Observations of Transformers in Natural Language Processing Applications

Jason Stanley
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
jtstanle@ucsd.edu

Elliot Lee
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
eal001@ucsd.edu

Michael Ma
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
jum002@ucsd.edu

Annabella Macaluso
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
amacalus@ucsd.edu

Abstract

In our work we investigated how to create custom models utilizing a pre-trained fine-tuned BERT Model as our baseline. We tuned hyper-parameters to get a well-performing fine-tuned BERT model and in addition looked into alternative custom fine-tuning strategies to try and improve our accuracy even further. The strategies we focused on were changing the warm up steps which altered our learning rate and re-initializing the top layers in order to get a better abstraction of our dataset. Please continue reading to see our justification behind these strategies and how we arrived at our specific custom model. The accuracy and loss we achieved at the end was 86.65% using our fine tuned custom model and 87.424% using our custom fine-tuned model using both of the advanced techniques. Additionally, when looking at the SimCLR and SupCON models, we can see that through fine tuning our hyper parameters, we are able to greatly increase the performance as well, going well beyond that of the baseline model. Which we see qualitatively though the tightness of groupings in the UMAP for these various models, tighter groups mean the model was better at grouping like phrases and distancing those with separate intent. Our best results for this were achieved with the Supervised Contrastive Model after we tweaked the hyper parameters to help the model perform better.

1 Introduction

Transformers have achieved state-of-the-art performance in many different tasks recently. Specifically, the pre-trained BERT (Bidirectional Encoder Representations from Transformers) model, proposed by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, was designed to pre-train deep bidirectional representations from unlabeled text. It's also made to be easilly fine-tuned by just adding one additional output layer. With this it can be applied to a variety of tasks such as question answering and language inference without having to greatly modify the model.[5]

For our purposes we wanted to train the BERT model to generalize to the Amazon Massive Intent dataset [6]. Other works that use this dataset are LINGUIST [8] and Alexa Teacher Model [7] which utilize the intents to better understand natural language systems. More information about them is given in the related works section of this paper.Given a sentence it should be able to output a specific intent. Thus, in our approach we attached a Classifier output layer in order to fine-tune the BERT model to generalize to the Amazon Massive Intent dataset.

We also wanted to understand how different training techniques and fine-tuning could effect the accuracy of the model. We experimented with different hyper parameters such as number of epochs, drop out rate and embedding size and assessed their affect on the model. For more advanced techniques we applies new training techniques obtained from the blog post "Advanced Techniques for Fine-tuning Transformers" by Peggy Chang [4]. We used her methods on warm-up steps and re-initializing the weights to achieve better convergence from the model through better abstraction of the data and less overfitting.

Finally, we trained the model with contrastive losses, a new method of loss that was proposed by the Supervised Contrastive Learning paper as an alternative to the current date, most widely used loss function; cross-entropy loss. Cross-entropy loss suffers from several shortcomings such as a lack of robustness to noisy labels and passibility of poor margins which lead to a reduction of generalization performance. The paper continues to state that due to no better alternative loss functions in the past, cross-entropy loss was thus consistently used. However, they proposed a new loss method for supervised learning. The sample code that came with the paper is located at the SupContrast Repository on GitHub. We investigated the differences between Constrastive Loss in SupCon (Supervised Contrastive Learning) and SimCLR (Simple Framework for Contrasive Learning) and how batch size, temperature, learning rate and epochs affected the way the models were able to converge and cluster data. We utilized UMAP embeddings and plots in order to visualize this clustering and assess how the model was performing. Please see our Experiments section for the results.

2 Related Work

Please view References section for work citations.

We utilized the pre-trained BERT model and optimizations from HuggingFace. Please visit their site for more information and their documentation.

[2] Supervised Contrastive Learning

This paper proposes a new alternative loss function to the popularly used cross-entropy loss function to achieve better generalization in contrastive learning. To view more code implementation please see the example code on their SupContrast Repository on GitHub.

[3] SimCSE: Simple Contrastive Learning of Sentence Embeddings

This paper proposes a contrastive learning approach that uses both supervised and unsupervised learning. The unique aspect of their unsupervised learning is the fact that they train SimCSE to predict the input itself using dropout as noise for the sentences, and received promising results.

[4] Advanced Techniques for Fine-tuning Transformers by Peggy Chang

Chang discusses different techniques that can be utilized in order to fine-tune transformers such as; Layer-wise Learning rate Decay, Warm-up Steps, Re-initializing Pre-trained Layers, Stochastic Weight Averaging and Frequent Evaluation.

The following lecture slides from Gary Cottrell's CSE 151b class on Deep Learning were also informative for the fundamental understanding of how Transformers work and how to build custom models from them.

- Vision Transformer Networks (ViT) pages 3, 13-26, 32-35
- SimCLR pages 6, 19-30
- Bert PA4 Discussion Slides 5-16

The dataset we're using is based on the Amazon MASSIVE Dataset [6]. As it is a relatively new dataset, there are not many related works.

Some related works that make use of the Amazon Massive Intent dataset are

[7] Alexa Teacher Model: Pretraining and Distilling Multi-Billion-Parameter Encoders for Natural Langauge Understanding Systems

This paper presents a large-scale experiment on pretraining encoders with non-embedding parameters and examining their application to Natural Language Understandign (NLU) for a virtual assistant (presumably, Alexa). They show a novel training method for natural language understanding and how an improvement in error rates for intent classification using their method along with a full virtual assistant experimentation platform where the models are utilizied.

[8] LINGUIST: Language Model Instruction Tuning to Generate Annotated Utterances for Intent Classification and Slot Tagging

This paper discusses a new method for annotating data through Intent classification and uses an instruction prompt to generate data with user requested data and model-generated data. It functions as a conversational agent that a user can talk to and is cross-lingual and multi-lingual making it a novel instruction prompt.

3 Experiments

3.1 Baseline

The final experiment settings we used for our baseline model is 10 epochs, 0.00008 learning rate, 0.45 drop-out rate, embedded dim of 768 and batch size of 32.

The experiment settings before fine-tuning the baseline model that were given to us was 1 epoch, 3 learning rate, 0.9 drop-out rate, embedded dim of 768 and batch size of 32. The loss and accuracies we achieved before and after fine-tuning:

epx idx	exp	loss	accuracy
1	Test set before fine-tuning	3.81	0.07
2	Test set after fine-tuning	1.11	86.583

We arrived at this experimental settings by doing the following steps

- First we increased the number of epochs to 10 as suggested by the write-up. When we evaluated this with a learning rate of 0.0005 we got an accuracy 83.759% which meant the spec
- Then we set the learning rate to 0.00001 (it was originally 3) and got a test accuracy of 54.97%. The learning rate is too slow and wasn't able to generalize quickly enough. Then we set the learning rate to 0.00004 and increased epochs to 15 and got a test accuracy of 84.297%. Since learning rate is slow it takes more epochs to generalize. We tried increasing learning rate to 0.00008 and set epochs to 10 again and got a test accuracy of 85.57%. We felt with the amount of time saved, increasing the learning rate worked better. However, we noticed that overfitting does begin to occur towards the end.
- We kept epochs to 10 and increased batch size to 64 (originally 32) and had a significant decrease in test accuracy to 76.8%. After decreasing batch size to 16 and we achieved a test accuracy to 83.89%. However, this trained extremely slowly because with a smaller

batch size we'd have to loop through more often and since we achieved only marginal improvement we decided not to include this parameter in our final fine-tuned model.

- We dropped the drop-out rate to 0.45 from 0.9 and got a test accuracy of 85.4%. We believe this allows the BERT transformer to generalize better by reducing the amount of noise during training. We found 0.9 drop-out to be far too much.
- We didn't have much success with other hyper-parameters.

This is the process we went through to arrive at our final experiment settings for the fine-tuned baseline BERT model.

3.2 Custom

The final experiment settings we used for our custom model is:

- epochs 10
- learning rate 0.00008
- drop rate 0.45
- warm up steps 10
- number of reinitialized layers 1

We achieved the following losses accuracies for the different methods

epx idx	exp	loss	accuracy
3	Test set with 10 warm up steps	1.103	86.785
4	Test set with 1 reinitialized layer	1.124	86.650
5	Test set using the 2 above methods	1.109	87.424

The hyper-paramters here with the exception of the warmup steps and number of reinitialized layers are the same as our fine-tuned BERT model from the previous subsection.

We arrived with these experimental settings by performing the following steps

- First set the reinit n layers to 3 which is what was described in the blog. However, this must've reinitialized too many important layers because our accuracy went down drastically. So we tried reinit n layers = 1 and we were able to get an improvement in our accuracy to 86.65%.
- With the amount of warm-up steps we found we got gradual improvement until we got to step 10. After 10 steps we experienced a decrease in accuracy. This meant our learning rate was affected too much and most likely wasn't able to return to its original value. So we chose 10 as our number of warm up steps as it gave us an accuracy of 86.95%.

This is the process we went through to arrive at our final experiment settings for the custom fine-tuned baseline BERT model.

3.3 Contrastive Learning

For SimCLR the final experiment settings we found were: 20 epochs, batch size 128, temp 0.3 and drop rate 0.4

The final experiment settings we use for our supervised contrastive model (SupCon) is: 5 epochs, batch size 128, temp 0.4, drop rate 0.5.

- We implemented a training script similarly to the first two tasks, with a loop over a constant amount of epochs and training the model over a mini-batch split of the dataset.
- We then implemented a contrastive learning model that can train over two different loss functions, either Supervised Contrastive or SimCLR (unsupervised).

- This model uses a pre-trained BERT model as a base with a dropout layer and a linear layer after. Then our desired loss function trains this linear layer and fine tunes the weights for the BERT model.

- After running into training complications, we realized that the model should be trained over duplicate inputs. The model's dropout layer produces a random noisy version of the final output features. So, the same input will produce a slightly different output because of the dropout layer. Then the model can compare one output with a 'duplicate' output and adjust the weights accordingly.

Our final result of validation and test accuracies were far lower than the previous two tasks. However the map of our embedding feature clusters produced good looking results.

Our baseline model produced some decent clustering with a few outlier points and some combined groups.

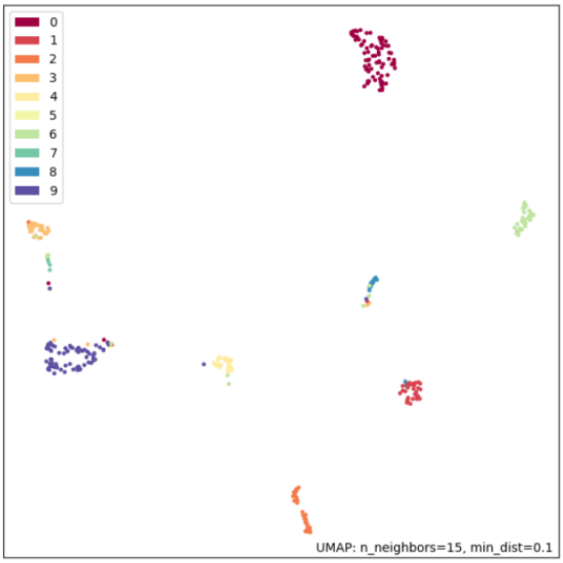

Figure 1: CrossEntropy Model UMAP Embeddings using fine-tuned baseline hyperparameters

Without tuning our hyper-parameters our supervised contrastive model produced curved line shaped embeddings.

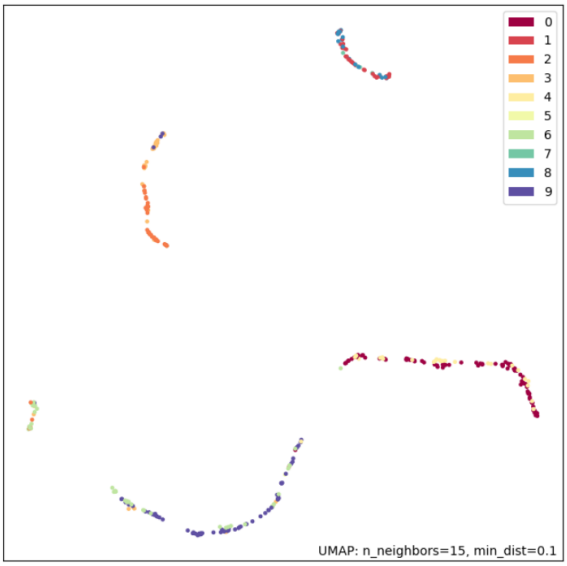

Figure 2: SupCon Model UMAP Embeddings
Batch Size: 32 Temperature: 0.7, Drop Rate: 0.9, Learning Rate: 5e-5, Epochs: 5

After tuning our hyper-parameters we saw that our SupCon Loss Model converged on groups quicker than the SimCLR model. This makes sense as the the SimCLR does not use labels to help train the model. Therefore it may take more time to converge on dense clusters.

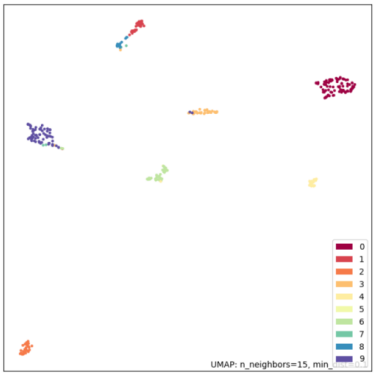

Figure 3: SupCon Model UMAP Embeddings
Batch Size: 128 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

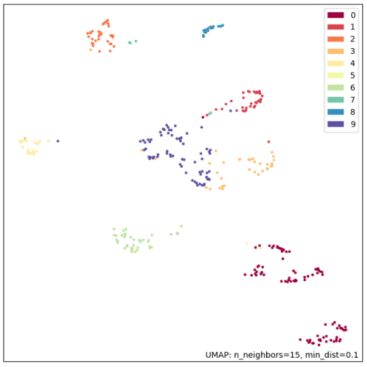

Figure 4: SimCLR Model UMAP Embeddings
Batch Size: 128 Temperature: 0.3, Drop Rate: 0.4, Learning Rate: 5e-5, Epochs: 20

We also tested embeddings for different size minibatches. It seems like the larger minibatches gave slightly better clusters, but there was not much difference between the different mini-batch sizes if no other hyper-parameters were changed.

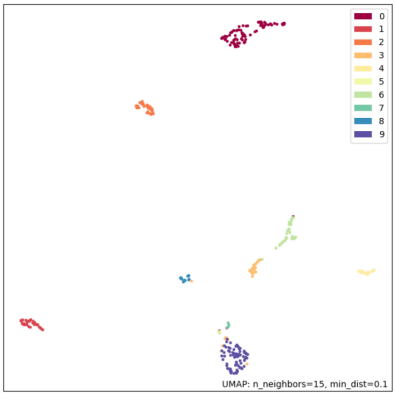

Figure 5: SupCon Model UMAP Embeddings
Batch Size: 16 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

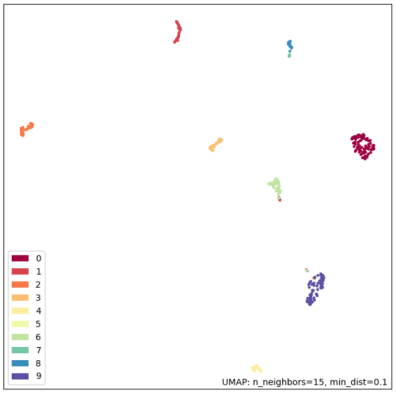

Figure 6: SupCon Model UMAP Embeddings
Batch Size: 32 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

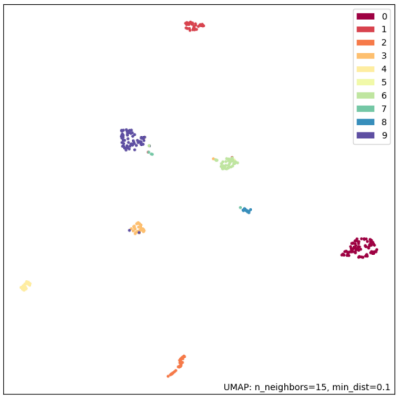

Figure 7: SupCon Model UMAP Embeddings
Batch Size: 64 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

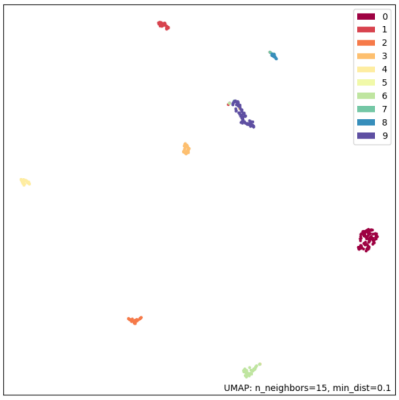

Figure 8: SupCon Model UMAP Embeddings
Batch Size: 128 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

4 Discussion

Before beginning the answers to the PA questions here are the tables with our results.

epx idx	exp	loss	accuracy
1	Test set before fine-tuning	3.81	0.07
2	Test set after fine-tuning	1.11	86.583

epx idx	exp	loss	accuracy
3	Test set with 10 warm up steps	1.103	86.785
4	Test set with 1 reinitialized layer	1.124	86.650
5	Test set using the 2 above methods	1.109	87.424

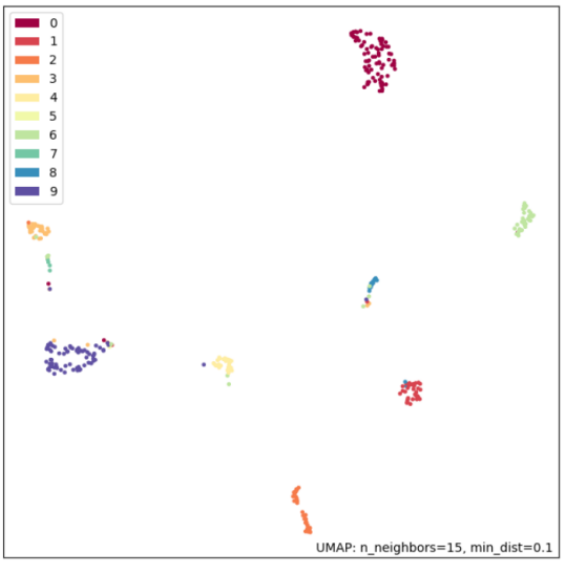

Figure 9: CrossEntropy Model UMAP Embeddings using fine-tuned baseline hyperparameters

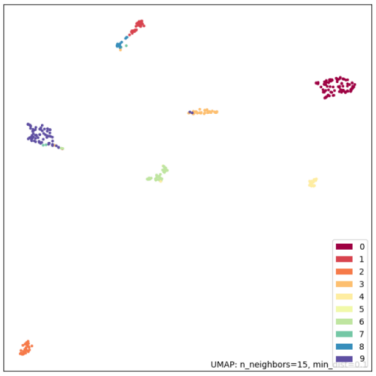

Figure 10: SupCon Model UMAP Embeddings
Batch Size: 128 Temperature: 0.4, Drop Rate: 0.5, Learning Rate: 5e-5, Epochs: 20

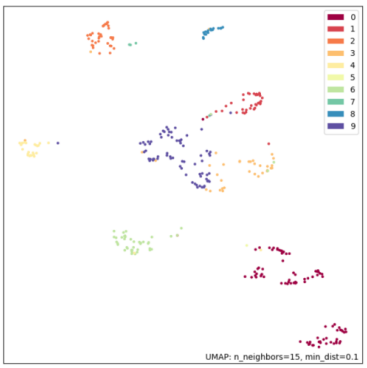

Figure 11: SimCLR Model UMAP Embeddings
Batch Size: 128 Temperature: 0.3, Drop Rate: 0.4, Learning Rate: 5e-5, Epochs: 20

- Q1: If we do not fine-tune the model, what is your expected test accuracy (1 sentence)? Why (1 sentence)?

  A1 (2 sentences):

  If we do not fine-tune the model, then our expected test accuracy should be low, most likely below 10%, because the Classifier that's attached has not been trained and is initialized with random weights. The pre-trained BERT model was trained to predict sentences from masked sentences, not made to classify amazon intents which is the role of our Classifier, so without training it shouldn't work well.

- Q2: Do results match your expectation(1 sentence)? Why or why not (1-2 sentences)?

  A2 (2 sentences):

  Yes, the results match our expectations. Initially the test set on the baseline pre-trained BERT model had a very low accuracy as the Classifier hadn't been trained and after training

the fine-tuned model with cross entropy loss it was able to classify intents and we got a high final test accuracy of 86.583%.

- Q3: What could you do to further improve the performance (1 sentence)?

  A3 (1 sentence):

  In order to further improve the performance we can continue to fine-tune the hyper parameters, look more closely at how overfitting is affecting generalization, and also look into alternative advanced methods such as warm-up steps and re-initializing pre-trained layers.

- Q4: Which techniques did you choose and why? (1 sentence)?

  A4 (1 sentence):

  We chose warm-up steps and re-initializing pre-trained layers because they adjusted the learning rate which we thought might help with overfitting and re-initialized weights to help get a more accurate abstraction of the data in the model to help with convergence.

- Q5: What do you expect for the results of the individual technique vs. the two techniques combined? (1 sentence)?

  A5 (1 sentence):

  I expect very minor improvements from the individual techniques, but with the both of them combined have a more significant improvement in accuracy.

- Q6: Do results match your expectation(1 sentence)? Why or why not (1-2 sentences)?

  A6 (2 sentences):

  The results somewhat match our expectations, that the individual improvements were very very minor between 0.001 0.002 improvement, but we were correct in guessing that the two of them together improved generalization greatly. We suspect that by re-initializing the weights and having a warm-up together, we were able to prevent some overfitting and get a better abstraction of the data in the top layer.

- Q7: What could you do to further improve the performance (1 sentence)?

  A7 (1 sentence):

  In order to further improve performance we could try stochastic weight averaging to modify the learning rate schedule and equally average the weights to see if it helps improve generalization.

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

  A8:

  SimCLR is a contrastive learning framework that can be run in a supervised an unsupervised manner. In its unsupervised form, SimCLR is run against a set of other sentences, one of which is the same as the input sentence. The other sentences have a random dropout appplied to their encoding, and so does the input sentence, making this network a bit more robust to overfitting. The supervised way of running SimCLR adds labels to sentence pairs, whether they express entailment, neutral or contradiction. It then uses a loss function that accounts for the difference in sentences and the difference in labels.

  SupCon is also a contrasting learning method, but this method uses some different data augmentation aside from dropout called 'mixup' and 'cutmix'. Image embeddings from the same class are 'pulled together' closer than image embeddings from other classes. The training consists of comparing multiple positive pairs of these labels and multiple negative pairs of these labels in a minibatch. This results in a different loss function that incorporates multiple positive and negative examples.

  SupCon and SimCLR share similarities in that they both utilize data augmentation and contrastive superised learning to train their models. However they differ in the loss functions used and the goals that they try to achieve. Supervised-Contrastive (SupCon) is using embedding to classify images with a final softmax layer, while SimCLR is evaluating the "correctness" of the vector representations of sentences using STS tasks.

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

  A9:

  SimCSE applies dropout in order to achieve minimal data augmentation. They utilize this during unsupervised learning. They pass two sentences into a pre-trained encoder were dropout is performed. Then two different embeddings of the original sentence are acquired.

SIMCSE should be able to predict that these two generated embeddings are "positives", or are the same as the original sentence. While other sentences taken from the same mini-batch should register as "negatives".

- Q10: If we do not fine-tune the model, how do the embeddings of the [CLS] tokens look like (1 sentence)?

  A10 (1 sentence):

  If the models' hyper-parameters are not fine tuned, we see that the embeddings are plotted in curved lines rather than clusters.

- Q11: What can be the differences between the the embeddings of the [CLS] tokens fine-tuned through cross-entropy, SupContrast, and SimCLR (1-2 sentences)?

  A11 (2 sentences):

  Between SupContrast loss and SimCLR loss training, we see that it takes SimCLR longer to converge the embeddings into clusters, possibly because SimCLR does not use labels to train the model. Through, cross-entropy we see that they begin to look similar to the SupContrast mappings loss after a longer training time.

- Q12: Do the results match your expectation(1 sentence)? Why or why not (1-2 sentences)?

  A12 (3 sentences):

  The differences in SimCLR and SupContrast match my expectations. With less refined information on the data points, it makes sense that the model would have a harder time converging on groups. The cross-entropy case is less expected, but it makes sense that the SupContrast loss would group embeddings quicker because the point of this loss function is to decrease the distance between other positive and negative example embeddings.

- Q13: What could you do to further improve the performance (1 sentence)?

  A13 (1 sentence):

  One way that we can improve the performance of these loss functions is to of course run training for longer and analyze our model for over-fitting, experiment with the umap library's hyper-parameters (n nearest neighbors, minimum distance) , and experiment further with higher dropout rate and lower temperature coefficients.

5 Authors' Contributions

5.1 Jason Stanley

On this PA I worked on implementing SupCon and Simclr with Elliot and helped debug several sections of the code base. I also read through all the resource papers and our group often talked about understanding the high level understanding of the project. I also dug into the SupCon Code base we referenced to ensure our implementation matched theirs. The group worked well together and discussing the topics together was a lot of fun, although a lot of work.

5.2 Michael Ma

I worked on part 3 of this PA to set up the baseline model of the network and helped tuning the hyperparameters to reach the expected accuracy. I read the papers for part 5 and helped with fixing the model and umap.

5.3 Elliot Lee

I read the SimCLR and Supervised Contrastive resource papers, helped implement those losses for task 5 with the help of Jason, helped implement Umap embedding visualization with the help of Anna and Michael, did hyper-parameter tuning for the contrastive learning and embedding visualization sections. Additionally I also helped write the corresponding sections of the report.

5.4 Annabella Macaluso

For this PA I contributed towards understanding the different custom fine-tuning strategies for improving upon our baseline BERT model. I helped implement the code for re-initializing pre-trained

layers. I did fine-tuning for Part 3 and custom training in Part 4 and helped with some training in Part 5. I also helped with general understanding of the PA by overlooking documentation, reading provided papers in the write-up and contributing towards writing the write-up. Everyone's great contributions made the PA go very smoothly.

References

[1] Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J. (2021) Dive into Deep Learning.

[2] Khosla, Teterwak, Wang, Sarna, Tian, Isola, Maschinot, Liu (2021) Supervised Contrastive Learning.

[3] Gao, Yao, Chen (2022) SimCSE: Simple Contrastive Learning of Sentence Embeddings.

[4] Chang (2021) Advanced Techniques for Fine-tuning Transformers

[5] Devlin, Chang, Lee, Toutanova (2019) BERTL: Pre-training of Deep Bidirectional Transformers for Language Understanding

[6] FitzGerald, Hench, Peris, Mackie, Rottmann, Sanchex, Nash, Urbach, Kakarala, Singh, Ranganath, Crist, Britan, Leeuwis Tur, Natarajan (2022) MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages

[7] FitzGerald, Ananthakrishnan, Arkoudas, Bernardi, Bhagia, Bovi, Cao, Chada, Chauhan, Chen, Dwarankanath, Dwivedi, Gojayev, Gopalakrishnan, Gueudre, Hakkani-Tur, Hamza, Hueser, Jose, Khan, Liu, Lu, Manzotti, Natarajan, Owczarzak, Oz, Palumbo, Peris, Prakash, Rawls, Rosenbaum, Shenoy, Soltan, Sridhar, Tan, Triefenbach, Wei, Yu, Zheng (2022) Alexa Teacher Model: Pretraining and Distilling Multi-Billion-Parameter Encoders for Natural Langauge Understanding Systems

[8] Rosenbaum, Soltan, Hamza, Versley, Bose (2022) LINGUIST: Language Model Instruction Tuning to Generate Annotated Utterances for Intent Classification and Slot Tagging