A Deep Dive into BERT and Contrastive Learning

Hargen Zheng
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92092
yoz018@ucsd.edu

Nathaniel del Rosario
Halıcıoğlu Data Science Institute
University of California, San Diego
San Diego, CA 92092
nadelrosario@ucsd.edu

Chuong Nguyen
Computer Science and Engineering Dept.
University of California, San Diego
San Diego, CA 92092
chn021o@ucsd.edu

Ziyue Liu
Electrical and Computer Engineering Dept.
University of California, San Diego
San Diego, CA 92092
zil085@ucsd.edu

Adam Tran
Mathematics Department
University of California, San Diego
San Diego, CA 92092
ant010@ucsd.edu

Abstract

Throughout this PA, we tackled the extensive Amazon massive intent dataset using the Bert Model and Contrastive Learning such as Supervised Constrastive Learning (SupCon) and Simple Framework for Constrastive Learning of Visual Representation (SimCLR). While implementing these models, we have tried fine-tuning the models by changing the dropout rate, batch size, learning rate, learning rate type (SGD), use and not use scheduler optimizer. In the end, we noticed that the performance of BERT without fine tuning is very poor, contrast to the BERT model with fine tuning that consistently scored in the 80 percent range for test accuracy. Among the tune parameters, we found the BERT model with the appliance of Layer-Wise Learning Rate Decay (LLRD) and Warm Steps to be the top scorer with the accuracy of 0.88, as measured by the number of correct predictions over total number of predictions. Even though contrastive learning seems to be a useful approach to train our model to well-embed the input text into the high dimensional feature space, it turns out the two techniques we have tried did not show improvements over the baseline model.

1 Introduction

In Natural Language Processing Research, Transformers have revolutionized tasks due to their ability to capture long-range contextual information effectively. In recent years, the Amazon Massive Intent dataset has emerged as a benchmark for understanding user intent through text. The dataset comprises a large number of examples across various domains, making it viable for training and evaluating the robustness and diversity models.

This report focuses on exploring transformer-based models for intent classification on the dataset.This task involves identifying the intention behind a user's query, which is crucial for building effective conversational agents and recommendation systems. By leveraging the power of transformers, we aim to achieve state-of-the-art performance on this challenging dataset.

CSE 151B: Deep Learning, Project 4 Report.

Previous research on the Amazon Massive Intent dataset has laid the groundwork for understanding the complexities of user intent classification. Papers such as "MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages" [2] and "MTEB: Massive Text Embedding Benchmark" [3] have provided insights into the nuances of the dataset and the current challenges research and evaluation methods in the field face for improving classification accuracy. Building upon this prior work, our study aims to explore current techniques by experimenting with different transformer architectures and training strategies.

In a greater scope, understanding user intent in interactions is a crucial step towards enhancing user experience and personalization in various applications, including e-commerce, customer support, and in general improving a product's user experience. This applies to more general cases as well; as artifical intelligence, and specifically NLP continues to improve, its use cases will continue to expand. Therefore, exploring transformer-based models on the Amazon Massive Intent dataset holds significant implications for advancing natural language understanding systems and facilitating more seamless human-computer interactions.

2 Related Work

The first referenced paper, "MTEB: Massive Text Embedding Benchmark" [3] introduces a pivotal framework designed to rigorously evaluate embedding methods across an array of tasks and languages. It encompasses tasks such as semantic similarity assessment, text classification, and clustering to comprehensively gauge the efficacy and versatility of text embeddings. The uniqueness of MTEB is its meticulous curation, ensuring a diverse set of tasks that reflect real-world NLP challenges. In a field where robustness is an important, yet difficult goal to achieve and evaluate, by providing a standardized evaluation methodology, MTEB not only enables benchmark reliably but more importantly facilitates fair and transparent comparisons among different approaches. The most important takeaway comes in the abstract of the paper, where it is described that "...no particular text embedding method dominates across all tasks [evaluation metrics]. This suggests that the field has yet to converge on a universal text embedding method and scale it up sufficiently to provide state-of-the-art results on all embedding tasks." Thus, this suggests that there are still uncertainties in the field on the best/most agreed upon way of evaluation. This benchmark serves as a catalyst for advancing the field of text embeddings and ensuring the development of more robust and generalizable models capable of capturing nuances across various language and domain inputs.

The second paper, "MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages" [2] is a significant contribution to the field of multilingual natural language processing because it introduces MASSIVE, a vast and diverse dataset comprising over a million examples across 51 distinct languages. This dataset is meticulously crafted to capture the linguistic diversity present in different regions and cultures, making it an invaluable resource for training and evaluating multilingual NLP models. Without a dataset with such a wide/diverse distribution, model robustness would be limited. By including a wide spectrum of tasks such as intent classification, entity recognition, and question answering/response generation, MASSIVE offers a comprehensive platform to develop and benchmark algorithms capable of handling the complexities of multilingual text understanding. Furthermore, MASSIVE's extensive coverage of typologically-diverse languages facilitates cross-lingual research, enabling insights into language universals and variations while fostering the development of more inclusive and globally applicable NLP solutions.

The third paper, "Supervised Contrastive Learning" [4] introduces a novel approach to contrastive learning. This paper helps with learning representations by maximizing agreement between similar instances and minimizing agreement between dissimilar ones. By agreement, it refers to the idea instances of data points that are closer / farther together should be brought closer / farther together in the embedding space. The paper proposes a variant of contrastive learning, which leverages labeled data to guide the learning process. It also uses a contrastive loss function, which simply refers to a design that operates by comparing representations of pairs of instances and penalizing the model based on the similarity between these representations. By incorporating class labelling into the contrastive loss function, the model learns to not only distinguish between similar and dissimilar instances but also respect class boundaries. This enables the model to learn more discriminative (and therefore more robust) and semantically meaningful representations, leading to improved performance on downstream classification tasks. SCL stands out for its ability to effectively utilize labeled data,

making it very well-suited for scenarios where labeled examples are abundant, such as in image classification or natural language understanding tasks.

The final related paper, "SimCSE: Simple Contrastive Learning of Sentence Embeddings" [5] suggests a straightforward yet highly effective method for learning sentence embeddings. Unlike traditional approaches that rely on complex architectures or pretraining strategies, SimCSE adopts a simple yet powerful framework based on contrastive learning, very similar to SCL. By formulating the task as a binary classification problem, SimCSE encourages the model to distinguish between semantically similar and dissimilar sentence pairs. Notably, SimCSE introduces a data augmentation technique that augments sentences (such as mixing the order of words) while preserving their semantic meaning, enhancing the model's ability to capture subtle linguistic nuances. This approach yields high-quality embeddings that show strong coherence and can be readily applied to a wide range of tasks, including text classification, paraphrase detection, and semantic similarity assessment. SimCSE's simplicity, coupled with its effectiveness in learning robust sentence representations, makes it a compelling choice for practitioners seeking to leverage contrastive learning for natural language understanding tasks.

3 Methods/Experiments

3.1 Baseline Model

Table 1: Hyperparameter Choice

Name	Value
Epochs	10
Batch Size	16
Learning Rate	$1e-4$
Learning Rate Scheduler	$CosineAnnealing$
Drop Out Rate	0.9
Maximum number of iterations (scheduler)	10
Minimum learning rate (scheduler)	$1e-6$
Optimizer	$AdamW$
Momentum	$False$

We employed transfer learning by integrating a pre-trained BERT model and appending two additional classifier layers atop it to fine-tune the model using the extensive Amazon massive intent dataset. The BERT architecture is comprised of a fully connected or pooler layer, eleven transformer layers, and a word embedding layer, as illustrated in the fig 1. For each of transformer layers, it will produce 768 outgoing element to the next transformer layers. We utilize the [CLS] representation, the first element of the BERT's last hidden layer, and input this representation into our top classifier [768, 10] with a dropout rate of 0.9. The output of this classification layer is then fed into the [10, 60] bottom classifier.

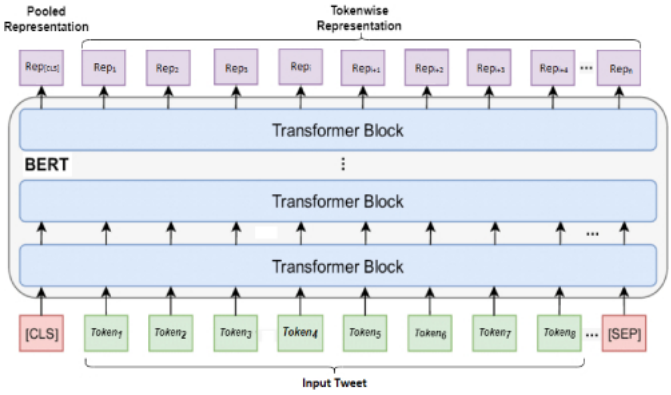

Figure 1: Bert model [6]

We fine tune our model using 10 epochs with batch size of 16. Because BERT is already well trained, we choose a low learning rate of $1e-4$, since setting too large of a learning rate will just undermine the result of the model. For the learning rate scheduler we used the Cosine Annealing scheduler, where we set $\eta_{min} = 1e-6, T_{max} = 10$. The cosine annealing learning rate scheduler helps us move towards the minimum of the loss function at a decreased speed and ensures we are not stuck at the local minimum by periodically resetting the learning rate to the original value.

The optimizer we used was the AdamW built in method from PyTorch. AdamW works by implementing weight decay or regularization only after controlling the parameter wise step size. Thus it makes the regularization term proportional to the weight itself instead of including it in the moving averages of the weights. This allows for the weights to tend towards smaller weights and lead to a better generalizing model.

3.2 Custom Fine-Tuning

In order to fine-tune our model, we implemented Layerwise Learning Rate Decay and Warm Up Steps separately and compare their results to the result of the techniques combined. The final hyperparameter choices can be seen in the tables below.

The idea behind Layerwise Learning Rate Decay is to have different learning rates depending on the layer of the model. At the top layers we have higher learning rates, which encodes task-specific information and broad dependencies. As we descend layers, the learning rate decays and decreases till we reach the bottom layer which has the smallest learning rate. The bottom layer encodes basic semantic information, syntax, and positional information. The reasoning behind using a decaying learning rate is the fact that different layers encode different information from the data.

To implement Layerwise Learning Rate Decay we use a multiplicative decay rate of 0.9 and a learning rate starting at $1\text{e}{-4}$ for our pooling layer at the top. As we descend a layer, the learning rate is multiplied by 0.9. This continues until the we reach the embedding layer where the learning rate is around $2\text{e}{-5}$. We have a separate learning rate of 0.01 and 0.009 for classification layers one and two respectively, which map the output of the pooling layer to the 60 classification categories. The classifier's learning rate is much higher than that of the BERT pretrained model. This is because we are training the classifier from scratch, and having a higher learning rate will reduce the training epoch. While BERT is already well-trained, we just need to make a small adjustment to make its model match our purpose.

The idea behind Warm Up Steps is to begin with a small, gradually increasing, learning rate, to facilitate stable training and prevent large deviations caused by new data. This helps the model converge to an optimal solution. By controlling the learning rate, the model can avoid overfitting or diverging from an optimal solution. Figure 2 below shows an example of how the learning rate should change.

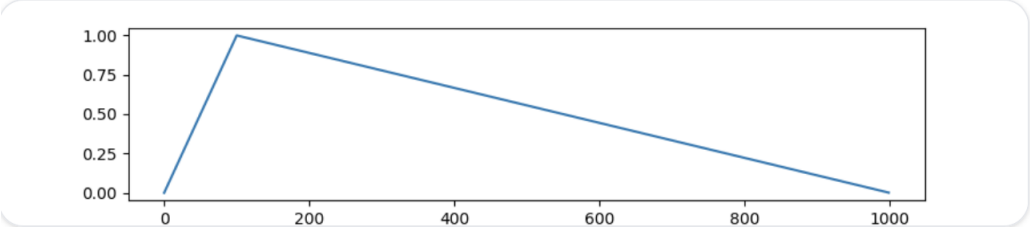

Figure 2: Warmup Steps Example

To use the Warm Up Steps technique, we have the learning rate grow linearly from 0 for 2 epochs until it reaches our initial desired learning rate of $1\text{e}{-5}$. The learning rate then decays to 0 throughout the remaining 8 epochs. For our implementation we use a linear scheduler with warmup from the Huggingface Transformer module, with two out of ten of our epochs serving as the warmup steps.

Table 2: Layerwise Learning Rate Decay Hyperparameter Choice

Name	Value
Epochs	10
Batch Size	16
Learning Rate (Classifier)	1e−2
Learning Rate (BERT)	1e−4
Learning Rate Decay	0.9
Learning Rate Scheduler	Layer-wise Learning Rate Decay (LLRD)
Optimizer	$AdamW$
Momentum	$False$

Table 3: Warm Up Steps Hyperparameter Choice

Name	Value
Epochs	10
Batch Size	16
Learning Rate	1e−4
Learning Rate Scheduler	Linear with Warmup Steps
Warmup Proportion	20%
Optimizer	$AdamW$
Momentum	$False$

Table 4: Combined Strategy Hyperparameter Choice

Name	Value
Epochs	10
Batch Size	16
Learning Rate (Classifier)	1e−2
Learning Rate (BERT)	1e−4
Learning Rate Decay	0.9
Learning Rate Scheduler	Linear with LLRD and Warmup Steps
Warmup Proportion	20%
Optimizer	$AdamW$
Momentum	$False$

3.3 Contrastive Learning

Before we go into technical details, we list the hyperparameter choices for our best reported model result in the following tables.

Table 5: Hyperparameter Choice with SupCon Loss

Name	Value
Epochs	10
Batch Size	64
Learning Rate (SupCon)	$1e-4$
Learning Rate (CrossEntropy)	$1e-4$
Learning Rate Scheduler	Linear with LLRD and Warmup Steps (Both)
Warmup Proportion	20%
Optimizer	$AdamW$
Momentum	$False$

Table 6: Hyperparameter Choice with SimCLR Loss

Name	Value
Epochs	13 for SimCLR and 20 for CrossEnt
Batch Size	256
Dropout Rate	0.9
Learning Rate (SimCLR)	$1e-5$
Learning Rate Scheduler	Linear with LLRD and Warmup Steps (Both)
Warmup Proportion	20%
Optimizer	SGD
Momentum	$True$
Momentum	0.99
Learning Rate (CrossEnt)	$1e-2$
Learning Rate Scheduler	Linear with LLRD and Warmup Steps (Both)

Since SimCLR is a self-supervised approach and it significantly relies on our model's ability to generate positive pairs and negative (dissimilar) pairs, we significantly increased the batch size to ensure that our model is trained on enough examples to learn useful hidden representations of the input texts, so that the model could well embed the input texts into the feature space.

We used two loss functions to train our embedding for the sake of comparison - one is Supervised Contrastive Learning (SupCon) and the other is Simple Framework for Contrastive Learning of Visual Representation (SimCLR). More detailed comparison between these two losses will be addressed in the discussion section of this report. Here, we mainly focus on the setup of the loss functions.

SupCon loss leverages the power of contrastive learning within a supervised learning framework. Essentially, the loss encourages the model to learn embedding in a way that input representations belonging to the same class are closer to each other in the embedding space, whereas input representations belonging to different classes are farther apart from each other. It extends the idea of supervised approach by using class labels to determine which pairs of embeddings need to be pulled together and which two pairs need to be pushed apart. Mathematically, the SupCon loss is presented as follows:

$$\mathcal{L} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a / \tau)}, \tag{1}$$

where $i \in I \in \{1, 2, \ldots, 2N\}$ for N to be randomly sampled pairs, z_ℓ is the projection of feature representation of the input in high dimensional space, $A(i) = I \setminus \{i\}$, $P(i) = \{p \in A(i) : \tilde{y}_p = \tilde{y}_i\}$ is the set of indices of all positive examples in the multiviewed batch distinct from i, and $|P(i)|$ is its cardinality [4]. τ denotes the temperature scaling parameter that helps to control the separation of similarity score

Now, we formalize the SimCLR loss functions that we used, as another method, to train the model embedding, or representation, of the text input. For the original paper, the goal of SimCLR is to learn effective visual representations without relying on explicit labels, by teaching the model to identify similar (positive) and dissimilar (negative) pairs of images based on augmented versions of the input data. In the context of working with text data, we put one pair of text embeddings into the loss function and the loss function would encourage the model, in a self-supervised way because we do not have labels passed in, to achieve the similar goal. The loss function is designed to maximize the similarity between representations of positive pairs (two different augmented views of the same text inputs) while minimizing the similarity between negative pairs (augmented views of different text inputs). This contrastive approach enables the model to learn rich and generalizable features by understanding what makes two different augmentations similar or different, without needing explicit label information. Mathematically, the loss could be formulated as follows.

Given a paif of augmented text inputs x_i and x_j, which serve as a positive pair, and a set of other images that act as negative examples, the loss for a positive pair (i, j) in a mini-batch of size N is defined as

$$\ell_{i,j} = -\log \frac{\exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(sim(z_i, z_k)/\tau)}, \tag{2}$$

where z_i, z_j denotes the representations of the two augmented text inputs x_i, x_j, $sim(z_i, z_j)$ denotes the cosine similarity between two input embeddings, τ denotes the temperature scaling parameter that helps to control the separation of similarity score, and $\mathbb{1}_{k \neq i}$ is an indicator function that equals to 1 if $k \neq i$ and 0 otherwise. More specifically, the cosine similarity for the pair z_i, z_j is given by

$$sim(z_i, z_j) = \frac{z_i^T z_j}{\|z_i\| \|z_j\|}. \tag{3}$$

Then, the total loss, or the cost is given by

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} \left[\ell(2k1, 2k) + \ell(2k, 2k-1) \right], \tag{4}$$

as detailed in the original paper [1].

After setting up the loss functions, we explain how the model is trained to compare the effect of these two loss functions with other model results we have obtained. The inputs to the loss function z_i and z_j are the embeddings for the text inputs. More specifically, we feed the input to the encoder, take the *last_hidden_state*'s [CLS] token as output of the encoder, like we did for the baseline model, feed it to a dropout layer with the preset dropout rate in the argparse argument, and pass the result to a projection head (linear layer). By adding noise using the dropout layer and repeating the process twice, we are able to augment the original embedding and pass the two augmented embeddings to the loss function to calculate the loss during training. More about how augmentation works will be discussed in the discussion section of this report.

However, the loss alone does not tell us how good we are at predicting the class labels. By freezing the classifier layer first, we are able to train the embeddings to pull positive pairs together and push negative pairs apart using either SupCon or SimCLR loss function. After that, we unfreeze the classifier layer and freeze all other layers to train the classifier using Cross Entropy loss function. This is done in a separate training loop after our model has learned a good mapping to embed the input text into a high dimensional feature space.

4 Results

In all test losses provided below, we provide the average loss, instead of the total loss when evaluating our model performance. Since we have 2974 testing examples, the total loss should be multiplied by 2974 to obtain the total test loss for our model. Accuracies are simply measured by the number of correct predictions over total number of predictions. To train the model with contrastive learning, we applied both fine-tuning strategies we explored.

The experiment results are summarized in the following table, including test statistics that are all rounded to 7 decimal places.

Table 7: Models and Their Performances

Exp Idx	Experiment Name	Test Loss	Test Accuracy
1	Test set before fine-tuning	0.0166073	0.0097512
2	Test set after fine-tuning	0.0641283	0.8493611
3	Test set with <Layer-wise Learning Rate Decay>	0.0665530	0.8117014
4	Test set with <Warmup Steps>	0.0595268	0.8540686
5	Test set with <LLRD and Warmup Steps>	0.0645733	0.8823134
6	Test set with SupContrast	0.0106450	0.8739072
7	Test set with SimCLR	0.0030572	0.8074916

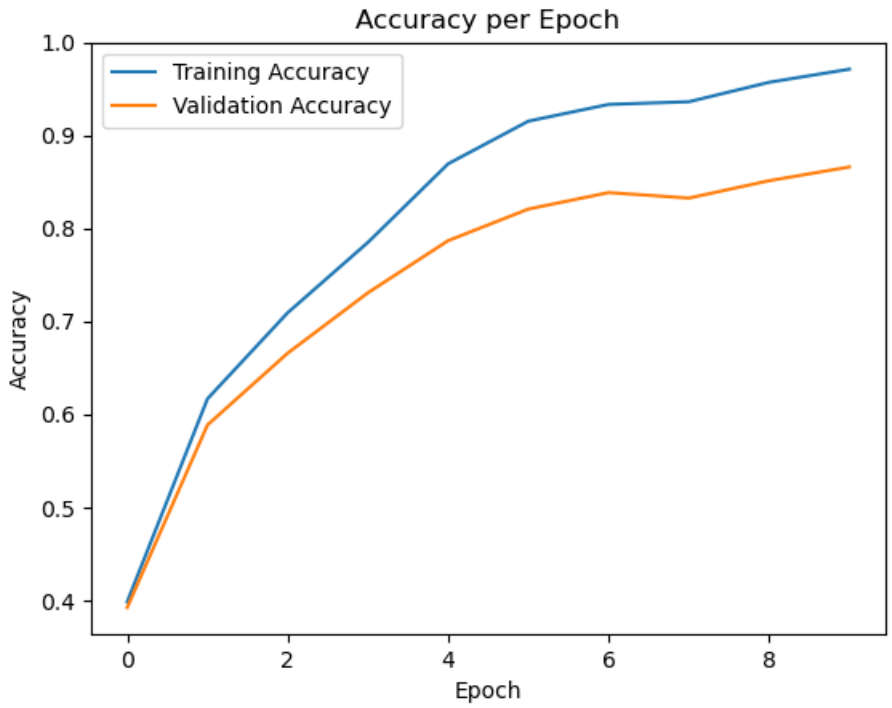

Figure 3: Baseline Model

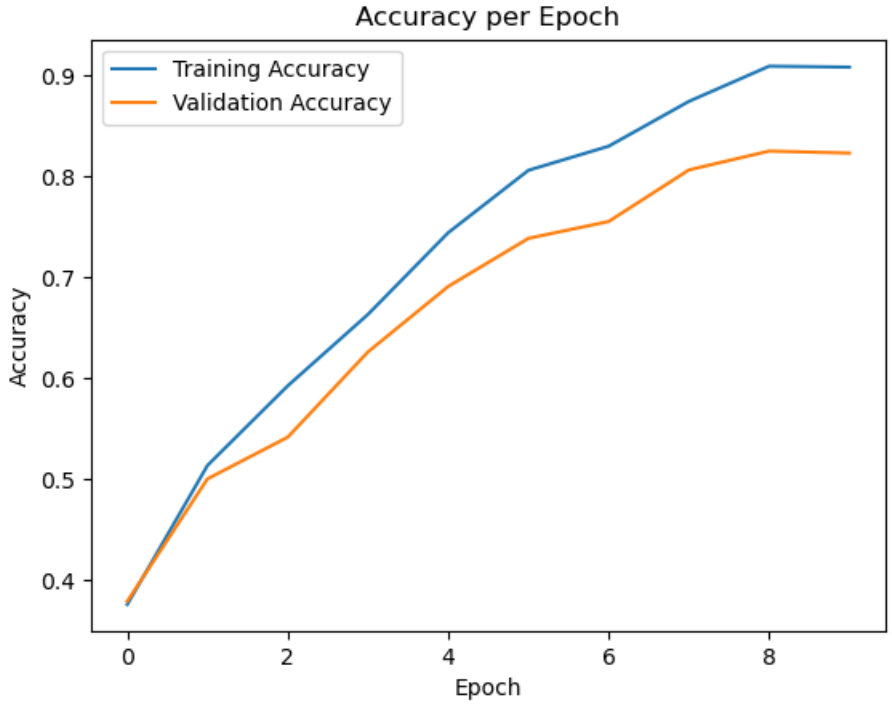

Figure 4: Fine-Tuning with LLRD Only

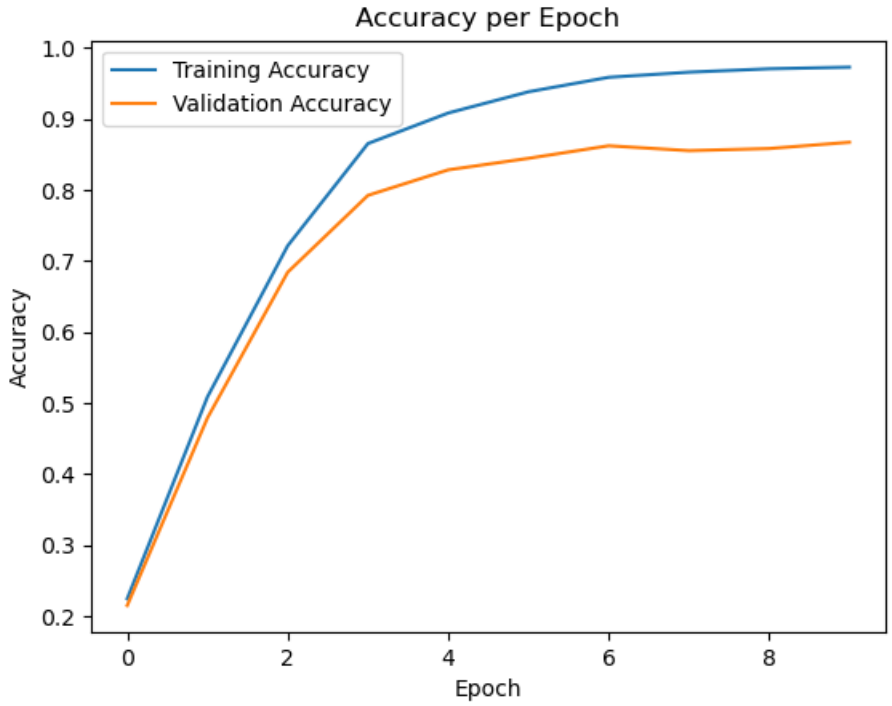

Figure 5: Fine-Tuning with Warmup Only

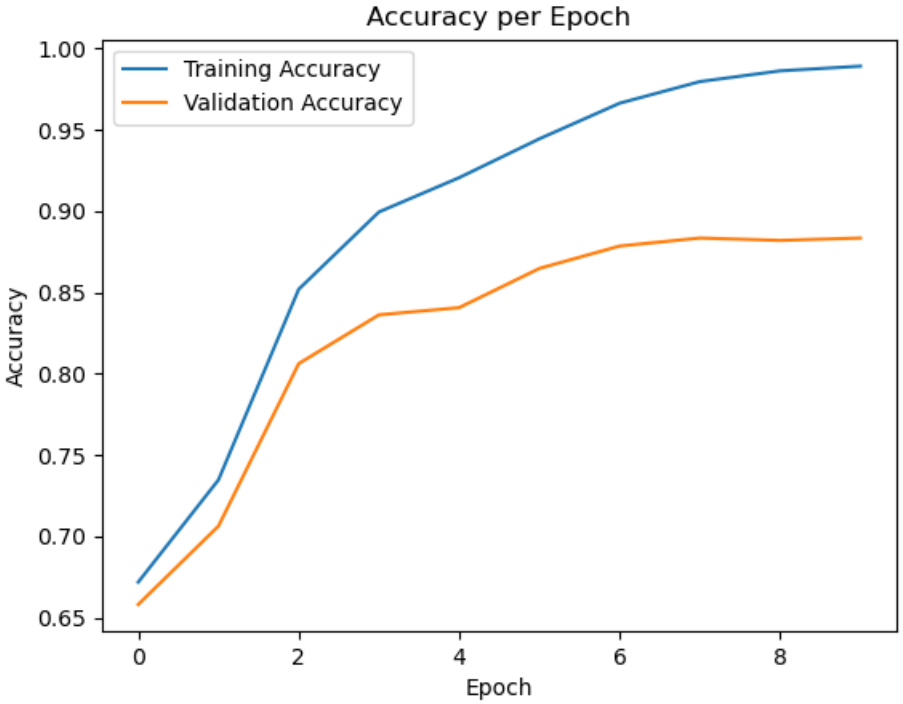

Figure 6: Fine-Tuning with LLRD and Warmup

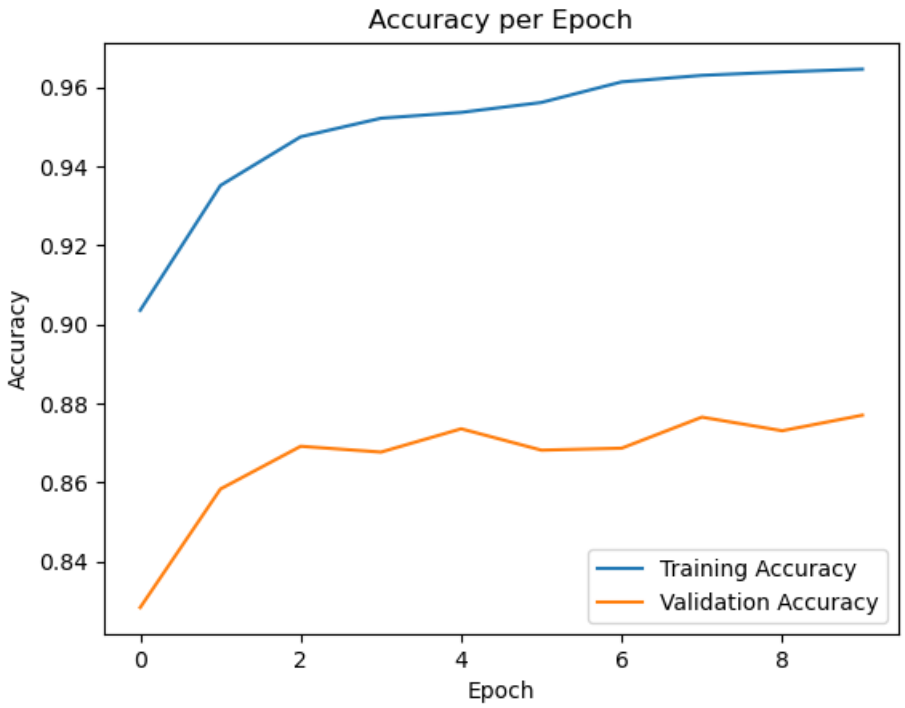

Figure 7: Contrastive Learning with SupCon Loss

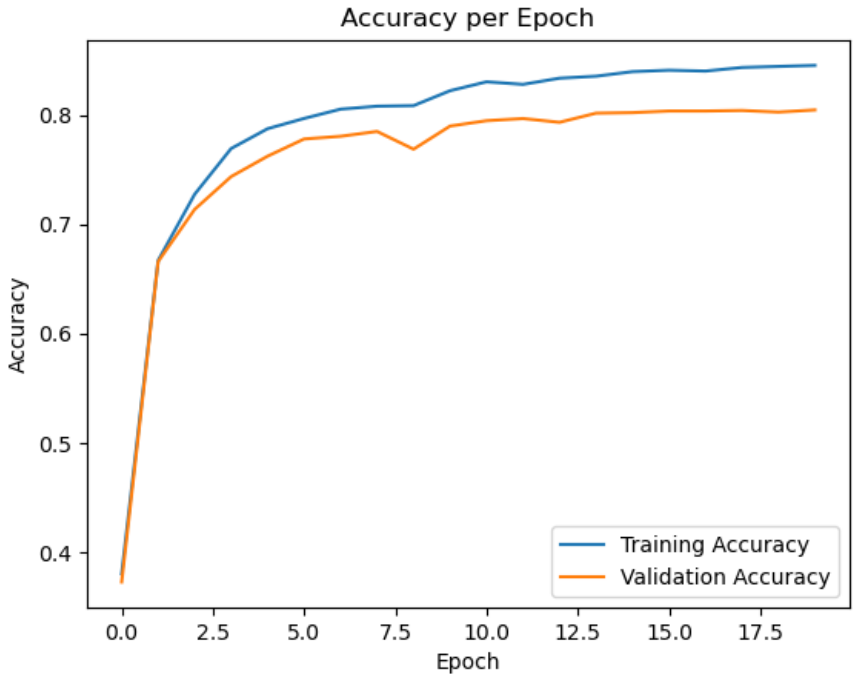

Figure 8: Contrastive Learning with SimCLR Loss

As a side note, we also provide the SupCon loss and SimCLR loss after training for 10 and 13 epochs respectively. The losses, rounded to 7 decimal places to keep consistancy, are summarized as follows:

Table 8: SupCon and SimCLR Loss Values

Loss Function	Loss Value
SupCon	285.7891606
SimCLR	119.4280379

5 Discussion

5.1 Q&A from Writeup

Q1: If we do not fine-tune the model, what is your expected test accuracy? Explain Why.

A1: If we do not fine-tune the model, the expected test accuracy will be slightly better than random guessing $(1/60)$ because the BERT model is trained on a large text corpus and we expect such a model to generalize well on other natural language tasks. Additionally, BERT works well on generalizing robust natural language tasks because its encoder implements the like of self-attention mechanism, bidirectional context, embeddings, multi-attention heads, which have allowed BERT to capture semantic meanings and relationships between texts. Since majority of the natural language tasks follow similar structures that BERT was trained to tackle, hence, BERT often performs exceptionally well on a variety of tasks. Therefore, the expected test accuracy to be between 70-80 percent accuracy seems reasonable.

Q2: Do results match your expectation(1 sentence)? Why or why not ?

A2: Based on our model result, we obtained around 0.00975% test accuracy before fine-tuning (training). This is even worse than random guessing and does not match with our expectations and we were expecting too much from the BERT model. The reason is that, despite BERT is robust at generalizing different tasks, the model is not good at generalizing to our classification task that we would have liked the model to do, unless we fine tune the model furthermore. Our model has two classifiers with weight initialized randomly without learning hidden representations in the service of our classification task. Therefore, even through BERT is known to generalize well for different tasks given that BERT is trained on a very large dataset and can correctly sort text into 768 categories, the classifier will just randomly assign each element of 768 input element to 10 hidden layer elements and eventually to 60 category. Therefore, expected initial test accuracy will be around $1/60 \approx 0.0167$.

Q3: What could you do to further improve the performance ?

A3: The general strategy to further improve the performance includes increasing the Epoch number as we notice that our baseline model with CosineAnnealing Scheduler is undertrained. Increasing the training epochs while adding technique of early stopping should improve the overall performance. Another strategy that we should train is to fine tune hidden layer size of top and bottom classifier. Because our model in essence is a unlinear bottleneck structure, adjusting the size of hidden layer will improve the overall performance of model to summarize all important features it needs to make correct prediction. Also, we could adjust layerwise learning rate, so that for the classifier layer, it can have higher learning rate and for the bert as it has already been well trained with large amount of data, we should use smaller learning rate for fine tuning BERT model.

Q4: Which techniques did you choose and why?

A4: We used the Layerwise Learning Rate Decay and Warm Up Steps techniques to fine-tune our model. Layerwise Learning Rate Decay allows us to maintain the model blocks that have already acquired general patterns. Instead, we focus the fine-tuning process on the top layers, which typically specialize in learning specific features for the particular purpose of the task. Therefore, by using layerwise learning rate decay, we can select more useful data features for our purpose instead of changing the well-trained source of data, which is the whole purpose of importing the BERT model.

Warm-up is also an important technique to reduce unfamiliar data. The Amazon massive intent dataset we use for fine-tuning might be quite different from the text dataset Bert used to train. Incorporating warm-up techniques allows us to reduce the negative of introducing unfamiliar data, or completely different data, in Amazon's massive intent dataset, as the warm-up technique, as described in section 3, will gradually increase the learning rate from zero to the expected learning rate. Also, reducing the learning rate after a certain epoch is also normally beneficial in model training, as we have already almost converged the model to its best performance; having a learning rate as high as before might cause the model's weight to jump back and forth instead of converging directly. Overall, controlling the learning rate in this way prevents overfitting and ensures convergence to an excellent solution. By using these techniques, we can have more stable training and a model that better adapts to our problem by modifying the learning rates. We expect these techniques to improve the performance of our models.

Q5: What do you expect for the results of the individual technique vs. the two techniques combined?

A5: We anticipate achieving higher validation and test accuracy by combining both techniques compared to using them individually. As elaborated in A4, each technique serves a distinct purpose in enhancing the fine-tuning of pre-trained models. The warm-up technique mitigates data unfamiliarity, while layerwise learning rate adjustment enables us to focus on refining feature representations rather than changing the pre-existing well-trained feature library.

Through this combined approach, we can effectively regulate the learning rate, facilitating improved generalization of our model to new data and tasks. While each technique applied separately is expected to yield enhancements over our baseline model, the synergistic effect of their combination is projected to deliver even more substantial improvements.

Q6: Do results match your expectation(1 sentence)? Why or why not?

A6: The results show a higher validation accuracy when using the two techniques together versus separate, which is as expected. As explained before, both techniques allow us to control the learning rate to prevent overfitting and so that the model converges to an optimal solution.

Q7: What could you do to further improve the performance?

A7: To further improve the performance we could implement other fine tuning techniques such as Re-initializing Training Layers or using Stochastic Weighting Average. Both of these techniques can lead us to a better and faster convergence. Furthermore we could further optimize both of the current techniques we used. For Layer-wise Learning Rate Decay we can modify the rate of decay or use a layer grouping implementation to improve our results. For Warmup Steps we can use different linear growths and decays and a different optimal learning rate to improve the performance of our model.

Q8: Compare the SimCLR with SupContrast. What are the similarities and differences?

A8: SimCLr and SupCon are both contrastive learning frameworks to learn feature embeddings of the input, where the goal of the loss is to bring closer the embeddings of similar (or positive) pairs of examples while pushing apart the embeddings of dissimilar (or negative) pairs of examples. More over, in our contextl, both loss functions take in embeddings of text input by utilizing data augmentation techniques to generate positive pairs. In SimCLR, these are two different augmented views of the same text input, whereas in SupCon, the two embeddings are the augmented views by leveraging the label inforamtion to define positive and negative pairs. Lastly, both loss functions incorporate temperature τ to help control the separation of similarity score.

However, these two loss functions are different in a sense that SimCLR operates in a self-supervised learning setting, where no labels are used to define positive and negative pairs. Positive pairs are generated from different augmentations of the same text input, and all other text inputs are considered negative examples. SupCon, on the other hand, utilizes a supervised learning method that leverages class label information to define positive pairs (different text inputs of the same class) and negative pairs (text inputs from different classes). In addition, SimCLR learns general features from the data without relying on labels, making it broadly applicable to various downstream tasks. SupCon, however, seeks to enhance the discriminative power of the representations by explicitly using class

labels during training, potentially offering better performance on tasks where label information is crucial.

Q9: How does SimCSE apply dropout to achieve data augmentation for NLP tasks?

A9: SimCSE introduces a simple yet effective method for generating positive pairs of sentences for contrastive learning by exploiting the dropout mechanism as a form of data augmentation. By adding noise through randomly dropping out a subset of neurons in a layer during the training phase, SimCSE leverages the inherent randomness introduced by dropout as a form of data augmentation for text inputs. When a text input is passed through our BERT-based model with dropout enabled, the randomness in dropout paths results in slightly different embeddings for the same text input at different forward passes. By feeding the same input text through the model's forward pass twice, SimCSE generates two semantically similar but not identical embeddings, which are then treated as a positive pair for the contrastive learning task.

Q10: Do the results match your expectation? Why or why not?

A10: Since SimCLR and SupCon are specific techniques used to train the embedding part of our model, we expected our test accuracy to improve by an edge because we are applying custom techniques to train each layer of the network. However, we can see that neither of the approaches showed improvement over the baseline model, especially the model whose embedding is trained with SimCLR loss function.

One potential reason could be the way we applied the data augmentation – using dropout alone could be a simple yet effective approach to perform data augmentation, but it might just be too simple in the case of our classification task. SimCLR relies heavily on data augmentation to generate positive pairs (two different augmented views of the same image) for training. If the data augmentation is too weak, the model might not learn more useful features than the baseline.

Another reason could be that our batch size is not large enough. In our case where we used 256 as the batch size. Though larger compared with other models experimented, it might still not be large enough to yield enough useful positive pairs of text embeddings, thus disallowing our model to learn enough useful hidden representations of the text input to embed positive pairs close together. Likewise, 256 might not be large enough of a batch size to provide enough negative samples for each positive pair, limiting our model's ability to distinguish between pairs of text inputs from different classes.

Additionally, we used to use AdamW optimizer to train the embedding part with contrastive learning, but the model is not performing well on the test set. Then, we switched to SGD optimizer, as done in the paper's implementation [4] and the model's performance gets better. As a result, we suspect that the choice of optimizer and the corresponding learning rate scheduler might be different than our previous models, thus requiring us to find a more suited set of optimizer choice in the service of our training task.

Finally, the low accuracy might come from our classifier layer. Even though the customized loss functions help us train the model in a way that similar text inputs are embedded closer together and dissimilar text inputs are embedded farther apart, the linear head that follows might not be powerful enough to capture the nuances of the feature embedding in the high dimensional space. Consequently, our model fails to classify text inputs in their corresponding categories correctly in some edge cases, which explains why our test accuracy is lower than the baseline model.

Q11: What could you do to further improve the performance?

A11: In this section, we address the possible issues one by one and propose ways to improve model performance.

Firstly, to address the data augmentation aspect, we can fine-tune our model further with more choices of dropout rates, so we can apply the augmentation in various ways the find the best way that's suited for our model architecture. Also, we can find a better augmentation approach, such as slicing different parts of the text input. By applying more careful data augmentation, the contrastive learning approach could take its effective to maximum and truly learn the hidden representations of the input texts that differentiate the class labels.

For the second point, we could find a machine with larger memory, so we can fit larger batch size in each epoch, thus training on more dissimilar examples during our training procedure and enabling the contrastive learning to learn more useful hidden representations of the input texts. By increasing the batch size to even larger, we expect our model to generalize better to unseen data, as it can do a better job at pulling positive pairs closer together and pushing negative pairs farther apart in the high dimensional feature embedding space. As a result, by providing better embedding on the input text, our classifier layer could potentially do a better job at classifying the input texts into its correct category.

For the third point, we only implemented LLRD and Warmup schduler fine-tuning strategies to improve the baseline, which might not be well-sutied for the contrastive learning with SGD optimizer. We could potentially incorporate approaches like *Re-initializing Pre-trained Layers*, *Stochastic Weight Averaging*, and *Frequent Evaluation* to check which one is better suited with the contrastive learning part. Consequently, we might be able to obtain a better feature embedding for a given input text. This helps us detecting more nuances inherent in the input text and thus classifying them better, as more hidden representations might have been learned through the contrastive learning process.

Lastly, instead of using one linear head as our classifier, we could have applied *ReLU* activation function, followed by another linear layer, together to serve as our projection head. With the help of *ReLU* and its added non-linearity in our model, our model might be able to learn more hidden representation in the service of the classification task, thus doing a better job at predicting the correct class label for a given input text.

6 Authors' Contributions

Hargen Zheng I helped Ziyue to debug the baseline model training, though it was not succesful and eventually Ziyue figured out the problem on his own. I was not significantly involved in the baseline part because I have prior experience in transfer learning with the BERT model. In this case, contrastive learning sounds new to me and I implemented the SupCon and SimCLR parts of the projects, along with help of Ziyue and Adam, who helped me debug my training loop and model.py to get it running. For the report, I mainly contributed to the discussion Q7-11 and all parts relevant to contrastive learning.

Chuong Nguyen I helped with starting and debugging with the baseline model. I additionally tried implementing Stochastic Weight Averaging (SWA) but didn't seems to work. I helped with writing the discussion and fine tune the SimClr model using different hyperparameters.

Nathaniel del Rosario I implemented the baseline model (the baseline class in model.py as well as the training loop) as well as got the dataloader working, and then pair programmed the debugging process with the rest of the members who worked on the baseline model. I also helped write the introduction and related work sections, which were peer reviewed by the other team members.

Adam Tran I helped implement the contrastive learning part of the model. Furthermore, I contributed to writing about the custom fine-tuning strategies within the report as well as parts of the discussion questions. I ran multiple experiments to plot the training accuracy and validation accuracy. Lastly, I reviewed the report and readied it for submission.

Ziyue Liu I implemented the Baseline and Custom training part of the model. I have also worked with Hargen to debug SimCLR and SupCon model. In the end, I contributed to the writing of the baseline model and contributed Q3-6 in the discussion section within the report.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[2] Charith Peris Scott Mackie Kay Rottmann Ana Sanchez Aaron Nash Liam Urbach Vishesh Kakarala Richa Singh Swetha Ranganath Laurie Crist Misha Britan Wouter Leeuwis Gokhan

Tur Prem Natarajan Jack FitzGerald, Christopher Hench. Massive: A 1m-example multilingual natural language understanding dataset with 51 typologically-diverse languages. In *International conference on machine learning*. PMLR, 2022.

[3] Loïc Magne Nils Reimers Niklas Muennighoff, Nouamane Tazi. Mteb: Massive text embedding benchmark. In *International conference on machine learning*. PMLR, 2022.

[4] Chen Wang Aaron Sarna Yonglong Tian Phillip Isola Aaron Maschinot Ce Liu Prannay Khosla, Piotr Teterwak. Supervised contrastive learning. In *International conference on machine learning*. Google Research, MIT, Snap Inc, Boston University, 2021.

[5] Danqi Chen Tianyu Gao, Xingcheng Yao. Simcse: Simple contrastive learning of sentence embeddings. In *International conference on machine learning*. Department of Computer Science, Princeton University, Institute for Interdisciplinary Information Sciences, Tsinghua University, 2022.

[6] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.