COMBINING ACTIVE LEARNING AND DATA AUGMENTATION TO REDUCE LABELLED TRAINING DATA FOR SENTIMENT ANALYSIS

by

Colton Aarts

B.Sc., University of Northern British Columbia, 2019

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

April 2025

Abstract

Creating a sentiment analysis classifier requires a large amount of labelled training data. Labelling this data is an expensive and time-consuming process. Because of this, reducing the amount of labelled data required leads to classifiers that are cheaper to train and more accessible to all disciplines. Many different methods can be used to reduce the amount of labelled data. For this research, we focused on combining active learning and lexical expansion techniques.

By combining these two techniques, this research examined an underutilized area of study. Active learning focuses on letting the classifier select the data to learn from, while lexical expansion creates more data for the classifier. While there are a larger number of different techniques in both fields, there is little work to be done to combine them. We felt this was a natural progression for these techniques as they complement each other well. The active learning technique will select the data to be labelled, and the lexical expansion technique will generate high-quality artificial data from this hand-selected information. In addition to combining these techniques, we examined how different neural network structures would interact with our new technique.

Our research found that the combination of active learning and lexical expansion improved the performance of our classifiers for very small amounts of data. We found a significant difference between the performance of our two classifiers. While there was an improvement at low levels of training data, at higher levels, we found that the combined techniques did not offer any improvements over the active learning technique. Overall, we found potential benefits to combining the two techniques and that future research is required to understand further how to leverage these improvements best.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AL	Active Learning
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CoLSTM	Convolutional Long Short Term Memory
LSTM	Long Short Term Memory
NLP	Natual Language Processing
PLSDA	Part-of-Speech Focused Lexical Substitution for Data Augmentation
POS	Part-of-Speech

Acknowledgements

I would first like to express my gratitude to my supervisor, Dr. Fan Jiang (Terry). Without his continued support and advice, I would never have started, let alone completed my Masters. He has constantly provided me with guidance as well as opportunities for success. Between research papers to industry grants, Terry has been instrumental in my success as a Masters student at UNBC.

I would next like to thank my committee, Dr. Chen and Dr. Monu, who have supported me as much as Terry has. They have provided advice and guidance for my research and work at the university. Without their input, I would not have been able to complete this research.

Next, I would like to thank the fantastic Computer Science department here at UNBC. From professors like Dr. Haque, who has been a constant source of support and motivation, to our outstanding Admin Assistants, who helped me navigate the mysteries of required paperwork, everyone in the department has helped me during my time at UNBC.

Finally, I would like to thank my friends and family. Without my friend group's unconditional love and support, I would never have been able to complete my research. Alongside this, my mom and my sister's motivation, drive, and love were instrumental in my ability to complete my Masters.

Chapter 1

Introduction and Motivation

Sentiment Analysis (SA) is a field of Natural Language Processing (NLP) that is concerned with detecting positive, neutral, and negative text. Currently, the best-performing models use various deep-learning techniques and neural network structures. While these models perform well, they suffer from a shared drawback of requiring a large amount of labelled data to achieve the desired performance. In addition, many SA models are domain-dependent, adding the additional constraint that a model that performs well on one domain may not perform well on a different domain. Acquiring large domain-specific datasets can be expensive as domain experts must provide the labels. Utilizing methods that reduce the required labelled data to combat these costs is helpful. Active Learning (AL) and Data Augmentation (DA) are popular methods to reduce the total data needed.

1.2 Motivation

A trend that has appeared in many different data mining disciplines is using more and more data in training and training longer to achieve better performance. While this has been achieving the desired results, it is not always feasible for smaller research groups or many industrial partners. For example, Open AI used over 40

GBs of data for Chat-GPT and built a custom supercomputer to train it, the BERT pre-trained word embeddings are trained on 13 GB of text, AlphaGo was trained on over 1000 CPUs and over 100 GPUs, and for text mining, there has been research showing that having over 200,000 labelled data points can improve performance [2]. While continually pushing the bleeding edge of performance is important, it is also important to remember that producing tools other domains can use is essential to research. With this in mind, finding ways to reduce the amount of data or computational resources required to achieve a model that still performs at a high level is essential. When focusing on sentiment analysis, acquiring high-quality labels can be expensive [3]. The cost of labelling is compounded by the fact that current sentiment analysis techniques remain domain-dependent, and creating high-quality labels requires experts in that domain. While there are methods that can acquire labels cheaper, these run the risk of creating a noisy dataset, which can affect the model's performance [4].

1.3 Objectives and Contributions

While there has been extensive research into different active learning and lexical expansion techniques, combining them is under-researched. In addition, the existing research tends to focus on a single model and demonstrate how the individual techniques improve the performance of the specific model. For our research, we proposed four research questions:

1. Does combining an AL and LE offer better performance?

2. Is it possible to use this technique to create a classifier whose performance improves faster?

3. Does the proposed algorithm behave differently with different classifier ar-

chitectures?

4. Is there a large variance between the average and best performance of the classifiers?

The main goals of our research are Questions 1 and 2. We expect that combining the two techniques will allow us to perform better with less training data. To do this, we will examine our classifiers' performance when they are trained with various training data. We propose Questions 3 and 4 to help get a better understanding of how our algorithm will interact with different classifier architectures and to investigate if our proposed algorithm creates reliable results. We expect that our proposed algorithm will increase the performance of any classifier it uses. While we expect that the overall performance of the classifier will improve, we predict that the variance of our proposed APLSDA algorithm will be higher than the base classifier as we introduce noise through the PLSDA technique.

1.4 Thesis Layout

The thesis is organized as follows. We introduce the background information in Chapter 2. In Chapter 3, we introduce the different AL and LE algorithms we are using and the neural network architecture we are using to create our classifiers. Our combined algorithm is presented in Chapter 4, and our experiment setup is found in Chapter 5. Finally, we evaluate the performance in Chapter 6, and the final Chapter 7 contains our conclusions and future work.

Chapter 2

Background

This chapter will focus on reviewing concepts related to our research. We will start by providing background knowledge on neural networks (NNs). We will then introduce the different types of Language Models (LMs) used in Sentiment Analysis. We will then explore how SA combines the different LM and NN structures. We will conclude this chapter by examining techniques used to improve the performance of SA models.

2.1 Neural Networks

The idea of neural networks was introduced in the mid-1940s by McCulloch and Pitts in [5]. In this work, the authors proposed a computational model to represent how the brain learns. Their work laid the foundation for creating NNs. The smallest component of a NN is the neuron or perceptron. A perceptron works by summing its inputs i_n multiplied by their associated weights w_n. The output from the perceptron results from applying an activation function $f(y)$ to the sum. The basic structure of a single perceptron can be seen in Figure 2.1. Many different activation functions are popular in research, including softmax and tanh. The weights are first initialized to random values. These values are updated during training to

minimize a given error function.

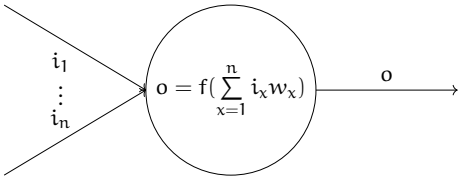

Figure 2.1: A Single Neuron

One of the main drawbacks of a single perceptron is that they can only learn linear relationships between their inputs. However, arranging multiple perceptrons into different layers can create a network that learns non-linear relationships. The weights of each perceptron are learned through backpropagation. This is the basic structure of neural networks today, and a simple example can be seen in Figure 2.2. This network would be described as a fully connected network whose first two layers have three perceptrons, and the final layer has a single perceptron. Fully connected networks are simply networks where the output from all perceptrons in the previous layer is provided as input to each perceptron in the following layer. Combining different combinations of layers that contain different numbers of perceptrons can allow NNs to learn a fast variety of relationships. While NNs with structures similar to Figure 2.2 are useful, different types of networks have been developed to help solve complex problems. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are particularly interesting to our research and will be examined in more detail in this section.

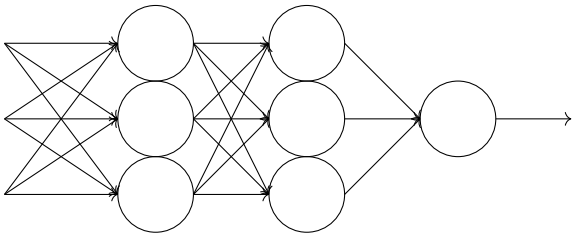

Figure 2.2: A Simple Network

2.1.1 Convolutional Neural Networks

CNNs are neural networks that contain at least one convolutional layer. These layers were developed initially to be used on image data but have been adapted to text processing. CNNs were inspired by the brain's visual cortex and how specific regions focus on sections of the visual field. CNNs also help reduce the neurons needed in the fully connected layers. This helps reduce the computation time needed to train the network.

CNNs use convolutions to detect different features in the dataset. A convolution is an integral whose result is the overlap between two functions. CNNs use this idea by applying several filters to the input. The filters will all be the same size and are moved across the data. Each of these filters will detect a different feature. Stacking multiple convolutional layers on top of one another is also popular. As you add more convolutional layers, the complexity of the features that can be detected increases. While convolutions will reduce the data's dimensionality, using a pooling layer is also common. This layer will look at a section of the output from the convolution and apply a function to return a single number. Two popular functions are max pooling and average pooling. After applying the pooling layer, the final results will be flattened and fed into a fully connected layer for classification.

CNNs were adapted to be used on text data by modifying how the filter is moved across the data. Instead of moving the filter in two dimensions, the filter is only moved in one. This allows the CNN to consider the words that occur near each other and allows the network to learn different relationships in the text.

2.1.2 Recurrent Neural Networks

While CNNs offer an advantage over regular NNs by changing the structure of the layers and reducing the number of neurons needed, RNNs change the internal

structure of the neurons, allowing them to remember the information they have previously seen. Adding memory to the neurons makes RNNs especially useful for time series data. This includes text, as the meaning of words is influenced by all the context that occurs before them. RNNs modify the internals of a neuron to include past information. These neurons work by taking the previous state h_{t-1} and combining it with the current input. The output from this is then used as the state for the next neuron and the output for timestep t. This can be seen in Equation 2.1.

$$h_t = f(W_x x_t + W_h h_{t-1} + b_h) \tag{2.1}$$

Where f is some activation function. W_x and W_h are the weights associated with the inputs and the previous state, respectively, and finally, b_h is the neuron's bias. This configuration allows for the passage of information between neurons in the same layer. An example RNN neuron can be seen in Figure 2.3. While RNNs provide advantages over a standard NN, they suffer drawbacks. Namely vanishing and exploding gradients. A solution to the vanishing gradient problem can be found in Long-Short Term Memory units (LSTMs).

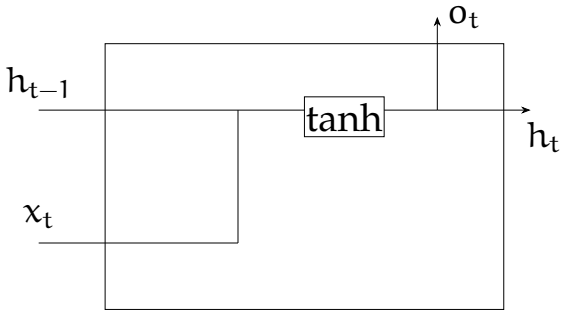

Figure 2.3: RNN Neuron

LSTMs were introduced in 1997 by Hochreiter and Schmidhuber [6]. LSTMs modify regular RNNs to allow the neuron to remember information longer. This modification comes in the form of three gates and an additional state. These gates

are the forget, output, and input gates. The state is a long-term state that accumulates and forgets information as time passes. The forget gate was added by Gers, Schmidhuber, and Cummins in 2000 [7], allowing the LSTM to remove information from the long-term state. The input gate allows information to be added to the long-term state, and finally, the output gate creates the next hidden state and the output for the current time step. An example LSTM neuron can be seen in Figure 2.4. Where $\times$ is the Hadamard product, $+$ is element-wise addition, and σ is the sigmoid function. We will start by examining the forget-gate, the out-gate, and the in-gate in Equations 2.2, 2.3, and 2.6.

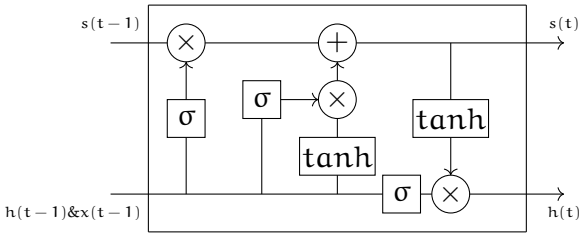

Figure 2.4: LSTM Neuron [1]

$$\mathrm{for}(t) = \sigma(W_{\mathrm{for}} x_t + V_{\mathrm{for}} h_{t-1} + b_{\mathrm{for}}) \tag{2.2}$$

$$\mathrm{out}(t) = \sigma(W_{\mathrm{out}} x_t + V_{\mathrm{out}} h_{t-1} + b_{\mathrm{out}}) \tag{2.3}$$

Both the forget-gate and the out-gate are straightforward processes. They both apply a sigmoid function to the sum of their input weights and inputs Wx, their state weights and the previous state Vh_{t-1}, and their bias b. The in-gate is somewhat more complicated as it has two parts that are combined by taking the Hadamard product. These are Equations 2.4 and 2.5.

$$\mathrm{in}_a(t) = \sigma(W_{\mathrm{in}_a} x_t + V_{\mathrm{in}_a} h_{t-1} + b_{\mathrm{in}_a}) \tag{2.4}$$

$$in_b(t) = tanh(W_{in_b} x_t + V_{in_b} h_{t-1} + b_{in_b}) \tag{2.5}$$

$$in(t) = in_a(t) \times in_b(t) \tag{2.6}$$

We now can calculate the long-term memory state $s(t)$ and the hidden state $h(t)$.

$$s(t) = for(t) \times c_{t-1} + in(t) \tag{2.7}$$

$$h(t) = out(t) \times tanh(s(t)) \tag{2.8}$$

With the addition of three gates and a long-term state, LSTMs help the network perform better on longer series.

The final improvement for recurrent networks is bi-directional networks. In traditional RNNs, the network can only look backward and not relate what it sees to what comes next. While this is suitable for some scenarios to determine the context of text data, it is necessary to consider the entire sentence, not simply what has come before. Bi-directional LSTMs (BiLSTMs) are one of the most popular bi-directional RNNs. To create a BiLSTM, you simply have an LSTM layer for both directions and combine the output of the neurons at each time step.

2.1.3 Encoder-Decoder

In natural language processing, several problems require a sequence of words to be transformed into a sequence of different words. Standard neural networks can struggle to accomplish this task [8]. Encoder-decoder neural networks were introduced to help with this task. In addition to helping with tasks such as machine translation researchers have discovered that using an encoder-decoder structure

a general representation of a language can be learned. The BERT model that will be discussed in 3 is of particular interest to this research. First, we will explore encoder-decoder models here, and in section 2.1.7, we will examine transformers. An encoder-decoder network can be broken into three important parts: the encoder, the encoded vector, and the decoder.

The encoder is composed of an RNN cell that will read each input symbol sequentially. The goal of the encoder is to read a sequence of any length and to create the encoded vector that is a specified length. This is accomplished by having the hidden state of the RNN be updated after each symbol. Cho et al. proposed a new method to update the RNN cell in the encoder. Their method can be seen in 2.9. Their method improved the RNN's ability to forget while remaining simpler to compute than an LSTM cell [9]. This method uses a reset gate r_j and an update gate z_j, seen in equations 2.11 and 2.12 respectively. Where the j subscript denotes the j-th element of a vector. While the network is reading through the input symbols, we discard the outputs from the RNN. After reading the entire input sequence, the hidden state of the encoder RNN will be the encoded vector.

$$h_j^t = z_j h_j^{t-1} + (1 - z_j)\tilde{h}_j^t \tag{2.9}$$

where $\tilde{h}_j^t$ is:

$$\tilde{h}_j^t = \tanh([Wx]_j + [U(r \odot h_{t-1})]_j) \tag{2.10}$$

$$r_j = \sigma([W_r x]_j + [U_r h_{t-1}]_j) \tag{2.11}$$

$$z_j = \sigma([W_z x]_j + [U_z h_{t-1}]_j) \tag{2.12}$$

In these equations, W_r and U_r are learned matrices. The logistic sigmoid func-

tion is represented by σ.

The encoded vector is a vector that summarizes the input sequence. This section of the Encoder-Decoder model is worth highlighting as this is the area that BERT focuses on. This vector aims to encapsulate the information that was contained in the input sequence. This vector acts as the initial hidden state of the decoder. There has been additional research in providing the decoder with more information about the input sequence that will be discussed in section 2.1.6.

The decoder is another RNN that is trained to generate the next word in the target sequence. The hidden state of the decoder is initialized as the encoded vector. As the decoder proceeds from one output to the next, the hidden state evolves. This can be seen in equation 2.13.

$$h_t = f(h_{t-1}, y_{t-1}, c) \tag{2.13}$$

Where c is the encoded vector, and f is some activation function that provides valid probabilities.

2.1.4 Transfer Learning

Unlike traditional neural networks, encoder-decoders work by exploiting the idea of transfer learning. The main idea behind transfer learning is that knowledge learned from one task should transfer to a similar task. This parallels human learning, where knowledge from one domain can be transferred to another if they are similar enough. This is useful for tasks where there is a lack of labelled data for the target task, but there is a large amount of data in a related task. Specifically for NLP, there is a large amount of unlabeled text available. Still, when we want to focus on a specific task such as sentiment analysis, question answering, named entity recognition, etc., we can run into scenarios where there is a lack of labelled

data available.

Transfer learning can be broken down into two sub-groups: transductive and inductive transfer learning. Inductive transfer learning is used in NLP, where the target task differs from the source task. Inductive transfer learning can be further sub-grouped into multi-task and sequential task learning. Of these two, sequential task learning is the most common in NLP. Unlike multi-task learning, there is only one target task in sequential task learning. We will explore the specifics of how sequential task learning works in the following section.

2.1.5 Sequentail Task Learning

STL can be broken down into two main stages: pretraining with source data and adoption with the target data. Ideally, the source data will be similar to the target data. We will break these two stages down in more depth.

2.1.5.1 Pretraining

In the pretraining stage, the network is presented with data that is similar to the target data, but the task that the network is learning is different. In NLP, this generally looks like training the network on unlabeled data to learn a general representation of the target language. This can be done with a variety of different source training tasks, but they are generally some forms of word prediction. A common pretraining task is next-word prediction. This involves training the network to create a probability distribution of all words, with the network predicting which words are the most likely to occur next in the sentence. Another task that has been used is next-sentence prediction, where, in addition to learning which words follow each other, the network is trained to identify which sentences follow each other. A drawback to this kind of learning is that you still require a large amount of unlabeled data, as seen in [When Do You Need Billions of Words of Pretrain-

ing Data]. For classification tasks, you need upwards of one billion words for the pre-trained model.

2.1.5.2 Adoption

After pretraining the model, the next step is to decide how to transfer the information to the target task. There are two approaches to facilitate this transfer: fine-tuning or embedding/feature extraction. The main distinction between these two is if the weights of the pre-trained model are adjusted in the new task (fine-tuning) or if the weights are kept as is and a new model is trained (embedding/feature extraction).

Fine-tuning is the more flexible of the two approaches as it does not require any specific changes to the pre-trained models' architecture. It can be accomplished by selecting specific outputs representing the target task or adding a final layer to compute the needed task [10]. One of the main drawbacks of fine-tuning is that the relationships between the tokens in the pre-trained network can be lost as the model learns the specifics of the target task. This is called "catastrophic forgetting," and there are some proposed solutions, including freezing learning rates and regularization.

In contrast to fine-tuning, the embedding approach freezes the weights of the pre-trained model. This allows the creation of a fixed-sized representation of the tokens to be extracted. For NLP, you can create word embeddings of a fixed size representing the word and its context regarding all other words in the pre-trained models' data set. You can build a new network from this embedding to take this contextual information and use it to complete the target task. A drawback of the embedding approach is that if the usages and meanings of words change between the pretraining and adoption stages, there is no way to update the embeddings.

2.1.6 Attention

In natural language processing, attention is the networks' ability to know which words relate to each other. The relationship between words can be demonstrated best with an example: consider the sentence:

"The chicken did not cross the road because it was tired."

Determining what the word "it" refers to requires the ability to recall previous information. Specifically, the network must know that the chicken is the associated noun, not the road. Basic attention accomplishes this by having different components share information. As the encoder-decoder section mentions, a neural network without attention will create a context vector c to predict outputs. One of the downsides to this vector is that it lacks information on how the different words relate to each other. Attention mechanisms work to add this information back to the vector. There are two general types of attention: inter or cross attention.

2.1.6.1 Inter-attention

Inter-attention networks are generally encoder-decoder models that are frequently used in machine translation. The context vector is created so that the decoder understands which inputs could influence the correct output. Hence, the name inter-attention refers to the interdependencies between the encoder and decoder networks.

The context vector is changed by adding a vector of hidden states. The decoder will use these context vectors to help create their hidden state. This gives the decoder a unique context vector for each time step. The context vector contains the dependencies between the current word and all the other words in the input. The context vector adds an alignment score to the summation of the hidden states as

seen in Equation 2.14.

$$c_t = \sum_{i=0}^{T_x} \alpha_{t,i} h_i \tag{2.14}$$

The hidden state h_i concatenates the bidirectional states for each element i. Calculated as:

$$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}] \tag{2.15}$$

The alignment score α_t represents how relevant the input is to the current time step of the decoder. This is calculated by taking the softMax of the score function as seen in Equation 2.16. This score function can be calculated in a variety of different ways, including concat, general, location-based, and dot-product seen in Equations 2.17, 2.18, 2.19 and 2.20 respectively.

$$\alpha_{t,i} = \text{softMax}(\text{score}(s_t, h_i)) \tag{2.16}$$

$$\text{score}(s_t, h_i) = v_\alpha^T tanh(W_\alpha[s_t; h_i]) \tag{2.17}$$

$$\text{score}(s_t, h_i) = s_t^T W_\alpha h_i \tag{2.18}$$

$$\alpha_{t,i} = \text{softmax}(W_\alpha s_t) \tag{2.19}$$

$$\text{score}(s_t, h_i) = s_t^T h_i \tag{2.20}$$

W and v are both learned matrices. This process will allow the attention mechanism to determine how each word relates to all other words in the sentence.

With the context vector calculated, we can now determine the current state of

the decoder. This is accomplished by applying the decoding function to the previous state, previous output and the context vector. This function may be multiple neural layers, but it will ultimately provide the state as seen in Equation 2.21. This state will be used to calculate the correct output by being fed into an additional neural network.

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{2.21}$$

2.1.6.2 Intra-attention

Intra-attention or self-attention networks do not generally have an encoder-decoder structure. These networks can be used for a variety of tasks that are not suitable for an encoder-decoder structure, including sentiment analysis.

Self-attention is calculated using three sets of vectors: query, key and value. These vectors are obtained by multiplying the word vector with three different weight matrices seen in 2.22.

$$\begin{bmatrix} Q \\ K \\ V \end{bmatrix} = H \begin{bmatrix} W_Q \\ W_K \\ W_V \end{bmatrix} \tag{2.22}$$

Where H is the hidden states of the output layer. The values of the $W_Q, W_K,$ and W_V matrices are learned during the network training. The final out layer is calculated by multiplying these matrices together as seen in 2.23.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \tag{2.23}$$

Where d is the dimensionality of the layers and $\sqrt{d}$ is the scaling factor.

Self Attention has been shown to help improve various NLP tasks, including machine translation and linguistic probing.

2.1.6.3 Multi-Headed Attention

Normal attention will learn the relationships between the words in the sequence using the entire word embedding. While this is effective, it can lose some of the nuance encoded into the embedding. Different parts of the embedding may be capturing different aspects of the word. Multi-headed attention looks to solve this problem by splitting the embedding into separate sections. Its attention head attends to each of these sections. In [11], the authors propose using eight attention heads and splitting the embedding dimension evenly between these heads. This changes the attention calculation into eight different attention calculations as seen in 2.24 that are concatenated together at the end 2.25.

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.24}$$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^A \tag{2.25}$$

The $\mathrm{Attention}$ function is similar to the one described in 2.23. The only change is that the $\sqrt{d}$ becomes $\sqrt{d_k}$. This is because the attention heads no longer examine the entire embedding space. They are now examining sections of size d_k. This d_k is calculated by taking the model's dimensions and dividing it by the number of heads.

2.1.7 Transformers

As language models have progressed, the computing constraints of using various RNN cell structures have become problematic. The desire for a more efficient neural network model has increased as the sequence length, and the number of sequences required to train have increased. This is where the Transformer was introduced in [11]. The transformer does not use any RNN cell or any convolutions.

This allows it to have a large amount of parallelization, significantly increasing training efficiency.

Transformers are encoder-decoder networks that are built from stacked, fully connected layers. The encoder is built from six layers. These layers have two components. The first is a multi-headed self-attention mechanism. The second is a position-wise fully connected network. A position-wise neural network uses the same layers to transform all the words from the input sequence [12]. This can be seen in equcation 2.26, and can be explained as two sets of matrix multiplication with a ReLU activation between them. In addition, each sub-layer has a residual connection that allows the input from each layer to bypass the network and be included in the output. This makes the output of each sub-layer:

$$PWN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.26}$$

$$SublayerOut = SubLayerNorm(x + Sublayer(x)) \tag{2.27}$$

The decoder is composed of six identical layers. These layers are composed of three sub-layers. The first and last sub-layers are the same as the encoder. The second sub-layer is composed of an additional multi-headed attention layer. The attention layer takes its Q and K values from the encoder, while its V value comes from the first sub-layer. In addition, the attention mechanisms in the decoder are modified to ensure that they only attend to words in the sequence that have already been seen. This prevents the decoder from looking into the future and ensures it only uses information that should be available.

Since transformers do not use RNNs or any convolutions, they need to add additional information about the position of the words in the sequence. This is

accomplished using the sine and cosine functions:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{2.28}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{2.29}$$

where pos is the word's position in the sequence, and i is the dimension. These positional encodings will have the same dimension as the model, allowing them to be summed into word encoding before being fed into the encoder or decoder.

With this setup, the authors found that the proposed transformer was nearly identical or better than all previous models. In addition to being competitive, their model requires significantly less training time than previous models.

2.2 Pre-processing

Before the computer can start to create a language model, it is important to pre-process the text. This entails various activities, including but not limited to removing unwanted symbols, stemming or lemmatization, and removing stopwords. A typical text analysis application will use some pre-processing techniques to ensure that the text being used is suitable for their chosen model.

Removing unwanted symbols is a common practice when the text being analyzed is collected from an environment where it is common for unusual or unique characters (or combinations of characters) to be included in the text. For example, if the text corpus is collected from Twitter, it is common to remove the @ sign as it is used to identify a specific account. When the data was collected using the Twitter API, it is typical for the character "RT" to be appended to the beginning of the text in certain circumstances. Removing these characters helps the model avoid

learning relationships that are overly specific to where the data was collected.

Stemming and lemmatization are techniques used to convert words with similar meanings into a single word. Stemming is the process of removing characters from a word to reduce it to a base form known as a lemma. This process is generally accomplished by following rules that can result in misspellings. For example, the words *funny* and *funnier* can be stemmed into the word *funni*. Lemmatization is similar to stemming in that the end goal is to convert words to a simpler lemma. However, lemmatization considers the context of the word and can replace the entire word with a lemma if one is suitable. An example of lemmatization would be the nouns *leafs* and *leaves*, both converting to the word *leaf*. By utilizing stemming and lemmatization techniques, it is possible to drastically reduce the number of words the language model needs to learn, thus helping both performance and run time.

Removing stopwords from the text is another process that can reduce the overall number of words that the model is exposed to and required to learn. Stopwords are words that have been identified in a language as common and not necessary for the content of the text to be learned. Some examples of stopwords for English are *the*, *a*, and *as*. Removing these words from the text reduces the number of words that are required to be learned and helps reduce the time required to run the model.

2.3 Language Models

Converting text into something a computer can understand is one of the most important parts of any text-processing application. Creating a systematic representation of a language is called creating a language model. Language models are a probabilistic representation of the language. This means that the model is trained

to determine the probability of words in a given sentence. The simplest LM is an n-gram model. These LMs are models trained to predict the next word based on the previous n words. This can be accomplished by using Equation 2.30.

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})} \tag{2.30}$$

Where $w_n|w_{n-N+1:n-1}$ is a word that is proceeded by N other specific words. $C(x)$ is a function that returns the frequency of a sequence of words. Several improvements have been proposed over the years; however, with the rise of neural networks, neural LMs have become more popular.

Neural language models (NLMs) are a type of LM created by a neural network. These models create embeddings to represent the text. These embeddings allow the LM to encode additional information about the language. Similar words will be encoded closer together in the embedding space, allowing the model to understand the language better while requiring less training data. A popular approach with neural networks is to use pre-trained language models. These pre-trained LMs can provide significant advantages compared to network learning in representing the language alongside their prescribed task. Pre-trained LMs leverage transfer learning, the idea that knowledge learned on one task can be transferred to a similar task. This applies to language models by training the pre-trained model on a general language task such as next sentence prediction or word prediction and then taking that model and fine-tuning it to more precise tasks. An additional advantage of pre-trained LMs is that they can be pre-trained on large unlabelled datasets. This is because the tasks they are initially learning can be trained using a self-supervised approach, eliminating the requirement of labelled data. Several different pre-trained LMs have recently become popular with these advantages [13, 14]. Of particular interest to our work is the BERT family of pre-trained LMs. We will introduce BERT here, and in Chapter 3, we will further explore the

specific architectures we will use.

In their work, Devlin, Chang, Lee, and Toutanova introduce BERT [13], an LM trained on a corpus composed from the BooksCorpus [15] and Wikipedia that totalled over 3 billion words. The main advantage of BERT over previous pre-trained LMs is that it is a fine-tuned bidirectional model. Being a fine-tuned model means that when BERT is adapted to a specific task, the only change that needs to happen is for an additional layer to be added to the output of BERT. This layer is trained on the specific task instead of the entire model. A bidirectional model means that BERT has been trained to look ahead in the sentence and behind when analyzing a particular word. The two tasks that BERT is pre-trained on are the two mentioned above. Training to predict masked words is fundamental to BERT's success. Without masking words, BERT would struggle to learn an accurate representation of the language. In a traditional unidirectional LM, when considering a word in a sentence, you can train it to predict what the next word most likely is. However, in a bidirectional model, the word has access to too much information about itself and to what words would come next in the sentence. Because of this, the model would struggle to learn an actual representation and not overfit the given data. The solution to this problem is to mask random words in the sentence and ask BERT to predict what that word would be. The authors tested various masking strategies that varied how a word is masked. The authors mask fifteen percent of the total words in each sentence. Each masked word has an eighty percent chance that the target token is replaced with the "MASK" token, a ten percent chance that the target token is replaced with a random token and a ten percent chance that the target token is replaced with the original token. The authors found that these percentages resulted in the best overall performance of the model.

2.4 Sentiment Analysis

Determining if a piece of text expresses positive or negative sentiment can be used in many fields, including security, finance, and medical [16]. This process is called Sentiment Analysis (SA) and has been an area of research since 1940 [16]. There are two broad categories for sentiment analysis: binary (positive or negative) or ternary (positive, negative, neutral). The specific models may differ between the two categories, while general strategies remain similar. Over time, the methods used have evolved into the sophisticated ones used today. Alongside the models' evolution, the domains in which they were being used evolved as well. The growth of Web 2.0 opened up new and exciting areas for applying sentiment analysis. The rise of social media, online blogs, and companies moving customer reviews online created a vast source of potential training data for SA. While this data is easily available, the amount of data gave rise to a new problem: acquiring high-quality labels is expensive. Coupling this with the fact that the best-performing models are neural networks that require a large amount of labelled data to train creates a problem for developers and researchers. There have been a variety of different solutions proposed, including:

- Creating models more resilient to noise.
- Improving the quality of labels acquired from cheap sources.
- Creating artificial data from high-quality labels.
- Reducing the number of labels required for current models.

For our purposes, we will focus on the last two ideas.

2.5 Improvements

One of the main drawbacks when creating a SA model is the requirement for a large training set. While there are several publicly available datasets, SA is domain-dependent. You must create your own if you are working on data that is not similar to any public datasets. Considering that it is expensive to obtain high-quality labels, ensuring that you are labelling the correct pieces of data and getting the most out of your labels is important. The areas of data augmentation and lexical expansion work to solve these problems.

2.5.1 Active Learning

In machine learning, Active Learning (AL) is a machine learning algorithm that chooses the data from which it will learn [17]. The main idea behind these algorithms is that if you allow the classifier to choose which data points to label, you will achieve a higher level of performance while requiring fewer labels. To train a classifier using AL, we first need two sets of data: U, the unlabeled data, and L, the labelled data. We will train our classifier C on L: $train(C, L)$. After completing this training step, we need to query U for the data we want to add to L to improve the classifiers' performance. This requires the creation of some query function f that will return a value that can be used to compare the data points in U. There are several different query strategies, including uncertainty sampling, query-by-committee, and expected model change. For our research, uncertainty sampling is the most important. The most common uncertainty sampling strategy is entropy:

$$E(x) = -\sum_{i \in Y} p_i(x) \log(p_i(x)) \tag{2.31}$$

where Y is the set of all possible labels and $p_i(x)$ is the probability that x belongs to label i. This allows us to create our function f as:

$$f(S) = max(E(x), x \in S) \tag{2.32}$$

where S is a data set for which the classifier has provided labels. With this function, our classifier can provide labels for U and pass U and the labels to f. This will give us the next data point to get the correct label. This process is repeated until the desired performance is reached.

In traditional AL, new data will be added one at a time. However, this is impractical in practice. Labelling potentially hundreds of data points one by one is expensive, and some classifiers will overfit if trained on datasets where there has been little change between one step and the next. This leads to the idea of batch learning. Batch learning is where the classifier will select multiple data points at each step to learn from. This will modify Equation 2.32 to return multiple points. While this can lead to some data points being labelled when they did not necessarily require it, the benefits achieved between cost reduction and preventing overfitting outweigh the cost.

2.5.2 Data Augmentation

Data Augmentation (DA) expands the provided training data to increase performance without manually labelling more data. It is widely used in computer vision and has started to be used more in other areas, including text classification tasks like sentiment analysis. There are a variety of different types of DA. However, we will be focusing on word-level replacement methods. These methods focus on how many sentiment analysis techniques work on a word level. This means that the classifier attempts to determine a representation of the individual words in the dataset and decide how the words relate to the label. Over the training steps, the classifier will encounter words and learn this representation. Problems arise

when the classifier encounters words in the testing stage that were not known in the training stage. When this happens, the classifier cannot use the new word and discard it. This can lead to losing large amounts of information, especially when the training data set is small. Word Level Data Augmentation looks to solve this problem by introducing a wider variety of words in the training step to ensure that the classifier can recognize as many words as possible.

Two main approaches have been proposed for word-level replacement in sentiment analysis. The first utilizes word embeddings, and the second depends on a thesaurus. There are a variety of different approaches that utilize word embeddings. These range from generating data to balance classes to using cosine comparison and k-nearest neighbours to select words for replacement. The other methods utilize thesauruses to replace selected words with their synonyms. The selection of the original words has been an area of active research. We have chosen to implement the PLSDA algorithm that selects words belonging to specified parts of speech and ensures that all replacement words belong to the same part of speech.

Chapter 3

Related Research

In this chapter, we will discuss the research directly related to ours. We will examine the dataset on which we chose to train our NNs. Then, we will introduce the different NN structures that we are comparing, and finally, we will look at the AL and DA techniques we are combining.

3.1 Data set

The data set we trained on is the IMDB Movie Ratings Sentiment Analysis dataset that can be found from [18]. This is a publicly available dataset that contains a large number of labelled movie reviews. There are 20019 negative reviews and 19981 positive reviews.

3.2 Literature Review

3.2.1 Neural Networks

There are a wide variety of neural networks used in natural language processing. The most common structures are CNN, LSTMs and pre-trained networks. We have

implemented one of each of these networks. We will introduce the networks that we have based our implementations on here. We discuss the exact structures of our implementations in Chapter 5.

We are comparing three different neural network structures. These are a stacked one-dimensional CNN proposed by Dang, Moreno-Garcia, and De la Prieta in [19], the second is a convolutional LSTM from the work of Behera et al. in [20]. The last NN is a BERT classifier based on the work of Devlin, Chang, Lee, and Toutanova in "BERT: Pre-trainings of Deep Bidirectional Transformers for Language Understandings." [13]

3.2.2 Stacked 1D CNN

The first neural network we use for our research comes from Dang, Moreno-Garcia, and De la Preieta's work in "Sentiment Analysis on Deep Learning: A Comparative Study." [19] The authors reviewed many different SA techniques and model structures in their work. They compare the performance of three different neural network structures and two sentence representation techniques across eight datasets. From these comparisons, the authors found that using a stacked CNN structure, you can create a SA model that achieves competitive performance while taking at most 65% of the time to train as the top-performing model. The F-score and run time comparisons can be seen in Table 3.1. The neural network structure can be found in Table 3.2.

The run times reported in Table 3.1 come from running the selected neural network structure on 100% of the data. While the authors found that the RNN outperformed CNN, the run times are much longer. This increase in run time can make the RNN unsuitable for certain tasks where getting results quickly is an important factor for the model. Overall, the authors' comparisons in this work provided us with a CNN structure that has been shown to perform well on various datasets.

Table 3.1: CNN vs. RNN

Datasets	CNN F Score	RNN F Score	CNN Time	RNN Time
Sentiment140	0.8006	0.8297	7 min 3 s	1 h 4 min 16 s
Tweets Airline	0.9406	0.9406	1 min 22s	2 min 41 s
Tweets SemEval	0.8288	0.8387	1 min 11 s	2 min 43s
IMDB Movie Reviews (1)	0.8591	0.8702	33 s	7 min 42 s
IMDB Movie Reviews (2)	0.8273	0.8697	37 s	8 min 23 s
Cornell Movie Reviews	0.7156	0.7759	21 s	4 min 40 s
Book Reviews	0.7773	0.7340	21 s	4 min 40 s
Music Reviews	0.7403	0.7321	17 s	4 min 42 s

Table 3.2: Stacked 1D CNN

Layer	Output Shape	Parameters
Embedding	40, 300	4500300
1D Conv	40, 64	57664
1D Conv	40, 32	6176
Max Pooling 1D	13, 32	0
1D Conv	13, 16	1552
1D Conv	13, 8	264
Global Avg 1D	8	0
Dense	1	9

3.2.3 CoLSTM

To create our Convolutional Long-Short Term Memory Neural network, we combined the ideas presented in the the following papers:

- ”Co-LSTM: Convolutional LSTM model for sentiment analysis in social big data” by Behera et al. [20].
- ”Multi-channel LSTM-CNN model for Vietnamese sentiment analysis” by Vo et al. [21]

We will explore these works in this chapter, and in Chapter 4, we will introduce how we have combined these ideas.

3.2.3.1 Behera et al.

With the introduction of their Co-LSTM model, Behera et al. provide an exciting framework for creating a neural network to perform sentiment analysis [20]. The model that the authors propose combines CNNs and LSTMs into a single network to leverage the advantages that the different layer types bring. The authors use a CNN layer to capture the relationships between different words. The LSTM layer is used to help identify how these different relationships interact with each other and how they relate to the overall sentiment of the input. The model proposed by the authors has six layers. The first is a standard embedding layer that embeds the input into 128-dimension space. The subsequent two layers are the convolutional layer and the max pooling layer, respectively. The authors use seven 3x3 filters for their convolutions. After the max pooling layer comes the LSTM layer. The output from the LSTM is then fed through two fully connected layers to determine the final label for the input.

3.2.3.2 Vo et al.

In their work, Vo et al. introduce a neural network structure that combines CNNs and LSTMs [21]. Of particular interest to our work, the authors propose a CNN structure that utilizes multiple kernels of different sizes. This allows their network to identify relationships between multiple different combinations of words. The authors created an LSTM-CNN network by having a separate LSTM and CNN network, concatenating the outputs from each and then feeding this through a final dense layer. For both the CNN and LSTM networks, they use word embeddings with a dimension of 200. They then use three different kernels to create 450 different convolutions. There are 150 convolutions of sizes 3, 5, and 7. The max of each output is selected and fed through a dense layer to get the final output with a dimension of 100. The LSTM network feeds the word embeddings into an LSTM layer with 128 nodes. The output of this matches the dimensions of the CNN network at 100. Finally, these two outputs are concatenated and fed into a final neural network that will predict the sentiment of the original input.

3.2.4 BERT

We have chosen the ALBERT model for our work [10]. We will summarize what BERT is here and the improvements that ALBERT offers. In Chapter 5, we will explore how we have used ALBERT for our experiment.

3.2.4.1 Original BERT

BERT (Bidirectional Encoder Representations from Transformers) was proposed in [13] in 2018. The main idea behind BERT was to improve on previous ideas of unsupervised pre-training by utilizing bidirectional architecture, attention, and transformers. Devlin, Change, Lee, and Toutanova achieved this by stacking many

transformers on top of one another and training these on two unsupervised tasks. BERT was trained on a masked word prediction task and next-sentence prediction. In the word prediction task, BERT was required to predict which word should occur in a sentence when a word has been hidden. For the next sentence task, BERT must predict if sentence A is followed by sentence B. The authors demonstrate that these tasks create a pre-trained model that can be used as the basis for many other more complicated NLP tasks.

After the model is pre-trained, it will have learned a general representation of the text in the training materials. This representation can then be used for the target task. To use the pre-trained representation, all that is needed is to supply the BERT model with the new task inputs, take the outputs from the model and feed them into a classifier. The classifier will be trained as usual and only need to learn how to solve the new task, not how to represent the language and the new task. BERT is pre-trained on large amounts of unlabeled data. The representation that BERT learns can be distributed. This saves training time as only one base is needed to create many different classifiers. This is one of the main advantages that BERT offers. Overall, the ability of BERT to learn accurate representations of a language and to transfer this learning to task-specific models can be a significant advantage when creating accurate models in many NLP domains.

3.2.4.2 ALBERT

One of the main drawbacks that arose from the base BERT model is that the pre-trained BERT model is large. This comes from the fact that it contains a large number of parameters that are used to represent the language. Lan et al. proposed ALBERT (A lite BERT), a model that reduces the number of parameters needed in the final pre-trained product [10]. Their research aimed to determine if having a larger model is required to produce good results for NLP tasks. The authors propose

two changes to the architecture of BERT and one change in how the model is pre-trained. The first proposed changes to the architecture were to reduce the number of embedding dimensions needed by taking the encodings into a lower-dimension space before embedding them. The authors found that this enhanced the model's performance in all the test cases. The authors proposed the second change to allow information to be shared between the different layers. Unlike the first change, the authors found that this caused the results of their models to lose performance. Finally, the authors tested a pre-training method different from the original BERT. The authors changed the sentence prediction training from next-sentence prediction to sentence ordering; instead of simply determining if one sentence occurs after another, ALBERT is trained to determine the order of sentences. They found that this change in pre-training increased the performance in all other tasks. Overall, the authors found that their model offers better performance when compared to the BERT models that contain a similar number of parameters.

For our research, we use the ALBERT implementation that TensorFlow provides [22]. We used this network as our baseline to compare the performance of our proposed algorithm.

3.2.5 Active Learning

Active learning in natural language processing is a diverse field with many different areas of active research. These include a large variety of querying strategies, annotation techniques, and various structures for the network and techniques for learning [23]. We will briefly explore the available research in these areas before exploring the research we use directly in our work.

We will start by examining the three general types of querying strategies: informativeness, representativeness, and hybrid [23]. These strategies are all involved in deciding how you select information from the unknown data set.

Informativeness strategies involve using different sampling methods that examine each data point individually. These strategies can range from uncertainty sampling strategies such as entropy-based to local divergence [24, 25] to gradient [26] and performance prediction methods [27]. The general motivation behind these strategies is to optimize how the new information is added to the model.

Representativeness strategies help to deal with the fact that informativeness selection strategies can be susceptible to sampling bias and outlier selection [23]. The representativeness strategies include density selection, discriminative selection and batch diversity [23]. Density selection attempts to avoid outliers, where data points are selected based on the points' average similarity to all other points. Discriminative selection selects samples that are different from data points with labels. This will hopefully give the classifier a good representation of the entire data set. Batch diversity strategies work to optimize the ability to select multiple data points at once. In other selection strategies, only a single data point is selected for each iteration. While this can ensure optimal data points are selected each iteration, acquiring labels one at a time can be expensive. Supplying the labellers with a more extensive selection of data points helps ensure their time is used well.

Hybrid strategies are a combination of the previous two. These strategies aim to combine the advantages of the other strategies and minimize the disadvantages. There are numerous different ways to combine informative and representative approaches. Common combinations include entropy and density combinations [28, 29] or combinations of uncertainty, representativeness, and diversity [30]. Many of these strategies naturally combine different approaches. Examples of these are weighted clustering or filtering for uncertain samples and then clustering to select a diverse set of the samples [23]. While these natural combinations can provide advantages, dynamic combinations can further improve the performance of active learning algorithms. Dynamic combinations are combinations where the

specific selection method evolves or changes over time. For example, a representativeness sampling method should be used at the start of the active learning process, then changing to uncertainty sampling as you acquire more data can be helpful.

While tasks like sentiment analysis and other classification tasks that require the classifier to provide a single label can be costly to train, other tasks like event extraction and named entity recognition are even more expensive to label. Different annotation strategies can help reduce the costs for these types of tasks [31]. While different annotation strategies are an essential aspect of AL, we will focus on the querying strategy we used in our research.

3.2.5.1 Exploration-Exploitation

In their work "Deep similarity-based batch mode active learning with exploration-exploitation," Yin et al. identified two limitations that previous AL algorithms presented. The first limitation was that the similarity measure used to compare instances was a feature space. The second limitation was that previous algorithms focused heavily on the "exploitation" of the data. This means that other techniques focused on information close to the decision boundary. The authors argue that focusing too heavily on the data around the decision boundary will limit the algorithm's "exploration" of the entire data distribution. With these limitations in mind the authors propose a AL algorithm that utilizes two different equations to ensure that the entire distribution is explored while ensuring that the decision boundaries are sufficiently explored.

The first equation is used in the exploitation step:

$$I(x) = E(x) - Sim(x, S) \tag{3.1}$$

where $Sim(x, S)$ is the similarity between an element x and the set of selected

elements S, this equation calculates the amount of information each data point contains. To use Equation 3.1, we first need to construct the set S. This set will represent the selected data, and we will initialize it with the data point that maximizes Equation 2.31. We will calculate Equation 3.1 for the U and select the element with the maximum value to add to S. We will repeat this step until we have added a total of $num_{exploit}$ data points. After we complete the exploitation step, we have ensured that we have added the top m points that contain the most information. The sudo code for this step can be found in Figure 3.1. The algorithm will now move into the exploration step.

Given: U, $num_{exploit}$
Find $x \in U$ such that $x = max(E(x))$
$S = set(x)$
$U = U - x$
While $|S| < num_{exploit}$:
    For $i \in U$:
        $I(i) = E(i) - Sim(i, S)$
    End For
    $x = max(I(x))$
    $S = S \cup x$
    $U = U - x$
End While
Return S, U

Figure 3.1: Exploitation Code

The equation used in the exploration step is:

$$i = min_i Sim(i, L \cup S) \tag{3.2}$$

This equation ensures that the element we select is the most dissimilar in the dataset. We will add i to S and recompute Equation 3.2 $num_{explore}$ times. The sudo code for the exploration step can be found in Figure 3.2.

Given: U, L, S, $num_{exploit}$, $num_{explore}$
$total = num_{exploit} + num_{explore}$
While $|S| < total$:
    For $i \in U$:
        $I(i) = -Sim(i, L \cup S$
    End For
    $x = max(I(x))$
    $S = S \cup x$
    $U = U - x$
End While
Return S, U

Figure 3.2: Exploration Code

After completing the exploitation and exploration steps, we can retrain our classifier and calculate the new performance. We repeat these steps until we achieve the desired performance. We perform minor modifications to the Exploration-Exploitation Algorithm for our research. These changes are discussed in Chapter 4.

3.2.6 Lexical Expansion and Data Augmentation

As discussed previously, one of the main challenges that NLP models can face is a lack of diversity in the training vocabulary. This is especially prevalent when the number of training samples is limited. LE and DA techniques aim to fix this problem by introducing a wider range of words into the training data. The motivation behind this is simple: unknown words cause errors. Following the work of [32], we can see four main areas of DA. These are token-level augmentation, sentence-level augmentation, adversarial argumentation, and hidden-level augmentation. We will examine these individually before highlighting the algorithm we have implemented for our research.

Starting from the individual token level, LE has been used to improve the per-

formance of NLP techniques. This level of augmentation focuses on changing the individual words or simple phrases found in the training data. This can take the form of replacing words with synonyms to insert and delete words. Many approaches have been used to decide how to insert new words into existing sentences. These include using part of speech information to ensure that the new words fulfill the same role as the original [33], to changing the structure of the classifier to detect the changes that are made with the new data [34], or to incorporate the identification of target and non-target words with as well as ideas from image data augmentation to handle incorrect word insertions [35]. The idea that modifying the individual tokens of the training sentences can help improve performance while not requiring more labelled data has been shown to increase the performance of the classifiers. This approach can be expanded to a sentence level as well. Tasks like machine translation and summarization depend on words and a wider understanding of the language structure. Generating sentences in different contexts can improve the performance of these tasks. Instead of targeting the words in the training sentences, augmenting the data by examining what the classifier has been learning is possible. Adversarial augmentation techniques can be created by directly challenging the classifier with unknown data. These techniques train the model to be more robust and help increase the overall performance [36]. Finally, it is possible to create artificial data by augmenting the hidden representations of the words. This means that humans cannot read these new samples, but they still help improve the classifier performance. An example of this can be found in [37], where the authors combine two real samples to create a new virtual sample. LE and DA techniques all aim to create more data from the limited amount of available labelled data. For our research, we focused on a token-level technique, PLSDA.

3.2.6.1 PLSDA

Xiang et al. proposed the PLSDA algorithm to ensure that any generated data followed the syntactic consistency principle [33]. This principle is that any changes to the data do not change the syntactic information that the data originally contained. To ensure this principle is followed, the authors proposed the constraint of selecting words for replacement based on specified parts of speech and ensuring any selected synonyms are of the same part of speech. Choosing a word from a sentence and replacing it with its synonyms is accomplished over two steps. The first is the Substitution Candidate Selection step, and the second is the Instance Generation step.

The first step is choosing which words could be replaced with their synonyms. We will assume a training sentence S composed of n words: $S = w_1, w_2, ..., w_n$. We will also have their associated POS tag for all these words. The authors present two ways to select potential words from this set. The first finds all possible synonyms with the same POS tag as the original word. While slightly more complicated, the second provides a higher degree of quality in the generated sentences. The second method incorporates word similarity into the synonym selection process. After generating the synonyms for a selected word and filtering them based on their POS tag, there is an additional comparison of the similarity of the synonym to the original word. Only synonyms that are above a threshold are considered for possible replacement. The algorithm will go through all n words and determine whether synonyms should be generated. The sudo code for this step can be found in Figure 3.3. Once complete, the words and their synonyms will be passed on to the next step.

```
Given: S, pos, sim
SCL_S = set()
FOR each word w in S:
    SCL_w = set()
    w_p = POS(w)
    IF w_p ∈ pos:
        syns = synonyms of w
        FOR sy ∈ syns:
            sy_p = POS(sy)
            IF sy_p=w_p AND
            SIM(syn, w) ⩾ sim:
                SCL_W ∪ syn
            END IF
        END FOR
    END IF
    SCL_S ∪ (w, SCL_W)
END FOR
Return SCL_S
```

Figure 3.3: Substitution Candidate Selection

The second step in the PLSDA process is instance generation. This is where the artificial data is created. The number of possible new sentences that could be created is relatively high. Consider a sentence where three words fulfilled the POS restraints. If each of these words has five possible synonyms, then the number of potential artificial sentences is 5^3. Creating all these possible sentences is computationally expensive; more importantly, they would require a large amount of storage—the authors propose an additional constraint exp, which is the expected number of generated instances. The PLSDA algorithm will stop generating new instances once it reaches this number. If the number of possible combinations is less than exp the PLSDA will generate all possible combinations.

To ensure that a random sampling of all possible instances is selected, the authors propose using a probabilistic distribution to choose the initial word for replacement and a random selection of all the potential synonyms. Each word is

selected for replacement based on a Bernoulli distribution. The synonyms for each selected word are selected randomly from all possible synonyms, with each synonym being equally likely to be selected. By utilizing this method, it is possible to generate artificial instances of a sentence.

Using the PLSDA algorithm can greatly increase the number of training samples that a classifier can access without manually labelling more samples. We discuss the details of our implementation of PLSDA in Chapter 4.

Given: S, SCL_S
$InsGen$ = set()
ben_S = $BenDis(SCL_S)$
FOR index in ben_S:
    $S_{new} = S$
    IF $ben_S[index]$ == 1:
        w = $SCL_S[index][0]$
        SCL_W = $SCL_S[index][1]$
        $prob_W$ = $Prob(SCL_W)$
        FOR $index_B$ in $prob_W$:
            IF $prob_W[index_B]$ == 1:
                S.replace(w, $SCL_W[index_B]$)
            END IF
        END FOR
    END IF
END FOR
$InsGen \cup S_{new}$

Return InsGen

Figure 3.4: Instance Generation

Chapter 4

Algorithm

This chapter will examine the proposed algorithm's structure and setup. We have designed the algorithm modularly so that either the active learning or the lexical expansion can be used separately or together. Additionally, the selection of the neural network is independent of the algorithm. This independence allows the easy insertion of any appropriate network into the algorithm. We will examine the algorithm sequentially, starting with the preprocessing, then moving into the AL and LE and ending in the training stage. The final section of this chapter will summarize the contributions this research has made to advancing sentiment analysis research.

4.1 Preprocessing

Our preprocessing stage can be broken down into five steps:

1. Remove over length sequences.

2. Generate intermediate information.

3. Tokenization.

4. Build vocabulary.

5. Convert words to numbers.

6. Padding.

4.1.1 Over Length Sequences

The first step in our preprocessing stage is to remove overly long sequences. We do this because our neural networks have a set input length. We chose the input length to be 800 words. We chose this length as the average sequence length in our training set is 757. We ensured we kept most of the data by selecting a number larger than the average. Another approach would be to truncate the longer sequences than the input size. We wanted to avoid the noise this approach would introduce, as the longest sentence in the dataset is 8574 words long. In addition, we felt that there may be some differences between classifying long sequences and short ones. By removing overly long sequences, we avoid any of the potential differences between these tasks.

4.1.2 Tokenization

The tokenization process that we use is composed of four steps. We first remove special characters. The second step is to remove single characters. The third step is to remove numbers and stop words. The stop word list we use is the English stop words from nltk [38]. We then finally convert the entire sentence to lowercase. These four steps help ensure we offer our neural network the highest quality sequences.

4.1.3 Intermediate Information

The second step in the preprocessing stage is the creation of two intermediate information processes. These are required as the active learning stage needs to compare the similarity of unknown sequences with sequences that the classifier has seen. We examined two approaches to this step as proposed by the authors in [39]. It is possible to use BERT embeddings to take all the sequences into an easy-to-compare state. However, we felt that using the BERT embedding for the similarity comparison but not for the actual network representation was utilizing information the network did not have access to. We wanted to compare the performance of the active learning technique when the similarity between sequences was limited to information the network had access to. To do this, we used a latent semantic analysis approach. We take all the sequences in the training set and create a term-frequency inverse document frequency matrix. We then use a singular value decomposition to reduce the dimensions of this matrix to 100. With this final matrix, we can easily compare any unknown sequence by running it through the same process. Any words unknown to the network will be given the same label, and similarity will only be calculated based on available information. This process can be seen in 4.1. We wanted to avoid using the network-learned embeddings as these are actively being learned and would be a poor representation of the information until the network is more established.

Define $svd()$ as:
Given: $dataTrain$, $dataTest$, $tfidf$, svd
$temp = tfidf.fit\_transform(dataTrain)$
$svd\_train = svd.fit\_transform(temp)$
$temp = tfidf.transform(dataTest)$
$svd_test = svd.transform(temp)$
Return svd_train, svd_test,

Figure 4.1: Create SVD Code

4.1.4 Build Vocabulary

Using SVD to determine the similarity between two sequences requires maintaining a vocabulary for the words we have seen. We do this by uniquely mapping the words in the training set to numbers. Additionally, we need to be able to map the numbers back to the words. When we add a sequence from the test set to the training set, we update the mapping with any new words. This process can be seen in algorithms 4.2 and 4.3.

```
Define vocab() as:
Given: dataTrain, STOPWORDS as SW
index = 1
map = dictionary()
indecies = dictionary()
For sentence ∈ dataTrain:
    For w ∈ sentence:
        If w ∉ SW and w ∉ map
            map[w] = index
            indecies[index] = w
            index++
        End If
    End For
End For
Return map, index, indecies
```

Figure 4.2: Create Vocab Code

4.1.5 Convert Words to Numbers

After building the vocabulary, we convert all the words to numbers. Using the dictionary created from 4.2, words that we present in the training set are replaced with numbers. Words that are not present in the training set are removed. This is applied to both the training and the testing set. This results in sequences that are of varying length. Our neural networks require sequences to be the same length;

```
Define addVocab() as:
Given: sent, map, index, indices, SW
For w in sent:
    If w ∉ SW and w ∉ map:
        map[w] = index
        indicies[index] = w
        index++
    End If
End For
Return index
```

Figure 4.3: Add Sequence Code

this is handled by the next step: padding.

4.1.6 Padding

Padding is a simple preprocessing step that ensures all sequences are the same length. This is accomplished by inserting zeros at the end of any sequences that are shorter than needed. It is essential to mask these zeros from the neural networks to ensure they are not learning from these extra inputs. This is why in 4.2 our indexing starts from one not zero.

4.1.7 Complete Preprocessing

Now that the different preprocessing stages have been examined, we can look at the code for the entire process.

4.2 AL + LE

We will now examine our algorithm's Active Learning and Lexical Expansion sections. These are based on the works from [39] and [33], respectively. The basics of these algorithms were discussed in Chapter 3. Here, we will examine the changes

that we have made and how we combine these two approaches. In addition, as we have created our algorithm modularly, we will assume the network is provided to us as a variable.

Define $\mathtt{preprocessing}()$ as:
Given: $\mathtt{data}$, $STOPWORDS$ as SW, $\mathtt{svd}$, $\mathtt{tfidf}$
$\mathtt{data} = \mathtt{removeOverLength}(\mathtt{data})$
$\mathtt{data} = \mathtt{tokenize}(\mathtt{data})$
$\mathtt{dTr}, \mathtt{dTe} = \mathtt{split}(\mathtt{data})$
$\mathtt{svd}(\mathtt{dTr}, \mathtt{dTe}, \mathtt{tfidf}, \mathtt{svd})$
$\mathtt{svdTr}, \mathtt{svdTe} = \mathtt{vocab}(\mathtt{dTr}, SW)$
$\mathtt{map}, \mathtt{index}, \mathtt{indices} = \mathtt{vocab}(\mathtt{dTr}, SW)$
$\mathtt{dTrNum} = \mathtt{convertNum}(\mathtt{dTr})$
$\mathtt{dTeNum} = \mathtt{convertNum}(\mathtt{dTe})$
$\mathtt{dTrNum} = \mathtt{pad}(\mathtt{dTrNum})$
$\mathtt{dTeNum} = \mathtt{pad}(\mathtt{dTeNum})$

Figure 4.4: Preprocessing Code

4.2.1 Combining AL and LE

We made small changes to both the Active Learning and Lexical Expansion algorithms. We will discuss these changes and then outline how we combine the techniques.

4.2.1.1 AL

The main change we make to the Exploration-Exploitation approach proposed in [39] is switching the Siamese network with a singular value decomposition (SVD) matrix. Training an additional network alongside the classification network is computationally expansive. The original paper did not focus solely on text analysis, so the authors needed a method of comparing the similarity of a wide variety

of data types. For our purposes, we can replace the Siamese network with a similarity comparison that is widely used in a text, a sparse term frequency-inverse document frequency (TF-IDF) matrix that is reduced using SVD. Using this matrix has been shown to have good results for similarity [40]. By removing the Siamese network, the overall computation time of the algorithm should be reduced, and the accuracy of the similarity calculation should improve as well.

4.2.1.2 LE

We also changed the PLSDA algorithm proposed in [33] by changing the similarity measurement. In the original paper, the authors used pre-trained Glove embeddings to determine the similarity between the old sentence and the new modified sentence. We did not want to use embeddings that were not being used by the classifier. Instead, we used the WordNet similarity comparison to compare words [41]. WordNet generates the synonyms used by the algorithm, and it contains a built-in word similarity comparison. We feel that relying on a pre-trained embedding that may not generalize to the current training set for similarity could introduce noise into the algorithm. Using the similarity comparison from WordNet removes some of this noise [41]

4.2.1.3 Combining AL and LE

To combine the two strategies, we first run the Active Learning approach to select the new sequences to be added to the data set. We then take these sequences and apply the Lexical Expansion technique to them. This will generate additional sequences. This can be visualized in the following Figures. The motivation behind our approach is that the exploration-exploitation algorithm will select data that confuses the classifier and is close to the decision boundary and data that is far from the decision boundary on which the classifier has little information. Adding

the PLSDA algorithm to this will generate more information on the areas that are either the most confusing or the most unknown.

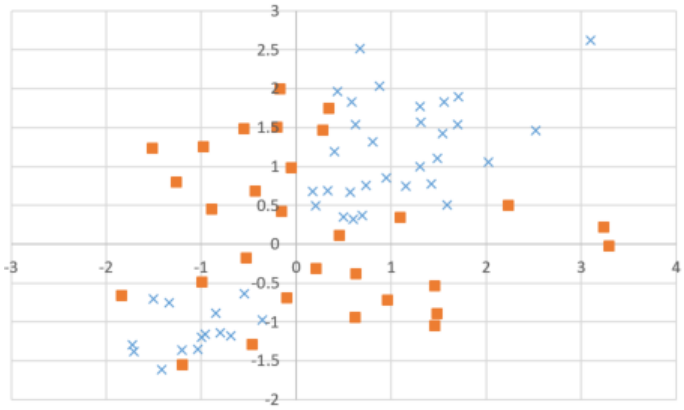

Figure 4.5: Generated Data

Figure 4.5 contains artificially generated data demonstrating our algorithm. The data is separated into two classes: the squares and the crosses.

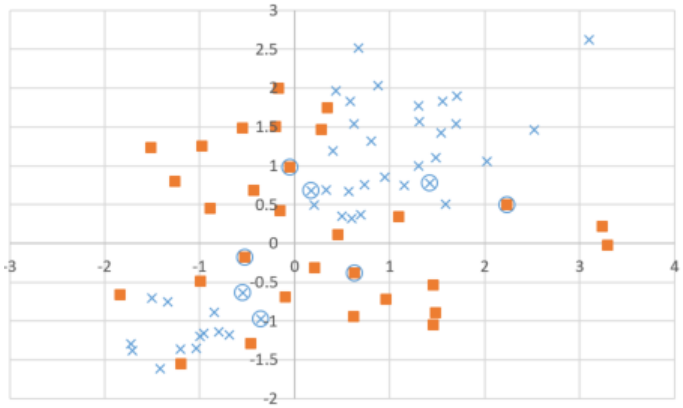

Figure 4.6: Training Data

In Figure 4.6, we have selected a small amount of the data to train our classifier. The circles around the data points denote this. The remaining data will be used to test the classifier.

We create an artificial decision boundary based on the selected training data. The

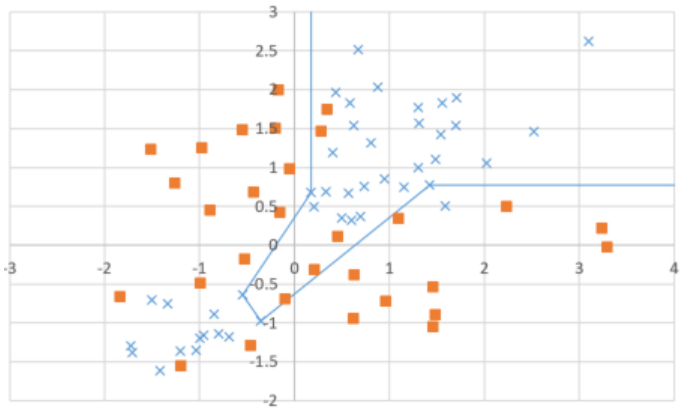

Figure 4.7: Decision Boundary

line in Figure 4.7 shows this boundary. The data points above and to the right of the line will be labelled as crosses, and the ones below and to the left will be labelled as squares.

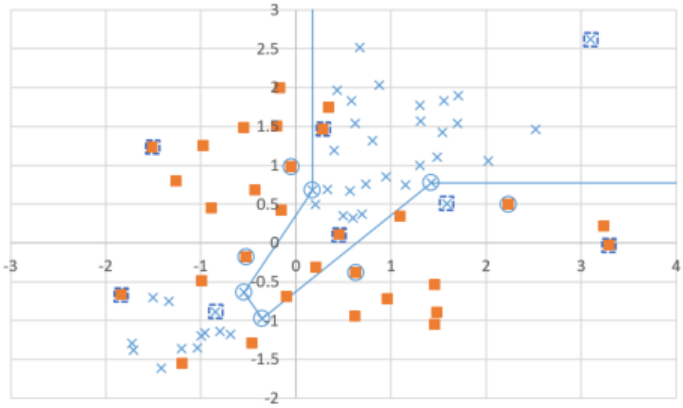

Figure 4.8: Active Learning Data

After the initial labelling, we can apply our proposed algorithm. The first step will be to run the active learning algorithm: exploration-exploitation. We will have it selected eight data points. Four from exploration and four from exploitation. The exploration stage will select four points far from the original training set. The exploitation stage will select four data points that are confusing to the classifier: four points that the classifier was wrong or unsure about in the labelling. If our

algorithm is used on real-world data, where the correct labels are unknown beforehand, the newly selected data will be provided to topic experts to label.

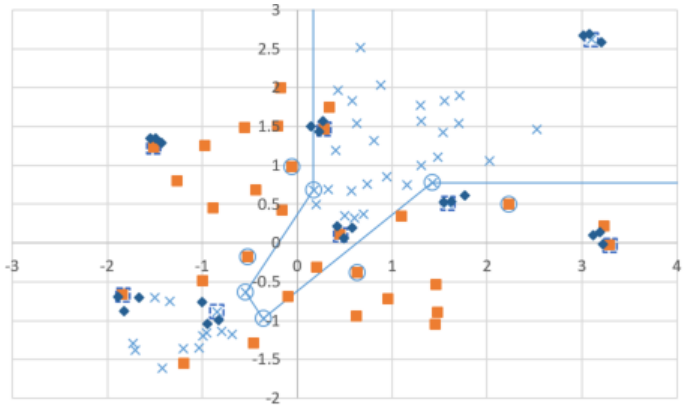

Figure 4.9: PLSDA Data

After the Active Learning section of our algorithm has been completed, we move on to the PLSDA section. For this demonstration, we will create three new data points. These will be within a preset similarity threshold to the originals. The newly created data points are shown in Figure 4.9 as the diamonds. The generated data will be labelled as the parent data point. The new training set will be assembled after the PLSDA section has processed all the data points selected in the Active Learning section. This can be seen in Figure 4.10.

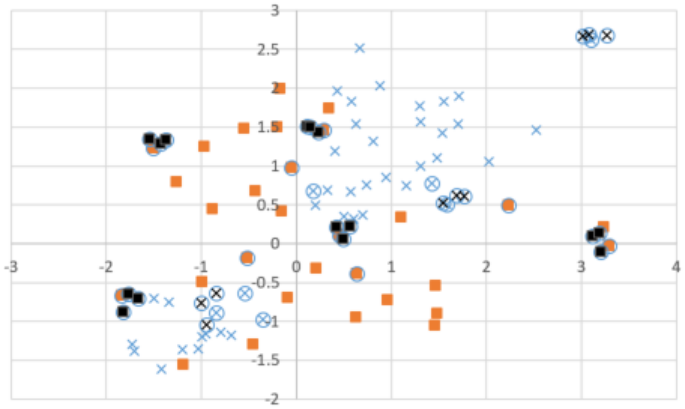

Figure 4.10: New Training Data

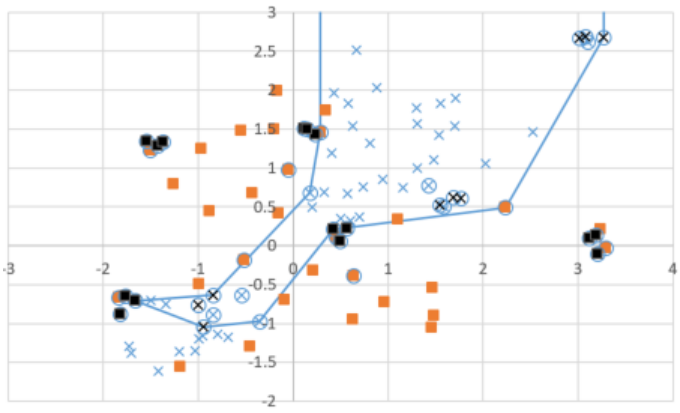

Figure 4.11: Updated Decision Boundary

After completing our algorithm, the classifier is retrained on the new training data. This will create a new decision boundary that could look something like the line in Figure 4.11. An aspect of the PLSDA approach that is not illustrated in Figure 4.11 is that when that training data is text data, the generated data can add information that was not initially present in the training set. Ideally, this new information will reshape the decision space, making the distinction between the classes more apparent.

4.3 Contributions

The contributions that this thesis has made to advance the current state of sentiment analysis research are:

- Combining EE and PLSDA.
- Testing a different similarity measure for EE.
- Testing a different similarity measure for PLSDA.
- Training and comparing two neural networks with EE, PLSDA, and APLSDA.

The main contribution of this work is the combined algorithm APLSDA. This newly synthesized algorithm tests to see if its component algorithms' advantages can be combined while minimizing the weaknesses that those algorithms introduce. By changing the similarity measures of both the EE and the PLSDA algorithms, this research aims to determine if pre-trained embeddings are necessary for sentiment analysis research. One of the main motivating factors for this research is to reduce the amount of labelled training data needed to create a classifier and to reduce the overall cost of the classifier. If pre-trained embeddings are required to complete this goal, then the cost of creating a classifier is still high, as a general end user will not be able to create these embeddings themselves. Finally, by training and comparing two different neural network structures, we examine how our APLSDA algorithm interacts and whether different neural networks interact with the algorithm differently.

In the next section, we will describe how we test this algorithm to determine if it offers better performance when compared to the baseline networks.

Chapter 5

Experiment Set Up

Our experiment aims to determine if combining the PLSDA lexical expansion algorithm with the Exploration-Exploitation active learning algorithm provides better results when compared to either of the algorithms alone or the base performance of the model. We are measuring each algorithm's average and maximum performance to determine this. In addition to changes in performance, we also analyze changes in processing time between the different combinations of approaches. We are examining three different neural networks. We are interested in seeing if the changes in performance depend on the network structure. We will first explore the three models we are evaluating before examining the structure of our experiments.

5.1 Models

We have two main types of models. These are general neural network models and BERT-based models. Overall, we tested three different neural networks. The two general neural network structures are a stacked CNN model based on [19] and a CoLSTM, which is a combination of the works from [20] and [21]. For the BERT-based model, we have selected an ALBERT model [10] for evaluation. All of our neural networks use the Python API for TensorFlow [22]. We use the Keras layers

for both the CNN and the CoLSTM. We use the pre-trained models for the BERT model, which are available through the TensorFlow Hub.

5.1.1 CNN

We used the CNN based on the network found in [19]. The structure of their model can be found in Table 3.2. Our structure can be found in Table 5.1. The main difference between the author's network and ours is that we allow the input sentences to be longer. In the original paper, the max length of a sentence was 40, while we allow sentences up to a length of 800. With this in mind, we follow the example of [19] by setting the kernel size for all of the 1DCNN layers to 3. The first 1DCNN layer has 64 filters, and the next has 32. A max-pooling layer follows this. After this, there are two 1DCNN layers, the first with 16 filters and the second with 8. The last two layers are a global average pooling layer and, finally, the fully connected dense layer. The authors in [19] showed that this neural network structure performs well while requiring relatively little training time.

Table 5.1: Our CNN

Layer	Output Shape	Parameters
Embedding	800, 300	18000000
1D Conv	800, 64	57664
1D Conv	800, 32	6176
Max Pooling 1D	266, 32	0
1D Conv	266, 16	1552
1D Conv	266, 8	264
Global Avg 1D	8	0
Dense	1	9

5.1.2 CoLSTM

As discussed in Chapter 3, our CoLSTM is loosely based on the works of [20] and [21]. From [20], we adopt the idea of first using a CNN layer to reduce the dimensionality of the data before providing it as input to the LSTM. We take the idea of using multiple different CNN kernel sizes from [21]. By combining these two ideas we get our final structure, where we have the word embeddings being fed into three separate 1DCNNs. These have kernel sizes of 3, 5, and 7, respectively. Each of these has five filters that are learned. An average pooling layer is then applied to each of the CNN layers. This will provide us with three tensors with 800 elements. We then stack these on one another to create a 3,800 tensor that is the input into our LSTM. The LSTM's output is fed into two dense layers to get our final output. The idea behind this structure is that the three CNN layers will learn the different word relationships in the text. The LSTM layer will take these different relationships and learn any temporal relationships between them. The layer-by-layer breakdown can be seen in Table 5.2.

Table 5.2: Our CoLSTM

Layer	Output Shape	Parameters
Input	800	0
Embedding	800, 300	18000000
1D Conv Kernal 3	800, 5	4505
1D Conv Kernal 5	800, 5	7505
1D Conv Kernal 7	800, 5	10505
Average Pooling 1D For 3	800	0
Average Pooling 1D For 5	800	0
Average Pooling 1D For 7	800	0
Stack	3, 800	0
LSTM	100	360400
Dense	100	10100
Dense	1	101

5.1.3 ALBERT

We use the large ALBERT model [10] that is available from TensorFlow [22]. We made this choice as it has comparable performance to the base BERT model while still providing an increase in performance. We did not allow the weights of the pre-trained embeddings to be updated. On top of the embeddings, we have a network that is similar to our CNN network. There is a single 1DCNN with a kernel of size 3 followed by a pooling layer. After the pooling, there are two more 1DCNN layers with kernels of 3 each. A final pooling layer feeds into a dense layer to produce the final classification. This can be seen in Table 5.3

Table 5.3: ABLERT

Layer	Output Shape	Parameters
Input	800	0
Embedding	128, 1024	17683968
1D Conv Kernal 3	128, 64	196672
Max Pooling	42, 64	0
1D Conv Kernal 3	42, 16	3088
1D Conv Kernal 3	42, 8	392
Global Avg Pooling	8	0
Dense	1	9

5.2 Algorithm Selection

We have four different algorithms that we are testing:

1. Basic
2. PLSDA
3. Exploration-Exploitation
4. PLSDA + Exploration-Exploitation

Depending on which algorithm we are testing, two different experiment setups are used. When we test the basic or PLSDA algorithm, we randomly select a portion of the data set for training and use the rest for testing. The amount of data selected for training starts from one percent and increases by one percent up to five percent. After the amount reaches five percent, we increase it by five percent until sixty percent. The motivation is to use the smallest training data possible while keeping the computational costs reasonable. We train ten of the selected neural networks at each increment and record their performances.

The iterative nature of the Exploration-Exploitation algorithm makes the previous experiment setup ineffective in testing it. Instead, we start the algorithm with one percent of the data, allowing it to choose 64 sequences from the testing set at each iteration. Each time the algorithm adds data to the testing set, we train ten networks and keep the best-performing one for the next iteration. Again, we track the performance of all the networks for comparison. We have the active learning algorithm run until it has selected sixty percent of the data to train from.

For our combined algorithm, we use the same setup as the Exploration-Exploitation algorithm. We start from one percent, and using the active learning section of our algorithm, we add 64 sequences. Additionally, our algorithm will add generated sequences from the lexical expansion step.

Chapter 6

Evaluation and Analysis

6.1 Evaluation Criteria

To measure the performance of our classifiers, we use three metrics. These metrics are Precision, Recall, and F1-Score. To understand these metrics, we must introduce the confusion matrix. The confusion matrix contains four quadrants. These quadrants are labelled as: True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (FN). TP and TN terms are used when the classifier correctly identifies whether a data point belongs to the target class. The FP and FN terms are used when the classifier incorrectly identifies a data point as belonging to the target class when it does not (FP) or says a data point does not belong to the class when it does (FN). By calculating a classifier's TP, FP, TN, and FN rates, we can start calculating that classifier's precision, recall, and F1-Score.

Precision measures how often the classifier correctly makes a positive prediction. The formula can be found in Equation 6.1

$$\mathrm{precision} = \frac{TP}{TP + FP} \tag{6.1}$$

Recall measures the classifier correctly identified that a data point belonged to the

target class. The equation for recall can be seen in Equation 6.2

$$recall = \frac{TP}{TP + FN} \tag{6.2}$$

Finally, the F1-Score is the harmonic mean of precision and recall. The formula for F1-Score can be found in Equation 6.3

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{6.3}$$

In addition to F1-Score, we measure the mean and variance of the different networks and algorithms.

F1-Score is measured across all labels in a testing set. We take the average F1-Score to measure our classifiers' overall performance. In both of our datasets, there are two labels, so our final metric is:

$$F1_{average} = \frac{F1_0 + F1_1}{2} \tag{6.4}$$

We also measure the statistical significance of the results of our different networks and algorithms. This is done using the Student's t-test. This equation can be found in 6.5.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_{\bar{\Delta}}} \tag{6.5}$$

Here $s_{\bar{\Delta}}$ is calculated in 6.6.

$$s_{\bar{\Delta}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \tag{6.6}$$

Where s_i^2 is the estimator of the variance of the ht two series, and n_i is the number of sequences in the two samples. We set our confidence interval to 0.95 and our p value to 0.05.

6.2 Evaluation

We use the IMDB movie review dataset [18] for our evaluation. We will analyze the average and the maximum for the classifiers. We have taken ten observations for each network and each algorithm. Although this is a small number of observations for significance testing, we believe these tests will still provide some helpful information.

6.2.1 CNN

We will start by analyzing the average from the CNN. The averages can be found in 6.1. We have graphed these values as well. This graph can be found in 6.1. The highest average at each sampling step is in bold. Values followed by an asterisk are statistically significant compared to the normal, unmodified results with a p-value of $p < 0.05$. The p-values are in Table 6.4 in the A-N, P-N and AP-N lines for the Active-Noraml, PLSDA-Normal and APLSDA-Normal, respectively.

Table 6.1: Average F1 Score CNN

	1	2	3	4	5	10	15	20
Normal	0.335	0.335	0.335	0.558	0.666	0.791	0.812	0.806
PLSDA	0.335	0.452	0.683*	0.684*	0.720	0.782	0.782	0.800
Active	0.335	0.433	0.626*	0.667	0.697	**0.801**	0.816	0.824*
APLSDA	0.335	**0.671***	**0.706***	**0.717**	**0.771**	0.788	**0.819**	**0.830***

	25	30	35	40	45	50	55	60
Normal	0.825	0.823	0.827	0.827	0.830	0.830	0.832	0.826
PLSDA	0.820	0.822	0.823	0.820	0.825	0.828	0.832	0.832
Active	**0.840***	0.848*	0.852*	**0.869***	**0.879***	**0.894***	0.894*	**0.914***
APLSDA	**0.840***	**0.851***	**0.864***	0.865*	0.875*	0.891*	**0.902***	0.913*

?* denotes statistical significance compared to the normal approach

This table shows that our proposed method outperforms the other approaches for training amounts less than ten percent. This difference is only statically sig-

nificant for values two and three. For values of ten and above, the AL approach or our APLSDA approach had the highest average. Among these values, the AL and PLSDA approaches are only significantly different from the normal approach for values greater than 15. There seems to be no predictable pattern in which one outperforms the other. It is also worth noting that we have calculated the p-values between the different approaches. These can be found in 6.4, and as seen in the A-AP lines, there is no significant difference between the two approaches for values greater than two.

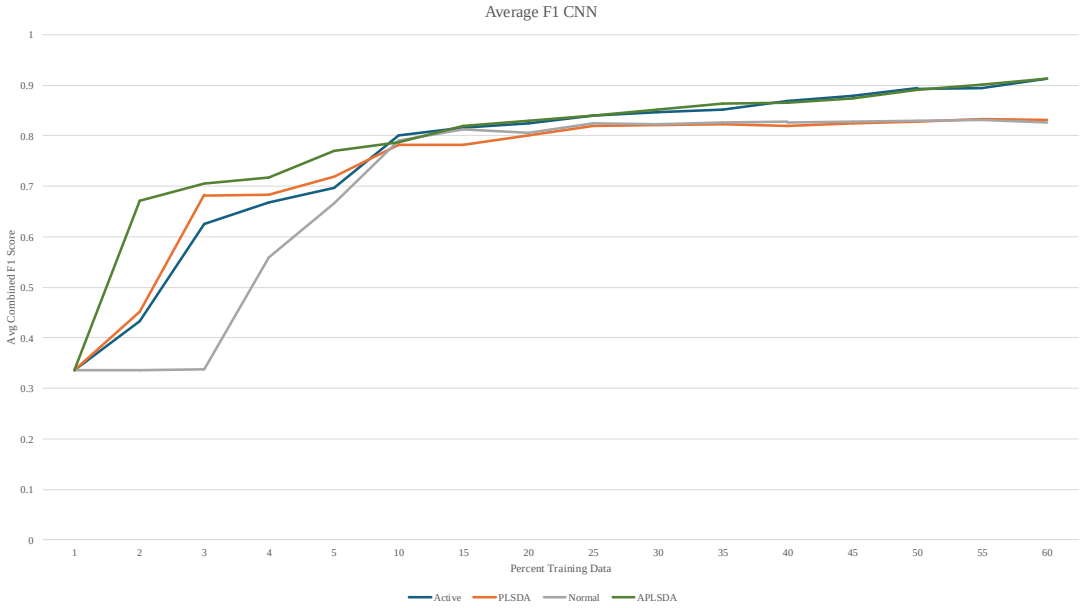

Figure 6.1: Average F1 Score CNN

The graph in Figure 6.1 helps to visualize the difference between the active learning and APLSDA algorithms, which is minor for values above 25. To further illustrate the differences between the different algorithms compared to the normal CNN network, we have plotted the differences in the average F1 score in 6.2. This figure shows that our APLSDA algorithm improves slightly faster than the Active learning algorithm. While the PLSDA algorithm offers some improvements, it

quickly falls off and offers slight improvement, and, in some cases, it is worse than the standard network.

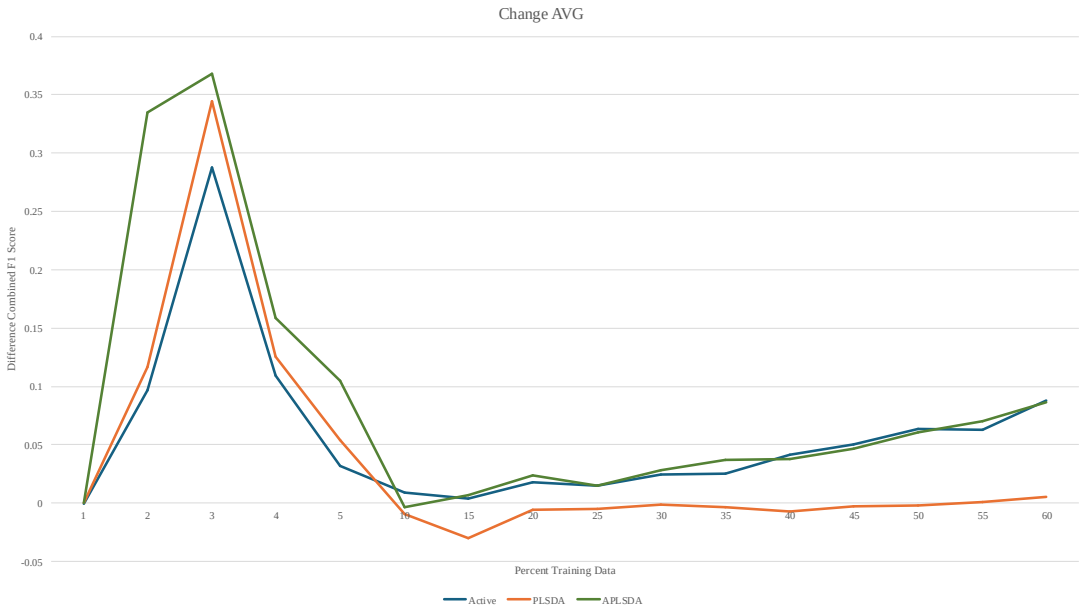

Figure 6.2: Change in Average F1 Score CNN

We also want to look at the maximum values the neural network achieves. These values are in Table 6.2 and Figure 6.3. From the table and the figure, we can see that the max value follows a similar pattern to the average value. Our APLSDA algorithm achieves the highest value four out of five times for values of five and less. There is little pattern to the best-performing algorithm for values of ten and above. We can see from Figure 6.4 that the three different algorithms provided are similar to the changes to the average F1 score.

Table 6.2: Max F1 Score CNN

	1	2	3	4	5	10	15	20
Normal	0.335	0.335	0.367	0.706	0.786	**0.817**	0.824	0.824
PLSDA	0.335	0.706	0.756*	0.766*	0.791	0.813	0.816	0.826
Active	0.335	0.651	**0.783***	0.753	0.795	0.815	**0.829**	0.837*
APLSDA	0.335	**0.745***	0.779*	**0.794**	**0.797**	0.814	0.826	**0.839***

	25	30	35	40	45	50	55	60
Normal	0.845	0.848	0.848	0.840	0.854	0.838	0.839	0.839
PLSDA	0.826	0.830	0.830	0.831	0.835	0.850	0.837	0.836
Active	**0.847***	0.854*	**0.870***	0.873*	**0.883***	**0.896***	**0.908***	**0.918***
APLSDA	0.846*	**0.858***	0.867*	**0.874***	**0.883***	0.893*	0.904*	0.915*

?* denotes statistical significance compared to the normal approach

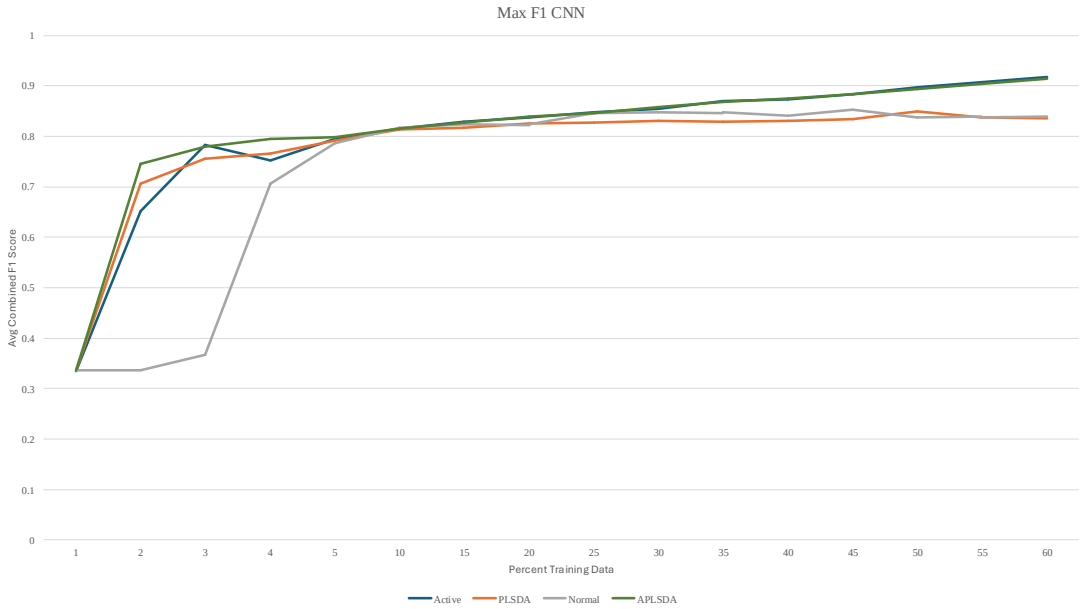

Figure 6.3: Max F1 Score

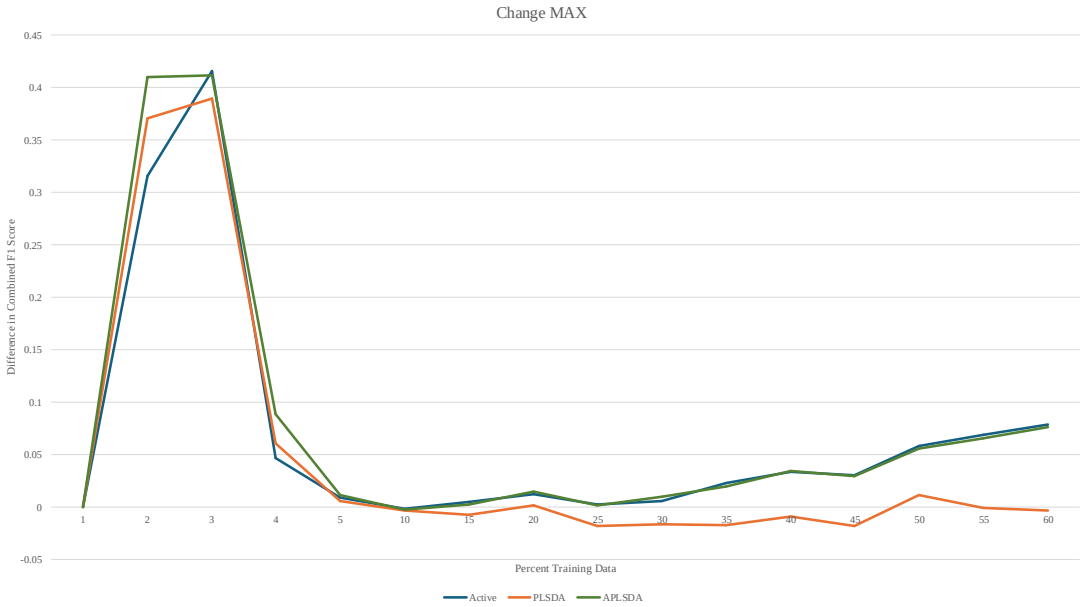

Figure 6.4: Change in Max F1 Score CNN

Table 6.3: Standard Deviation CNN

	1	2	3	4	5	10	15	20
Normal	5.551E-17	0.0743	**0.0331**	0.0842	0.1580	0.0194	0.0108	0.0119
PLSDA	0.1006	0.1881	0.1203	**0.0696**	**0.0247**	**0.0105**	**0.0101**	0.0143
Active	5.551E-17	0.1148	0.0940	0.0913	0.1257	0.0254	0.0225	0.0109
APLSDA	5.551E-17	**0.0488**	0.0997	0.1127	0.0229	0.0220	0.0164	**0.0096**

	25	30	35	40	45	50	55	60
Normal	0.0132	0.0077	0.0043	0.0034	0.0038	0.0021	0.0051	0.0024
PLSDA	**0.0046**	0.0057	0.0273	0.0047	**0.0020**	0.0038	0.0039	0.0110
Active	0.0100	**0.0037**	0.0234	**0.0020**	0.0033	**0.0015**	0.0267	0.0029
APLSDA	0.0059	**0.0037**	**0.0028**	0.0111	0.0114	0.0023	**0.0022**	**0.0016**

In Table 6.3, the standard deviation for the four different algorithms is listed. Unlike the previous tables, the smallest value is highlighted in bold. We are interested in seeing if there is any algorithm that constantly provides a smaller standard deviation. The change in standard deviation can be seen in Figure 6.5. While the PLSDA approach does offer a smaller standard deviation in six of the sixteen points, it is important to remember that it resulted in worse performance than the baseline. The PLSDA approach is statistically significant in only one of these points. The APLSDA and Active algorithms reliably produce values with a lower standard deviation for values above 15. The Active algorithm has a smaller standard deviation in seven out of the nine values, and the APLSDA has smaller values for six. The Active algorithm has the smallest values for three of the points.

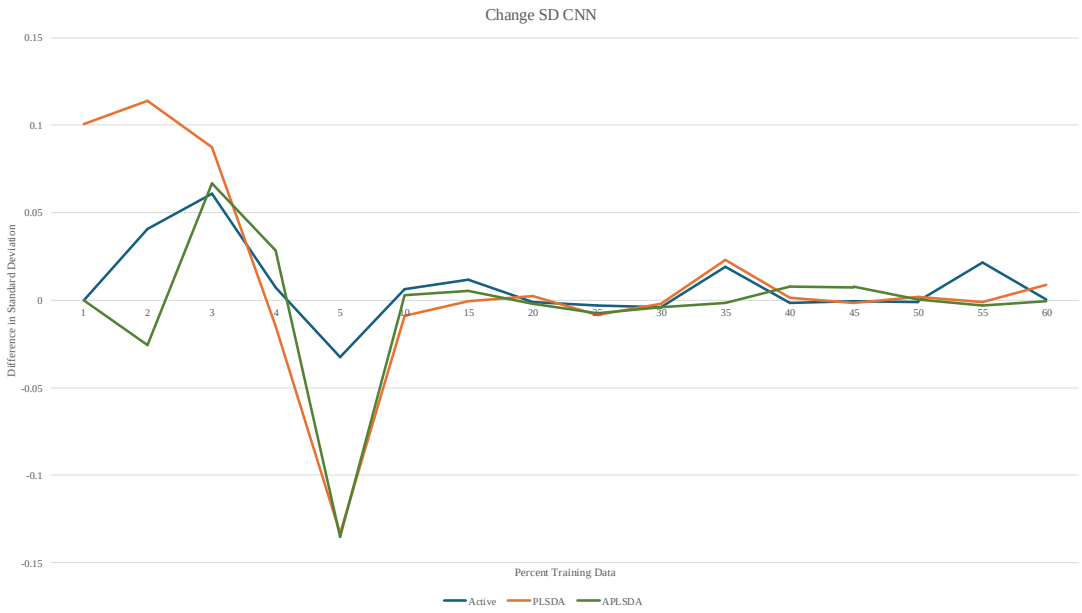

Figure 6.5: Change in Standard Deviation F1 Score

When looking at the statistically significant difference between the different algorithms, we can see a clear difference in how the algorithms perform for values of five and less, the values of ten and fifteen, and the values of twenty and higher.

For the values of one to five, there is little pattern to the statistical significance between the different algorithms. For ten and fifteen, none of the algorithms are significantly different from each other. Finally, for all the training values of twenty-five and higher, all but Active-APLSA and PLSDA-Normal are significant.

Table 6.4: P-Values CNN

	1	2	3	4	5	10	15	20
A-P	0.348	0.829	0.217	0.087	0.482	0.63	0.308	**0.032**
A-N	4E-225	0.136	**4E-06**	0.741	0.479	0.444	0.667	**0.034**
A-AP	1E-223	**6E-05**	0.073	0.24	0.187	0.309	0.873	0.217
P-N	0.343	0.427	**0.001**	**0.035**	0.166	0.612	0.318	0.784
P-AP	0.344	**0.002**	**0.009**	0.812	**0.018**	0.391	0.131	**0.003**
AP-N	6E-212	**1E-08**	**5E-07**	0.139	0.069	0.732	0.442	**0.002**

	25	30	35	40	45	50	55	60
A-P	**1E-04**	**7E-09**	**0.001**	**6E-12**	**2E-16**	**1E-14**	**4E-05**	**3E-10**
A-N	**0.001**	**6E-07**	**0.005**	**9E-15**	**9E-17**	**1E-21**	**4E-05**	**2E-22**
A-AP	0.819	0.315	0.154	0.461	0.607	**0.012**	0.449	0.164
P-N	0.63	0.98	0.08	0.276	0.156	0.29	0.989	0.2
P-AP	**7E-07**	**3E-09**	**1E-04**	**3E-07**	**4E-07**	**8E-17**	**3E-17**	**1E-09**
AP-N	**0.001**	**3E-07**	**1E-13**	**1E-06**	**9E-08**	**1E-21**	**4E-14**	**1E-22**

Taking all the results from this section together, we can see that our APLSDA algorithm and the Active algorithm perform similarly. Overall, the performance at five percent or less is unpredictable. Our algorithm produces slightly more consistent results for values over fifteen when considering the standard deviation. For the CNN, the APLSDA and AL algorithms perform similarly, and neither outperforms the other.

6.2.2 LSTM

Similarly, for the LSTM, we will first examine the average performance of the network. The numeric values can be found in Table 6.5, and the graphical representations can be found in Figure 6.6 and 6.7. We can see that the Active and APLSDA

outperformed the other two algorithms in all cases. In all cases, either the Acitve or the APLSDA algorithm performed the best. However, the statistical significance of these improvements is unreliable for values less than 30.

Table 6.5: Average F1 Score LSTM

	1	2	3	4	5	10	15	20
Normal	0.335	0.516	0.519	0.565	0.646	0.708	0.748	0.751
PLSDA	0.381*	0.505	0.541	0.59	0.668	0.721	0.751	0.755
Active	0.34	0.441	**0.596**	**0.649***	**0.703**	**0.75**	**0.774**	**0.783**
APLSDA	**0.531***	**0.552**	0.568	0.608	0.638	0.735	0.767*	0.775

	25	30	35	40	45	50	55	60
Normal	0.767	0.783	0.795	0.8	0.798	0.806	0.809	0.81
PLSDA	0.775	0.776*	0.784*	0.791	0.801	0.812	0.815	0.81
Active	**0.794**	**0.805***	**0.818***	0.819*	0.831*	**0.848***	0.847*	0.854*
APLSDA	0.785	0.789	0.813*	**0.826***	**0.836***	0.844*	**0.848***	**0.866***

?* denotes statistical significance compared to the normal approach

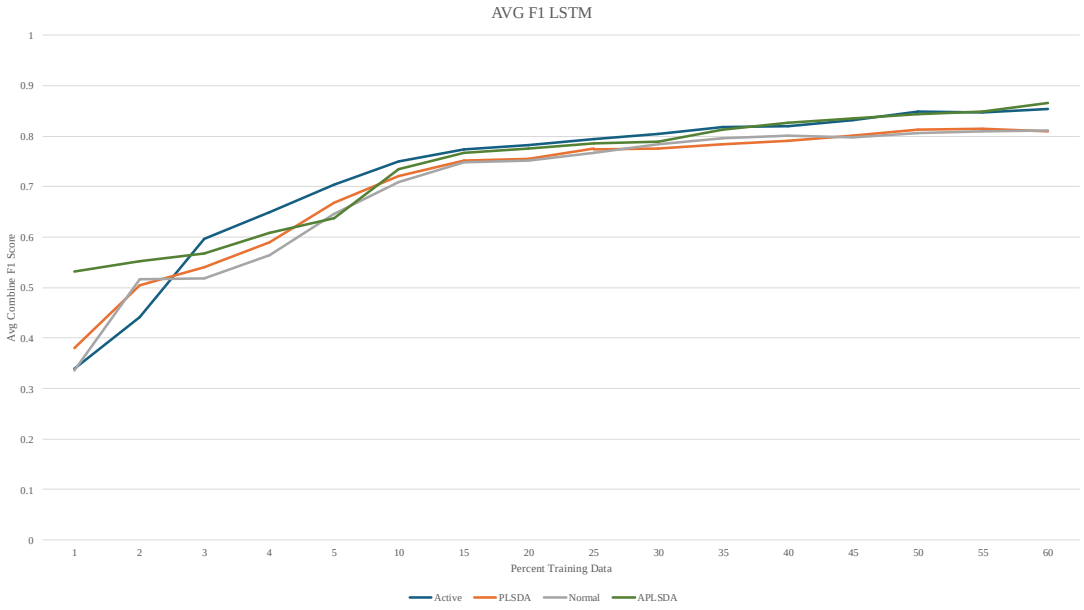

Figure 6.6: Average F1 Score LSTM

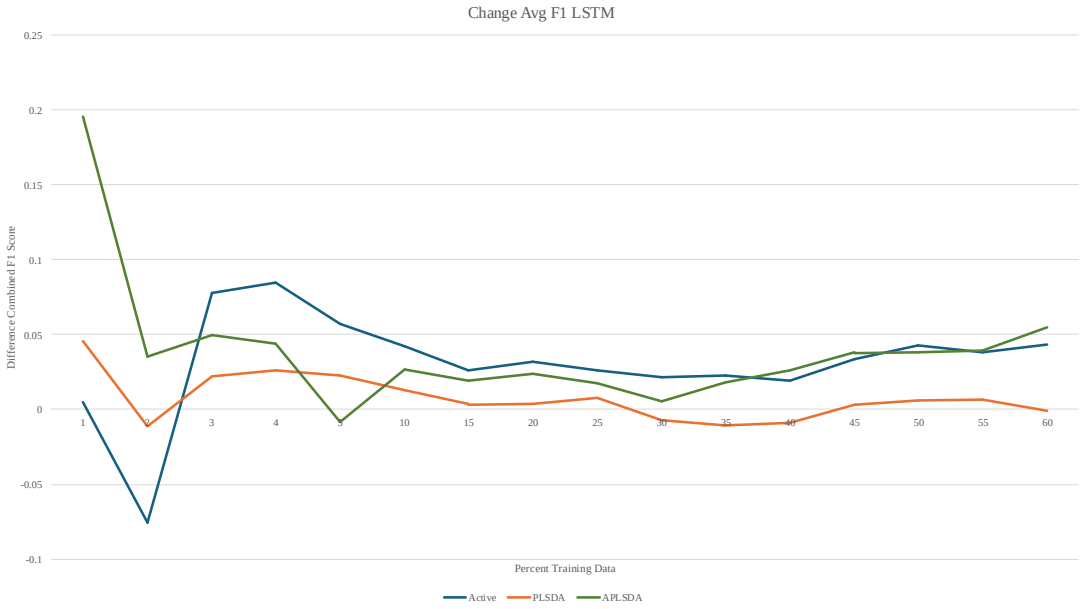

Figure 6.7: Change in Average F1 Score LSTM

The max F1-Scores for the LSTM network can be found in Table 6.6, Figure 6.8, and Figure 6.9. Again, the Active and the APLSDA algorithms outperform the PLSDA algorithm at all values. The statistical significance is the same as the average, so there is no pattern to the significance for values less than 30.

Table 6.6: Max F1 Score LSTM

	1	2	3	4	5	10	15	20
Normal	0.336	0.541	0.582	0.667	0.709	0.757	0.765	0.794
PLSDA	0.458*	0.532	0.618	0.7	0.715	0.746	0.773	0.795
Active	0.366	0.651	**0.684**	0.718*	**0.732***	**0.767**	**0.795***	**0.809**
APLSDA	**0.571***	**0.652**	0.661	**0.747**	0.722	0.763	0.794*	0.8

	25	30	35	40	45	50	55	60
Normal	0.798	0.801	0.81	0.811	0.818	0.816	0.815	0.826
PLSDA	0.796	0.798	0.802*	0.808	0.814	0.823	0.824	0.823
Active	0.806	**0.825***	**0.836***	0.837*	0.843*	**0.86***	**0.868***	0.872*
APLSDA	**0.81**	0.819	0.833*	**0.844***	**0.852***	0.856*	0.864*	**0.875***

?* denotes statistical significance compared to the normal approach

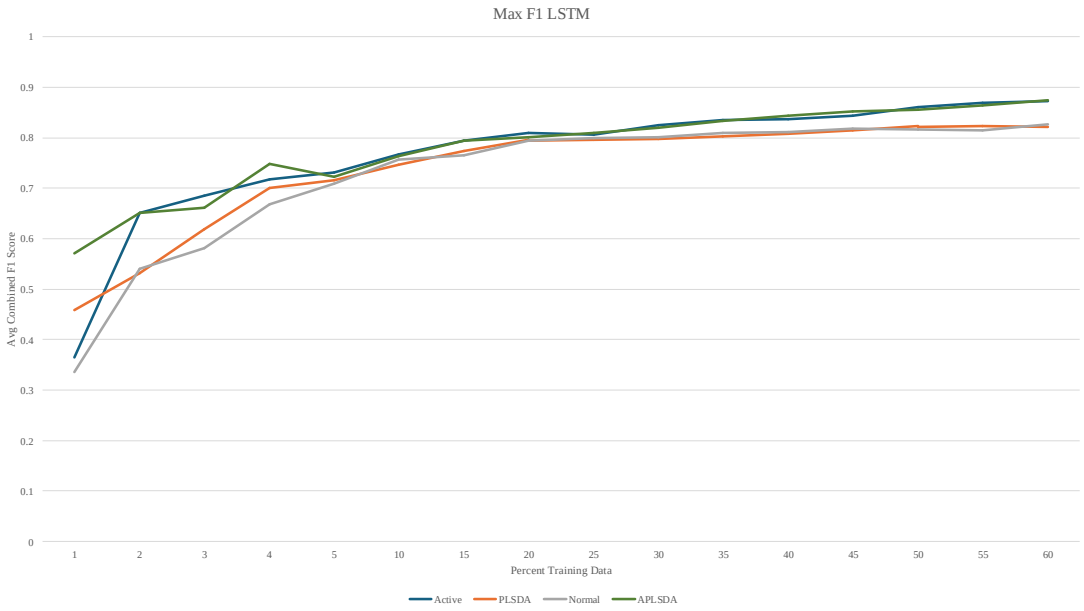

Figure 6.8: Max F1 Score LSTM

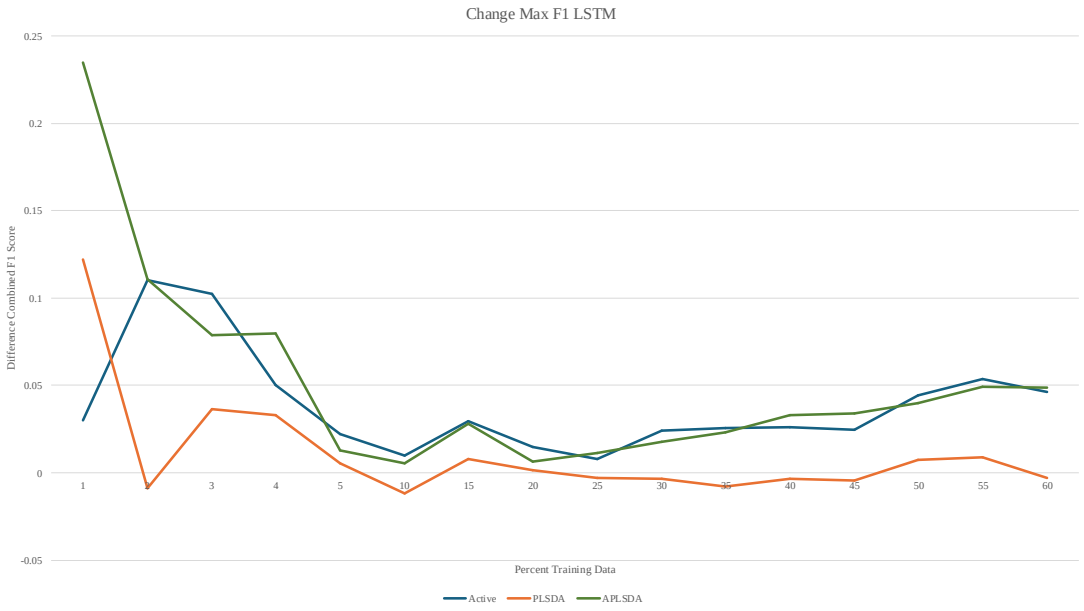

Figure 6.9: Change in Max F1 Score LSTM

We will next examine the standard deviation of the LSTM network. Unlike the CNN, our APLSDA algorithm with the LSTM produced a smaller standard deviation for the training values of 2, 50 and 60. All algorithms produced the smallest deviations for some values. There is no observable pattern for which algorithms will produce small standard deviations.

Table 6.7: Standard Deviation LSTM

	1	2	3	4	5	10	15	20
Normal	**0**	0.0134	0.0461	0.0659	0.041	0.072	0.0206	0.062
PLSDA	0.0432	0.0159	**0.042**	0.0692	0.0415	**0.0234**	0.0165	0.045
Active	0.0096	0.1146	0.0883	**0.0474**	**0.0407**	0.0087	**0.0113**	**0.0119**
APLSDA	0.0372	**0.0566**	0.0733	0.0798	0.0828	0.0247	0.0125	0.0123

	25	30	35	40	45	50	55	60
Normal	0.0544	**0.0142**	**0.0092**	**0.0097**	0.0167	0.0105	0.0047	0.0151
PLSDA	0.0148	0.0232	0.0113	0.0111	0.0132	0.0085	**0.0075**	0.0103
Active	**0.0082**	0.0114	0.0116	0.0107	**0.0102**	0.0119	0.0223	0.0174
APLSDA	0.015	0.0332	0.0157	0.0266	0.0124	**0.0103**	0.0214	**0.0094**

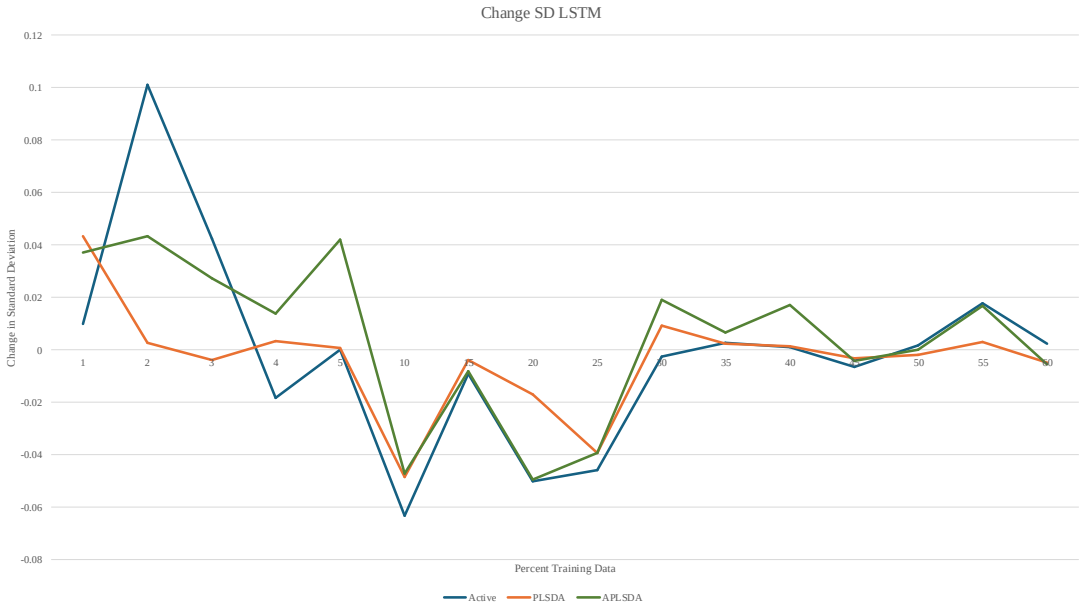

Figure 6.10: Change STD LSTM

The p-values found in Table 6.8 show that the APLSDA and AL algorithms are only statistically different at three percent. Overall, we can see that all algorithms except the PLSDA algorithm are significantly different from all others for values greater than thirty. For values less than thirty, there are no clear patterns for the significant differences between algorithms except for the previously mentioned APLSDA and AL algorithms.

Table 6.8: P-Values LSTM

	1	2	3	4	5	10	15	20
A-P	0.032	0.136	0.167	0.062	0.151	**0.008**	**0.003**	0.081
A-N	0.189	0.072	0.072	**0.003**	**0.017**	0.125	**0.016**	0.188
A-AP	**5E-08**	**0.009**	0.274	**0.032**	**0.001**	**1E-06**	**2E-04**	**9E-07**
P-N	**0.012**	0.133	0.385	0.513	0.204	0.582	0.728	0.902
P-AP	**1E-04**	**0.01**	0.601	0.839	**0.014**	**3E-05**	**5E-04**	**4E-05**
AP-N	**7E-08**	0.098	0.107	0.222	0.789	0.313	**0.03**	0.287

	25	30	35	40	45	50	55	60
A-P	**0.012**	**0.009**	**3E-04**	**8E-05**	**3E-04**	**5E-06**	**0.002**	**1E-04**
A-N	0.185	**0.003**	**0.002**	**0.006**	**0.001**	**5E-06**	**0.001**	**0.002**
A-AP	**5E-04**	**2E-05**	**5E-06**	**2E-05**	**7E-06**	**9E-09**	**5E-07**	**5E-07**
P-N	0.622	0.312	**0.048**	0.184	0.582	0.216	0.052	0.903
P-AP	**0.001**	**0.003**	**1E-04**	**3E-04**	**9E-05**	**3E-07**	**2E-08**	**1E-05**
AP-N	0.372	0.657	**0.01**	**0.017**	**5E-05**	**3E-07**	**3E-04**	**1E-07**

Overall, the Active and APLSDA algorithms significantly improved the performance of the LSTM network. These two algorithms achieved the best average performance and the best max performance. They were both consistently statistically significant for values greater than 25. However, the standard deviation of these algorithms is not consistently lower than the normal algorithm. Similar to the CNN, these two algorithms are not significantly different from each other at nearly all points. The most notable result from the LSTM is that the APLSDA algorithm started with a max F1 score of 0.571. This is over twenty points above the performance of the CNN. This result is surprising when examining the rest of

the LSTM's performance. This indicates that with one percent of the data being used for training, the LSTM is slightly better than guessing. While the LSTM does not progress to the same levels as the CNN, this result indicates some benefit to the LSTM structure. Notably, neither the PLSDA nor the AL algorithms offer the same performance at one percent. This could indicate that there is some use in combining them.

6.2.3 BERT

The final model that we tested was our BERT-based model. We had to treat this model differently than the others. We did not apply the different algorithms and only ran the algorithm once for each dataset division. We had to do this from a computational standpoint, as the BERT model took over three hours per epoch. While this prevents us from being able to calculate an average or find an actual max, we can treat this as a baseline for a large pre-trained model. The results from our testing can be found in Table 6.9 and Figure 6.11.

From this table, we can see that the BERT model is unpredictable for values less than 15. In addition, the performance of the BERT model is lower than what we would have expected from other research. This may be because the ALBERT model may struggle to transfer its learning from its pre-trained knowledge base of books and Wikipedia to movie reviews.

Table 6.9: BERT F1 Score

	1	2	3	4	5	10	15	20
Normal	0.384	0.539	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
Normal	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

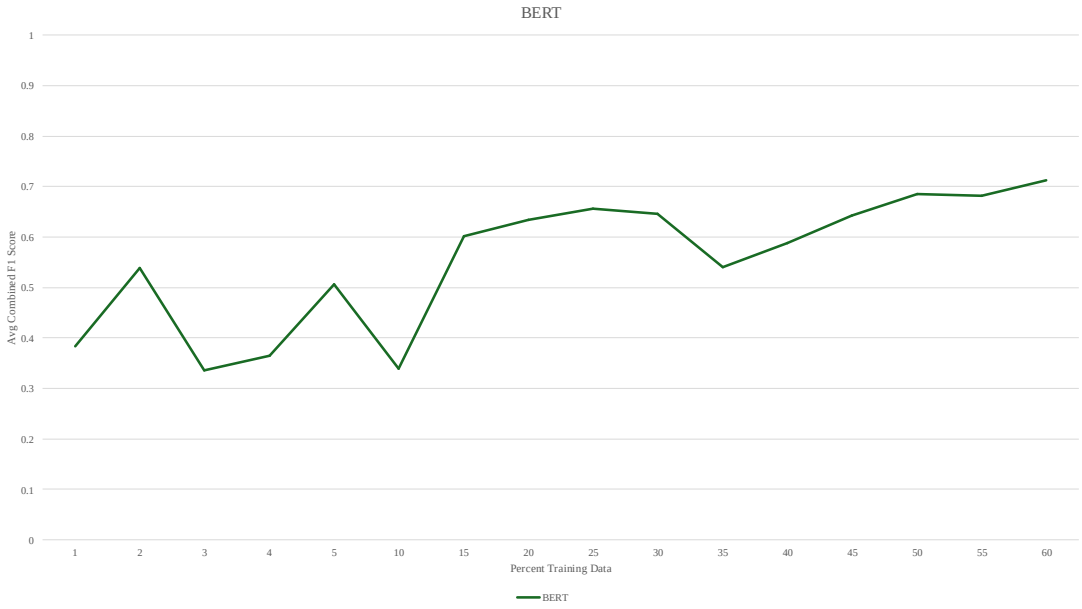

Figure 6.11: BERT F1 Score

6.3 Comparison and Analysis

We will compare the different techniques and networks. We will first examine to see if our approach performed better than its components, then compare the overall performance of the networks.

6.3.1 Compare Networks

Comparing the CNN, LSTM, and BERT networks, we can see that the CNN outperforms the other networks in almost all cases. We can see this clearly in Tables 6.10, 6.11, 6.12, 6.13, 6.14, and 6.15. For all the algorithms, either BERT or the LSTM starts from a higher F1 score. The CNN outperforms the other networks as early as three percent training data. Even in the notable case with the APLSDA algorithm, where the LSTM starts from 0.571, the CNN reaches an F1 score of 0.745 at two percent.

When examining the standard deviation of the networks, we can see that the

CNN has a smaller deviation. In general, the deviation decreases as the amount of training data increases. There is some variability in the change in deviation, but it trends down. This implies that the CNN is learning a better representation of the data in addition to being able to classify positive and negative sentiment.

Finally, we will examine if there is a significant difference in how the different networks performed with the different algorithms. From Table 6.16, we can see a significant difference between the networks for values greater than twenty. There is no clear pattern of significance for values of twenty and under. There are no significant differences for the values of two, three, and four. This implies that for small values of training data, the networks are too variable to be able to statistically pick one as better than the other, regardless of the technique used. We can see that the CNN is statistically the better choice for higher values.

6.3.2 Compare Algorithms

In addition to comparing the differences between the networks, we want to examine the different algorithms to see any patterns in how they performed with the two networks. Looking back at Figures 6.3 and 6.8, we can see that with less than ten percent of the training data, all the algorithms improved the performance of both networks. From values equal to or greater than ten percent, the AL and APLSDA algorithms outperform the PLSDA algorithm. The AL and APLSDA algorithms start outperforming the PLSDA algorithm for both networks by 20 points and outperforming the PLSDA algorithm by at least 50 points by the end. Comparing how the different algorithms affect the average and max scores as the amount of training data changes shows some interesting differences between the networks. For the CNN, the APLSDA algorithm increases the average performance by 0.336 from one percent to two and the max by 0.310. All the algorithms created a significant increase in the max performance for the CNN, but only the APLSDA al-

gorithm created such a significant increase in the average. For the LSTM, there were no sudden changes in performance, but the APLSDA algorithm did start surprisingly high. The standard deviation is unpredictable when determining if the algorithms create a smaller concentration of results. There are no apparent patterns for either network.

6.4 Overall Comparisons

Overall, we can conclude that the CNN outperformed the other networks across all algorithms. The only exception is for values one and two, where either the LSTM or BERT would perform better. As for which algorithm performs best, the AL and APLSDA algorithms outperformed the PLSDA algorithm. However, we can not differentiate between these two algorithms as they performed almost the same. There are two notable exceptions to this. One is for the CNN. The APLSDA algorithm increases the average performance from one percent to two significantly more than the AL algorithm. The other exception is the LSTM, where the APLSDA algorithm produced surprisingly high values for one percent of the training data.

Table 6.10: Active Average F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	0.433	**0.626**	**0.667**	**0.697**	**0.801**	**0.816**	**0.824**
LSTM	0.34	0.441	0.596	0.649	0.703	0.75	0.774	0.783
BERT	**0.384**	**0.539**	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	**0.84**	**0.848**	**0.852**	**0.869**	**0.879**	**0.894**	**0.894**	**0.914**
LSTM	0.794	0.805	0.818	0.819	0.831	0.848	0.847	0.854
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.11: Active Max F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	**0.651**	**0.783**	**0.753**	**0.795**	**0.815**	**0.829**	**0.837**
LSTM	0.366	**0.651**	0.684	0.718	0.732	0.767	0.795	0.809
BERT	**0.384**	0.539	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	0.847	**0.854**	**0.87**	**0.873**	**0.883**	**0.896**	**0.908**	**0.918**
LSTM	0.806	0.825	0.836	0.837	0.843	0.86	0.868	0.872
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.12: PLSDA Average F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	0.452	**0.683**	**0.684**	**0.72**	**0.782**	**0.782**	**0.8**
LSTM	0.381	0.505	0.541	0.59	0.668	0.721	0.751	0.755
BERT	**0.384**	**0.539**	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	**0.82**	**0.822**	**0.823**	**0.82**	**0.825**	**0.828**	**0.832**	**0.832**
LSTM	0.775	0.776	0.784	0.791	0.801	0.812	0.815	0.81
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.13: PLSDA Max F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	**0.706**	**0.756**	**0.766**	**0.791**	**0.813**	**0.816**	**0.826**
LSTM	**0.458**	0.532	0.618	0.7	0.715	0.746	0.773	0.795
BERT	0.384	0.539	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	**0.826**	**0.831**	**0.83**	**0.831**	**0.835**	**0.85**	**0.837**	**0.836**
LSTM	0.796	0.798	0.802	0.808	0.814	0.823	0.824	0.823
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.14: APLSDA Average F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	**0.671**	**0.706**	**0.717**	**0.771**	**0.788**	**0.819**	**0.83**
LSTM	**0.531**	0.552	0.568	0.608	0.638	0.735	0.767	0.775
BERT	0.384	0.539	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	**0.84**	**0.851**	**0.864**	**0.865**	**0.875**	**0.891**	**0.902**	**0.913**
LSTM	0.785	0.789	0.813	0.826	0.836	0.844	0.848	0.866
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.15: APLSDA Max F1 Score Comparison

	1	2	3	4	5	10	15	20
CNN	0.335	**0.745**	**0.779**	**0.794**	**0.797**	**0.814**	**0.826**	**0.839**
LSTM	**0.571**	0.652	0.661	0.747	0.722	0.763	0.794	0.8
BERT	0.384	0.539	0.336	0.365	0.506	0.339	0.602	0.635

	25	30	35	40	45	50	55	60
CNN	**0.846**	**0.858**	**0.867**	**0.874**	**0.883**	**0.893**	**0.904**	**0.915**
LSTM	0.81	0.819	0.833	0.844	0.852	0.856	0.864	0.875
BERT	0.656	0.647	0.540	0.588	0.642	0.685	0.681	0.713

Table 6.16: P Values Between LSTM and CNN

	1	2	3	4	5	10	15	20
Normal	**1E-05**	0.097	0.095	0.088	0.119	0.069	**0.036**	0.055
PLSDA	0.089	0.144	0.093	0.1	**0.049**	**0.043**	**0.031**	**0.044**
Active	**0.007**	0.118	0.095	0.075	0.096	**0.033**	**0.027**	**0.024**
APLSDA	0.104	0.084	0.116	0.115	0.091	**0.037**	**0.029**	**0.031**

	25	30	35	40	45	50	55	60
Normal	**0.048**	**0.024**	**0.016**	**0.016**	**0.019**	**0.015**	**0.012**	**0.015**
PLSDA	**0.026**	**0.03**	**0.024**	**0.02**	**0.017**	**0.011**	**0.01**	**0.014**
Active	**0.026**	**0.024**	**0.026**	**0.027**	**0.025**	**0.025**	**0.035**	**0.034**
APLSDA	**0.031**	**0.04**	**0.028**	**0.029**	**0.024**	**0.025**	**0.032**	**0.025**

Chapter 7

Conclusion

We will examine our research questions individually, starting with Question 1.

1. Does combining AL and LE offer better performance?

Our combined APLSDA algorithm performed significantly better than the PLSDA algorithm and the unmodified approaches. However, it was only significantly different from the AL approach for the CNN at the value 3. For the LSTM, the differences between the APLSDA and AL approach were significantly different for all values but 3 and 4. However, the performance of the algorithms was not consistent with both the APLSDA and the AL algorithm performing the best with no clear pattern. In regards to our expectations, our algorithm did result in better performance than the unmodified approach, while our APLSDA and the AL approach we comparable. While the average performance of our APLSDA algorithm was better than the AL algorithm for the CNN for training values up to 40 percent, the differences were minor and not significant.

2. Is it possible to use this technique to create a classifier whose performance improves faster?

Our algorithm did help the classifiers improve faster than all the other techniques. For training values of one or two percent, our algorithm outperformed

the other techniques significantly. This is most apparent in the LSTM, where the APLSDA algorithm started over ten points higher than the other algorithms for both its max F1 and its average. For higher training values the APLSDA and the AL algorithms are similar again when examining the rates of improvement. These results match our expectations that our technique would help improve the classifiers faster than the other results.

3. Does the proposed algorithm behave differently with different classifier architectures?

To answer question 3 we can conclude that the CNN network outperformed both the LSTM and the BERT model for training values higher than two percent. For the very small values, the LSTM performed surprisingly well. While this performance is notable, overall the CNN is the more useful architecture. Our expectations that the APLSDA algorithm would help improve the performance of any classifier regardless of architecture were supported by our research. Both the LSTM and the CNN saw significant improvements with our APLSDA algorithm. We did not expect that the CNN would outperform the LSTM so drastically.

4. Is there a large variance between the average and best performance of the classifiers?

By examining the standard deviation of the different algorithms, we were able to see how their performance changed over time. Overall, all the algorithms trended towards a smaller standard deviation. Our APLSDA algorithm had the smallest deviation at 60 percent training data. However, the other algorithms had smaller deviations leading up to 60 percent. We can conclude that all algorithms increase the accuracy of their predictions as the amount of training data increases. While the PLSDA algorithm performed the worst out of the three improvements tested, it

did have much lower standard deviations for some of the smaller training values. This result did not align with our expectations. We had predicted that introducing artificial data from the PLSDA and APLSDA approaches may hurt the deviation of those classifiers. However, this does not appear to be the case. This result is encouraging, as it shows that the data points generated by the PLSDA and APLSDA algorithms are similar to the real data.

Overall, our algorithm and the AL algorithm performance are too similar to say that one is better than the other. However, we have some useful results from our research. We found significant improvements in all training percentages when using either technique. When focusing on very small amounts of training data, we found that our algorithm outperformed the AL algorithm. Finally, by examining the average performances of the classifiers as well as the max, we saw that there could be a significant amount of variance between the performance of two classifiers that are trained on the same data. All these findings open up interesting future avenues for continuing this research.

7.1 Future Work

Our research brought forth several future research avenues. These include investigating the LSTM, trying different combinations of techniques, further investigating into average performances of classifiers, and the potential to examine several classifiers. One of the most interesting results from this research was the incredible performance of the LSTM with the APLSDA algorithm at one percent training data. It achieved a performance that was slightly better than guessing when the other classifiers were only capable of supplying a single label. Investigating what this classifier has learned and how it differs from the CNN and the BERT model is an area of potential future research. The second avenue of research could be

combining other techniques. We selected two techniques to try and combine. Several other AL and LE algorithms could be beneficial to use together. There is also potential to examine different ways of combining the techniques. We only examined a complete combination of the techniques; it would be useful to examine if starting from one technique before transitioning to another would offer any benefits. We found that the performance of the classifiers was not as consistent as we would have hoped. This was especially clear with training values of less than five. Future research could focus on reducing the variance of the classifiers at low training values. Finally our research showed that there can be increable differences in the performance of one classifier when compared to another. This encourages further research into different architectures and experimentation with unusual neural network structures.

Bibliography

[1] G. Chevalier, "Lstm cell." `https://commons.wikimedia.org/wiki/File:LSTM_Cell.svg`, May 2018. Accessed: 2023-05-11.

[2] J. Prusa, T. M. Khoshgoftaar, and N. Seliya, "The effect of dataset size on training tweet sentiment classifiers," in *2015 IEEE 14th International Conference on Machine Learning and Applications*, ICMLA 2015, pp. 96–102, Dec 9-11, 2015.

[3] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," in *35th Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*, NeurIPS 2021, Dec 6-14, 2021.

[4] V. S. Sheng, F. Provost, and P. G. Ipeirotis, "Get another label? improving data quality and data mining using multiple, noisy labelers," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2008, pp. 614—-622, Association for Computing Machinery, 2008.

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), NIPS 2017, Curran Associates, Inc., Dec 8-13, 2014.

[9] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (A. Moschitti, B. Pang, and W. Daelemans, eds.), EMNLP 2014, pp. 1724–1734, Association for Computational Linguistics, Oct 25-29, 2014.

[10] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *8th International Conference on Learning Representations*, ICLR 2020, Apr 26-30, 2020.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), NIPS 2017, Curran Associates, Inc., Dec 4-9, 2017.

[12] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), NAACL 2019, pp. 4171–4186, Association for Computational Linguistics, Jun 2-7, 2019.

[14] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, Dec 2020.

[15] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, ICCV 2015, pp. 19–27, Dec 7-13, 2015.

[16] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—a review of research topics, venues, and top cited papers," *Computer Science Review*, vol. 27, pp. 16–32, 2018.

[17] B. Settles, "Active learning literature survey," Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[18] M Yasser H, "Movie ratings sentiment analysis." https://www.kaggle.com/datasets/yasserh/imdb-movie-ratings-sentiment-analysis. Accessed: 2022-02-15.

[19] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics*, vol. 9, no. 3, pp. 1–29, 2020.

[20] R. K. Behera, M. Jena, S. K. Rath, and S. Misra, "Co-lstm: Convolutional lstm model for sentiment analysis in social big data," *Information Processing & Management*, vol. 58, no. 1, p. 102435, 2021.

[21] Q.-H. Vo, H.-T. Nguyen, B. Le, and M.-L. Nguyen, "Multi-channel lstm-cnn model for vietnamese sentiment analysis," in *2017 9th International Conference on Knowledge and Systems Engineering*, KSE 2017, pp. 24–29, 2017.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[23] Z. Zhang, E. Strubell, and E. Hovy, "A survey of active learning for natural language processing," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), EMNLP 2022, pp. 6166–6190, Association for Computational Linguistics, Dec 7-11, 2022.

[24] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[25] K. Margatina, G. Vernikos, L. Barrault, and N. Aletras, "Active learning by acquiring contrastive examples," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, eds.), EMNLP 2021, pp. 650–663, Association for Computational Linguistics, Nov 7-11, 2021.

[26] B. Settles, M. Craven, and S. Ray, "Multiple-instance active learning," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), vol. 20 of *NIPS 2007*, Curran Associates, Inc., Dec 3-8, 2007.

[27] H.-S. Chang, S. Vembu, S. Mohan, R. Uppaal, and A. McCallum, "Using error decay prediction to overcome practical issues of deep active learning for named entity recognition," *Machine Learning*, vol. 109, pp. 1749–1778, Sep 2020.

[28] J. Zhu, H. Wang, T. Yao, and B. Tsou, "Active learning with sampling by uncertainty and density for word sense disambiguation and text classification," in *2008 - 22nd International Conference on Computational Linguistics, Proceedings of the Conference*, COLING 2008, pp. 1137–1144, Aug 18-22, 2008.

[29] V. Ambati, *Active learning and crowdsourcing for machine translation in low resource scenarios*. PhD thesis, Carnegie Mellon University, 2012.

[30] A. Erdmann, D. J. Wrisley, B. Allen, C. Brown, S. Cohen-Bodénès, M. Elsner, Y. Feng, B. Joseph, B. Joyeux-Prunel, and M.-C. de Marneffe, "Practical, efficient, and customizable active learning for named entity recognition in the

digital humanities," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), NAACL-HLT 2019, pp. 2223–2234, Association for Computational Linguistics, Jun 2-7, 2019.

[31] M.-A. Rocha and J.-A. Sanchez, "Towards the supervised machine translation: Real word alignments and translations in a multi-task active learning process," in *Proceedings of Machine Translation Summit XIV: Posters* (A. Way, K. Sima'an, and M. L. Forcada, eds.), MTSummit 2013, Sep 2-6, 2013.

[32] J. Chen, D. Tam, C. Raffel, M. Bansal, and D. Yang, "An empirical survey of data augmentation for limited data learning in nlp," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 191–211, Mar 2023.

[33] R. Xiang, E. Chersoni, Q. Lu, C.-R. Huang, W. Li, and Y. Long, "Lexical data augmentation for sentiment analysis," *Journal of the Association for Information Science and Technology*, vol. 72, no. 11, pp. 1432–1447, 2021.

[34] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III, "Deep unordered composition rivals syntactic methods for text classification," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (C. Zong and M. Strube, eds.), ACL-IJCNLP 2015, pp. 1681–1691, Association for Computational Linguistics, Jul 26-31, 2015.

[35] Z. Miao, Y. Li, X. Wang, and W.-C. Tan, "Snippext: Semi-supervised opinion mining with augmented data," in *Proceedings of The Web Conference 2020*, WWW 2020, pp. 617—628, Association for Computing Machinery, Apr 20-24, 2020.

[36] Y. Cheng, L. Jiang, and W. Macherey, "Robust neural machine translation with doubly adversarial inputs," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (A. Korhonen, D. Traum, and L. Màrquez, eds.), ACL 2019, pp. 4324–4333, Association for Computational Linguistics, Jul 28 - Aug 2, 2019.

[37] J. Chen, Z. Yang, and D. Yang, "MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, eds.), ACL 2020, pp. 2147–2157, Association for Computational Linguistics, Jul 5-10, 2020.

[38] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st ed., 2009.

[39] C. Yin, B. Qian, S. Cao, X. Li, J. Wei, Q. Zheng, and I. Davidson, "Deep similarity-based batch mode active learning with exploration-exploitation,"

in *2017 IEEE International Conference on Data Mining*, ICDM 2017, pp. 575–584, Nov 18-21, 2017.

[40] O. Shahmirzadi, A. Lugowski, and K. Younge, "Text similarity in vector space models: A comparative study," in *2019 18th IEEE International Conference On Machine Learning And Applications*, ICMLA 2019, pp. 659–666, Dec 16-19, 2019.

[41] C. Fellbaum, *WordNet: An electronic lexical database*. MIT press, 1998.