10.

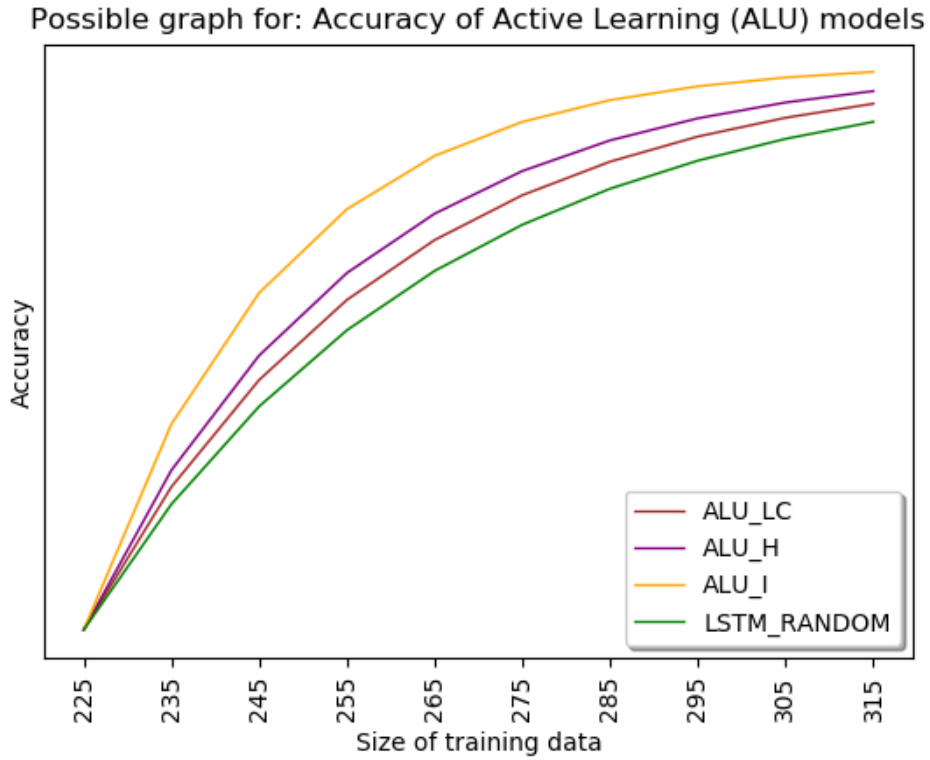

Figure 4.10: *Example of a graph that will be used to compare Monte Carlo Dropout AL with an standard DL model. The lower bound is represented in the graph by the* ***LSTM_RANDOM*** *model. The dotted lines represent the other active learning methods. The values on this graph are only for demonstration purposes, since no experiment has been performed yet*

In Figure 4.10, we can see that the "x" axis represent the amount of labeled data presented to the model and the "y" axis represent the accuracy measure of the model.

The 4.10 will be mainly used to address **Q1**. This means that we will observe if any of the **ALU** models reach a better accuracy than the **LSTM_RANDOM** model with the same amount of data. If any of our **ALU** models achieve a better accuracy curve than our **LSTM_RANDOM**, we will further address **Q1** by testing the CEAL approach 3.5.1. We will verify if CEAL can reach a higher accuracy than using just our **ALU** models alone. To compare that we will create a CEAL model using the metric that provided the best result for our **ALU** model. We will them be able to compare both curves as shown in Figure 4.11

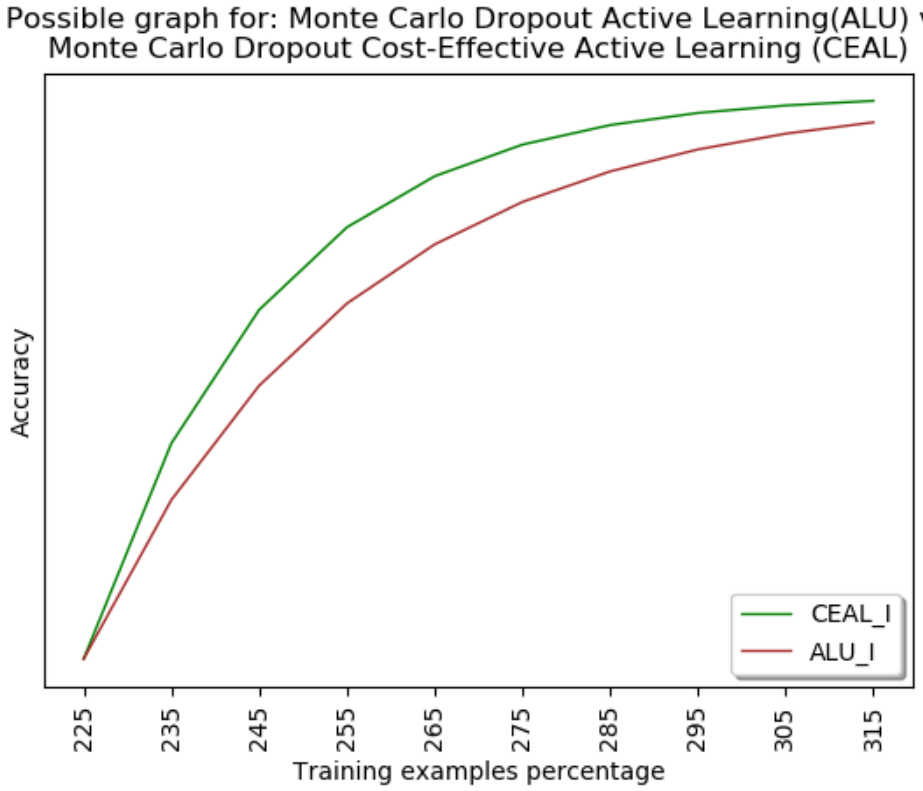

Figure 4.11: *Example of a graph that will be used to compare the best **ALU** model found on 4.10 with our CEAL model. We can see that both **ALU** and **CEAL** use the same uncertainty metric. The values on this graph are only for demonstration purposes, since no experiment has been performed yet*

Again, if any of our **ALU** models achieve a better accuracy curve than our **LSTM_RANDOM**, we will select the uncertainty metric (*Least Confident, Entropy* or *Mutual Information*) that achieved the best accuracy curve and use it to make the comparisons needed to address **Q2**. With this metric defined, we will create an AL model that uses both softmax and the selected uncertainty metric to choose examples from the unlabeled group. Different from the *Monte Carlo Dropout* technique, we use only **one** softmax result from our network to perform our uncertainty computations.

After creating this model, we will evaluate if the softmax underperforms for the AL setting in comparison with our model that uses the *Monte Carlo Dropout* technique. This can be better visualized in Figure 4.12.

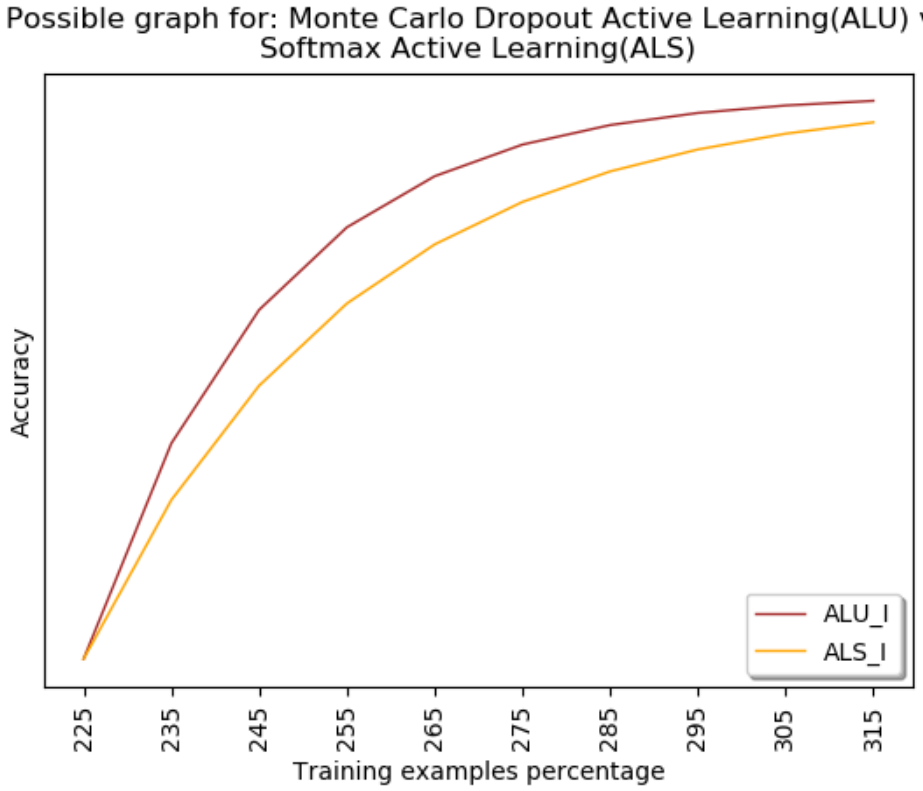

Figure 4.12: *Example of a graph that will be used to compare the best **ALU** model found on 4.10 with the **ALS** model. We can see that both **ALU** and **ALS** use the same uncertainty metric. The values on this graph are only for demonstration purposes, since no experiment has been performed yet*

We can see on Figure 4.12 that similar to 4.10, the "x" axis indicates the number of training data used and the "y" axis, the accuracy of the model.

Additionally, all experiments will be performed 3 times and the results will be averaged when presented in both graphs. The main reason to run all the experiments 3 times is to use different model's parameters at each run. For example, our models will have weight parameters, in each experiment the initial values of these parameters will be different. This will guarantee that our results will not be based on a single random initialization of these parameters. Furthermore, we believe that using a high amount of experiments, we would achieve more sound results, but due to practical considerations, we have chosen to do minimum of 3 rounds of experiments.

4.3.2 Labeled and Unlabeled Group

To perform an AL cycle, we need an initial labeled and unlabeled group. For both LMRD and SD, we will create the labeled and unlabeled group based only on their **training** datasets. For LMRD, we will use 1% of the train set as the initial labeled group and the rest of the train set as the unlabeled group. However, for SD, we have decided to use only 10 examples as our initial labeled group and the rest of the train set as the unlabeled group. This can be better visualized in Table 4.4. We have chosen this number because of the SD dataset is a simpler dataset than the LMRD and we belive that using a big quantity of data initially would allow our first run of active learning to achieve an expressive result earlier, therefore, we have chosen a smaller initial size for our initial labeled group.

Dataset	Labeled Group	Unlabeled Group
Large Movie Review Dataset	225	22275
Subjectivity Dataset	10	8090

Table 4.4: *Number of examples in labeled and unlabeled groups for both the Large Movie Review Dataset and Subjectivity Dataset.*

4.3.3 Unlabeled Group Samples

One of the crucial steps of an AL cycle is to use our model to evaluate the examples in the unlabeled group to select the most informative cases to be labeled by the oracle. Nevertheless, due to some limitations, such as memory size and the time to run our experiments, we cannot evaluate all the unlabeled group at each AL step. Before using our model to evaluate the examples in our unlabeled data group, we will sample **2000** examples from our unlabeled group and select the most informative examples from these **2000** examples. This behavior will happen for both LMRD and SD as well.

4.3.4 Number of Monte Carlo Dropout Passes

For the *Monte Carlo Dropout* technique, we must perform t stochastic passes and collect t different classifications from our model. When selecting the value of t, we must consider a trade-off between the speed of performing *Monte Carlo Dropout* and the accuracy of our uncertainty metric. Based on this trade-off and the t values used in both [GIG17] and [GG15a], we have chosen to use $t = 100$ and then use the collected 100 classifications to calculate our uncertainty metrics defined in 3.5.

4.3.5 CEAL Parameters

For the CEAL cycle, we have chosen to use the values of α as 0.005 and dr as 0.000001. We have used these values based on [WZL+17], but we have not performed any random search for these parameters, as has happened for the hyperparameters of our proposed model.

4.3.6 Overall Approach

To summarize our approach to perform our active learning experiments, we can take a look at Figure 4.13 that describes the whole process we will use to perform our active learning experiment.

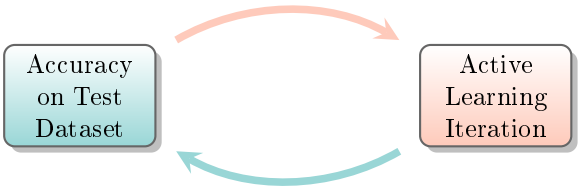

Figure 4.13: *The process to perform our Active Learning experiments.*

We can see on 4.13 that for each iteration of our AL cycle, we will collect the accuracy measurement on the **test** set of the dataset we are currently using (LMRD or SD). This will be the procedure used to build the graphs displayed in Figures 4.10, 4.11 and 4.12. To define our AL iteration, we must perform the following steps:

- **Train model:** We use the labeled group to train our model. This model will be used to evaluate the examples in our unlabeled group.

- **Sample Unlabeled Group:** We sample 2000 examples from the unlabeled group. These 2000 samples will be the ones evaluated by our model.
- **Perform 100 Monte Carlo Dropout Predictions:** We use the *Monte Carlo Dropout* technique to generate 100 predictions for our 2000 unlabeled examples, creating a 2000×100 matrix.
- **Calculate Uncertainty Measurements:** Using our 100 predictions for each of our sampled unlabeled examples, we calculate the model's uncertainty measurement for each of those samples. These measurements can be the metrics of **Variation Ratio, Entropy** and **Mutual Information**.
- **Rank Unlabeled Samples:** using the value from our uncertainty metric, we rank our sampled unlabeled examples in descending order.
- **Select Most Informative Samples:** We select the k items from the ranked unlabeled examples. These examples are the ones considered the most informative ones.
- **Oracle labelling:** The oracle will them label our k most informative unlabeled samples.
- **Add examples to labeled group:** Once the oracle has finished labelling our k most informative examples, we can add these examples to our labeled group and restart our cycle.

This process can be fully visualized in Figure 4.14. It must be said that at the end of each AL iteration we will have a **new** model to evaluate the test set of one of our datasets.

This procedure is the same one used for our AL experiments when using both random and softmax approach. For our random approach, it is not necessary to perform any *t Monte Carlo Dropout* predictions and we don't need to any uncertainty measurement on the unlabeled samples. This means that we will rank our unlabeled examples **randomly**. For the softmax AL, we do not need the *t Monte Carlo Dropout* predictions, but instead, we collect a single prediction from the network and then calculate the uncertainty measurements. All of the other steps displayed in Figure 4.14 remain the same for both random and softmax Active Learning.

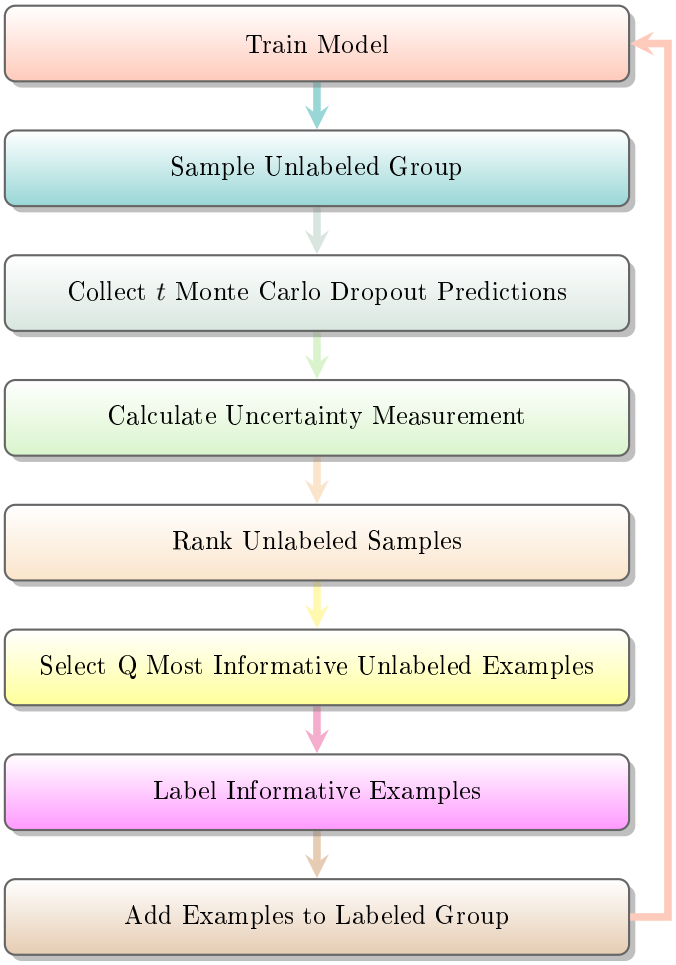

Figure 4.14: *The necessary steps to perform an iteration of our Active Learning cycle using the Monte Carlo Dropout technique.*

Chapter 5

Experimental Evaluation

This chapter covers all the investigations performed on this work. They mirror the details provided in Chapter 4. The first part of this chapter specifies the technologies employed to implement the code used in this work. Following that, we will present the comparison of our model with the baseline model described in Section 4.2.1. Finally, we will show our Active Learning (AL) experiments and explain the iterative process we have used to perform our experiments.

5.1 Technologies Used

To implement the model proposed in Section 4.2.2 we have used the Python programming language version 3.6 together with TensorFlow version 1.7.0 [1]. TensorFlow was used to create our Deep Learning model and the *Extract-Transform-Load* (ETL) pipeline, used to feed data into the DL model. It is also essential to detail all the machines used to conduct the experiments in this thesis. In total, three distinct computers were used to perform the experiments in our work, Tables 5.1, 5.2 and 5.3 describe the components of each of these machines.

Furthermore, all the code used in this research is open source and can be found on GitHub [2].

5.2 Baseline Evaluation

Before proceeding with our Active Learning experiments, it was necessary to evaluate if our proposed model was adequate to solve the sentiment analysis task we have proposed.

Our first step was to implement the model and perform a random search on the model hyperparameters, as described in Section 4.2.2. We have conducted 60 random search steps, evaluating 60 different hyperparameters configurations in the LMRD validation set. The hyperparameter configuration that achieved the best result (91.9%) on the validation set is described in Table 5.4.

Component name	Description
Distribution	Ubuntu 18.04
Central Processing Unit (CPU)	Intel(R) Core(TM) i7-6900K CPU 3.20GHz
Random-Access Memory (RAM)	64Gb DIMM DDR4
Graphics Processing Unit (GPU)	GeForce GTX 1070
Graphics Processing Unit (GPU) Memory	8Gb

Table 5.1: *Configuration for the first machine used in this research experiments.*

[1] https://www.tensorflow.org/
[2] https://github.com/LIAMF-USP/deep_active_learning

Component name	Description
Distribution	Ubuntu 18.04
Central Processing Unit (CPU)	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
Random-Access Memory (RAM)	64Gb DIMM DDR4 Synchronous 2133 MHz
Graphics Processing Unit (GPU)	GeForce GTX 1060
Graphics Processing Unit (GPU) Memory	6Gb

Table 5.2: *Configuration for the second machine used in this research experiments.*

Component name	Description
Distribution	Debian 9.5
Central Processing Unit (CPU)	Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
Random-Access Memory (RAM)	32Gb DIMM DDR4
Graphics Processing Unit (GPU)	GeForce GTX TITAN X
Graphics Processing Unit (GPU) Memory	12Gb

Table 5.3: *Configuration for the third machine used in this research experiments.*

After that we evaluated our proposed model with the hyperparameters described in Table 5.4 using the LMRD test set. We have achieved an accuracy of 88.9%. Since our baseline model achieved an accuracy of 88.89%, we can understand that our model has reached a similar result than our baseline, being fit for the task of sentiment analysis provided by the LMRD.

Furthermore, using the same hyperparameter configuration provided by Table 5.4, we have found an accuracy of 90.6% for or SD validation set and an accuracy of 88.5% for our SD test set.

5.3 Active Learning Experiments Evaluation

This section describes all the Active Learning experiments performed in this work. We adopted an iterative approach when performing our experiments. First, we identified if there is any difference between the Monte Carlo Dropout models (ALU models) and our random (LSTM_RANDOM) model; as a result, our first iterations were then focused on discovering if the curves differ. If we reached that point, we would be able to further address both of our research questions.

Additionally, the reason for limiting the scope of our experiments at the beginning of this work

Hyperparameter	Description
Batch Size	64
Number of epochs	16
Embedding Matrix Dropout	0.267
Word Embedding Dropout	0.568
Variational Dropout	0.855
LSTM Output Dropout	0.694
L2 Weight Decay	0.0003097047
LSTM Layer Size	285
Fully Connected Layer Size	285
Learning Rate	0.00410

Table 5.4: *Best hyperparameter configuration found using the validation set of the LMRD*

Parameter	Description
Unlabeled Data Queries (Q)	The number of example we will select from the unlabeled group to be labeled by the oracle.
Number of epochs (EPO)	At each AL cycle, we will train our model for a given number of epochs. This variable defines this quantity.
Dropout Values (DROP)	The dropout probability for the weights in our network.
Number of Active Learning Cycles (NC)	The number of AL cycles we have run for a given experiment.

Table 5.5: *Parameters that will be address in the Active Learning Experiments Evaluation*

is due to the large amount of time taken to run a complete AL experiment. Since at the beginning of this research we were testing the initial hypothesis, we believe it was not a sound decision to make experiments with all the available models (ALU_LC, ALU_E and ALU_I), since we did not know how much time it took to perform a full AL experiment cycle.

With that decision in mind, the rest of this Section describes all the iterations used for our Active Learning Experiments, addressing the results in each iteration and what has changed from the last iteration.

5.3.1 Notation

Before addressing the AL experimental iterations, we must define the notation adopted through this section. We will talk about four different parameters concerning our AL experiments. These parameters can be seen on Table 5.5.

5.3.2 Active Learning Experiments - Iteration 1

In our first iteration, we have focused on generating accuracy curves for two different models, the *Monte Carlo Dropout* model using the **Mutual Information** metric (ALU_I) and the random model (LSTM_RANDOM). We have chosen the **ALU_I** model beforehand because, in the works of [GIG17] and [SYL+17], the Mutual Information metric was the one that achieved the best result. Therefore, we believe it was the safest metric to test first.

We have also decided to use the LMRD dataset for this experiment, with the assumption that if there were a significant difference in the model's curves for this dataset, the same result would follow for the SD as well.

In this initial iteration, we tested if using the same hyperparameter displayed in Table 5.4 we would achieve different curves for our models. Table 5.6 presents the parameters configuration used for our first experiment.

Parameter	Value
Unlabeled Data Queries (Q)	50
Number of epochs (EPO)	16
Number of Active Learning Cycles (NC)	50

Table 5.6: *Parameters used for our first iteration. Dropout Values (DROP) was omitted because in the first iteration, we have not created a single value for all our dropout masks. Instead we have used the individual dropout values found on Table 5.4*

After running the experiments, the results obtained did not follow our assumptions. Not only

both of our curves do not differ by much, but the model, using almost 2700 samples could not even achieve an accuracy of 60%. After that results, we decided to make a few adjustments to our experiments.

5.3.3 Active Learning Experiments - Iteration 2

After the results found on Iteration 01, we decided to look back in the papers and code that created and applied the technique of *Monte Carlo Dropout* into DL tasks. We have perceived that both [Gal16] and [GIG17] used a distinct concept we were not exploring. At each AL cycle, they have trained the DL model for a high number of epochs. For example, in [GIG17], the researchers trained their convolutional model for over 50 epochs for the task of digit recognition on the MNIST dataset, a rather simple problem that doesn't require that number of epochs for convergence. One of the rationales found on [Gal16] for this behavior is the fact that as we have a high number of epochs, the number of weight configurations that the model will see will increase. This makes the network more robust when we sample different weights for predicting unseen data.

Based on that finding, we have decided to increase the number of epochs we train a model in each AL cycle from 16 to 200, as can be seen on Table 5.7.

Parameter	Value
Unlabeled Data Queries (Q)	50
Number of epochs (EPO)	200
Number of Active Learning Cycles (NC)	50

Table 5.7: *Parameters used for our second iteration. Dropout Values (DROP) was omitted because in the first iteration, we have not created a single value for all our dropout masks. Instead we have used the individual dropout values found on Table 5.4*

After making that modification, we did another experiment using the LMRD, obtaining the graph displayed in Figure 5.1

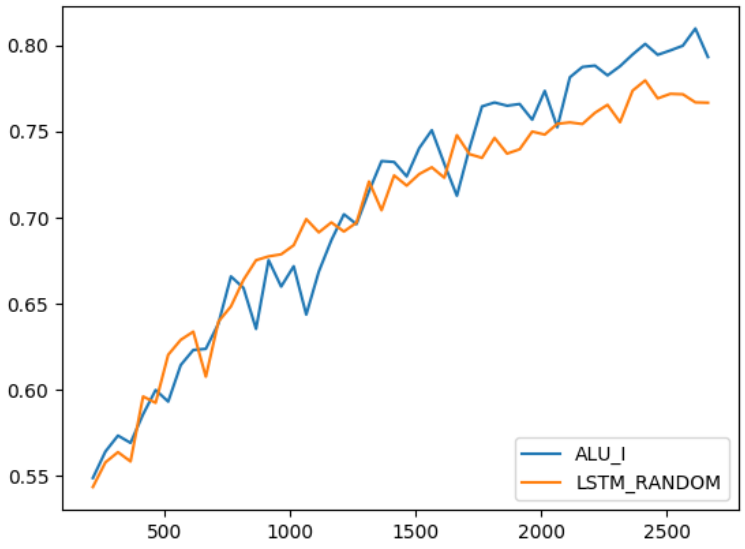

Figure 5.1: *Comparison between our ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=200, Q=50 and NC=50*

We can see from Figure 5.1 that we have different curves now. After 1600 examples, we can clearly see that our **ALU_I** model starts to distance further away from our **LSTM_RANDOM** model. We have observed a positive result towards the *Monte Carlo Dropout* technique. However, by using now EPO=200 and Q=50, we have taken too much time to perform an experiment, which is something we were not comfortable with.

5.3.4 Active Learning Experiments - Iteration 3

Although we had a positive result in Iteration 2, we had the problem that our experiments took to much time to run. We explored the effects of both [GIG17] and [Gal16] and found two other aspects in the code they have provided for their findings. The first aspect is that, at each AL cycle, they selected from the unlabeled group 10 examples for the oracle to label. Therefore, they have significantly reduced the number of data points added, at each cycle, to the labeled dataset. The second aspect we have found is that they have used a value of 0.5 for every *Dropout* probability in their network. This is very different from the approach we have used since we are using different probabilities for different parts of our network.

Based on the results of Iteration 2 and these new findings, we decided to update our AL parameters, as can be seen on Table 5.8. Additionally, we have also expanded the value of our NC value to allow for a greater quantity of data to be used by our models, since now every AL cycle will only add 10 new data points to the labeled group, instead of 50.

Parameter	Value
Unlabeled Data Queries (Q)	10
Number of epochs (EPO)	150
Dropout Values (DROP)	0.5
Number of Active Learning Cycles (NC)	100

Table 5.8: *Active Learning Parameters used for our third iteration.*

Using these new configurations, we have reached the result displayed in Figure 5.2. We can now see a permanent trend were our **ALU_I** model has achieved a better accuracy with the same amount of data over our **LSTM_RANDOM** model.

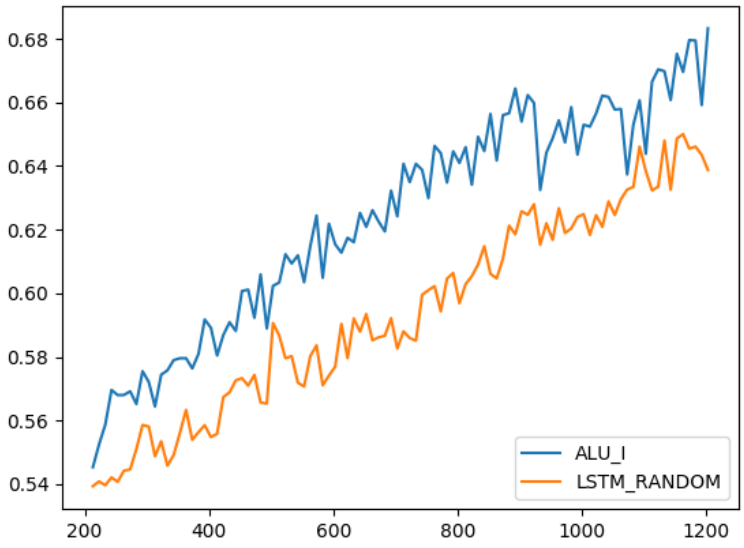

Figure 5.2: *Comparison between our ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

To further validate the result achieved, we have decided to plot the same graph displaying the standard deviation around each point to verify if there is still an overlap between the curves, as can be seen in Figure 5.3.

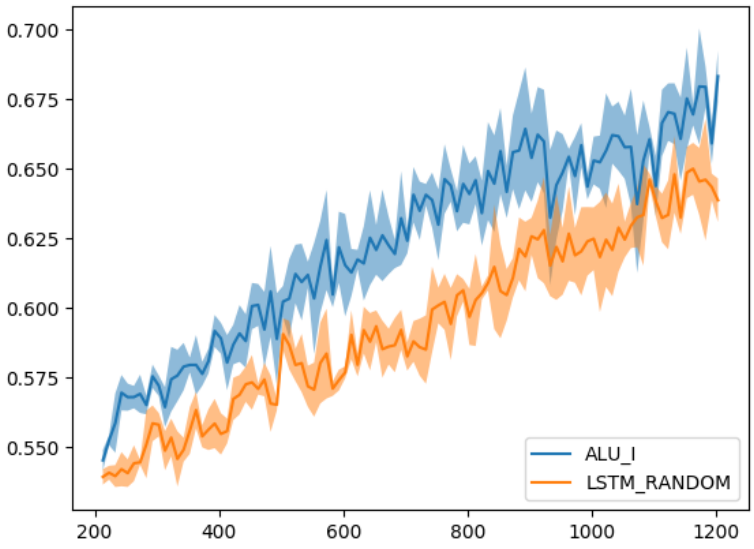

Figure 5.3: *Comparison between our ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5 using the standard deviation around each point.*

We can see that there is no overlap between the curves. Based on that result, we believe we have found a significant difference between the curves. Therefore, we have decided to make the same experiments using different uncertainty metrics, Entropy and Least Confident.

5.3.5 Active Learning Experiments - Iteration 4

After the positive results in Iteration 3, we have decided to make our complete AL experiments and verify the accuracy curves for both of our Entropy and Least Confident metrics as well. In this iteration, we decided not to change any of our AL parameters, meaning that they remain the same ones displayed in Table 5.8.

Therefore, in this iteration, we created two new models:

- **ALU_E:** *Monte Carlo Dropout* together with the Entropy metric.
- **ALU_LC** *Monte Carlo Dropout* together with the Least Confident metric.

After running the AL experiments for these new models, we have achieved the results displayed in Figures 5.4 and 5.5.

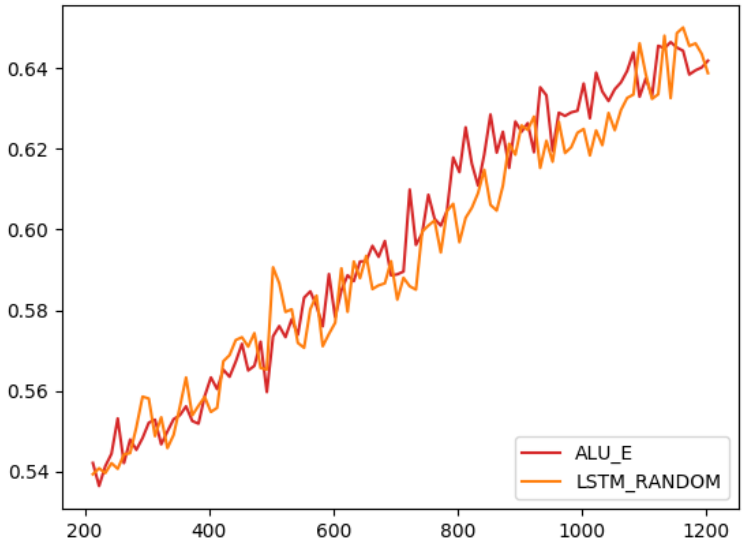

Figure 5.4: *Comparison between our ALU_E and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

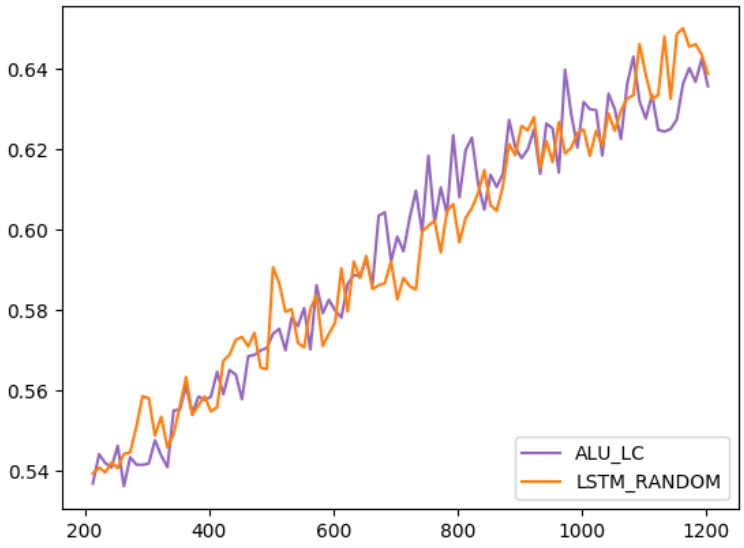

Figure 5.5: *Comparison between our ALU_LC and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

We can see from both Figures 5.4 and 5.5 that the curves don't differ much. This is the complete opposite result we have achieved in Iteration 3, especially when we put all the curves in the same graph, as can be seen in Figure 5.6.

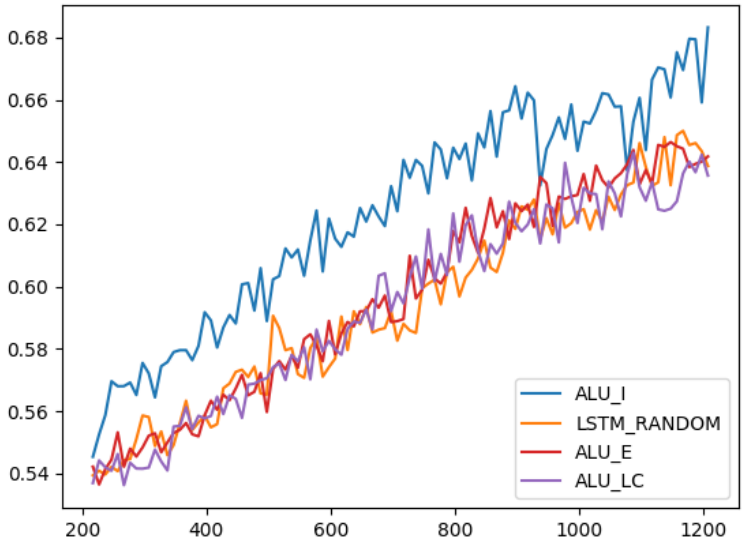

Figure 5.6: *Comparison between our ALU_I, ALU_E, ALU_LC and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

We can see that the **ALU_E** and **ALU_LC** behave almost precisely as the **LSTM_RANDOM**, meaning that measuring the uncertainty with their proposed metrics, have provided no benefit for this task. Only the **ALU_I** model has provided a significantly better accuracy curve.

Finally, we have also decided to combine our **ALU_I** model with CEAL cycle and see if we

would achieve a better result for this task. We have named the model **CEAL_I**. After running our experiments, we have found the result displayed in Figure 5.7.

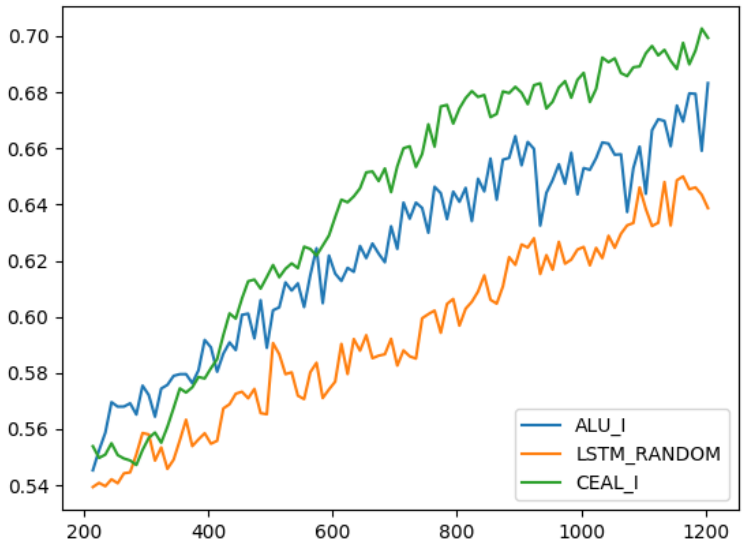

Figure 5.7: *Comparison between our CEAL_I, ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

We can see in Figure 5.7 that our **CEAL_I** model has achieved a better accuracy curve than **ALU_I** and **LSTM_RANDOM**. To further validate this result, we decided to plot the same graphs, but displaying the standard deviation too, in a similar manner than the graph shown in Figure 5.3. This new graph can be seen in Figure 5.8.

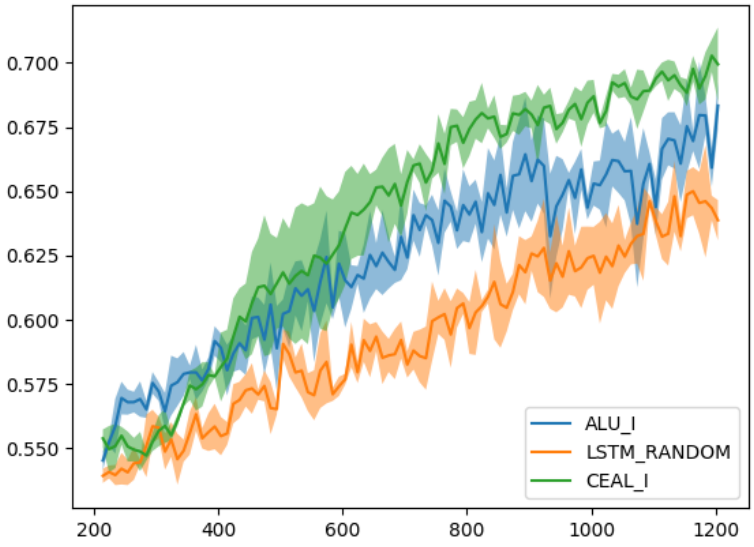

Figure 5.8: *Comparison between our CEAL_I, ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5 using the standard deviation around each point.*

We can see from Figure 5.8 that at around 600 training examples, the **CEAL_I** model has a

very high standard deviation compared to both **ALU_I** and **LSTM_RANDOM**. However, after 800 training examples, the **CEAL_I** achieves better results than both of the other models. One of the possible reasons for this behavior is that, at the beginning of the CEAL cycle, its auto-labeling step may not be too efficient yet. As new data is added to the labeled group, this mechanism may improve further and further, allowing the model to perform better.

With that results in hand, we have to perform similar comparisons, but now, comparing our best **ALU** model with a model that measures uncertainty using the softmax function.

5.3.6 Active Learning Experiments - Iteration 5

In this Iteration, we will compare our **ALU_I** (best model found in Iteration 4, without using CEAL) with a model that uses softmax instead of the *Monte Carlo Dropout* to handle uncertainty. Therefore, we will create a softmax model that uses the same metric as our **ALU_I** model, the Mutual Information, the **ALS_I** model. Additionally, we will use the same AL parameters displayed in Table 5.8.

After creating this model and performing the AL experiments, we have reached the results displayed in Figure 5.9.

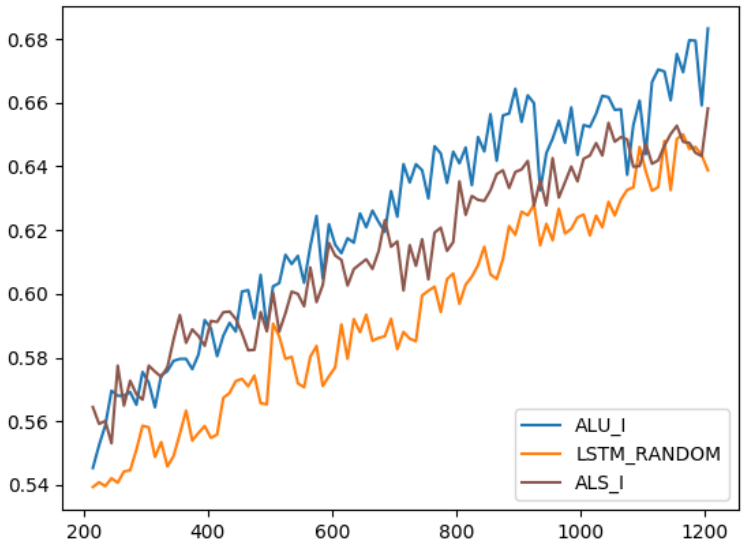

Figure 5.9: *Comparison between our ALU_I, ALS_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

We can see in Figure 5.9 that our **ALS_I** model performs better than our **LSTM_RANDOM**, but worse than our **ALU_I** model. We can further visualize this by plotting the standard deviation for the **ALS_I** model, as can be seen in Figure 5.10.

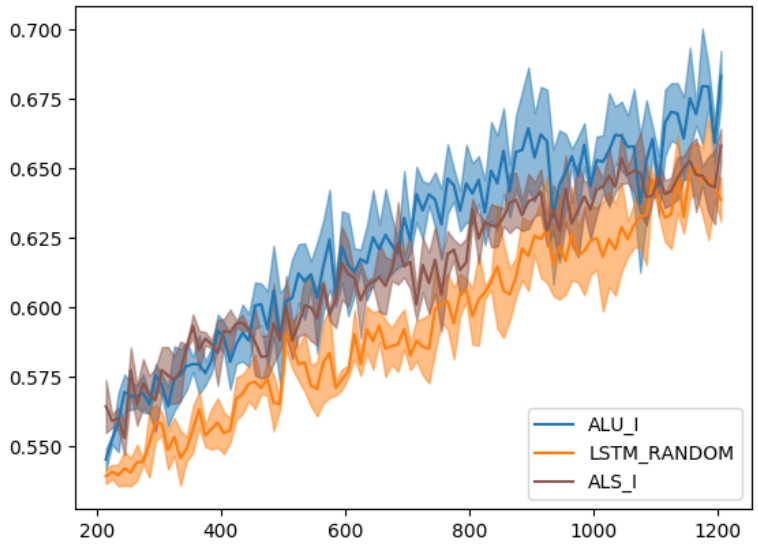

Figure 5.10: *Comparison between our ALS_I, ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5 using the standard deviation around each point.*

We can see that the **ALS_I** has a lower standard deviation that out**ALU_I** curve, but still, they do not overlap. Therefore, for the LMRD, we can understand that modelling the uncertainty with the *Monte Carlo Dropout* and the Mutual Information metric has given positive results for achieving better accuracy results with the same amount of data than the other approaches.

Based on these results on the LMRD, we have decided to apply the same experiments in the SD. In that case, we would have significantly smaller model than the one used for the LMRD. We would them verify if the results found here also apply in a smaller dataset and model.

5.3.7 Active Learning Experiments - Iteration 6

In this iteration, we have decided to perform the same experiments we have performed for our LMRD using the SD instead. Since the model will be smaller and the number of data is also smaller than the LMRD, we have decided to increase our Number of Cycles (NC) from 150 to 400, in order to reach a more robust accuracy curve. Besides that change, the AL parameters will remain the same ones established in Table 5.8.

We performed the experiments for our four models: **ALU_I, ALU_E, ALU_LC** and **LSTM_RANDOM** using our SD, instead of the LMRD. After creating these models, we compared each one of them with the **LSTM_RANDOM** model, as can be seen in Figures 5.11, 5.12, 5.13.

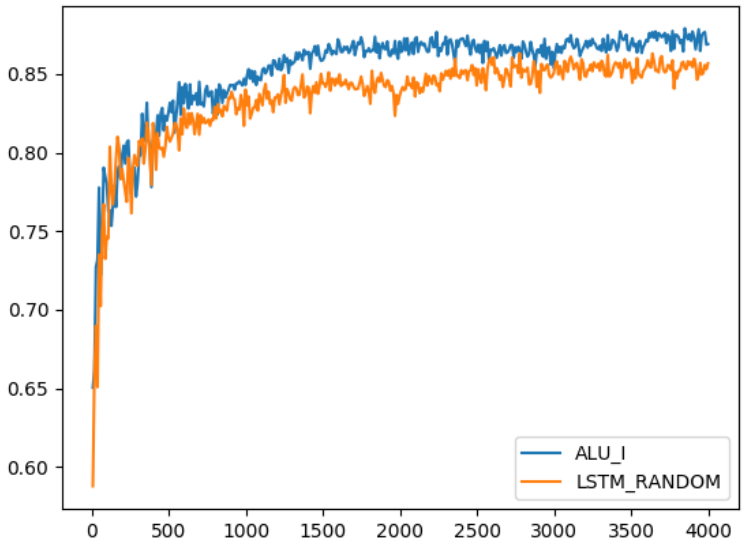

Figure 5.11: *Comparison between our ALU_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

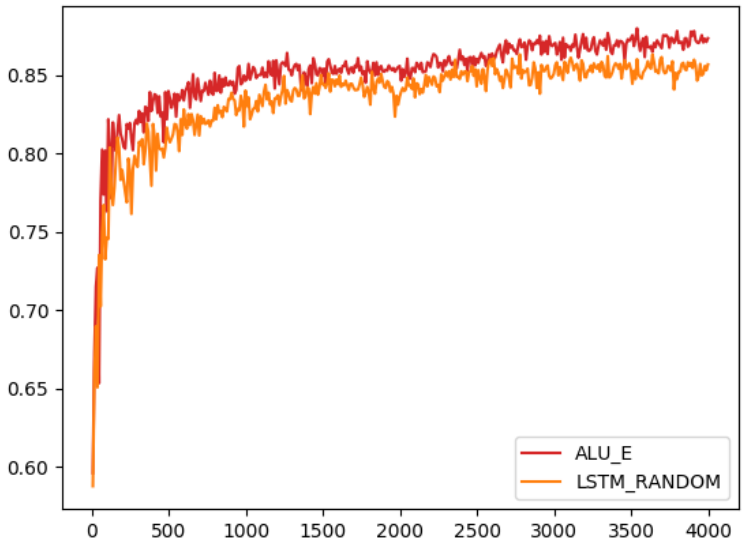

Figure 5.12: *Comparison between our ALU_E and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

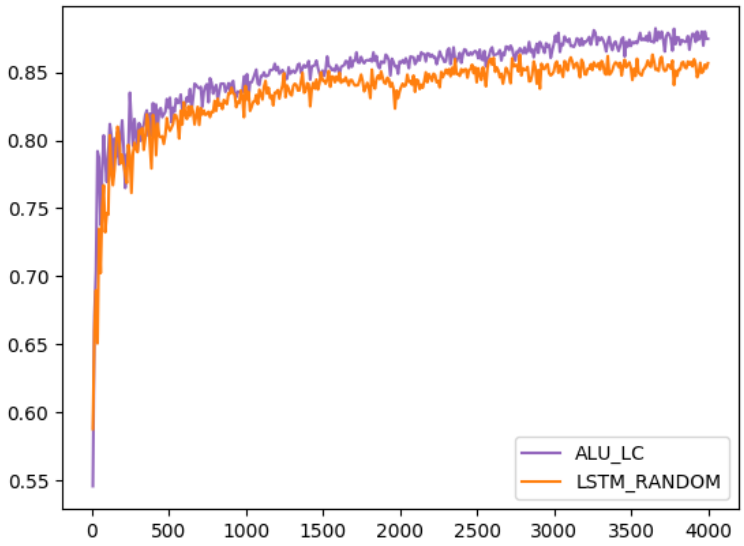

Figure 5.13: *Comparison between our ALU_LC and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

Figures 5.11, 5.12, 5.13 illustrates that all of the **ALU** models have achieved a better accuracy curve than our **LSTM_RANDOM**. This outcome respresents a different result than the one found for our LMRD, in which only the **ALU_I** performed better than our random model.

To compare the models, we need to visualize them together, as can be seen in Figure 5.14.

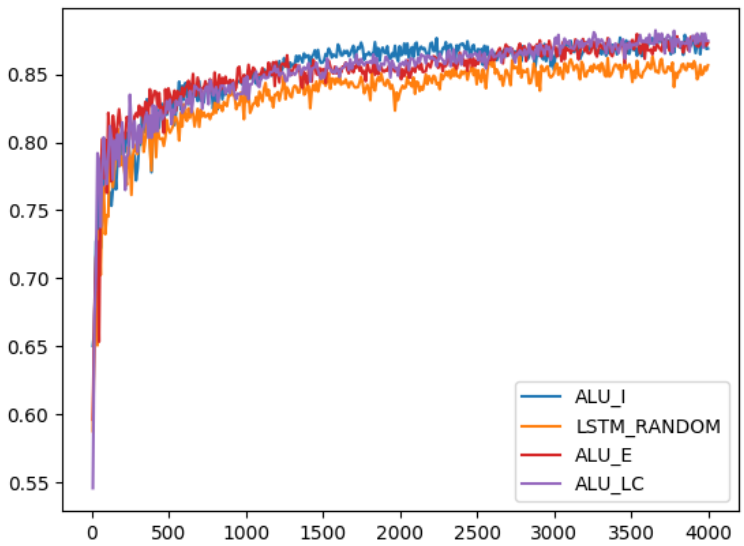

Figure 5.14: *Comparison between our ALU_I, ALU_E, ALU_LC and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

We can see that the **ALU** curves look similar to each other. Also, the curves reach a plateau after 2500 examples. Although the curves reach a similar accuracy value at the end of the experiments, there is one **ALU** model that reaches this value first, which is the **ALU_I** model. This model

Metric name	1009	1509	2009	2509	3009
ALU_I	0.85 ± 0.005	0.86 ± 0.007	0.86 ± 0.010	0.87 ± 0.001	0.86 ± 0.008
ALU_E	0.85 ± 0.007	0.85 ± 0.012	0.85 ± 0.008	0.86 ± 0.004	0.87 ± 0.004
ALU_LC	0.85 ± 0.002	0.85 ± 0.008	0.86 ± 0.001	0.86 ± 0.003	0.88 ± 0.006
LSTM_RANDOM	0.83 ± 0.007	0.84 ± 0.010	0.84 ± 0.008	0.85 ± 0.004	0.86 ± 0.008

Table 5.9: *Comparison of accuracy for each model given a number of training examples. Every accuracy value is presented with the standard deviation for that number of training examples.*

reaches the best accuracy in the graph roughly at 1500 training examples, while the other **ALU** models reach that accuracy approximately at 2600 examples. See Table 5.9 for better visualization of this finding.

We found a similar result for LMRD dataset, where the **ALU_I** model also performed better. However, we must remember that the other **ALU** models did not perform better than our random model for the LMRD. Furthermore, different from the LMRD, the accuracy difference between the **ALU** models is not that strong, since it is a 1% difference in accuracy between the **ALU_I** and the other **ALU** models.

After finding these results, we decided to test our CEAL approach on the SD as well. Since the best model for SD was the **ALU_I**, we will use our CEAL model with the same metric as the **ALU_I**, the Mutual Information. After performing this experiment, we have reached the result displayed in Figure 5.15.

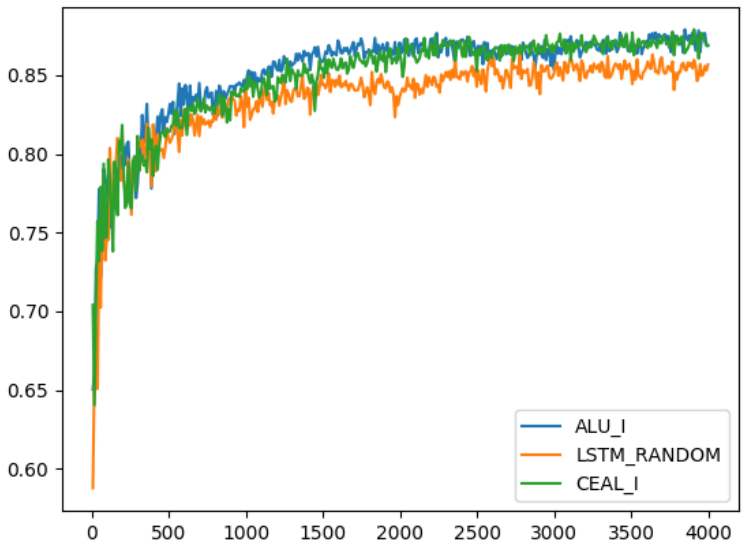

Figure 5.15: *Comparison between our ALU_I, CEAL_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

For our SD, we have not obtained any gain using the CEAL approach, as we have seen for the LMRD dataset. However, the **CEAL_I** does not underperform in comparison with the **ALU_I**, it just does not reach a better accuracy curve.

Finally, we compared out **ALU_I** model with a model which uses the softmax as the uncertainty

estimative together with the Mutual Information metric, the **ALS_I** model. After performing the experiments, we reached the result displayed in Figure 5.16.

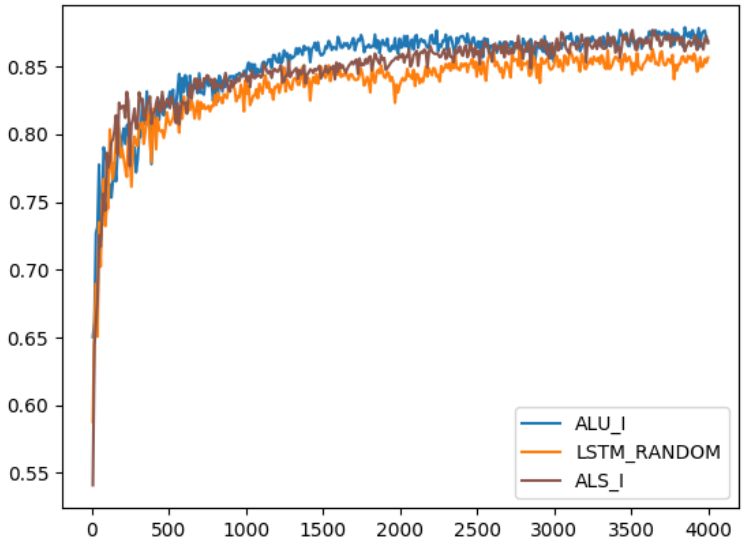

Figure 5.16: *Comparison between our ALU_I, ALS_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5.*

Although the **ALS_I** model reaches the same accuracy as the **ALU_I** model, we can see that the **ALU_I** model reaches this accuracy at 1500 training examples while the **ALS_I** model reaches it roughly at 2600 training examples. Therefore, we can conclude that out **ALU_I** model provided a better accuracy curve. This was the same result achieved in the LMRD, where **ALU_I** performed better than our **ALS_I** model. We believe this further reinforces that the *Monte Carlo Dropout* technique overperforms the softmax for the case of uncertainty handling.

Finally, the graphs with the standard deviation plotted can be seen in the Appendix Chapter B. We have not include these graphs in this Iteration because there is no significant overlap between the curves, except at the end of the graph.

5.4 Final Analysis

After performing the full set of experiments for both LMRD and SD, we can now properly address the two proposed research questions of this work. The research question **Q1** aimed to verify if measuring uncertainty with the *Monte Carlo Dropout* technique would provide any improvement when we perform Active Learning for the task of sentiment analysis. Although for the LMRD, only the combination of *Monte Carlo Dropout* and Mutual Information provided better accuracy than the random model, this was not the case for the SD, where all metrics performed better than the random model. This validates one of the conclusions found on [SC08], which states that the uncertainty metrics are problem dependent, indicating that there is no overall best metric for all situations. However, we can positively verify that measuring the uncertainty with *Monte Carlo Dropout* helped in the case of Active Learning.

Nevertheless, we must also address that it seems that larger models using *Monte Carlo Dropout* produce a better uncertainty estimate than smaller models. This can be seen in the difference between the curve in Figures 5.6 and 5.14. In the LMRD, the accuracy difference between the best

ALU model and the random model is around 5% while it is approximately 2% in the SD. This also reinforces one of the findings of [Gal16], which states that large models produce better uncertainty measurements.

For **Q2**, we want to verify if *Monte Carlo Dropout* would be a superior approach for uncertainty than using the softmax results as uncertainty measures. Figures 5.9 and 5.16 show that the *Monte Carlo Dropout* models performed better than the softmax models, meaning that the right uncertainty metric together with the *Monte Carlo Dropout* can provide a superior result than merely using the softmax result as an uncertainty evaluation.

Although we had positive results for both research questions, there are issues that we have found relating to both *Monte Carlo Dropout* and AL that needs to be addressed, since they are at the core of this work.

The first problem was related to the use of *Monte Carlo Dropout*. Finding the right hyperparameters was a hard task to accomplish. Although trial and error is common when finding hyperparameters, AL is a slow procedure, making that search costly. We tried to use random search using all the available training data we had, but we have not achieved the expected results, as seen in Section 5.3.2. In a real situation, where the user only has a small quantity of labeled data, finding these hyperparameters would be a difficult task and even time consuming. This is true because the user would need to perform a considerable amount of Active Learning cycles to verify if the results are improving, which for a real problem, can be a huge barrier.

However, the main problem with Active Learning found through this research is the amount of time taken to run a single experiment. At each AL cycle, we need to retrain the model for a large amount of epochs. This is an extremely slow procedure while also consuming a lot of computational resources (CPU and GPU included). Moreover, we have used an LSTM Network for performing our experiments. This network takes the size of the largest sentence we are using to train our model. For example, if we have a sentence with 600 words, as it happens in LMRD, we would have a network with 600 layers. Furthermore, the LSTM is a sequential architecture, meaning that it requires that we process one input at a time. This makes the use of LSTM in Active Learning extremely slow, since training an LSTM is a slow process and cannot be easily parallelized. Although this is not always the case for Deep Learning architectures, i.e., Convolutional Neural Networks don't suffer from this issue, the models produced are still large, meaning that they have multiple processing layers. If we want to to use Active Learning with Deep Learning, we reason that this is one of the main problems that should be dealt with.

Because of these findings, we strongly believe that further research in engineering approaches need to be done to handle that inherent problem with AL. This has happened for DL, with libraries such as TensorFlow and PyTorch, which allowed a huge improvement in the development and use of DL models. We believe that this is a necessary step for enabling Active Learning to be performed with larger datasets and on real problems too.

Finally, AL assumes one assumption that can be problematic. We expect that a model will fit our data before actually understanding the data distribution. We can choose a model that may not be able to fit our data or even select a model that is too complex over a simpler model that would fit our data. This can lead to both underfitting and overfitting in the use of AL. We have found that this is an inherent problem of Active Learning, but we must shed light on it, to alert any research that wants to replicate or use AL for their experiments.

Chapter 6

Conclusion

Uncertainty play a key role on the use of many techniques regarding Deep Learning models. From measuring the confidence of a model's prediction to using techniques such as Active Learning, model's uncertainty measurement is a necessity. However, there are few approaches to measure uncertainty in Deep Learning models. One of these approaches is the *Monte Carlo Dropout* technique.

This work aimed at evaluating this technique using the Active Learning framework for the task of sentiment analysis. We have evaluated if measuring the uncertainty using the *Monte Carlo Dropout* would allow for a better performance using Active Learning in contrast of randomly selected unlabeled examples. We further explored if the *Monte Carlo Dropout* would provide a better uncertainty measurement than using the softmax as an uncertainty measurement. Therefore, our work provides the following contributions:

- An intrinsic comparison of *Monte Carlo Dropout* with random sampling strategies for the Active Learning framework for the task of sentiment analysis.
- An intrinsic comparison of *Monte Carlo Dropout* with the softmax uncertainty measurement for the Active Learning framework for the task of sentiment analysis.

We conducted an iterative set of Active Learning experiments aimed at finding the best parameters for performing the experiments while also answering our research questions. After performing all our iterations, we have found that in fact *Monte Carlo Dropout* provides gains in comparison with both random sampling and softmax uncertainty measurements. In both cases, the accuracy curve obtained by models using the *Monte Carlo Dropout* was superior than the random and softmax approaches. For example, the *Monte Carlo Dropout* model using the Mutual Information metric achieved, on average, 5% more accuracy than the random model using the same amount of data using the Active Learning framework.

However, we have also found that some results where not consistent between datasets. While in one dataset a given technique performed better, in the other dataset, the same technique did not perform as well. Furthermore, we have also found that the *Monte Carlo Dropout* technique tends to work better for large Deep Learning models than small Deep Learning models as theorized in [Gal16].

Additionally, this master's research also provides contributions on additional questions related to Active Learning:

- An iterative experimentation scheme for performing Active Learning Experiments.

- Analysis of important parameters used to perform Active Learning Experiments together with *Monte Carlo Dropout*
- Practical considerations for the use of Active Learning on real world tasks.

We believe this discussion is essential for further research approaches using Active Learning and also for the practical use of Active Learning in real world problems.

6.1 Future Work

Although we have made an intrinsic comparison using the *Monte Carlo Dropout* technique, there is still a huge amount of experiments we can cover to better address this technique. From comparing *Monte Carlo Dropout* with other techniques, such as Semi-supervised techniques or model ensembles, to updating Active Learning approaches with *Monte Carlo Dropout*, there is a huge number of experiments that should be made. However, we strongly believe that in order to further allow Active Learning to be used by more researches and developers, engineering research should be made.

This Section will cover some ideas we believe are interesting paths to follow after performing this research, aiming both at *Monte Carlo Dropout* and the engineering behind the use of Active Learning.

6.1.1 CEAL and Monte Carlo Dropout

In this research, we have combined the CEAL technique (Section 3.5.1) with the *Monte Carlo Dropout* technique. Although the CEAL approach was not the best overall method to use, it achieved a great result for Large Movie Review Dataset and has reached almost the best result for the Subjectivity Dataset. This lead us to believe that the combination of the CEAL technique with *Monte Carlo Dropout* can be fruitful, especially if we consider the original paper that presented CEAL. In [WZL+17], the research have used the CEAL technique together with Active Learning for the task of image recognition. Although the CEAL approach achieved better results that the other models that they compared CEAL to, CEAL still uses softmax as an uncertainty measurement for selecting unlabeled data. As he have seen in Section 5.4, the softmax underperformed in comparison with the *Monte Carlo Dropout* approach. Therefore, we strongly believe that we could make experiments combining the CEAL with *Monte Carlo Dropout* for the task of Image Recognition. We would compare this new approach and verify if it performs better than the CEAL approach. This could potentially lead to a new Active Learning technique that uses a sound uncertainty measurement strategy to use both certain and uncertain example to guide the next steps of the Active Learning cycle.

Furthermore, the experiments would be performed using a Convolutional Neural Network instead of a Long-Short Term Memory (LSTM) network. A Convolutional Neural Network is faster to train and to run than a LSTM. That is because the convolutional operations can be easily parallelized in a GPU. This cannot happen for a LSTM, that reads its input sequentially, not allowing for parallelized approaches. This will lead to faster and easier experiments to run as well.

6.1.2 Active Learning Parameters

In our research, we have explored only a few of the Active Learning parameters, such as the number of epochs to train the model. However, more parameters can be evaluated. The size of the unlabeled group to evaluate, the DL architectures used and the number of labels for our data can provide valuable information when experimenting with Active Learning. For example, in our research, we have used sentiment analysis task that consider only two possible labels, Positive and

Negative, therefore all our samples are divided in two distinct groups. We believe this allows for random models to perform better, since the chance for them to selecting meaningful examples from both classes is higher. In [SYL+17] and [GIG17], the task they have used for Active Learning has a bigger amount of labels. In [GIG17], the researches experiment with the MNIST dataset, which has 10 different classification classes. In both of these works, the difference between the accuracy curve of the *Monte Carlo Dropout* models and the random model was higher than the one found in this research. We believe one of the reasons may be the number of classes between the task. Therefore, this would be an useful topic to further research on.

6.1.3 Active Learning Selection Visualization

Although we are using the *Monte Carlo Dropout* technique to select unlabeled examples, we have not explored in this research the type of examples the model is selecting. This means that are not visualizing what is the model uncertain about. For example, the model may initially take longer sentences first over smaller ones. Or maybe, the model is selecting sentence which possess words rarely seen on other sentences. By visualizing the sentences the model is taking, we would be able to further understand the uncertainty being measured by it. This would allow us to further improves our uncertainty selection policies, as has happened in [SYL+17] where they created a new uncertainty selection policy based on the samples the model was selecting from the unlabeled group.

6.1.4 Active Learning Engineering

One of the main reasons for the increasing popularity of Deep Learning models was the creation of big datasets and the use of Graphical Processing Units (GPU). However, in research and business, we can also see that this advancement was aided by open source libraries that allowed developers to easily implement these models and use GPUs resources. Libraries such as TensorFlow[1] and PyTorch[2] ease the process of developing Deep Learning models. Not only that, these libraries address issues such as the ETL pipeline and model evaluation as well. If we want Active Learning to be used by more researches and outside of academia we believe that a similar path must be taken, and a library must be created.

The library should address the whole Active Learning cycle and easily allow the integration of Deep Learning models developed from a different range of libraries. Furthermore, the library should also focus on engineering questions related to how we sample the unlabeled group and how we train our model. Additionally, we need to focus on finding better solutions for not retraining the model at each Active Learning step. As we have discussed in Section 5.4, retraining a DL model is a costly and time consuming operation. We understand that we use Maximum Likelihood Estimation to find the weight parameters of a DL model, and because of that, these parameters cannot be considered a valid posterior for when we retrain a new model with new data. However, we must think of a more practical approach than retraining the model at each Active Learning cycle, even if that approach is not mathematical our statistically sound. We have made an initial experiment considering this approach, as can be seen on Appendix B.2, but further research must still be made to solidify this scenario.

Finally, we understand that this path is one of an engineering research, but we further state that this is a necessary research to do. If we want to enable the use of technology to more and more people, we need engineering solutions for this mean and Active Learning is not different.

[1] https://github.com/tensorflow/tensorflow
[2] https://github.com/pytorch/pytorch

6.1.5 Update Active Learning cycle for Deep Learning models

In our work, in every step of the Active Learning cycle, we have trained the whole DL model. Although this is the classical approach in Active Learning, we have experienced that it possess major drawbacks when using DL, since training the whole model is a costly process. One approach to better handle that issue is to not train the whole model again at each cycle, but just the **Classification** layer of the network. To better understand what that means, we can visualize our DL model as a combination of two distinct networks, the **Feature Learner** layer and the **Classification** layer. The **Feature Learner** layer is the one responsible for receiving the raw input data and convert it into a more optimal format that allow to better separate the classification instances of the problem. Once the **Feature Learner** produces this result, the **Classification** layer produces the final decision of the network. In our case, we could see the *LSTM* layer as the **Feature Learner** and our *Fully Connected and Softmax* layers as the **Classification** learner.

Based on that distinction, we strongly believe that a more adequate Active Learning cycle to use together with a DL model is the following one:

- Train the model with the available labeled data.
- Use the model to select the most informative samples from the unlabeled data.
- Give the data for the oracle to label it
- Retrain only the **Classification** layer of the model with the new labeled data (Once we have *enough* data, retrain the whole model, both **Feature Learner** and **Classification** layers)
- Restart the Active Learning cycle

By using this cycle, we will guarantee that we will train our model faster during each Active Learning cycle, since we will only train a subset of its network. Furthermore, we believe that we would have less noise results, since we will not retrain our **Feture Learner** layer at each cycle. Because of that, we thing that it will be worth to experiment with that new Active Learning cycle, however, we understand that this setting now possess new hyperparameters, such as the threshold on the amount of data needed to retrain the whole model at an Active Learning cycle. This would need to be found empirically, but we still believe that this cycle is more optimal than the one used in this research and could potentially achieve better results than the ones found on this work.

Appendix A

Evidence Lower Bound

This chapter will be used to described how the *Evidence Lower Bound*(ELBO) was derived when we are measuring the similarity between $q_\lambda(\theta)$ and $p(\theta|\mathbf{X}, \mathbf{Y})$. Remember that to measure the similarity between the distributions we are measuring the KL divergence between them A.1:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta|\mathbf{X}, \mathbf{Y})} d\theta \tag{A.1}$$

We can start rewriting this equation using the following steps:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) log \frac{q_\lambda(\theta)}{p(\theta|\mathbf{X}, \mathbf{Y})} d\theta \tag{A.2}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) [log q_\lambda(\theta) - log p(\theta|\mathbf{X}, \mathbf{Y})] d\theta \tag{A.3}$$

Now, recall that:

$$p(\theta|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \theta) p(\theta)}{p(\mathbf{Y}|\mathbf{X})} \tag{A.4}$$

Therefore we can expand A.3 into A.8:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) [log q_\lambda(\theta) - log \frac{p(\mathbf{Y}|\mathbf{X}, \theta) p(\theta)}{p(\mathbf{Y}|\mathbf{X})}] d\theta \tag{A.5}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) [log q_\lambda(\theta) - [log p(\mathbf{Y}|\mathbf{X}, \theta) p(\theta) - log p(\mathbf{Y}|\mathbf{X})]] d\theta \tag{A.6}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) [log q_\lambda(\theta) - log p(\mathbf{Y}|\mathbf{X}, \theta) p(\theta) + log p(\mathbf{Y}|\mathbf{X})]] d\theta \tag{A.7}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X}, \mathbf{Y})) = \int q_\lambda(\theta) [log q_\lambda(\theta) - log p(\mathbf{Y}|\mathbf{X}, \theta) p(\theta)] d\theta + log p(\mathbf{Y}|\mathbf{X}) \tag{A.8}$$

Because:

$$\int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X}) d\theta = logp(\mathbf{Y}|\mathbf{X}) \int q_\lambda(\theta) d\theta \tag{A.9}$$

Since $q_\lambda(\theta)$ is a valid probability distribution, $\int q_\lambda(\theta) d\theta = 1$. Therefore, $\int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X}) d\theta = logp(\mathbf{Y}|\mathbf{X})$. With that step explained, we can further expand A.8 into A.14.

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[logq_\lambda(\theta) - logp(\mathbf{Y}|\mathbf{X},\theta)p(\theta)]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.10}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[logq_\lambda(\theta) - [logp(\mathbf{Y}|\mathbf{X},\theta) + logp(\theta)]]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.11}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[logq_\lambda(\theta) - logp(\mathbf{Y}|\mathbf{X},\theta) - logp(\theta)]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.12}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[logq_\lambda(\theta) - logp(\mathbf{Y}|\mathbf{X},\theta) - logp(\theta)]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.13}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[log\frac{q_\lambda(\theta)}{logp(\theta)} - logp(\mathbf{Y}|\mathbf{X},\theta)]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.14}$$

We can see that:

$$\int q_\lambda(\theta)\frac{q_\lambda(\theta)}{logp(\theta)}d\theta = KL(q_\lambda(\theta)||p(\theta)) \tag{A.15}$$

After identifying that, we can further expand A.14 into A.18:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = \int q_\lambda(\theta)[log\frac{q_\lambda(\theta)}{logp(\theta)} - logp(\mathbf{Y}|\mathbf{X},\theta)]d\theta + logp(\mathbf{Y}|\mathbf{X}) \tag{A.16}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) = -\int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X},\theta)d\theta + KL(q_\lambda(\theta)||p(\theta)) + logp(\mathbf{Y}|\mathbf{X}) \tag{A.17}$$

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) + \int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X},\theta)d\theta - KL(q_\lambda(\theta)||p(\theta) = logp(\mathbf{Y}|\mathbf{X}) \tag{A.18}$$

Now, if we call $\int q_\lambda(\theta) logp(\mathbf{Y}|\mathbf{X},\theta)d\theta - KL(q_\lambda(\theta)||p(\theta))$ as F, we can see that A.18 turn into A.19:

$$KL(q_\lambda(\theta)||p(\theta|\mathbf{X},\mathbf{Y})) + F = logp(\mathbf{Y}|\mathbf{X}) \tag{A.19}$$

Since the KL divergence is always ≤ 0, we can see that $F \leq logp(\mathbf{Y}|\mathbf{X})$. Since in the Baye's theorem A.4, the constant term $p(\mathbf{Y}|\mathbf{X})$ is called the *evidence* of our model, we can see that our F

is an *Evidence Lower Bound* of that term. Therefore, that's why our F variable is actually called the *Evidence Lower Bound*(ELBO) when we are using Variational Inference.

Appendix B

Additional Experiments

In this section we will cover additional performed experiments that were not displayed in Section 5.

B.1 Active Learning - Iteration 6 Additional Experiments

In Section 5.3.7 we have displayed the results of performing our Active Learning experiments for the SD. However, in this Section, we did not displayed the graphs with the standard deviation plotted. This section will present these graphs and analyze their results.

First, we can analyze the individual **ALU** models in comparison with the **LSTM_RANDOM** model in Figures B.1, B.2 and B.3.

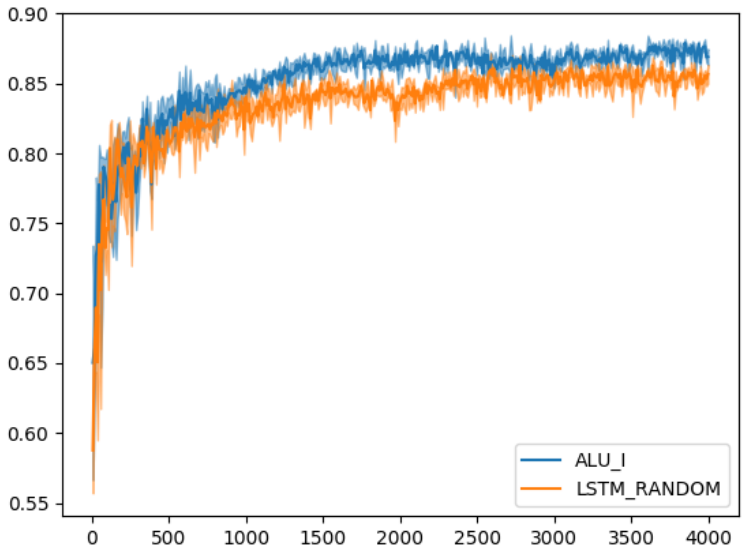

Figure B.1: *Comparison between our ALU_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

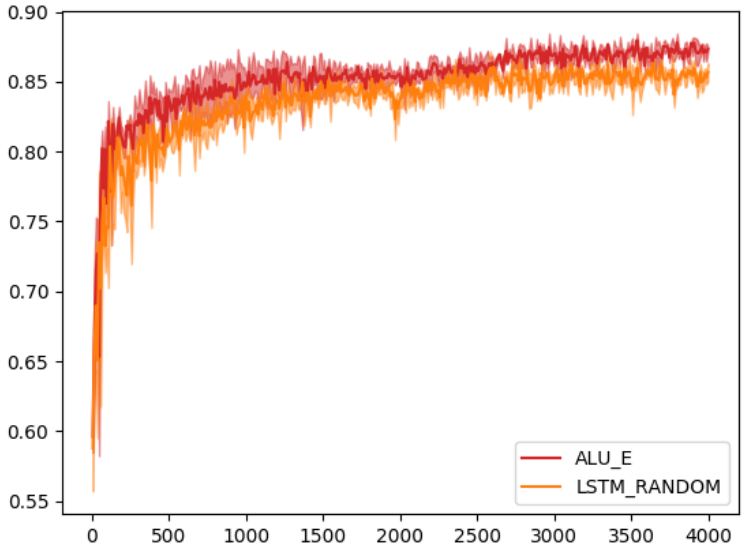

Figure B.2: *Comparison between our ALU_E and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

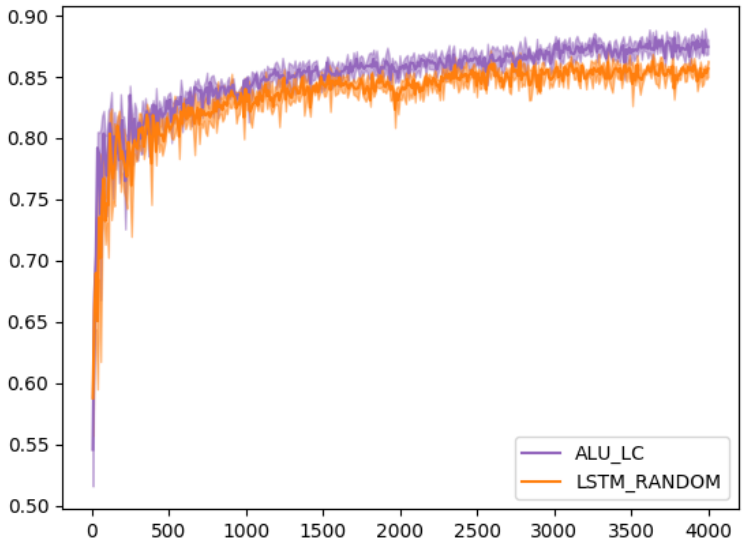

Figure B.3: *Comparison between our ALU_LC and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

We can see that even plotting the standard deviation, the curves do not overlap. We can see more overlap when we plot all models together, as can be seen in Figure B.4, but as we have seen in Table 5.9, although there is overlap in the end of the curves, the **ALU_I** model reaches a higher accuracy earlier than the other models.

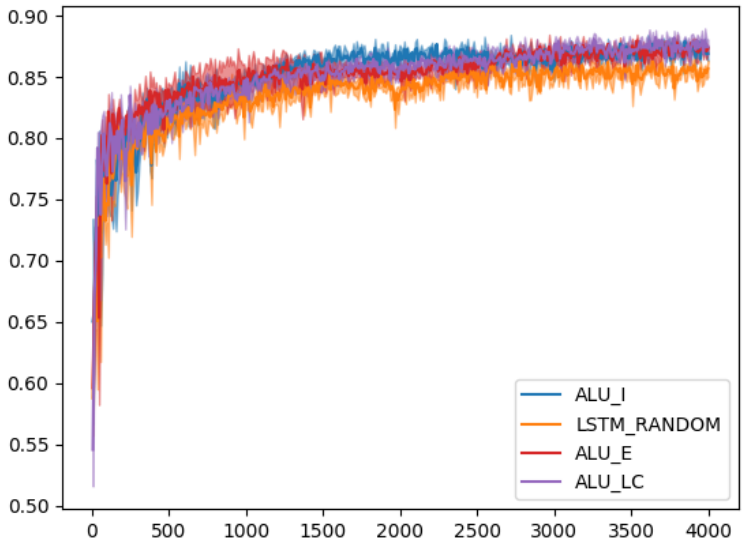

Figure B.4: *Comparison between our ALU_I, ALU_E, ALU_LC and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

We can see the same scenario for the comparison of the **CEAL_I** and the **ALU_I** models, were there is overlap in the end of the curve, but the **ALU_I** reaches a better accuracy earlier, as can be seen in Figure B.5.

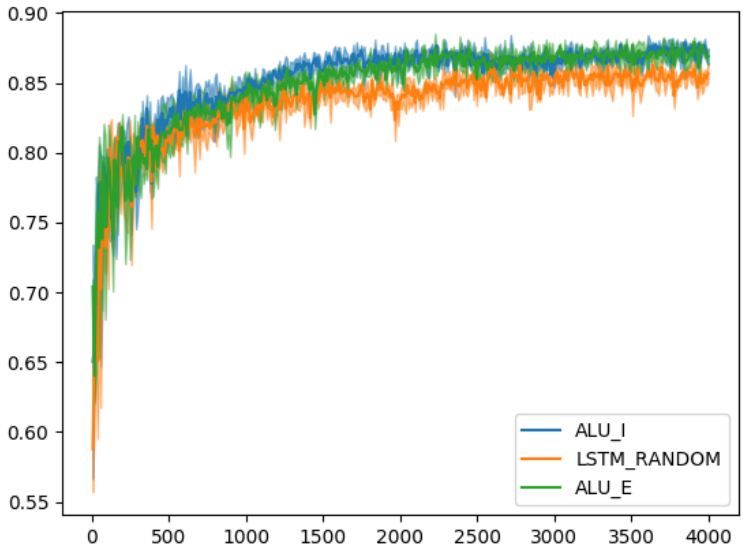

Figure B.5: *Comparison between our ALU_I, CEAL_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

Finally, this scenario is also present in the comparison of the **ALU_I** and the softmax model **ALS_I**, where the overlap happens, but the **ALU_I** reaches a better accuracy earlier, as can be seen in Figure B.6.

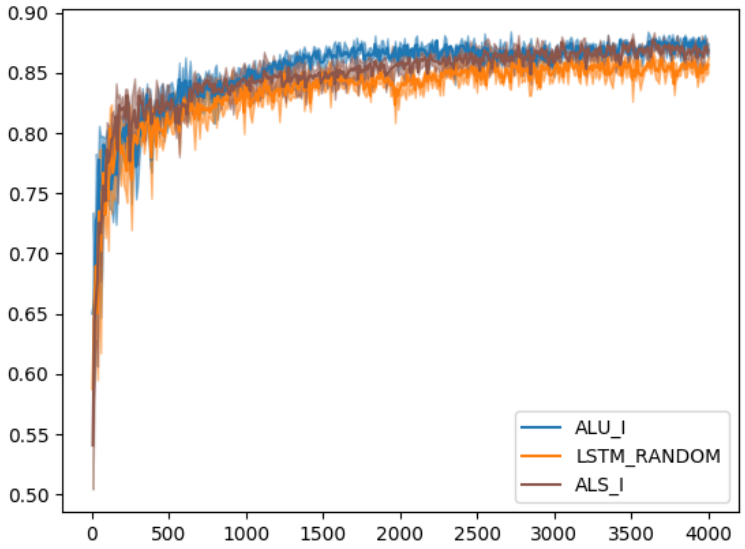

Figure B.6: *Comparison between our ALU_I, ALS_I and our LSTM_RANDOM model for the SD dataset using EPO=150, Q=10, NC=400 and DROP=0.5 using the standard deviation around each point.*

B.2 Continuous Active Learning

One approach that we have researched in the Continuous Active Learning. This scenario work as follows: we train our initial model with our defined number of epoches (150) and in the next AL cycles, we do not reset the model, but instead train it for an addiational number of epoches. This approach aims at solving one of the biggest problems of Active Learning, that is to retrain the model at each step.

Based on this approach, we have defined a new AL mode, **CONTINUOUS_ALU_I**, based on the *Monte Carlo Dropout* and Mutual Information metric. We have decided to further train our continuous model for 50 epoches at each AL cycle. After creating this model, we have achieved the results displayed in Figure B.7.

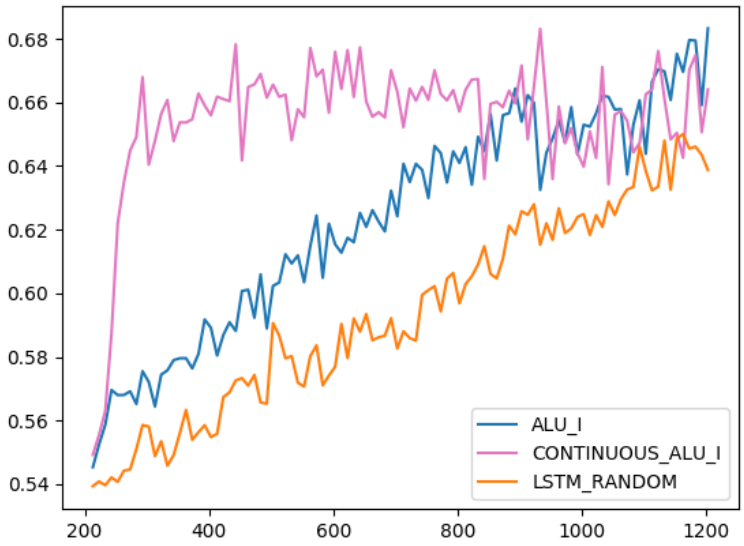

Figure B.7: *Comparison between our ALU_I, CONTINUOUS_ALU_I and our LSTM_RANDOM model for the LMRD dataset using EPO=150, Q=10, NC=100 and DROP=0.5.*

We can see in Figure B.7 that the **CONTINUOUS_ALU_I** model achieves a higher accuracy early on, different from the **ALU_I** and **LSTM_RANDOM**. However, we can also see that the model achieves a plateau at around 500 training examples. During this research, we could not understand why this model was behaving like that, but with more time in hands, we believe this can be an useful path to further research on, specially considering the speed up we would enable in using AL together with DL.

Bibliography

[BOK+13] J. Bayer, C. Osendorfer, D. Korhammer, N. Chen, S. Urban e P. van der Smagt. On Fast Dropout and its Applicability to Recurrent Networks. *ArXiv e-prints*, November 2013. 22

[CCH14] Bor-Chun Chen, Chu-Song Chen e Winston H. Hsu. Cross-age reference coding for age-invariant face recognition and retrieval. Em *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. 4

[Das11] Sanjoy Dasgupta. Two faces of active learning. *Theor. Comput. Sci.*, 412(19):1767–1781, April 2011. 24

[Gal16] Yarin Gal. *Uncertainty in Deep Learning*. Tese de Doutorado, University of Cambridge, 2016. 5, 9, 15, 17, 19, 27, 50, 51, 62, 63

[GBC16] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 11

[GG15a] Y. Gal e Z. Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *ArXiv e-prints*, December 2015. 22, 24, 44

[GG15b] Y. Gal e Z. Ghahramani. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. *ArXiv e-prints*, June 2015. 1

[GG16] Yarin Gal e Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. Em *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org, 2016. 16

[GHP07] G. Griffin, A. Holub e P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. 4

[GIG17] Y. Gal, R. Islam e Z. Ghahramani. Deep Bayesian Active Learning with Image Data. *ArXiv e-prints*, March 2017. 1, 2, 5, 44, 49, 50, 51, 65

[GJ14] Alex Graves e Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. Em Eric P. Xing e Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR. 20

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998. 20

[HS97] Sepp Hochreiter e Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. 21

[HSK+12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever e R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, July 2012. 16

[HvC93] Geoffrey E. Hinton e Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. Em *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 5–13, New York, NY, USA, 1993. ACM. 15

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola e Lawrence K. Saul. An introduction to variational methods for graphical models. *Mach. Learn.*, 37(2):183–233, November 1999. 12

[Kim14] Yoon Kim. Convolutional neural networks for sentence classification. Em *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014. 4

[KL15] Andrej Karpathy e Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. Em *CVPR*, pages 3128–3137. IEEE Computer Society, 2015. 22

[KSH12] Alex Krizhevsky, Ilya Sutskever e Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Em F. Pereira, C. J. C. Burges, L. Bottou e K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[KSP+15] Janez Kranjc, Jasmina Smailović, Vid Podpečan, Miha Grčar, Martin Žnidaršič e Nada Lavrač. Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the clowdflows platform. *Information Processing & Management*, 51(2):187 – 203, 2015. 5

[Kul59] Solomon Kullback. *Information Theory and Statistics*. Wiley, New York, 1959. 12

[KW13] D. P Kingma e M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013. 13

[LG94] David D. Lewis e William A. Gale. A sequential algorithm for training text classifiers. Em *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 3–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc. 25

[MCCD13] T. Mikolov, K. Chen, G. Corrado e J. Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, January 2013. 5, 38

[MDP+11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng e Christopher Potts. Learning word vectors for sentiment analysis. Em *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 2, 37

[MFC+10] Daniel S. Marcus, Anthony F. Fotenos, John G. Csernansky, John C. Morris e Randy L. Buckner. Open access series of imaging studies: Longitudinal mri data in nondemented and demented older adults. *Journal of Cognitive Neuroscience*, 22(12):2677–2684, 2010. PMID: 19929323. 1

[NYC14] A. Nguyen, J. Yosinski e J. Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. *ArXiv e-prints*, December 2014. 5

[PL04] Bo Pang e Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. Em *Proceedings of the ACL*, 2004. 2, 35

[PMB12] R. Pascanu, T. Mikolov e Y. Bengio. On the difficulty of training Recurrent Neural Networks. *ArXiv e-prints*, November 2012. 20

[PSM14] Jeffrey Pennington, Richard Socher e Christopher D. Manning. Glove: Global vectors for word representation. Em *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. 38

[RHW88] David E. Rumelhart, Geoffrey E. Hinton e Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. 11

[SC00] Greg Schohn e David Cohn. Less is more: Active learning with support vector machines. Em *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 839–846, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 1

[SC08] Burr Settles e Mark Craven. An analysis of active learning strategies for sequence labeling tasks. Em *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1070–1079, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. 27, 61

[Set09] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. 1

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever e Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. 17

[SPdC17] Davi Santos, Ricardo PrudÃªncio e Andre de Carvalho. Empirical investigation of active learning strategies. *Neurocomputing*, 09 2017. 24

[SVL14] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. Sequence to sequence learning with neural networks. Em *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press. 1, 20

[SYL+17] Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod e A. Anandkumar. Deep Active Learning for Named Entity Recognition. *ArXiv e-prints*, July 2017. 5, 6, 49, 65

[TK02] Simon Tong e Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002. 1

[TQL15] Duyu Tang, Bing Qin e Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. pages 1422–1432, 2015. 4

[VG14] Sudheendra Vijayanarasimhan e Kristen Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. *Int. J. Comput. Vision*, 108(1-2):97–114, May 2014. 1

[WZL+17] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang e Liang Lin. Cost-effective active learning for deep image classification. *IEEE Trans. Cir. and Sys. for Video Technol.*, 27(12):2591–2600, December 2017. 4, 27, 41, 44, 64

[XCQH16] J. Xu, D. Chen, X. Qiu e X. Huang. Cached Long Short-Term Memory Neural Networks for Document-Level Sentiment Classification. *ArXiv e-prints*, October 2016. 4

[XZS16] C. Xiong, V. Zhong e R. Socher. Dynamic Coattention Networks For Question Answering. *ArXiv e-prints*, November 2016. 22

[ZSV14] W. Zaremba, I. Sutskever e O. Vinyals. Recurrent Neural Network Regularization. *ArXiv e-prints*, September 2014. 22

[ZW16] Ye Zhang e Byron C. Wallace. Active discriminative word embedding learning. *CoRR*, abs/1606.04212, 2016. 5

[ZWL18] L. Zhang, S. Wang e B. Liu. Deep Learning for Sentiment Analysis : A Survey. *ArXiv e-prints*, January 2018. 35