Project Milestone 3
Team: Hex HyperCity

Mason Lien
Matthew Penne
Mrinal Rawool
Ketemwabi Yves Shamavu

Contents

1 Milestone 1: Project Ideas **1**
- 1.1 Introduction . . . 1
- 1.2 Project Idea 1: Natural Language Inference using Deep Learning 1
- 1.3 Project Idea 2: Evaluating Freeze Injury in Winter Wheat with Deep Learning . . . 2
- 1.4 Project Idea 3: Clinical Diagnosis Based on the NIH Chest X-ray Dataset . . . 4
- 1.5 Conclusions . . . 4

2 Milestone 2: Project Selection **5**
- 2.1 Introduction . . . 5
- 2.2 Problem Specification . . . 6
- 2.3 Proposed Method 1: Long Short Term Memory . . . 7
- 2.4 Proposed Method 2: Reusing Embeddings pre-trained Through Self-Supervision . . . 8
- 2.5 Conclusions . . . 10

3 Milestone 3: Progress Report 1 **11**
- 3.1 Introduction . . . 11
- 3.2 Experimental Setup . . . 12
- 3.3 Experimental Results . . . 13
- 3.4 Discussion . . . 13
- 3.5 Conclusion . . . 13

4 Milestone 4: Progress Report 2 **15**
- 4.1 Introduction . . . 15
- 4.2 Experimental Setup . . . 15
  - 4.2.1 Output Layers . . . 15
  - 4.2.2 Training BERT . . . 16
  - 4.2.3 Comparing pre-trained BERT models with two transformer layers . . . 16
- 4.3 Experimental Results . . . 16
  - 4.3.1 Output Layers . . . 16
  - 4.3.2 Training BERT . . . 18

4.3.3 Results of the comparison of pre-trained BERT variants on the SNLI task . . . 19
4.4 BERT Embedding Visualization . . . 21
4.4.1 Visualizing Context Learning . . . 22
4.4.2 Visualizing Token Neighborhoods . . . 28
4.4.3 Visualizing BERT Pooled Output for Sentence Pairs . . . 29
4.5 Discussion . . . 32
4.6 Conclusion . . . 33

5 Milestone 5: Final Report **35**
5.1 Introduction . . . 35
5.2 Progress at Each Milestone . . . 35
5.3 Experimental Setup . . . 36
5.3.1 Dropout Rate . . . 36
5.3.2 Training Data . . . 36
5.3.3 BERT Layers and Hidden Units . . . 37
5.3.4 Specific Models . . . 37
5.3.5 Custom BERT Model . . . 37
5.4 Experimental Results . . . 39
5.4.1 Dropout Rate . . . 39
5.4.2 Training Data . . . 40
5.4.3 BERT Layers and Hidden Units . . . 40
5.4.4 Specific Models . . . 40
5.4.5 Results of Custom BERT Model . . . 41
5.5 Discussion . . . 41
5.6 Model Recommendation . . . 43
5.7 Conclusion . . . 43

Bibliography **45**

Abstract

Natural language inference (NLI) is the process of inferring the meaning of a sentence. Transfer learning consists of importing a model pre-trained on a similar task and modifying it to the desired dataset. Bidirectional encoder representations from transformations (BERT) is a large model developed by Google for encoding sentences. This project uses transfer learning from various BERT models and additional output layers for NLI. In the NLI task, two statements, a hypothesis and a premise, are fed to the model. The model subsequently predicts whether the hypothesis is true, false, or undetermined given the premise. When the hypothesis is true, it is said to be entailed by the premise. When it is false, it is contradicted by the premise, and when it is undetermined, it is neutral with respect to the premise.

The group compared different output layers on the BERT model for accuracy and training time. Additionally, impacts on how many epochs the BERT model is trained, The training accuracy and loss for several different BERT sizes are compared. Visualisations of the BERT embedding are shown.

The best model found used a BERT model with 8 layers, 512 hidden units, and 8 attention heads and was trained with 100k examples. The model output layers had 64 bi-directional LSTM units, followed by average and max pooling layers which were concatenated, passed through a dropout layer with a 30% dropout rate and then a densely connected layer with three neurons and softmax activation. The model was trained for two epochs where only the output layers were trained, followed by two epochs were the entire model was trained. The model produced an accuracy of 83.1% with an average training time of 16.5 minutes per epoch.

Chapter 1

Milestone 1: Project Ideas

1.1 Introduction

Project ideas were inspired from online datasets sharing platforms such as Kaggle and group members personal research. The group also looked at published literature for other curated datasets, including the papers by [30, 10, 29, 24, 19].

Since the data sets mentioned by [24] and [19] required extensive module-specific CITI training and given the limited timeframe for our project, we excluded these papers from the project ideas detailed below.

1.2 Project Idea 1: Natural Language Inference using Deep Learning

Introduction

Natural Language Processing (NLP) is a domain that deals with the field of computer science and linguistics to devise ways for humans to interact with machines using human language. NLP includes some low-level tasks whose objective is to learn linguistic context. Examples of such tasks are parts-of-speech tagging, named entity recognition, information extraction, relationship extraction etc. Such tasks fall under shallow learning as the machine does not seek to understand the meaning or context of the content. On the other hand, high-level tasks such as question-answering, reading comprehension, and inference require the use of reasoning and knowledge in order to achieve the intended objective. According to Oxford dictionary, inference is defined as the process of reaching a conclusion based on evidence and reasoning. Currently, research within the realm of NLI can be broadly classified in two categories; textual inference and plausible inference[25]. Textual inference tasks is characterised by a definite, concrete hypothesis for every premise-hypothesis pair while plausible inference tasks require abductive reasoning and external knowledge. In this project, our objective is to build a deep learning model to solve a 3-way textual inference

task.

Problem Statement
The inference task can be described as semantically determining whether a **hypothesis** statement is **true**, **false**, or **undetermined** given the **premise**. A true, false, and an undetermined statement is termed as *e*ntailment, contradiction, and **neutral**, respectively.

Examples:
*P*remise: A man inspects the uniform of a figure in some East Asian country
*H*ypothesis: The man is sleeping
*L*abel: Contradiction
*P*remise: A soccer game with multiple males playing.
*H*ypothesis: Some men are playing a sport.
*L*abel: Entailment
*P*remise: A black race car starts up in front of a crowd of people.
*H*ypothesis: A man is driving down a lonely road.
*L*abel: Neutral

Applications
Natural Language Inference tasks are a stepping stone towards progress in question answering or semantic search tasks. Any task that requires the model to look beyond keywords and syntax and grasp the meaning and context of text uses some kind of inference-based technique to solve the task at hand. NLI is also used as an evaluation technique for machine translation task.

Approaches and Data set
The team plans to use a benchmark data set released in 2015 called Stanford Natural Language Inference corpus, also known as SNLI [11]. This data set is a collection of 570k human-written English sentence pairs manually labeled as entailment, contradiction, and neutral. The corpus includes a train, test, and development split with the last two having close to 10K examples each [6]. Additionally, we would be able to track our model performance by comparing it with the performance metrics of other models that use SNLI. This information is made available through a leaderboard [4].

1.3 Project Idea 2: Evaluating Freeze Injury in Winter Wheat with Deep Learning

Introduction
Freeze injury (FI) is an abiotic stress that significantly impacts normal growth in plants [16]. For winter wheat grown in the Midwest, the probability of experiencing FI is high, since it is planted in the fall where normal growth consists

of germination, emergence, and tillering [9]. Low temperatures are known to kill plants by damaging the crown of the wheat plant, which is the main point of growth. The severity of the injury is influenced by both low temperatures and the duration of low temperatures. When FI occurs, stem growth can be delayed or terminated, resulting in plants tillering at different times. This response causes moderate to severe yield reductions and prolongs harvest due to fields having a non-uniform dry down [9], reducing profits and food supplies.

In regard to the spring of 2020, most of the state of Nebraska experienced significant freezing temperatures between April 10-16th, where lows were below 20 degrees Fahrenheit for an extensive period of time. At the University of Nebraska-Lincoln Havelock research farm, current growth stages were at a tillering stage for the winter research plots being conducted for agriculture studies. At this growth stage, the research plots were moderatly susceptible to FI and had increased the risk of significant injury the closer to jointing. Typical injury observed during this growth stage is leaf burning, where leaf tissue turns from a healthy green to yellow and finally brown if the plant does not recover. One of the main objectives of the UNL small-grains breeding program is to develop wheat varieties that are tolerant to these types of environmental conditions experienced in The Great Plains. Traditionally these plots are subjectively screened visually for agronomic and performance traits that dictate the advancement into the breeding program. Visual scoring is subjective and introduces inter-rater bias and not the most robust method to produce quality data. Recent advancements in high-throughput phenotyping (H-TP) has allowed the use of imagery analysis to replace traditional scoring methods with more robust and repeatable methods of accurately quantifying agronomic traits like FI.

Objectively, H-TP paired with deep learning models like convolutional neural networks (CNN) can accelerate the rating of FI severity on research plots and identify cold-tolerant varieties that would be ideal for advancement into commercialization. Developing a deep learning method to evaluate FI in winter wheat is novel approach and the datasets are rare to collect due to the randomness of when freeze events occur. There is much interest in this work from the UNL agricultural engineering department as well as the UNL small-grains breeding program to better understand FI and develop advanced models for improving breeding methodologies.

Approaches and Data set

The FI dataset consists of 4,000 individual plot images ranging from healthy to severely injured winter wheat plots with annotated severity ratings (1-9), where 1 is healthy and 9 is severely injured.

The objectives of this study are the following:

1. Train a number of differing deep learning CNN models to predict FI.
2. Compare models and evaluate which generalizes the best for FI.
3. Suggest future work for further developing this research.

1.4 Project Idea 3: Clinical Diagnosis Based on the NIH Chest X-ray Dataset

Chest X-rays are a cost-effective and frequently medical imaging examination. But, it is hard to establish a clinical diagnosis using chest X-rays compared to other expensive medical imaging techniques such as CT imaging.

The goal of this project is to achieve clinically relevant computer-aided detection and diagnosis using chest X-rays with convolutional neural networks.

The dataset includes about 112,000 X-ray images with disease labels from about 30 thousands patients. The labels were created using text extraction from radiological reports [29]. Hence, the labels are not 100 percent accurate and are suitable for weakly-supervised learning.

The X-ray images can belong to one or more of the following 15 classes: Atelectasis, Consolidation, Infiltration, Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural_thickening, Cardiomegaly, Nodule Mass, Hernia, and No findings.

This project is interesting because several intensive and emergency care units across the United States rely on X-ray imaging diagnosis to decide on the best treatment in a time-sensitive context.

Unfortunately, besides the low accuracy of the provided true labels, only about one thousand images come with predefined bounding boxes. The detection of regions of interest will be unguided for the most part, which can increase training time and impinge on the convergence.

1.5 Conclusions

The topics are presented in the order the group is interested in them. The natural language inference project seems to be the most interesting and has a complete dataset, but the group is unsure which model will be the best fit. Evaluating freeze injury in winter wheat has be partially done by one of the group members for their research. But the dataset is not a large as the group would like. The project may be expanded to additional crops to compensate. The third project is a more classical project in convolutional neural networks. There are multiple data sets available for this project and models that can be adapted for this project.

Chapter 2

Milestone 2: Project Selection

The group decided to work on a Natural Language Inference (NLI) project. The goals of NLI is to represent the meaning of a sentence as a vector. This will be useful for question answering, information retrieval and extraction, and text summarizing. The project will focus on single sentences but can eventually be scaled to paragraphs or entire documents. Our specific project will input a statement and a hypothesis and the output will be if they support each other, if they are neutral to each other, or if they contradict each other.

2.1 Introduction

Natural Language Inference (NLI) task is falls under the purview of Natural Language Processing (NLP). Recognizing the notions of entailment and contradiction are core to the process of natural language understanding. Thus, modelling inference can further enhance semantic representation of language.

In this task, we try to infer the relationship between a pair of sentences using a deep learning model. When such a task is presented to a person, the relationship between sentences is inferred from the meaning of words or semantics. In a neural network model, this semantic modelling is achieved using two mechanisms, attention and language representation. At a high level, this process entails two tasks to be performed by a deep learning model. First, every word in a sentence is represented numerically such that the numbers associated with a word represent it's similarity to other words in the sentence. Additionally, each sentence is parsed by the model to look for important words, or terms that it needs to pay 'attention' to. This step is done for both premise, and the hypothesis. Once such words are identified, the model measures the similarity between the key terms from the pair of sentences and predicts the relationship between them.

As stated before, this task can be broken down into two sub tasks. The first

one aims at capturing the meaning of words through language representation. This can be achieved by creating a vocabulary of all words present in the corpus or data set and representing them as vectors that capture the relationship of a particular word with respect to the rest of the words in the vocabulary. However, this is a naive approach that has been proven insufficient to capture the complexities of natural language due to its vast nature and inherent complexities. Thankfully, research in the area of language representation has led to the development of pre-trained models such as GLoVE [3], Word2Vec [22] etc that could be used directly as an embedding layer or could be fine tuned for the task at hand.

Furthermore, research has shown that using pre-trained models on a task similar to the one being undertaken help the model achieve better performance on downstream tasks. For this reason, using pre-trained models such as BERT [13], and ELMo [23] that have been trained for question-answering or auto-summarization tasks are a good candidate to use for the inference task. Moreover, LSTM and transformer based architectures are better suited to handle natural language processing tasks due to their ability to retain context while working with sequential data. Thus models that are LSTM based (ELMo) or transformer based (BERT) have attention mechanism included making them suitable candidates for the inference task.

While creating an accurate semantic representation of natural language is a complicated task in itself, progress in this area started with the introduction on generic pre-trained word representations. Research in this area continues as pre-trained models for specific tasks have been introduced to facilitate building better models and reducing the time required to train such models through transfer learning. At present, there does not exist a pre-trained model that is trained on an inference task. One of the impacts of this project is availability of a pre-trained model that is trained specifically for inference task. Furthermore, such a model could be extended to provide explanations for inference or fine tuned for domain specific inference.

2.2 Problem Specification

This project aims at building a fast, efficient and accurate model that learns language representation to apply it on an inference task. The motivation behind taking up this challenge is to

1. Build a model that effectively learns the representation and not just relies on artifacts present in the data.

2. Create a model that could be fine-tuned for inference on a domain specific task such as scientific data or clinical data.

The dataset consist of a pair of sentences called premise and hypothesis. The task is to predict whether the hypothesis entails or contradicts the premise. Three types of inference labels are possible, 'entailment', 'contradiction', 'neutral'.

The project will be developed in two phases. The data set that we would be using for training, validating, and testing during the first phase is the Stanford Natural Language Inference dataset [11]. Based on the leaderboard published at the Stanford NLP website [7], the top test accuracy achieved by a deep learning model on this dataset is 92.1. We aim to build a model that attains a test accuracy within the top 20 range.
Phase two consist of testing the quality of language representation of the model. The proposed plan is to fine tune the model on the Heuristic Analysis for NLI Systems (HANS) dataset [21], which is a controlled evaluation set containing examples that do not contain any artifacts or heuristics that could be used by a model for inference label prediction. Performance of our model on a challenging dataset like HANS will help us evaluate the learned representations of our model better.

Our experiments would be performed on crane and will use libraries such as Numpy and Pandas for data processing. Seaborne and Matplotlib for data visualization, Transformers and Tensorflow Hub for pre-trained models, and Keras and Tensorflow for building model layers.

2.3 Proposed Method 1: Long Short Term Memory

The Long Short Term Memory (LSTM) technique helps alleviate the vanishing gradient problem for longer phrases is text based recurrent neural networks. The structure of a LSTM cell is shown in 2.1 [15].The inputs are the current data input x(t), the previous state long term memory c(t-1), and the previous state short term memory h(t-1). The goal of the LSTM cell is to have a sense of long term memory of the data throughout the RNN and to have a short term memory of more recent data.

For the forget signal f(t) the previous short term data, h(t-1) and the current input data x(t) are input into a logistic function whose output is then multiplied by the previous state long term memory c(t-1) at the forget gate. This "forgets" some of the previous long term data. The next gate, g(t) is the main gate that takes h(t-1) and x(t) through a tanh function to have the new data that is added to the previous long term memory. Before g(t) is added to the long term memory it is multiplied by i(t), which is similar in activation to f(t), which decideds how much to g(t) is added to the long term memory. That long term memory is passed to the next LSTM cell. This long term memory is sent through another tanh function then multiplied by o(t), which serves the same function as i(t) in that it decides how much information is passed to the short term memory h(t). The output of the cell y(t) is equal to the short term memory output h(t).

The group plans on using a model including LSTM's among other techniques to achieve high validation accuracy on different NLI corpus's. The main data set is the Stanford natural language inference corpus, with the multi-genre natural language inference corpus and the SciTail entailment dataset also being

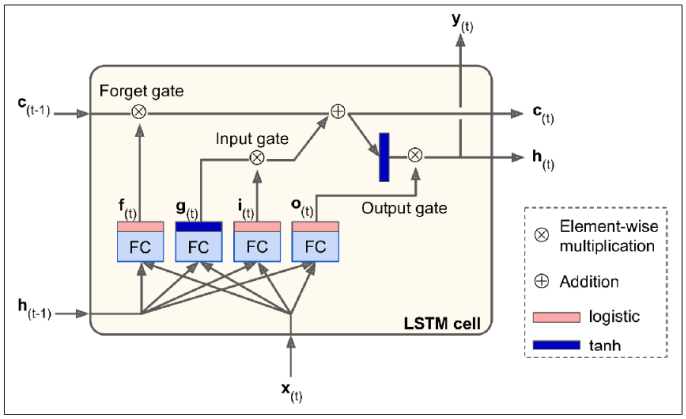


Figure 2.1: LSTM cell block diagram.

applicable.

The performance of our model can be compared with a scoreboard of notable NLI models found at [5]. Also, notable models can be used for comparison on inspiration [28], [26].

2.4 Proposed Method 2: Reusing Embeddings pre-trained Through Self-Supervision

This approach will use pre-trained embeddings from a Biredirectional Encoder Representation from Transformers (BERT) Tensorflow module hosted at the TensorFlow Hub project.

Each word or sub-word appearing in a text has to be treated as an individual text token for a model to understand an unstructured text. Thus, each token's representation can be pre-trained on a much larger corpus, such as word2vec, GloVe, or other subword embedding models.

Upon pretraining, words or tokens can have a vector representation, allowing a model to learn and recognize particular patterns [31].

However, the vector representation remains the same no matter what the context is. For instance, the vector representation of "state" is the same in both "the state of Nebraska" and "it is in a good state."

Recent start-of-the-art (SOTA) models attempt to provide a context-based vector representation of tokens or words in a sentence. BERT [13], based on the Transformer Encoder, is amongst such SOTA architectures. Figure 2.2 depicts how BERT fits into our second approach.

In the next two weeks, we will explore a range of BERT modules from the Tensorflow Hub project to find which can be adequately fine-tuned for the SNLI task.

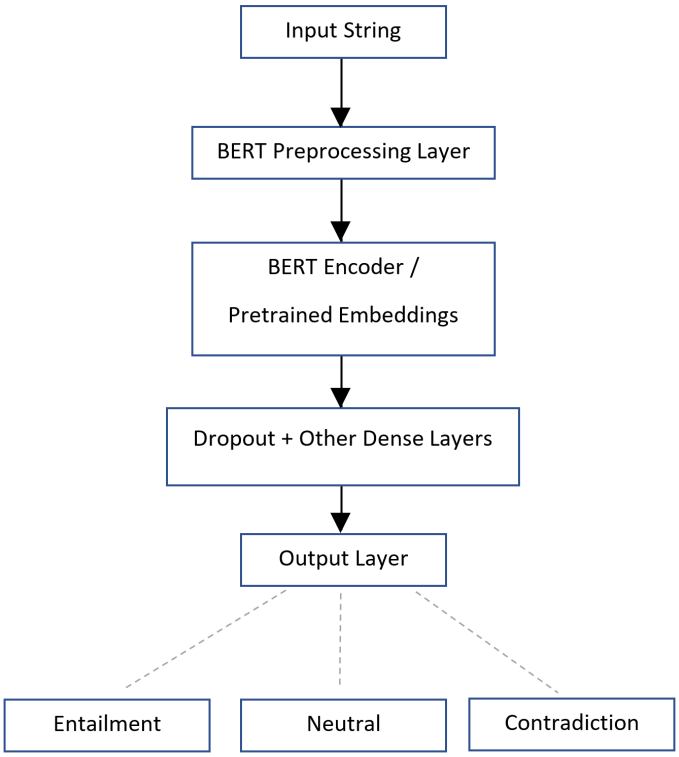


Figure 2.2: High-level representation of an architecture reusing embeddings pre-trained on large corpora through self-supervision with BERT or BERT-like models. Adapted from [2, 8].

Some of the BERT models we will explore are described by [1, 8] and include: the BERT-Base modules, which BERT authors initially released; the Small BERTs, which might be suitable to assess a trade-off between speed, size, and quality of predictions; ALBERT or A Lite BERT, which allows parameter sharing across layers thus reducing the model size; BERT Experts, a series of eight BERT modules providing specific embeddings based on the task; etc.

By the third milestone, we shall report on the BERT module that provided the task's best performance.

In the next steps, we will try out different hyperparameter values through cross-validation. We shall also tweak the architecture in a bid to attain a performance that may rank among the top based on the benchmark at the SNLI web page.

2.5 Conclusions

The goal of using natural language inference is to represent sentences as vectors. This project will compare those vector together and decide if they support each other, or neutral to each other, or are contradictory. With enough progress similar models can represent paragraphs or entire documents that can be used for question answering and summarizing. For model architecture, LSTMs, GRUs, and other RNN techniques will be explored since they have been used to different degrees of success in literature for models on the scoreboards [[**?**]]. Similar models can also be leveraged using BERT, which will hopefully yield high accuracy with lower training times.

Chapter 3

Milestone 3: Progress Report 1

3.1 Introduction

The group is working on providing a fast and accurate model to predict semantic relationships between a pair of statements, or commonly known as natural language inference (NLI). The goal is to infer the relationship between two sentences with a deep learning model. The model predicts whether the two statements, the premise and the hypothesis have an entailment, contradictory, or neutral relationship, whereby the hypothesis truthfulness is dependent on the premise. The output is zero for entailment, one for neutral, and two for contradiction. The dataset used for training, validation and testing is the Stanford natural language inference dataset (SNLI) [10] and it was retrieved from the Tensorflow Datasets collection.

The group is using transfer learning from bidirectional encoder representations from transformers, or BERT, for NLI [14]. A BERT model is imported from tensorflow hub, and with additional preprocessing layers and output layers the BERT model is trained on the SNLI dataset. The BERT model is very large, with the best MultiGenre NLI model having close to 110 million parameters and requires a significant amount of training time. Our goal is to use smaller versions of BERT with additional input and output layers to achieve a high accuracy with the SNLI data set, and perhaps other data sets, while maintaining a reasonable training time.

For this milestone, the group used three different models of BERTs, a small, medium, and large. As expected training time increased with model complexity but non linearly. Also, test accuracy increased with model complexity but again did not scale linearly. Results are discussed further in 3.3.

Table 3.1: Other Network Parameters for Search Space

Models	L	H	A	Trainable Parameters	Non-trainable Parameters
Tiny BERT	2	128	2	99587	4385921
AlBERT	24	1024	16	262659	17683968
MNLI-BERT	12	786	12	427267	109482241

3.2 Experimental Setup

The SNLI data set is obtained from [10]. It includes 500,000 training examples, 10,000 validation, and 10,000 test. The preprocessing layer is taken from [14]. The preprocessing tokenizes the input sentences, or splits it into individual words and word parts. The tokenizer uses 'wordpiece tokenization'. This is based on a sub-word segmentation algorithm popular in NLP tasks. This algorithm retains frequently used sub-words while splitting the infrequently occurring words. For example, the word 'annoyingly' is split into tokens 'annoy', 'inly' since there are far many words that end with 'inly' than the original word. This kind of tokenization helps in limiting the size of the corpus vocabulary which in turn makes the tokenization process efficient and economical. The inputs to the BERT model are the input words, the segment or phrase embeddings, and the position embedding of where the word is in the dataset [17]. The maximum sentence length used is 128, with an embedding dimension of 768.

The BERT models are loaded from Tensorflow hub. Three different BERTs are tested according to their size of parameters: small, medium and large. The small model, or Tiny BERT, has two layers, with a hidden size of 128, and 2 self-attention head [27]. It is a trimmed down version of the original BERT model. The medium model is an AlBERT model [20]. AlBERT is similar to BERT, but it can be trained faster due to the vocabulary being split into two smaller matrices and layers sharing parameters. It has 24 layers, a hidden size of 1024, and 16 attention heads. The large model is the original BERT that was fine tuned on the MNLI dataset for 12 epochs [14]. It has 12 layers, a hidden size of 786, and 12 attention heads. A summery of the models is shown in Table 3.1.

The output of the BERT is passed to a bidirectiononal LSTM layer with 64 units. The LSTM layer is then used as an input to a max pool and average pool layer that are concatenated. The outputs of this layer are then dropped at a rate of 30% as they are passed to the output layer. The output layer is 3 dense neurons that have softmax activation. The three neurons are for each of the possible outputs.

A batch size of 32 is used to help limit training time. Adam optimization with an initial learning rate of 1e-5 is used. For each model two epochs are trained without training the imported BERT model, then two epochs are trained with training the BERT model, or fine tuning. The first two epochs allow the non-BERT layers to be trained, while the last two epochs fit the entire model to the problem space. The accuracy and the training time for each model is measured.

Table 3.2: Other Network Parameters for Search Space

Models	Total Parameters	Training Time	Test Accuracy
Tiny BERT	4,485,508	1.9 hr	77%
AlBERT	17,946,627	2.2 hr	85%
MNLI-BERT	109,482,241	2.8 hr	90%

3.3 Experimental Results

As expected, tiny-BERT performed the fastest and the least accurate while the MLNI-BERT model took the longest to compute and had the most accurate results. The results are summarized in Table 3.2.

3.4 Discussion

The test accuracy vs. model parameters is shown in Fig 3.1 and the computation time vs. model parameters is shown in Fig 3.2. The MNLI-BERT increased the accuracy of the Tiny-BERT model by 13% but took 47.4% longer to compute. The AlBERT model increased accuracy by 7% and only increased computation time by 15.8%. This leads one to believe that the methodology behind the AlBERT model decreases training time while maintaining high accuracy. The additional training on the MNLI-BERT model on a similar dataset likely improved the accuracy.

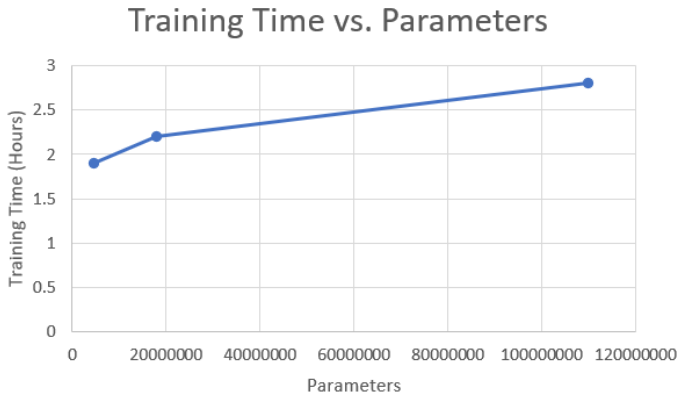


Figure 3.1: Training time of the models for 4 epochs vs. number of parameters

3.5 Conclusion

The group has used transfer learning from BERT models on Tensorflow datasets for natural language inference on the SNLI dataset. The group is trying to find

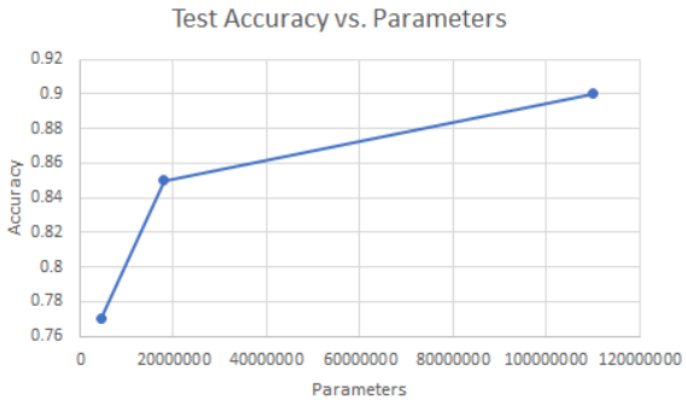


Figure 3.2: Accuracy of the models vs. number of parameters

the right combination of BERT model and additional layers to produce the most accurate network while achieving fast computation times.

AlBERT produced results that were better than expected. The group will begin looking at other improvements to the BERT model: RoBERTa, ERNIE, and DistilBERT. The group is also planning on doing a grid search of output layers to find the best in terms of accuracy and training times. Other output layers like convolutional layers and GRUs will be explored. The group will also consider training the model on a portion of the training data, instead of all 500,000 samples.

When importing models, there appears to be compatibility issues between Tensorflow, numpy, and other modules. The group is working on overcoming some of these issues. The projects usually work in Google Colab, but the group has problems when trying to run the code on HCC.

Overall, the group plans to provide a somewhat comprehensive list of available BERT models and implementations measured by accuracy and training time for the SNLI dataset.

Chapter 4

Milestone 4: Progress Report 2

4.1 Introduction

The goal of the group of to provide an accurate BERT model for NLI applications that can be trained quickly. This is a difficult task due to the size of BERT models and the size of the datasets necessary to train them. The importance of being able to train a model quickly or with limited hardware was highlighted during this milestone as HCC was under maintenance.

For this milestone, the group used tiny-BERT, which was the smallest model selected previously tested in Milestone 3, to test how changing the output layers effected the performance for the NLI data set. The group also tested how training the BERT model for all epochs or just the last half impacts training accuracy and time.

4.2 Experimental Setup

4.2.1 Output Layers

Since the BERT model cannot be easily modified, the most obvious way to impact performance is to change the output layers that come after the BERT model. Since reducing training time is a goal, the model use the smallest BERT model, one with 2 layers, 128 hidden units, and an attention head of 2. The models are trained on Google Colab GPU's. The preprocessing used is the same as decribed in 4.2. The output parameters are trained for 2 epochs while the BERT parameters are frozen. Then the BERT parameters are set to trainable and the model is trained for 2 more epochs. The performance metrics used are the validation accuracy and the training time.

Commonly bidirectional LSTMs are used following the BERT model. This experiment on the effect of output layers used bidirectional LSTMs with 16,

32, 64, and 128 units. The outputs of the LSTMs were passed through average 1D pooling and max 1D pooling whose outputs were then concatenated. The output of this layer was then dropped out at 30% to help with over fitting. The output of the dropout layer is sent to a dense layer with 3 neurons and softmax activation. The experiment is repeated with 16 bidirectional GRU units instead of LSTM units.

The effect of not using conventional RNN techniques as output layers was also observed. The output of the BERT model is passed through 1D average pooling and 1D max pooling and then concatenated. The concatenation layer is the dropped out at a rate of 30%. The drop out layer is passed through a densely connected layer of 16 neurons with relu activation. This layer is then connected to the output layer with three densely connected neurons and softmax activation.

4.2.2 Training BERT

Another way to change how a BERT model is trained is how long one trains the BERT parameters. The previous example trained only the output layers for 2 epochs, then the entire model for 2 epochs. What is the entire model were trained for the full 4 epochs or the BERT model not at all?

The experiment is repeated using Google Colab for the BERT model with 2 layers, 128 hidden units, and 2 attention heads. The structure with 64 bidirection LSTM units is used. When training four epochs, the BERT model is trained not at all, for only the last two epochs, and for all four epochs.

4.2.3 Comparing pre-trained BERT models with two transformer layers

We measured the performance of the base model illustrated in Figure 4.1 with four different variants of pre-trained BERT model on the SNLI test set. Thus far, only the variants in the top row of Table 4.1 were implemented inside the base model. The variants include: bert_en_uncased_L-2_H-128_A-2 with two transformer layers with 128 embedding sizes and two self-attention heads, bert_en_uncased_L-2_H-256_A-4 with two transformer layers with 256 embedding sizes and four self-attention heads, bert_en_uncased_L-2_H-512_A-8 with two transformer layers with 512 embedding sizes and eight self-attention heads, and bert_en_uncased_L-2_H-768_A-12 with two transformer layers with 768 embedding sizes and 12 self-attention heads.

4.3 Experimental Results

4.3.1 Output Layers

A table showing the training times, parameters, and final test accuracies for changing the output layers of the BERT model are shown in Table 4.2, with

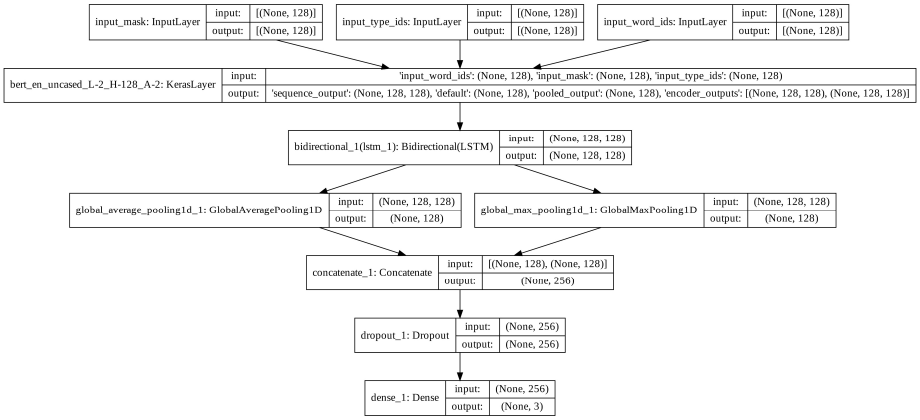


Figure 4.1: Architecture of the base model. This picture includes the bert_en_uncased_L-2_H-128_A-2 pre-trained model for illustration purpose.

Table 4.1: Pre-trained BERT models with their respective number of parameters (in millions) [27, 1]. For this reporting, we focused on the underlined models in the top row. H is the hidden embedding size, L is the number of transformer layers. The number of self-attention head is H divided by 64.

	H=128	H=256	H=512	H=768
L=2	<u>4.4</u>	<u>9.7</u>	<u>22.8</u>	<u>39.2</u>
L=4	4.8	11.3	29.1	53.4
L=6	5.2	12.8	35.4	67.5
L=8	5.6	14.4	41.7	81.7
L=10	6.0	16.0	48.0	95.9
L=12	6.4	17.6	54.3	110.1

None referring to only having the dense output layer with 3 neurons. The training time of each epoch is shown in Table 4.3, and the validation accuracies of each epoch are shown in Table 4.4.

Table 4.2: Training time, Accuracy, and Parameters of Different BERT Output Layers

Model	Training Time	Total Parameters	Non BERT Parameters	Test Accuracy
LSTM 16	139.7 min	4,404,676	18,755	0.7692
LSTM 32	145.2 min	4,427,524	41,603	0.7727
LSTM 64	148.2 min	4,485,508	99,587	0.7792
LSTM 128	273.5 min	4,650,628	264,707	0.7779
GRU 16	557.8 min	4,400,132	14,211	0.7633
Dense 16	120.8 min	4,390,084	4,163	0.7598
None	126.4 min	4,386,692	771	0.7633

Table 4.3: Epoch Training Time

Model	Epoch 1 (min)	Epoch 2 (min)	Epoch 3 (min)	Epoch 4 (min)
LSTM 16	33.3	33.1	36.6	36.9
LSTM 32	34.8	34.4	38.1	38.0
LSTM 64	35.8	34.6	38.9	39.0
LSTM 128	66.2	65.8	70.7	70.9
GRU 16	134.6	135.4	143.4	144.4
Dense 16	28.5	28.3	32.0	32.0
None	29.6	29.4	33.5	33.9

Table 4.4: Epoch Validation Accuracy

Model	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Test Accuracy
LSTM 16	.6855	.7083	.7582	.7791	.7692
LSTM 32	.7069	.716	.7719	.7832	.7727
LSTM 64	.7155	.7389	.7725	.787	.7792
LSTM 128	.723	.7452	.7766	.7898	.7779
GRU 16	.6946	.7112	.763	.782	.7713
Dense 16	.5024	.5076	.74	.7737	.7595
None	.4654	.4535	.7364	.7735	.7633

4.3.2 Training BERT

Table 4.5 shows the training times, validation accuracies at each epoch, and final test accuracy for training the BERT model for zero, two, and four epochs.

Table 4.5: Epoch Validation Accuracy

BERT Trained	Training Time	Epoch 1 Acc.	E 2 Acc.	E 3 Acc.	E 4 Acc.	Test Acc.
Zero Epochs	138.3 min	.7171	.7343	.7422	.7439	.7332
Two Epochs	148.2 min	.7155	.7389	.7725	.7870	.7792
Four Epochs	141.7 min	.7125	.7353	.7728	.7864	.7746

Table 4.6: Test set accuracy comparison of four models on the SNLI task. Each model includes a different pre-trained BERT variant with two transformer layers of different hidden embedding sizes as described by [27] and as illustrated in Figure 4.1. All the models were trained for two epochs and fine-tuned on two additional epochs.

BERT variant	Number of Parameters	Accuracy
bert_en_uncased_L-2_H-128_A-2	4,485,508	79.70%
bert_en_uncased_L-2_H-256_A-4	9,756,164	81.65%
bert_en_uncased_L-2_H-512_A-8	22,755,076	83.48%
bert_en_uncased_L-2_H-768_A-12	39,030,787	84.88%

4.3.3 Results of the comparison of pre-trained BERT variants on the SNLI task

The results are illustrated in Table 4.6. Moreover, the confusion matrices illustrated in Figures 4.6, 4.7, 4.8, and 4.9 show an improvement in the number of correctly classified labels in the diagonal cells as the model's complexity increases.

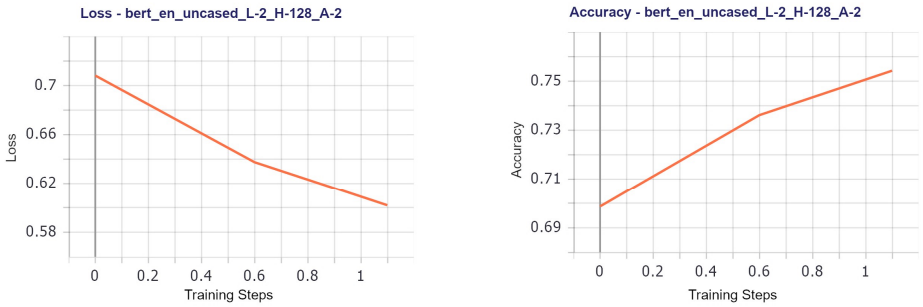


Figure 4.2: Loss and Accuracy plots on the training set with 550,152 records for the small BERT with two transformer layers with hidden embedding sizes of 128 and two self-attention heads.

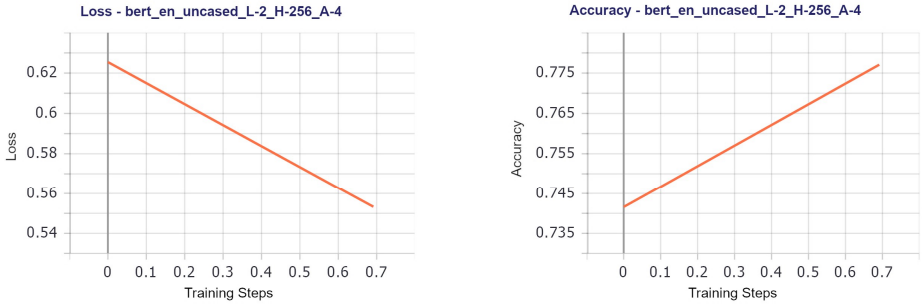


Figure 4.3: Loss and Accuracy plots on the training set with 550,152 records for the small BERT with two transformer layers with hidden embedding sizes of 256 and four self-attention heads.

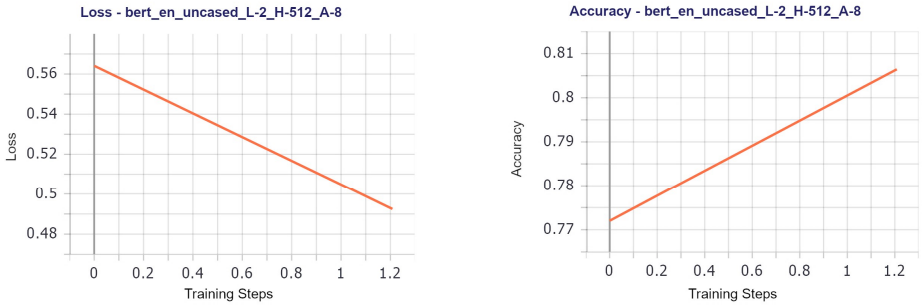


Figure 4.4: Loss and Accuracy plots on the training set with 550,152 records for the small BERT with two transformer layers with hidden embedding sizes of 512 and eight self-attention heads.

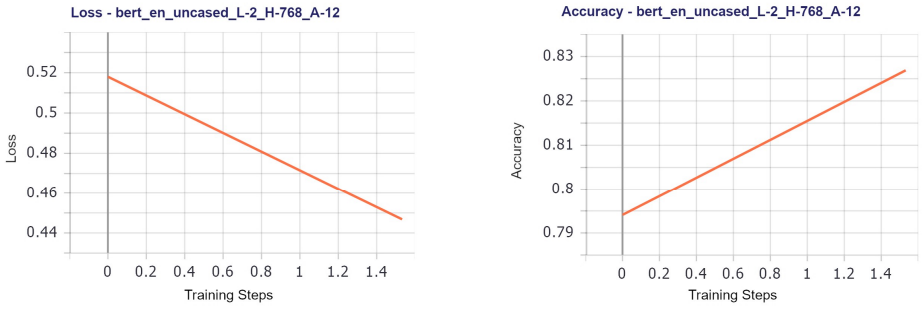


Figure 4.5: Loss and Accuracy plots on the training set with 550,152 records for the small BERT with two transformer layers with hidden embedding sizes of 768 and 12 self-attention heads.

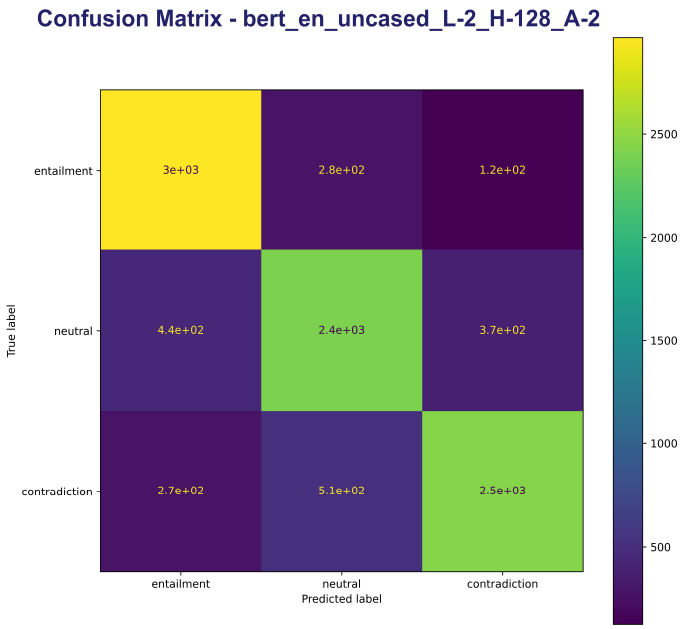


Figure 4.6: Confusion matrix on the test set with 10,000 records for the small BERT with two transformer layers with hidden embedding sizes of 128 and two self-attention heads.

4.4 BERT Embedding Visualization

Visualizing BERT's functionality would not only help in allowing explainability, but also provide guidance in designing outer layers of the model in a way that best utilizes BERT's output. In this section, we briefly describe various levels of visualizations that the group attempted in order to understand BERT layers and their output better.
The general setup used for this task is described below:

1. Import libraries for visualization such as matplotlib.pyplot, seaborn, pylab, mpl_toolkits. The library scipy is used for calculating distances between the embedding vectors. This library offers a variety of distance metrics like hamming, euclidean, jacobian etc. For the purpose of this experiment, the group opted for cosine distances as it was the most suitable metric for the task at hand. Furthermore, we have used T-SNE from sklearn to reduce the dimensionality of embedding vectors.

2. BERT model used is "bert-base-uncased". The maximum length of BERT input (which is the length of the hypothesis and premise together) is fixed to 32 because it is easier to visualize at this length. Anything above 32 makes the plots crowded and difficult to interpret. Furthermore, the

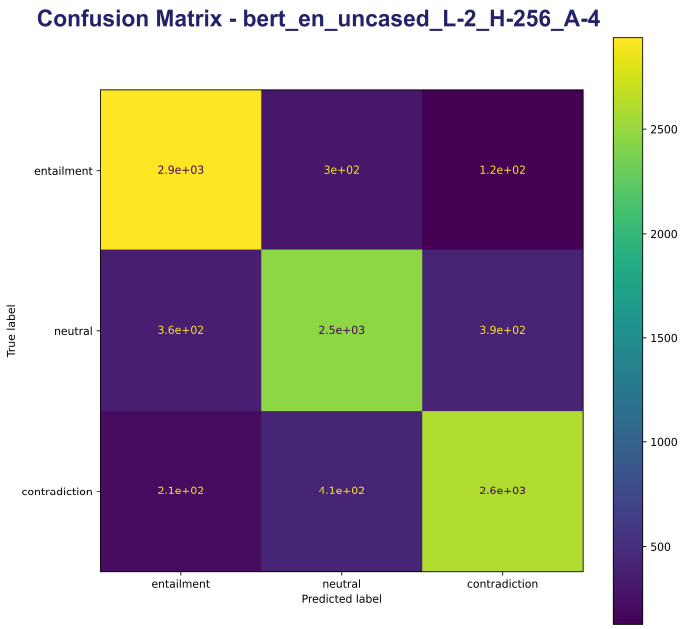


Figure 4.7: Confusion matrix on the test set with 10,000 records for the small BERT with two transformer layers with hidden embedding sizes of 256 and four self-attention heads.

BERT layers trainable feature was set to False and output_hidden_states set to True so that all the 12 hidden layers are available for visualization instead of just the last two.

3. The examples used in this section were picked from the SNLI dataset hosted by the library tensorflow_datasets. We ensured that the examples consisted of short sentences to avoid truncation. Furthermore, we also opted for sentences with less complexity to encourage clarity in the plots.

4. BERT model output is a dictionary of three items: last_hidden_state, pooler_output, and hidden_states. We have only used the last item for generating context heatmaps and neighborhood plots. Pooler output is used for visualizing sentence pairs belonging to different categories.

4.4.1 Visualizing Context Learning

BERT has an input layer (dimension $1 \times 32 \times 768$) followed by 12 hidden layers (dimension $12 \times 32 \times 768$) followed by a pooled output layer (dimension 1×768). To visualize how BERT learns context, we extract a layer, calculate the cosine distance between every token pair and present it as a heatmap. Such plots are generated for a sentence pair belonging to each of the three labels. In the

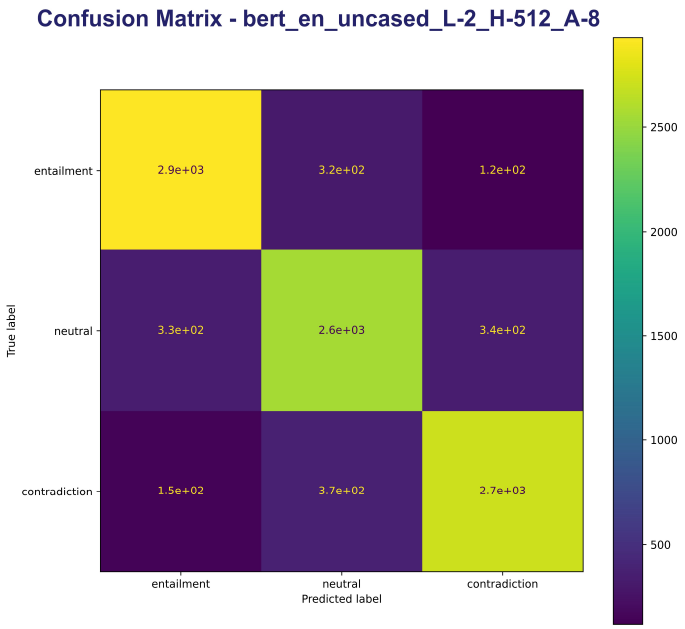


Figure 4.8: Confusion matrix on the test set with 10,000 records for the small BERT with two transformer layers with hidden embedding sizes of 512 and eight self-attention heads.

plots, higher distances are represented by lighter colors while lower distances are represented using darker colors. It can be easily noticed that the overall darkness of the plots increase as we move from Layer 1 (no context) to layer 12 (high context).

Contradiction

The example used for generating figures 4.10,4.11, 4.12, 4.13 is:

- Premise: a man standing at the front of the room is pointing up while several people who are seated look towards him.
- Hypothesis: a man is sitting outside.

The heat maps for different layers for above sentence pair show how context is built locally within a sentence as well as globally across the sentences. Layer 1 has darker areas mostly when both tokens are the same. However, the distance between consecutive words is high since this is a shallow layer. For example, the token 'him' in the premise refers to the token 'man' in the same sentence. In layer 1, the distance between him and man is high (0.74 w.r.t 'man' in hypothesis and 0.73 w.r.t the one in premise). However, in layer 10, the distance is reduced

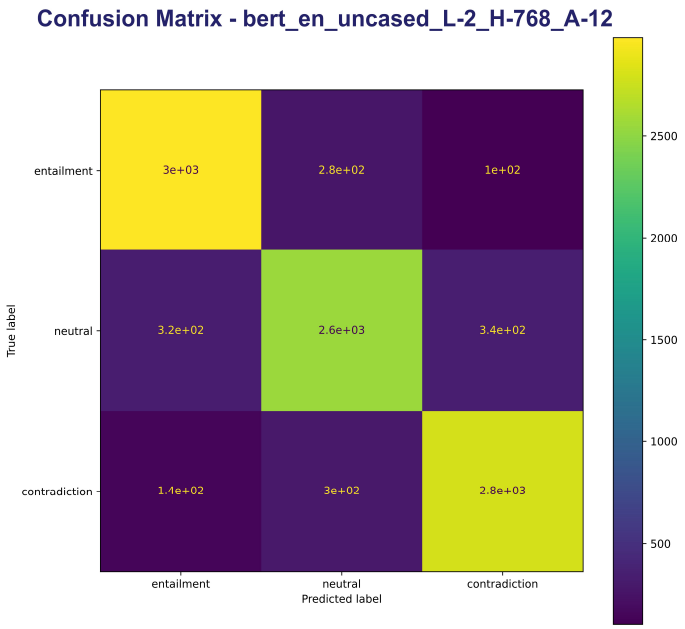


Figure 4.9: Confusion matrix on the test set with 10,000 records for the small BERT with two transformer layers with hidden embedding sizes of 768 and 12 self-attention heads.

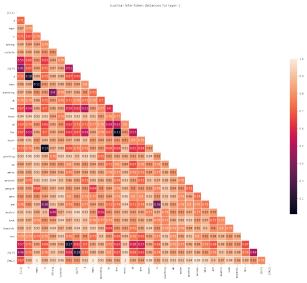


Figure 4.10: Visualizing inter-token cosine distances for Layer 1 (Label Contradiction)

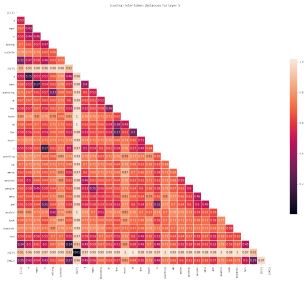


Figure 4.11: Visualizing inter-token cosine distances for Layer 5 (Label Contradiction)

to 0.5 and 0.51 respectively while in layer 12, it is reduced to 0.46 and 0.44 respectively.

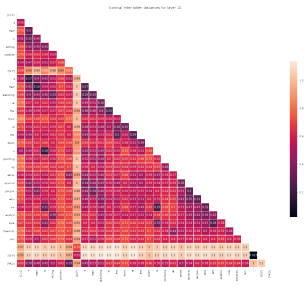


Figure 4.12: Visualizing inter-token cosine distances for Layer 10 (Label Contradiction)

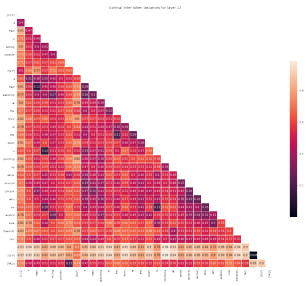


Figure 4.13: Visualizing inter-token cosine distances for Layer 12 (Label Contradiction)

Entailment

The example used for generating figures 4.14,4.15, 4.16, 4.17 is:

- Premise: three people are resting.
- Hypothesis: a mother and her two children sit down to rest.

In layer 1 fig. 4.14, the distance between the word 'people' in the premise and the words 'mother' and 'children' is 0.83 and 0.68 respectively. In layer 10 fig. 4.16, it is 0.58 and 0.55 respectively while in layer 12 fig. 4.17, it is 0.57 and 0.54 respectively.

Neutral

The example used for generating figures 4.14,4.15, 4.16, 4.17 is:

- Premise: a man snowboards.
- Hypothesis: a boy is snowboarding down a large hill.

In layer 1, the distance between 'man' and 'boy' is 0.56. In layer 10, it is 0.42, while in layer 12, it is 0.36.

Overall, the heatmaps show that layers learn local and global context in terms of usage of words in the English language. However, we did not find any evidence of the layers being able to distinguish if the same word occurring in two sentences refers to different entities just based on intra-layer cosine distances.

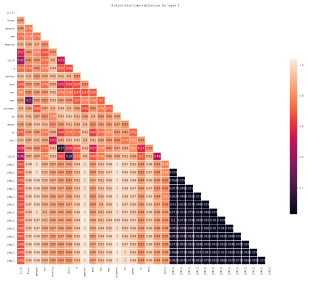


Figure 4.14: Visualizing inter-token cosine distances for Layer 1 (Label Entailment)

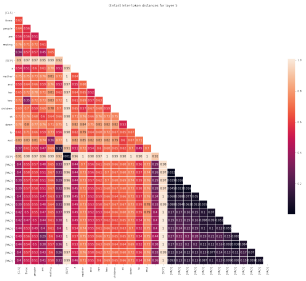


Figure 4.15: Visualizing inter-token cosine distances for Layer 5 (Label Entailment)

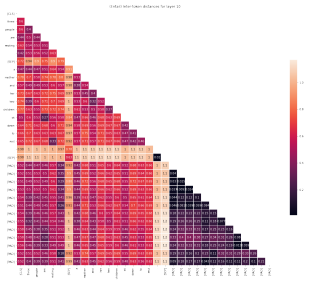


Figure 4.16: Visualizing inter-token cosine distances for Layer 10 (Label Entailment)

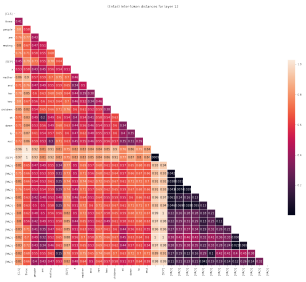


Figure 4.17: Visualizing inter-token cosine distances for Layer 12 (Label Entailment)

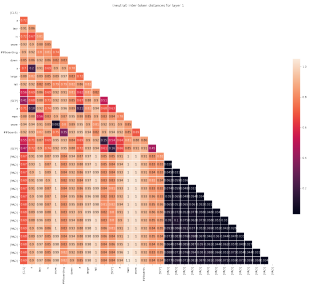


Figure 4.18: Visualizing inter-token cosine distances for Layer 1 (Label Neutral)

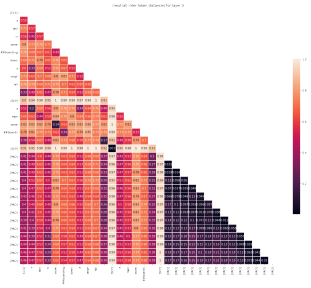


Figure 4.19: Visualizing inter-token cosine distances for Layer 5 (Label Neutral)

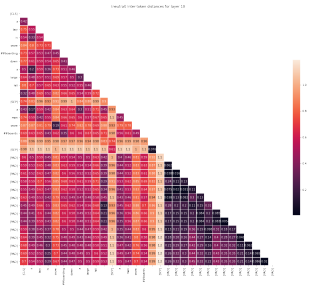


Figure 4.20: Visualizing inter-token cosine distances for Layer 10 (Label Neutral)

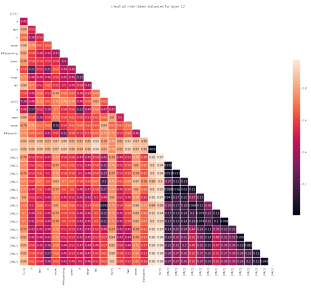


Figure 4.21: Visualizing inter-token cosine distances for Layer 12 (Label Neutral)

4.4.2 Visualizing Token Neighborhoods

Another way of visualizing BERT layers is by manipulating hidden layers using concatenation, summation etc., and representing token vectors as a scatter plot to get a quick glance at the neighborhood of a particular token. Since each token is high dimensional, T-SNE is used to map it onto a two-dimensional space. The results vary with different combination of layers and manipulation techniques. The set-up for plots is summarized as follows:

1. Extract different combination of layers and reduce the value for each token to a single number using summation. Here, we used combination of layers 6 through 9, layers 11 through 12, layers 10 through 12, and layers 9 through 12. It is observed that combining the last 3 layers provides meaningful plots compared to earlier layers.

2. Once the layers are combined, inter-token cosine distances are calculated.

3. For every token excluding the key tokens added by the tokenizer such as ['CLS'] and ['SEP'], the nearest n tokens are retrieved. This is controlled by a 'proximity' factor. The plots are generated using a proximity factor of 1. Generating plots for higher proximity factors will require further research and will be taken up by the group in the next phase.

4. Tokens are color coded to identify them as belonging to the premise or hypothesis. This is helpful when the same word is present in both sentences.

5. Figures 4.22, 4.23, 4.24, 4.25 are generated for the label contradiction.

6. Figures 4.26, 4.27, 4.28, 4.29 are generated for the label entailment.

7. Figures 4.30, 4.31, 4.32, 4.33 are generated for the label neutral.

The observations noted fro various labels are:

- Overall, the combination of layers 10 through 12 depicted word relationships better compared to other layer combinations.

- The simple summing up of deep layers does not offer any new insight into whether BERT incorporates the word sense. That is, when a word occurs in different sentences but each occurrence refers to distinct entities, BERT's vector representation does not capture this aspect.

- There could be two reasons for this. First, BERT layer weights were frozen at the time of this experiment. Second, BERT is trained on next sentence prediction and masked language modelling task. Thus, it is good at identifying relationship between a word and a group of words in a bidirectional context window. However, in order to perform well on inference task, it needs trainable intermediate and output layers that consume BERT's encoded output and learn to classify the relationship between sentence pairs.

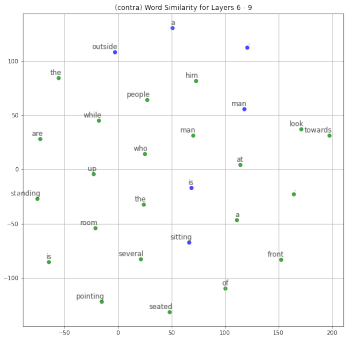


Figure 4.22: Visualizing layers 6-9 for word similarity (Label Contradiction)

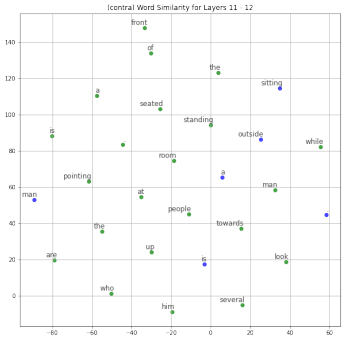


Figure 4.23: Visualizing layers 11-12 for word similarity (Label Contradiction)

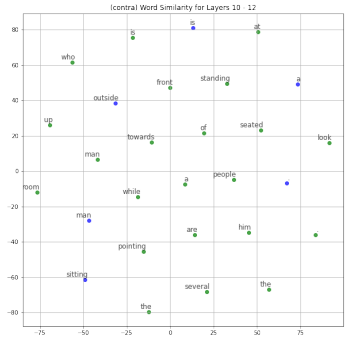


Figure 4.24: Visualizing layers 10-12 for word similarity (Label Contradiction)

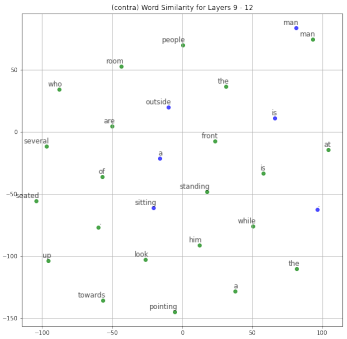


Figure 4.25: Visualizing layers 9-12 for word similarity (Label Contradiction)

- On the other hand, it would be interesting to make BERT layers trainable and then visualize token representations to see if word sense gets incorporated into token representation.

4.4.3 Visualizing BERT Pooled Output for Sentence Pairs

The plots 4.34, 4.35 are our initial attempts at visualizing BERT model's Pooler Output to

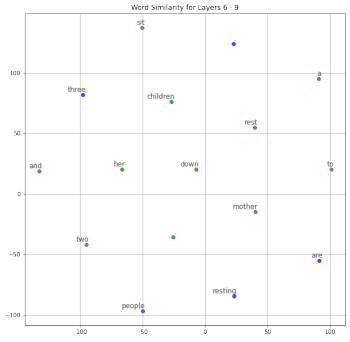


Figure 4.26: Visualizing layers 6-9 for word similarity (Label entailment)

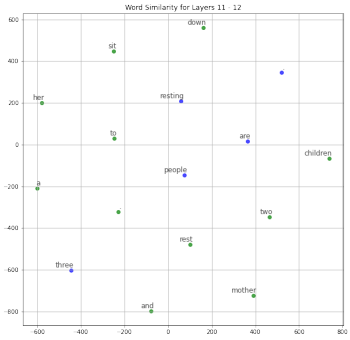


Figure 4.27: Visualizing layers 11-12 for word similarity (Label entailment)

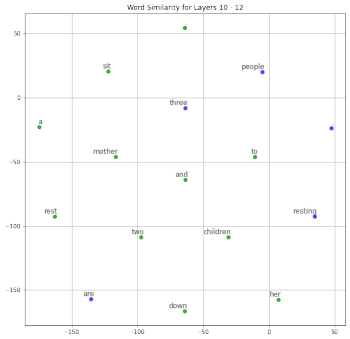


Figure 4.28: Visualizing layers 10-12 for word similarity (Label entailment)

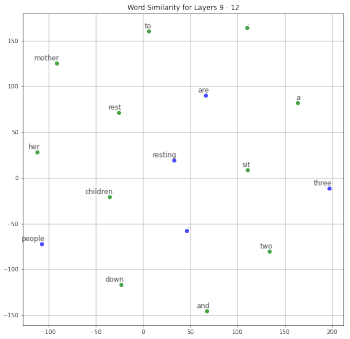


Figure 4.29: Visualizing layers 9-12 for word similarity (Label entailment)

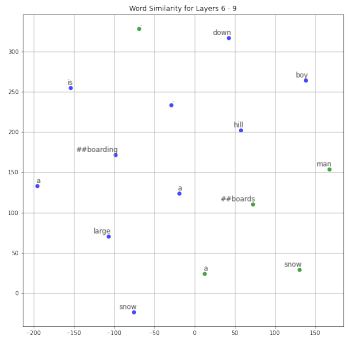


Figure 4.30: Visualizing layers 6-9 for word similarity (Label Neutral)

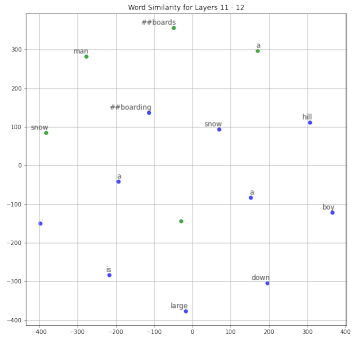


Figure 4.31: Visualizing layers 11-12 for word similarity (Label Neutral)

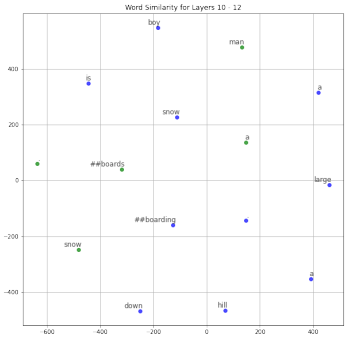


Figure 4.32: Visualizing layers 10-12 for word similarity (Label Neutral)

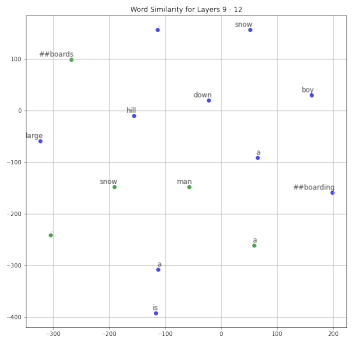


Figure 4.33: Visualizing layers 9-12 for word similarity (Label Neutral)

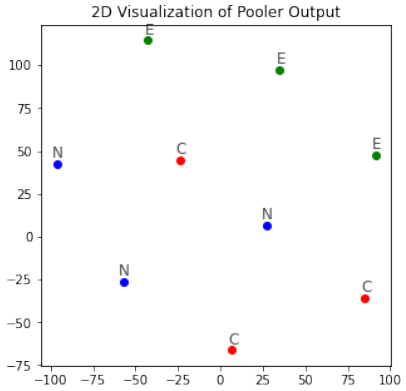


Figure 4.34: Visualizing BERT's Pooler Output in 2D

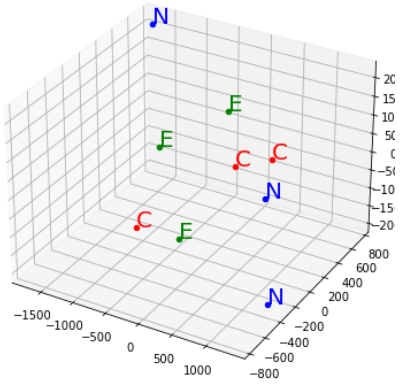


Figure 4.35: Visualizing BERT's Pooler Output in 3D

- explore the presence of discernible patterns in the way it interprets entailment, contradiction and neutrality relationship.
- exploit this layer to improve model predictions in the future experiments

Since this was an initial attempt of using the Pooler Output, the plots are generated on a tiny subset of nine sentences; with three sentences belonging to each category. Since the dataset is really small, it is not appropriate to derive any conclusions from the graphs. The data points seem to have a slightly better separation in 2D space compared to 3D. We will attempt at creating plots using a larger balanced subset of sentences picked randomly. The 2D and 3D plots were generated by projecting the 768 dimensional vector onto 2 and 3 dimensions respectively using T-SNE with a perplexity factor of 35.

4.5 Discussion

This milestone was mainly focused how the manipulation of output layers and testing variants of BERTs for the SNLI task affected training time and accuracy. When changes were made to the output layers, there was a linear relationship between training time and the increased non-BERT parameters when LSTM was increased from 16 to 128. with LSTM 16, the training time was around 140 minutes, where LSTM 128 was close to 274 minutes. Test accuracy stayed relatively similar at 77 percent, where there was a slight increase at the beginning of LSTM 16 to LSTM 32, suggesting that increasing LSTM layers does not significantly improve model performance. The longest model to train was GRU 16, which took almost 558 minutes. This was somewhat interesting as the GRU 16 has The idea behind comparing GRU vs LSTM was how they both utilize different method of gating information to prevent vanishing gradient. GRU has

also been proven to be on par with LSTM in respects to performance, but has a less complex structure compared to LSTM. Although GRU is more simple in its framework, LSTMs remember longer sequences than GRUs and outperform them in tasks requiring modeling long-distance relations. More investigation is needed to understand why GRU performed slower than the LSTM for this project.

It is assumed that training the entire BERT model takes more time, but the results in Table 4.5 do not support this hypothesis. Training the BERT model for no epochs takes the least amount of time, followed by training the BERT for all the epochs then for only two epochs. This leads one to believe that there is enough discrepancy in the Google Colab system to account for the 4% decrease in training time when BERT is trained for 2 epochs vs. all 4. Regarding accuracy of training the BERT model, as long as BERT is trained it appears to yield good results. Training the BERT model for no epochs produces the worst accuracy of any tested model. This leads one to beleive that the output layers can be improved to a point, but training of the BERT model to a specific dataset produces the best results in terms of accuracy.

The self-attention heads in a language model like BERT helps direct more weight to learn specific parts of a sentence like verbs or nouns, and ultimately form connections between words, enabling BERT to learn a variety of linguistic relationships. BERT has the ability to learn multiple attention mechanisms, which is why they are called heads, and operate in parallel to one another. This ability allows BERT to capture a broader range of relationships between words than would be possible with a single attention mechanism. It is important to note that each attention head does not share parameters, which allows for each to learn its own pattern. When comparing BERT variants, there was again a relationship between an increase in model parameters, which relates to the number of self-attention heads and accuracy. Overall, the bert_en_uncased_L-2-H-768_A-12 produced the highest accuracy of almost 85 percent, but had the greatest number of parameters of about 39 million.

4.6 Conclusion

The unique opportunity of HCC being down shown the group the practical sense of the project proposal. Large networks like the one used in the project need substantial hardware to perform quickly, or even at all. The user friendly nature of Google Colab and even the availability of python itself on personal machines make using medium size models (like BERT) and large data sets (like SNLI) usable for almost everyone who would wish use them. Small businesses and individuals can take advantage of resources, hopefully without sacrificing accuracy for training speed.

Google Colab is fine for testing the smallest models for output layer benchmarks, but more powerful hardware is needed to reasonably test some of the larger models. The group intends to use the information found from view the impact of output layers, how many epochs to train the BERT, and the visual-

Table 4.7: Contributions by team member for Milestone 4.

Team Member	**Contribution**
Mason Lien	Worked on report
Matthew Penne	Output layers, Changing BERT training epochs, worked on report
Mrinal Rawool	Visualization, worked on report
Yves Shamavu	Pre-trained BERT variants, worked on report

ization of the embedding to apply to the larger models shown in Table 4.1. The goal remains to produce an accurate but quickly trainable BERT model for NLI projects. The methodology in balancing the accuracy and training speed can hopefully be applied to other projects even outside textual projects.

Chapter 5

Milestone 5: Final Report

5.1 Introduction

The goal of this project is to find the best available BERT model, additional layers, and hyper parameters to achieve the high accuracy and low training time for NLI applications. Blending speed and accuracy is a difficult task and it is difficult to improve one without the compromising with the other.
The group tested different drop out layers, different amounts of training data, and compared BERT models with different hidden units and layers. The group also attempted a custom BERT model from scratch. After all the comparisons were made for hyper parameters, amount of training data, and different BERT models, the group tested different combinations of models and training data to find the best model in terms of accuracy and training time. The best model had 8 layers, 512 hidden units, 8 attention heads, and was trained with 100k examples.

5.2 Progress at Each Milestone

Here is a short summary of what our team achieved at every milestone:

1. Milestone 1: The team brought several topics related to deep learning applied to image analysis and language analysis. The team decided to go with the topic natural language inference since it was a new area of research to explore. The team also looked at various projects that have been already done in this area for ideas as well as explore datasets available at our disposal.

2. Milestone 2: The team zeroed on the Stanford Natural language Inference (SNLI) dataset for this project. We also tried to adapt working code examples that used BiLSTMs and transformer based models to test the feasibility of this project.

3. Milestone 3: Initially, team experimented with pre-trained BERT offered by Huggingface. However, after facing issues with providing input to the model, we moved on to the version offered by Tensorflow Hub. We experimented with various flavours of BERT such as AlBERT, RoBERTa and our experiments were framed around observing how these models performed when run for the same number of epochs on the entire dataset. We also noted the time taken by these models per epoch.

4. Milestone 4: In this milestone, we attempted different ways of visualizing the BERT in an attempt to understand how BERT learns contextual information present in a sentence. The group experimented on changing the ouput layers after the BERT model affected accuracy and training time. The group also experimented with training the entire BERT model, or just the additional layers. Lastly, the group viewed the impact of using BERT models with different numbers of hidden units.

5. Milestone 5: Based on the information gathered on the computational requirements in Milestone 3, we shortlisted BERT model configurations that could be used to conduct a comparative study about the performance of BERT by varying parameters like number of hidden layers, number of attention heads, amount of training and validation data, number of epochs etc. The motivation for this exercise was to identify the optimum combination of resources that leads to a decent level of accuracy.

5.3 Experimental Setup

To compare different hyperparameters of how to improve the BERT model, the smallest BERT model was used. This BERT model had 2 layers, 128 hidden units, and 2 attention heads. This was the same model that was used in Milestone 4 comparisons. The best output layer found from Milestone 4 had 64 bidirectional LSTM units, so that output layer was copied going forward. Also, Milestone 4 shown that training only the output layers for half the training time and then training the whole model including the BERT for the final half yielded good results. This method of not training the BERT for the first 2 epochs was repeated for the following examples.

5.3.1 Dropout Rate

In Milestone 4, the dropout rate from the concatenation layer to the dense output layer was fixed at 0.3. The group now tested dropout rates of 0, 0.1, 0.2, 0.3, and 0.4 to see how they affected test accuracy.

5.3.2 Training Data

The group also tested how much training data was needed to produce accurate results. The SNLI dataset used has 550k examples, which were all used for

training in previous examples. The group tested how test accuracy and training time were effected when the model was trained with 250k examples, 100k examples, and 25k examples. The validation set and test set were both set at 10k examples used for all other experiments.

5.3.3 BERT Layers and Hidden Units

Previously the group only explored using only the smallest BERT model with 2 layers and 128 hidden units to see how other attributes affected accuracy and training time. With the other attributes tested, how would changing the BERT model itself affect performance? The available models of BERT are shown in Figure 5.1, where the attention heads are defined as the number of hidden units divided by 64. The group tested each model in the first column in Figure 5.1 to compare have changing the layers of the BERT model affected performance. Then the group tested the models in the first row to see how changing the number of hidden units affected performance. These experiments used a dropout rate of 0.3, and used all 550k training examples.

	H=128	H=256	H=512	H=768
L=2	2/128	2/256	2/512	2/768
L=4	4/128	4/256	4/512	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768

Figure 5.1: BERT models developed in [27]

5.3.4 Specific Models

Once the BERT models changing both the layers and hidden units were compared, several models were expected to produce acceptable test accuracy and training time. These models, 4 layers-256 units-4 attention heads, 8 layers-256 hidden units-4 attention heads, 8 layers-512 hidden units-8 attention heads and were tested with 550k, 250k, and 100k training examples to find the best blend of test accuracy and training time.

5.3.5 Custom BERT Model

In the custom model, we developed the BERT layer from scratch and did the tokenization using the Tensorflow TextVectorization layer from the experimental

preprocessing package. In contrast, when using the pre-trained BERT models from TFHub, they were all shipping with a specific processing module.

In the custom implementation, the tokenization step was followed by a custom masking step to ensure the model did not treat padding as part of the input. Again, to contrast with the pre-trained BERT from TFHub, the specific preprocessing module returned three tensors, one with the tokenized inputs, one with masked padding, and one with the indexes of the input segments that gave rise to the input tokens at their respective position [18]. However, for the custom implementation, we did not produce the input_type_ids. Instead, we implemented a line of code to set a value of 0 at each actual input token position and a value of 1 for all padding tokens. This step returned a custom input mask.

The input pipeline described above was followed by three crucial steps for any BERT model: positional encoding, scaled dot-product attention, and multi-head attention [12]. Since the BERT architecture does not contain any convolutional layers nor recurrent cells, positional encoding allows the model to gain some information about the relative position of words in a sentence. Positional encoding is added to the embedding layer output such that with the latter, words are closer to each other based on their similarity, and with the former, words are brought closer based on their positions in the sentence.

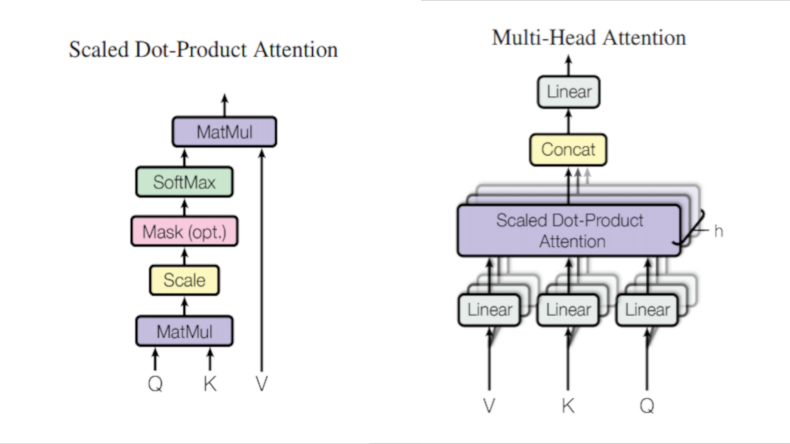


Figure 5.2: Scaled dot-product attention on the left, and multi-head attention on the right.

Figure 5.2 depicts the scaled dot-product attention on the left. The Q and K matrices represent the query and key items. In our case, the input tokens returned from the input pipeline served as both Q and K.
Thus, the scaled dot-product attention computes a similarity measure between each item in Q and the corresponding key in K. It then uses a softmax function to convert these similarity scores to weights that add up to 1. And finally, it

computes a weighted sum of the corresponding items in the value matrix V. For our purpose, V was also equal to the input tokens. The input mask is applied to scaled tensors before computing the softmax function.

The multi-head attention, on the right of Figure 5.2, receives the matrices Q, K, and V as inputs in the form of dense linear layers, i.e., without any specified activation. These linear layers are then split into multiple heads, and the scaled dot-product (as described in the previous paragraph) is applied to each head. This process allows the model to attend to information at different positions in different representational spaces [12]. In the end, the attention outputs for the different heads are concatenated and fed into another dense linear layer.

BERT itself is made up of one or several encoder layers. The encoder layer comprises two or more attention heads to form multi-head attention followed by feed forward or residual networks.
In our implementation, we used eight encoder layers with four attention heads.

Describe the setup of your experiments in sufficient detail for the reader to reproduce them. Include data sources, preprocessing used, architectures/other approaches used, hyperparameter values used, performance measures used, other relevant items (e.g., cross-validation).

5.4 Experimental Results

5.4.1 Dropout Rate

The accuracy and loss of the changing the dropout rate between the concatenation and the output layer are shown in Figure 5.3. Increasing the dropout rate past 30% appears to degrade performance. Regardless, all dropout rates produced similar performance. The group chose to keep 30% as the dropout rate to help over fitting the model to training data and because it produced the highest accuracy, even if slightly.

Dropout	Accuracy	Loss
0%	0.7751	0.5559
10%	0.7755	0.5593
20%	0.7769	0.5571
30%	0.7792	0.5536
40%	0.7725	0.5643

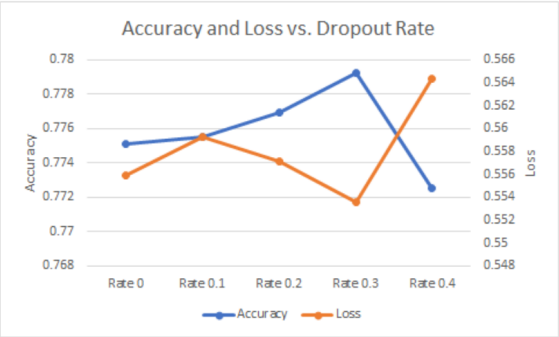


Figure 5.3: Table and graph showing the impact of changing the dropout rate

5.4.2 Training Data

When changing the amount of training data used, Fig 5.4 shows how the scaling of 25k samples to 550k samples directly affects average training time per epoch. As expected, there is an increase in accuracy with the increased number of samples in the training dataset. The jump from 25k to 100k shows the greatest improvement of accuracy, but computational cost of training time is over 3.25 times longer to train per epoch. Adding additional samples after 100k increased accuracy by 6 percent from 0.72-0.79.

Training Data	Average Time per Epoch (minutes)	Accuracy	Loss
550,000	31.0	0.79	0.51
250,000	16.60	0.75	0.60
100,000	12.55	0.72	0.67
25,000	3.85	0.64	0.83

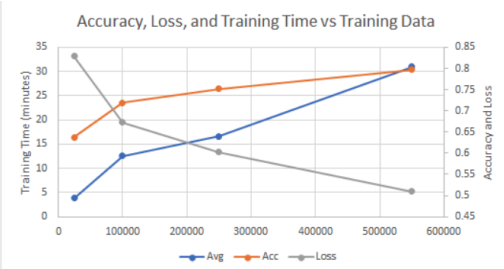


Figure 5.4: Table and graph showing the impact of changing the total amount of training data

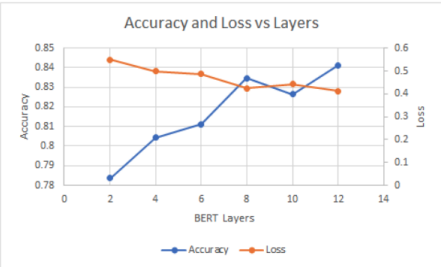
5.4.3 BERT Layers and Hidden Units

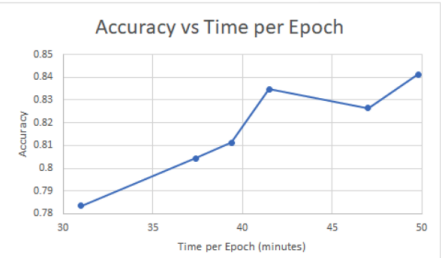
While keeping the number of hidden units fixed at 128, the changing in BERT layers from 2-12 was tested to understand the affect on accuracy, loss, and training time per epoch. Overall, there was a general increase in training time as the number of layers was added to the BERT model. There was a decrease in accuracy between layers 8-10, which was the only time in the study that showed a drop in accuracy when adding layers to the BERT model. The maximum amount of training time was 49.8 minutes which was the BERT model with 12 layers, and produced the highest accuracy of 0.84%.

Changing the number of hidden units is shown in Milestone 4 and repeated here for better comparison. Keeping the layers fixed at 2, the hidden units were increased from 128 to 768 by a multiple of 2. Compared to the base model with 128 hidden units, the final BERT model with 768 hidden units had an increase in training time by 200%, while the accuracy only increased by 6% (see Fig 5.6).

5.4.4 Specific Models

As shown in Fig 5.5, there is a large increase of accuracy at 4 layers and at 8 layers. There doesn't appear to be a great increase in training time going from 4 to 8 layers. Also, there appears to be big increases in accuracy going to 256 or 512 units compared to 128. This is why to models chosen to be tested were 4





Layers	Hidden Units	Time per Epoch (minutes)	Time Percent dif. From Base	Accuracy	Loss
2	128	31	N/A	0.797	0.5.96
4	128	37.4	21%	0.8044	0.4987
6	128	39.4	27%	0.8112	0.4865
8	128	41.5	34%	0.8347	0.4245
10	128	47	52%	0.8265	0.4432
12	128	49.8	61%	0.8412	0.4125

Figure 5.5: Table and graphs showing the impact of changing the BERT Layers

layers-256 units-4 attention heads, 8 layers-256 hidden units-4 attention heads, 8 layers-512 hidden units-8 attention heads.

The results of testing these models with 550k for the 1st two models, 250k, and 100k training examples is shown in Figure 5.7. The best model appears to be the largest used, 8 layers-512 hidden units-8 attention heads. Using 100k training samples produces reasonable accuracy at a relatively low training time. Stepping up to 250k training samples increases the accuracy by 2.3%, but takes 139% longer to train.

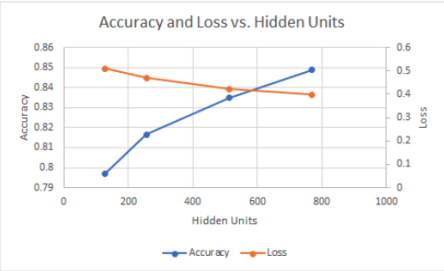
5.4.5 Results of Custom BERT Model

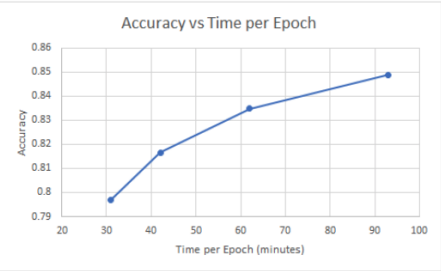
The training for the custom BERT model has not been adequately conducted due to several issues with the development environment and time-constraint. However, from the initial training that lasted five epochs, the accuracy was still low, between 30 to 40%. It is possible that the custom BERT layer has to be trained on independent English corpora before using it on a downstream task, such as the SNLI. Indeed, this is how the pre-trained BERT models from TFHub are trained and then provided for downstream NLP tasks.

5.5 Discussion

The purpose of running all the experiments was to methodically predict and find what would model would produce the best blend of accuracy and training. Exploring output layers and how long to train the BERT model were discussed in milestone 4. Graphing the accuracy and training time when changing the BERT hidden units and layers provided a good basis on where to predict effective models would be. This led to the models discussed in Section 5.4.4 being chosen.

The process of finding the right output layers shown that a moderate number





Layers	Hidden Units	Time per Epoch (minutes)	Time Percent dif. From Base	Accuracy	Loss
2	128	31	N/A	0.797	0.5096
2	256	42	35%	0.8165	0.471
2	512	62	100%	0.8348	0.4224
2	768	93	200%	0.8488	0.3987

Figure 5.6: Table and graphs showing the impact of changing the Hidden Units

Model	Layers	Units	Training data	Training Time per Epoch (minutes)	Accuracy	Loss
1	4	256	550000	46.7	0.8369	0.4174
1	4	256	250000	19.80417	0.8048	0.5039
1	4	256	100000	8.345833	0.7778	0.5538
2	8	256	550000	52.1	0.8455	0.4204
2	8	256	250000	24.12917	0.8203	0.461
2	8	256	100000	10.11667	0.7914	0.5203
3	8	512	250000	39.65	0.854	0.3965
3	8	512	100000	16.54167	0.831	0.4487

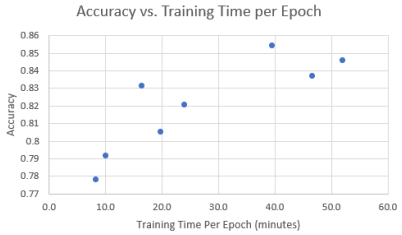


Figure 5.7: Table and graph showing results of specific models and training data

of Bidirectional LSTM units produced the best result. It was found that the BERT model needs to be trained some to fit the data, but fine tuning the output layers first while keeping the BERT weights frozen could save time and help with over fitting. Changing the dropout rate did not impact performance, but was kept relatively high to help with possible over fitting. Changing the BERT model layers and hidden unit shown how each parameter effect accuracy and training time. Increasing the number of hidden units greatly increases the training time compared to the number of layers, but doesn't greatly increase accuracy.

The largest model used appeared to be the best. The model produced higher accuracy when using little data, which is the best of both worlds. If not a lot of data is present, larger models could be used to produce reasonable accuracy. If a lot of data is available, as in our case, less data could be used to still produce an accurate result but with significantly faster training times.

5.6 Model Recommendation

The best output layers found were 64 Bidirectional LSTM units, that were then average and maximum pooled, concatenated, passed through a dropout layer of 30% and connected to a densely connected layer with 3 neurons and softmax activation. The model was found to produce best results when only the output layers were trained for 2 epochs, then the entire BERT model for 2 epochs. The best BERT model found had 8 layers, 512 hidden units, and 8 attention heads. Training this model with 100k training examples produced reasonable accuracy at good training times. A comparison between the recommended model and the original Tiny model are shown in Fig 5.8. The recommended model offers a 3.4% increase in accuracy and a 46.7% decrease in training time.

Model	Layers	Units	Training data	Training Time per Epoch (minutes)	Accuracy	Loss
Original	2	128	550000	31	0.797	0.5096
Recommended	8	512	100000	16.5	0.831	0.4487

Figure 5.8: Comparison between original Tiny BERT and Recommended BERT

5.7 Conclusion

The efforts made throughout this study has shown the exploration and testing a multitude of differing BERT models to find the one with the highest performance while minimizing computation cost of training. The reporting of the best model lead to the investigation of additional layers, and hyperparameters to achieve the accuracy and low training time for NLI applications. The team faced challenges of managing training runs on many models, which demanded a high level of time management due to long training times, limited number of models to train at once, and intermittent HCC outages which made the team agile in rethinking how to complete our tasks through Google Colab and local jobs. Overall, the team was successful in completing the objectives described throughout the previous milestones to ultimately recommend the best performing BERT model as seen in Fig 5.8.
We hope that this report serves as a helpful resource to anyone exploring the area of natural language understanding or inference. This report provides the reader with some useful resources for learning about the inference task and implementing related projects using various BERT configurations.

Team Member	**Contribution**
Ketemwabi Yves Shamavu	Custom BERT, BERT Layers and Hidden Units, Final Code, and reporting
Mrinal Rawool	Running different versions of BERT models and reporting
Mason Lien	Running different versions of BERT models and reporting
Matthew Penne	Dropout Layers, training data comparisons, versions of BERT models, and reporting

Table 5.1: Contributions by team member for Milestone 5.

Bibliography

[1] URL: https://tfhub.dev/.

[2] Classify text with bert nbsp;: nbsp; tensorflow core. URL: https://www.tensorflow.org/tutorials/text/classify_text_with_bert.

[3] GloVe: Global Vectors for Word Representation. URL: https://nlp.stanford.edu/projects/glove/.

[4] Natural language inference on snli. https://paperswithcode.com/sota/natural-language-inference-on-snli. Accessed: 2021-02-10.

[5] Nlp-progress. http://nlpprogress.com/english/natural_language_inference.html. Accessed: 2021-03-4.

[6] Snli. https://nlp.stanford.edu/projects/snli/. Accessed: 2021-02-10.

[7] The Stanford Natural Language Processing Group. URL: https://nlp.stanford.edu/projects/snli/.

[8] Martín Abadi and et al. Ashish Agarwal. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[9] Douglas S. Alt. Managing risks of soft red winter wheat production: Evaluation of spring freeze damage and harvest date to improve grain quality. Accessed: 2021-02-11.

[10] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[11] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[12] Tensorflow Core. Transformer model for language understanding, 2021. accessed: 2021-04-16. URL: `https://www.tensorflow.org/tutorials/text/transformer`.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. `arXiv:1810.04805`.

[15] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, CA, 2017.

[16] Abdul Khaliq Umair Ashraf Shakeel A. Anjum Shengnan men Longchang Wang Hafiz A. Hussain, Saddam Hussain. Chilling and drought stresses in crop plants. *Front. Plant Sci. 9:393. doi: 10.3389/fpls.2018.00393*, 2018.

[17] Rani Horev. Bert explained: State of the art language model for nlp, Nov 2018. URL: `https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270`.

[18] Tensorflow Hub. Common savedmodel apis for text tasks, 2021. accessed: 2021-04-21. URL: `https://www.tensorflow.org/hub/common_saved_model_apis/text#transformer-encoders`.

[19] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020. `arXiv:1909.11942`.

[21] R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*, 2019.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[23] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[24] Alexey Romanov and Chaitanya Shivade. Lessons from natural language inference in the clinical domain. *arXiv preprint arXiv:1808.06752*, 2018.

[25] Shane Storks, Qiaozi Gao, and Joyce Y Chai. Recent advances in natural language inference: A survey of benchmarks, resources, and approaches. *arXiv preprint arXiv:1904.01172*, 2019.

[26] Aarne Talman, Anssi Yli-Jyrä, and Jörg Tiedemann. Sentence embeddings in nli with iterative refinement encoders. *Natural Language Engineering*, 25(4):467–482, Jul 2019. URL: `http://dx.doi.org/10.1017/S1351324919000202`, `doi:10.1017/s1351324919000202`.

[27] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019. `arXiv:1908.08962`.

[28] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019. `arXiv:1804.07461`.

[29] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017.

[30] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.

[31] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. `https://d2l.ai`.