Experiment of Transformer Network with Amazon Massive Intent Dataset

Dongze Li
Department of Cognitive Science
University of California, San Diego
La Jolla, CA 92093
`dol073@ucsd.edu`

Xiaoyan He
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093
`x6he@ucsd.edu`

Yunfan Long
Halıcıoğlu Data Science Institute
University of California, San Diego
La Jolla, CA 92093
`yulong@ucsd.edu`

Abstract

Transformers have recently demonstrated cutting-edge performance in a variety of tasks. In this paper, we investigate and employ a pre-trained BERT model to categorize user intent. BERT will be adjusted as a baseline model initially, followed by the use of various training methods learned from a blog to create a unique model, and ultimately, the training of models with contrastive losses. For training our model, we'll use the Amazon Massive Intent dataset. This dataset includes 60 different user intent categories. With train/validation/test splits, Huggingface makes it simple to become familiar with the data samples and labels. For the classification tasks, the text serves as the model's input and the intent label serves as its output (with the exception of contrastive learning at Task 5). We use accuracy, namely, the correct predictions over total number of samples, as our evaluation metrics. In our baseline model, we employed pre-trained BERT model as our encoder and implemented a classifier which will provide the final classification result. Our best final result of classification task which achieves test accuracy around 88.9711% for test set. After custom fine-tuning, we improve the test accuracy to 90.0471% by reducing batch size to 16 and adding both warm up and LLRD to get this improvement more than 1%. Finally, we explored contrastive learning by changing loss function to SimCLR and SupContrast. We found that with SimCLR, we get a test accuracy of 61.829% as our best result while with SupContrast we got a similar result (test accuracy 88.1977%) as our baseline model. This is not surprising because unlike SupContrast, SimCLR is not supervised, which means it can be hard for it to get high accuracy for a classification task, given the real result is not directly used to force training.

1 Introduction

Voice user interfaces are increasingly prevalent in modern society, including chatbots, social robots, and Amazon Alexa. These systems need to be able to interpret the user's spoken utterance's intent in order to conduct effective communication. To effectively respond to or direct the conversation, intent classification involves analyzing a written or spoken input and figuring out which of several classes it matches. Building a variety of conversational technologies, such as chatbots, automated phone systems, and social robots, requires effective intent recognition. We build an intent classification model in this paper, and performance at each step of fine-tuning were evaluated.

For training our model, we'll use the Amazon Massive Intent dataset. This dataset includes 60 different user intent categories. For our BERT model, we'll make use of a terrific program called Hugging Face. Along with many other fantastic models and datasets, Hugging Face provides a ton of pre-trained BERT models for various purposes. In particular, we'll be utilizing the 110M parameter BERT Base Uncased model, which was pre-trained on both the BookCorpus (800M words) and English Wikipedia (2,500M words). With train/validation/test splits, Huggingface makes it simple to become familiar with the data samples and labels. For the classification tasks, the text serves as the model's input and the intent label serves as its output (with the exception of contrastive learning at Task 5).

We'll utilize BERT, a transformer-based model that has been pre-trained on a substantial portion of English data, for our intent recognition model. This pre-training teaches BERT to represent language very effectively, which allows it to be tailored for a particular task. Although beginning from random model weights can be used to train a whole model, we will use pre-trained weights for BERT and then perform extra training to modify it for our intent recognition task.

2 Related Works

We acknowledge the great help from our instructor Garrison Cottrell for briefly introduced and explained the model of Transformer[3] to us. We also thanks for the book ”Dive into Deep Learning” [9] which gives a lot of insight on model construction, and the ideas of how transformer map the sequence as input to learn in corresponding embedding space.

The data set we used in this experiment, MASSIVE[2], gives a lot of good data to help us to understand the advantages and the shortages of our model for fine tuning. Since MASSIVE is a parallel dataset, each utterance is provided in each of the 51 languages. Through cross-linguistic training on natural language understanding (NLU) tasks, this enables models to acquire shared representations of utterances with the same intentions, regardless of language. Additionally, it enables adaption to additional NLP tasks like automatic translation, multilingual paraphrasing, fresh linguistic evaluations of imperative morphologies, and more. Previously, lack of labeled data for training and testing, particularly data that is relaistic for a given task and natural for a given language, is one challenge in building massively mulyilingual NLU models. High naturalness often comes with expensive human vetting. However, Massive dataset provides invaluable resources for us to tackle this problem.

We also thanks for the SLURP and PyTorch Library[1][10] provides useful function and method that saved a huge amount of time.

Here, we also thanks the previous works that involves Transformers for providing us a lot of insights. In the paper “Attention is all you need”[4], they invented the transformer so that we have such a great tool at our disposal. We also went through some paper[6][7][8] that gives us more ideas about how to better tune the hyper-parameters of our model. We also thank Alexander and the Harvard NLP Group for sharing a wonderful blog post about the fundamental ideas of the Transformer.

3 Experiments

3.1 Baseline Model

3.1.1 Final Setting

Hyperparameters	Value
Number of epochs	10
Batch size	64
Learning rate	5e-5
Hidden units of classifier	512
Dropout rate	0.1
Optimizer	AdamW
Lr Scheduler	None
Loss function	Cross entropy

Table 1: Final experiment setting of baseline model

In our baseline model, we employed pre-trained BERT model as our encoder and implemented a classifier which will provide the final classification result. Also, the evaluation metric of our experiment is straight forward. We simply apply the following:

$$\text{Accuracy: percent correct predictions} = \frac{\text{total correct predictions}}{\text{total number of samples}}$$

For the baseline model, we put the input into our encoder, then took the last hidden state's [CLS] token as output of the encoder, put it to a dropout layer, then feed it into the classifier to get the final output.

Our final setting of the hyperparameters are listed as the table 1 above. It correspond to our best final result of classification task which achieves cross entropy loss around 0.464395 and accuracy around 88.9711% for test set.

3.1.2 Milestones

In the process of tuning our baseline model, we initially start with epoch 10, batch size 16, learning rate of 5e-4, hidden units as 512, and dropout rate as 0.1. Our expectation of this experiment setting is around 80% test accuracy. We got the test result as 0.702741 cross entropy loss and 84.7680% test accuracy. So the result is better than we expected.

We tried to find out if we can further improve the performance. We thought the learning rate might be too high, and set the learning rate as 5e-5 to see if lower learning rate can help the model to reach closer to the optimal point. It turned out that we have 0.652141 as loss and 87.8951% as accuracy. Then, we tried some other learning rates and adjusted number of epochs accordingly and found that the result given by 5e-5 is the best.

After that, we also thought that the batch size might limited our performance, because if we add more sample in a batch, the model can learn more at one time. Therefore, we tried other batch sizes while hold everything else constant. It turned out that batch size 64 gave the best performance. The loss is 0.464395 and the test accuracy is 88.9711%.

Then, we tried some other number for the hidden units. Because generally, the more the hidden units, the more complex the feature that can be learned by the network, but there is also the possibility that our number of hidden units is too high and causes overfit of the network. So we tried 256, 1024, 2048, and 4096 as the hidden size. We found really interesting here, because it turned out that 2048 hidden units had the same performance in test accuracy but the cross entropy loss is 0.477792 which higher than the model with 512 hidden units. We still decided to consider model with 512 hidden units as the best model because cross entropy loss tells more about does the model classify a sample to the correct class with a strong or weak definiteness. Therefore, we thought model with 512 hidden units a better general performance.

Finally, we tried some different dropout rates, but the drop out rate we initially used still performed best which gave the reported result (0.464395 as loss, 88.9711% as accuracy).

3.2 Custom Fine-tuning Model

3.2.1 Final Setting

Hyperparameters	Value
Number of epochs	10
Batch size	16
Learning rate	5e-5
Hidden units of classifier	512
Dropout rate	0.1
If LLRD is turned on	True
If warm up is turned on	True
Optimizer	AdamW
LLRD factor	0.95
Lr Scheduler	Linear
Warm up ratio	0.07
Loss function	Cross entropy

Table 2: Final experiment setting of custom tuning model

To achieve a better result, we tried to use some advanced fine-tuning techniques. We picked two strategies and applied them to our baseline model. The first strategy is layer-wise learning rate decay (LLRD), it applies higher learning rate to the layers close to the output of encoder and lower learning rate to the layers close to the input. It makes sense, because the layers close to the input are usually deal with more general and broad-based information, and layers close to the output are usually deal with more localized and specific information to our task.

Warm up is that we increase learning rate linearly from 0 to the initial learning rate we set as the hyperparameter in warm up steps, and then learning rate will start to decrease linearly to 0. We use warm up ratio times our training steps to get the warm up steps. The advantage of applying warm up is that reduce early training sample's effect on weight updating.

Our final experiment setting is listed above as table 2. With that setting, we can get test loss as 0.474755 and test accuracy as 90.04707%. Comparing to our best baseline model, we reduced batch size to 16 and turned on both warm up and LLRD to get more than 1% improvement in test accuracy.

3.2.2 Milestone

In the process of tuning custom model, we divided it as 3 parts (as test result of exp 3, 4, 5 listed in table 5). We first only turned LLRD on and warm up off, then turned warm up on and LLRD off, finally we turned both on.

For the first part (exp 3 in in table 5), we start with the final setting of baseline model (listed in table 1) and set learning rate decay factor as 0.9. By that setting, we get the test loss as 0.442607 and test accuracy as 89.2065%. It gave a performance that was better than we expected, then we first tried to tune the learning rate, but resulted that no other learning rate achieved a better performance.

Then, we tried to change the learning rate decay factor, we found that if we decrease the decay factor, the model will performed worse than before with test accuracy round 87% or 88%. We conclude the poor performance is that for each layer, the learning rate is exponentially decaying, since BERT model has 12 layers, with a decay factor lower than 0.9 (i.e. 0.8), the actual learning rate for the layer closest to the input is the learning rate which we set as hyperparameter times 0.8^{12} which is almost zero. By figuring out this fact, we then tried to test the combination of lower decay factor (i.e. 0.8, 0.85) with smaller learning rate and higher decay factor (close to 1) with larger learning rate. But none of them gave a better performance better than our initial setting. Lower decay factor with smaller learning rate suffered from underfitting, even we trained the model over more epochs, it didn't achieve a better performance. Higher decay rate with larger learning rate gave a performance similar to our baseline model (test accuracy as 89.1056%). So we conclude from this part of experiment that setting decay factor around 0.9 is most sufficient.

We assumed that number of hidden units and dropout rate should be same as baseline model since LLRD should have really small effect on them. To verify our assumption, we experiment with some different number of hidden units (i.e. 256, 1024, 2048) and dropout rate (i.e. 0.3, 0.5, 0.7), and it didn't come out with a better performance (the closest one is accuracy as 89.0720% provide by 2048 hidden units) which verified our assumption. Therefore, we finalize our final setting as decay factor as 0.9 and else as same as final setting of baseline model with test loss 0.442607 and test accuracy 89.2065%.

For the second part (exp 4), we start with warm up ratio 0.1 and the final setting of baseline model. It gave the result as test loss 0.5100 and test accuracy as 87.5925%. The initial result was lower than what we expected. We inferred that it's due to the batch size. Because as batch size increase, the train steps will decrease, and the number of training step will decrease since it's proportional to the training steps in our setting. Then the training process will be warmed up in fewer steps which indicates the learning rate will increase from 0 to what we set in a steeper rate. Thus we tried to decrease the batch size and increase the number of training steps to flatten and smooth the increase rate of learning rate. By figuring out this issue, we get a better performance with batch size 16 and all else constant. The test loss is 0.494092 and test accuracy as 89.8453%.

This result reached our expectation of warm up strategies. But we still wanted to improve our model's performance. We thought if we lower warm up ratio, it may have a better performance. Thus, we changed warm up ratio to 0.08. As we expected, this model had better performance with test loss 0.485392 and test accuracy 89.9126%.

Then we tried other combinations of batch size, learning rate, and warm up ratio. But none of them had a better performance. We also tried to change the dropout rate and hidden units, and still none of other dropout rates and number of hidden units had a better performance than 0.1 dropout rate and 512 hidden units as we set in the baseline model. Therefore, the final setting of exp 4 is 0.08 warm up ratio, 16 batch size and others same as final baseline model with test loss 0.485392 and test accuracy 89.9126%.

Finally, for the experiment with both strategies turned on (exp 5), we started with final setting of baseline model and decay factor as 0.9, warm up ratio 0.1. It had test loss as 0.541294 and test accuracy as 86.8863%. By the experience of exp 4, we figured out the performance was limited by the batch size. As we changed batch size to 16, we got test loss 0.464522 and test accuracy as 89.6436%. Then we lower the warm up ratio to 0.08 as exp 4. The model had the same test accuracy but a slightly lower test loss 0.463917. Thus we saved this setting and continue our tuning.

We thought our performance might be limited by current decay factor and warm up ratio, because both strategies made the learning rate really small at the start point of training. Although these strategies can reduce early training sample's effect on weight updating and preserve the bottom pre-trained layers of BERTmodel, combining both strategies would make our model had very slow update in first few training steps. So, we increased the decay factor to 0.95, it resulted a better performance as test loss 0.478344 and test accuracy 89.7781%. Then, we changed warm up ratio to 0.07 which is our final setting, and get the best result test loss as 0.474755 and test accuracy as 90.04707%.

3.3 Contrastive Learning Model

3.3.1 Final Setting

Hyperparameters	Value
Number of epochs	30
Batch size	16
Learning rate	5e-5
Hidden units of classifier	512
Dropout rate	0.1
If warm up is turned on	True
Optimizer	AdamW
Lr Scheduler	Linear
Warm up ratio	0.07
Loss function	SupContrast
Temperature	0.07

Table 3: Final experiment setting of contrastive learning model

In this experiment, we replace the loss funtion (Cross entropy) with contrastive learning function (SupContrast and SimCLR). Since we cannot directly compare the losses produced by SupContrast, SimCLR, and Cross entropy, we have to develop some other way to compare the performances. We decided to plot the embedding of BERT encoder of first 10 classes in test set and evaluate the performance of contrastive learning primarily based on the plot, and we added a classifier trained with Cross entropy (with BERT freeze when training the classifier) to the BERT encoder that trained with contrastive learning to get the accuracy of classification and use the accuracy as the compliment.

With this final setting, our model can get a test accuracy as 88.1977%, with the plot of embedding as figure 4 shows. Also, we found an overall better performance of SupContrast loss function than SimCLR in the task of classification.

3.3.2 Milestones

During this experiment, we fixed the hidden units of classifier as 512, since the major task is developing the understanding of contrastive learning of BERT encoder but not tuning the hyperparameters of classifier. We first started with the setting of batch size 64, learning rate 5e-5, dropout rate 0.1, temperature 0.07, and warm up ratio 0.07 with SimCLR function. We got the plot embedding as figure 1 below with classification accuracy 49.6301% on test set.

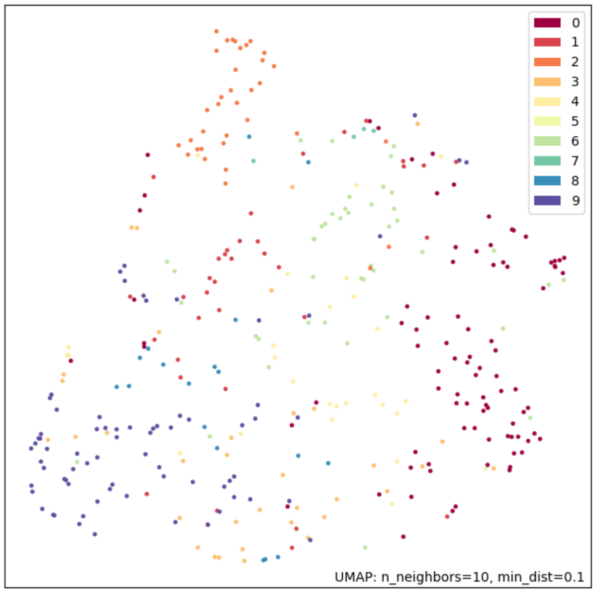

Figure 1: Plot of embedding of SimCLR loss function with batch size 64, lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07, trained with 30 epochs

We considered about the logic behind contrastive learning, and suggested that the batch size might limit our performance. Because for SimCLR, we pass the same sentence twice to the encoder in order to get two different embeddings by applying dropout, then consider these two embeddings as the positive pair, and take other sentences in the same mini-batch as negatives. If we don't have large enough batch size, then we can't learn enough negatives from a mini-batch. But we also thought that compare to our dataset, current batch size might be too large. Thus, we tested different batch sizes next.

We first increase the batch size and run 10 epochs to get an idea of the model performance. Increasing the batch size from 64 to 128, we get a test accuracy of 48.554%. It takes only 10 epochs to achieve the test accuracy almost as high as that of batch size 64 after 30 epochs. We decided to increase the batch size again to see whether a further improvement is possible. When we increased the batch size to 256 and again run 10 epochs, it turns out to have accuracy as low as 28.581%. Therefore, our next step is to increase the epochs for the batch size 128.

However, after we run 20 epochs with batch size 128, we found the accuracy only achieved 51.412%, which is not much improvements from the baseline model. It is clear that bigger batch size doesn't work as we expected. We believe that for our small and somewhat redundant dataset, it is better to use stochastic gradient descent with a smaller minibatch. We decided to try smaller batch size to see whether our training procedure would be more effective with a smaller batch size.

We then start to explore whether reducing the batch size from 64 to 16 would help further increase the accuracy. It turns out that the batch size 16 worked exceedingly well. After running 20 epochs, it achieving a test accuracy of 61.829%. We got the plot embedding as figure 2 below with classification accuracy 61.829% on test set.

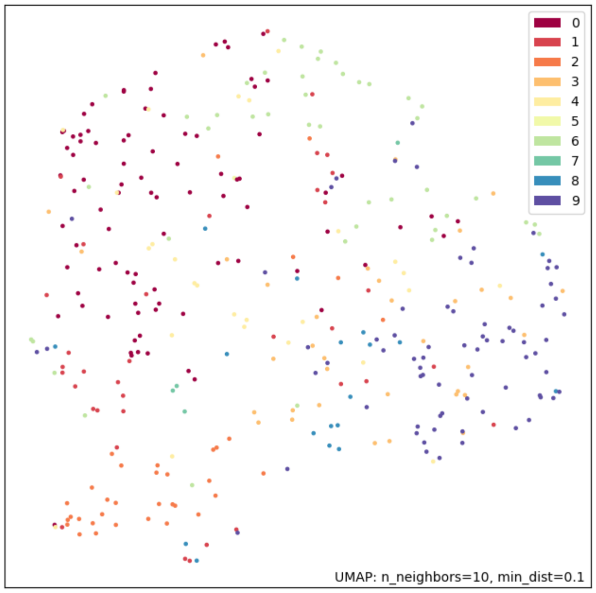

Figure 2: Plot of embedding of SimCLR loss function with batch size 16, lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07, trained with 20 epochs

We then try to modify the temperature of the model. Since the temperature of our model is already low (0.07), we want to see whether a higher temperature makes our result better. We first experiment with a higher accuracy of 0.2, which yields very poor result with test accuracy of 9.394% after 15 epochs. What's more, it achieved 18.888% after 2 epoch but soon over-fitted and the accuracy drop continuously. Then we adjust the temperature to 0.1, which is only a slight change compare to the original temperature. This time the accuracy improved but ended up at 51.412%, which is about 10 percent apart from our model with temperature 0.07. Therefore, we decided to fix the temperature at 0.07.

Since the training set can be somehow redundant, we want our model to be able to generalize better on unseen examples. Therefore, we decided to adjust the drop-out ratio. This turns out to be less effective than we expected, which gets a test accuracy of 56.758% after 20 epochs, which is 5.071% lower than our batch size 16 model with default drop out rate as 0.1.

Therefore, we decided to use the batch size 16 SimCLR model with all other parameters being default (lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07, trained with 20 epochs).

After that we changed the loss function to SupContrast, we start with batch size 64, lr 5e-5, dropout rate 0.1, temperature 0.07 and warm up ratio 0.07. After we trained the model through 20 epochs, we get the accuracy as 87.6597% and the plot of embedding is shown as figure 3 below.

As we can see, comparing to the embeddings of SimCLR, the embeddings of SupContrast forms clusters in the plot below. We inferred the reason causes such difference is that SupContrast is supervised contrastive learning. The model trained with SimCLR can only distinguish between samples, while the model trained with SupContrast can distinguish between classes. Thus, the plot of embedding of SimCLR looks more discrete and the plot of embedding of SupContrast looks like clusters.

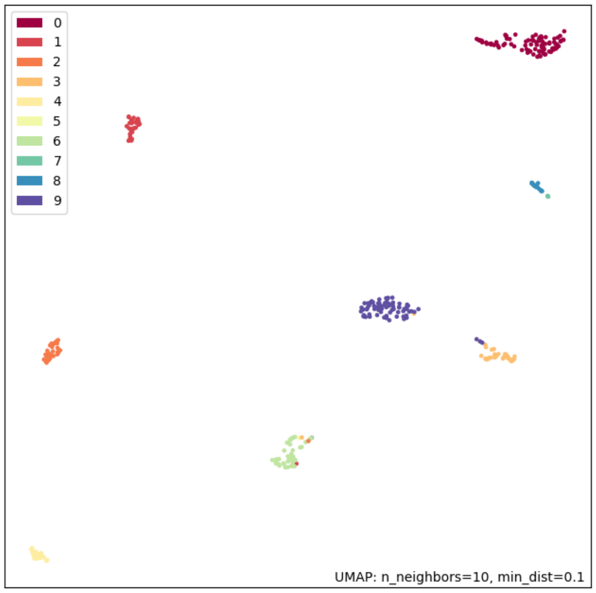

Figure 3: Plot of embedding of SupCon loss function with batch size 64, lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07, trained with 20 epochs

Then by the experience of tuning SimCLR, we still change the batch size first. The performance of batch size 128 is 84.7680% test accuracy, and the performance of batch size 32 is 87.0881% accuracy. As we decrease the batch size to 16, the performance increase to 88.1977% accuracy, and the plot of embedding is shown as figure 4 below. We can see here, each elements of the cluster become more compact.

Then we change the temperature of the model. However, increase the temperature worse the performance like what happened in SimCLR to around 87% accuracy. Just like what we did when tuning SimCLR, we then changed the dropout rate, but with dropout rate 0.2, 0.3, and 0.5, then test accuracy is still around 87%. Since there is no further improvement in the performance, we therefore set batch size 16, lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07 as our final setting of this section.

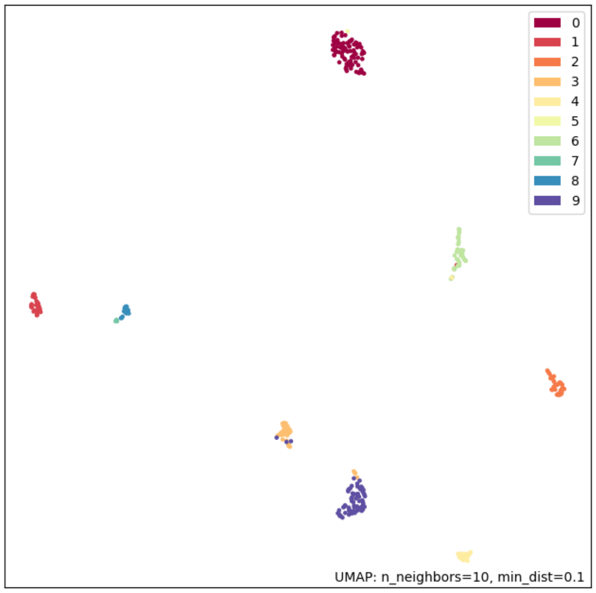

Figure 4: Plot of embedding of SupCon loss function with batch size 16, lr 5e-5, dropout rate 0.1, temperature 0.07, warm up 0.07, trained with 30 epochs

4 Discussion

During the process of tuning the model, we also tried to come up and answer some discussion question to enhance our understanding of the process of fine-tuning.

4.1 Baseline Model

Question 1: If we do not fine-tune the model, what is your expected test accuracy (1 sentence)? Why (1 sentence)?
Answer for question 1: (2 sentences) We expect an extremely low test accuracy (less than 1%). Because although the model is pre-trained, it's not trained to do our specific experiment, the layers closer to the output of the encode are trained to detect specific features of pre-train task's dataset but not our dataset.

exp id	experiment name	loss	accuracy
1	Test set before fine-tuning	4.119877	0.2459%
2	Test set after fine-tuning	0.464395	88.9711%

Table 4: Test performance before fine-tuning and best performance after tuning baseline model

Question 2: Do results match your expectation(1 sentence)? Why or why not (1-2 sentences)?
Answer to question 2: (3 sentences) Both results matches our expectation. Because before we fine-tune the model, the layers closer to the output of encoder which process more specific information are not trained to detect the feature of our dataset, although the layers closer to the input are pre-trained to be able to deal with more general information. After we fine-tuning and

training the model with our training set, our encoder is able to detect the specific information of our dataset, thus we expect it will have a high performance.

Question 3: What could you do to further improve the performance (1 sentence)?
Answer for question 3: (1 sentence) We think we can do grid search to find if there is some other combination of hyperparameters can achieve better result or employ some advanced fine-tuning strategies to fine-tune the model.
Other comments for question 3: The answer above are some possible ways we come up with to improve the final result (the result of exp 2). The strategies and attempts of tuning hyperparameters we tried during the experiment are described in the milestone part of baseline model (part 3.1.2).

4.2 Custom Fine-tuning Model

Question 4: Which techniques did you choose and why (1 sentence)?
Answer for question 4: (1 sentence) We chose layer-wise learning rate decay and warm up steps, because we thought that each layer of the encoder process different kinds of information, applying different learning rate layer-wise may have a better performance and additionally adding warm up steps can reduce the primacy influence of the early training examples.
Other ideas of question 4: We first pick layer-wise learning rate decay because the reason we answered above, and we also considered that re-initialize some pre-trained layers has some overlapped function of LLRD, since both strategies concern about the different roles each layer takes in processing information. So we didn't choose to re-initialize some pre-trained layers.

Question 5: What do you expect for the results of the individual technique vs. the two techniques combined (1 sentence)?
Answer for question 5: (1 sentence) We expect that for separately layer-wise learning rate decay and warm up steps, both techniques will improve the accuracy around 89%, and combining both together will give accuracy above 90%.

exp id	experiment name	loss	accuracy
3	Test set with 1st technique	0.442607	89.2065%
4	Test set with 2nd technique	0.485392	89.9126%
5	Test set with both techniques	0.474755	90.04707%

Table 5: Test performance before fine-tuning and best performance after tuning custom model

Question 6: Do results match your expectation(1 sentence)? Why or why not (1-2 sentences)?
Answer for question 6: (3 sentences) All three finally experiments match our expectation. Because LLRD makes BERT's layers close to the input are not affected too much by our training sample and preserves its generalization while keep the layers close to output good at detecting specific information of our samples. Also warm up steps reduces the early training sample's effect on weight updating and makes the model not overfit too early.
Other comments on question 6: Although all final results match our expectation, the initial setting of those three experiments don't all give a result that reach our expectation. For exp 4, we initially start with baseline model and warm up ratio 0.1, and results test loss 0.5100 and test accuracy as 87.5925% which is lower than baseline model. We think it's due to large batch size which cause less training steps. Thus learning rate need to increase from zero to what we set in fewer steps and also then decrease to zero in fewer steps. After we fix that issue through adjusting batch size and warm up ratio, we get the performance we expected. Similar thing happened in exp 5. Initially, we have test loss as 0.541294 and test accuracy as 86.8863% which is much lower than we expected, with baseline model setting and decay factor 0.9, warm up ratio 0.1. We also think with that setting, the learning rate is really small at the first few training steps, thus the model respond very slow to the samples in first few training steps. And we reach the result we expected by adjusting decay factor and warm up ratio.

Question 7: What could you do to further improve the performance (1 sentence)?
Answer for question 7: (1 sentence) We can try some other optimizer or other learning rate scheduler, do a grid search, and also use stochastic weight averaging strategy and try to combine it with LLRD and warm up steps to see if it can achieve a better performance.
Other comment for question 7: The answer above are some possible ways we come up with to improve the final result (the result of exp 5). The strategies and attempts of tuning hyperparameters we tried during the experiment are described in the milestone part of custom fine-tuning model (part 3.2.2).

4.3 Contrastive Learning Model

Question 8: Compare the SimCLR with SupContrast. What are the similarities and differences?
Answer for question 8: Both of them are contrastive learning which uses data augmentation to achieve contrasting samples against each other to learn the common attributes between data samples and features that set apart a data sample from another. They both need to define a metric that measures the distance between the features of data. The major difference between them is that SimCLR is self-supervised contrastive learning and SupConstrast is supervised contrastive learning. So SimCLR doesn't necessarily include the labels of classes, and SupConstrast need to include the labels. By SimCLR, we want to push the features from different samples far away from each other and pull the features from same sample close to each other. By SupConstrast, we want to push the features that are from different classes far away from each other and pull the features from same class as close as possible. Because SimCLR doesn't the use the label, so it may not be able to learn the correlation between features from different samples that are in the same class. And SupContrast is able to achieve that since it uses label just as figure 5 shows below.

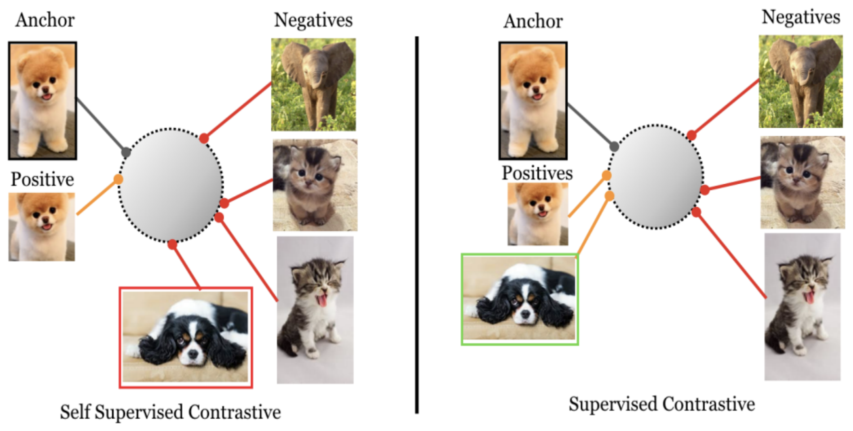

Figure 5: Different learning outcome of self-supervised and supervised learning

Question 9: How does SimCSE apply dropout to achieve data augmentation for NLP tasks?
Answer for question 9: In SimCSE paper, given a set of sentences, they pass the same sentence twice to the encoder in order to get two different embeddings by applying dropout. Then they use these two embeddings as the positive pair for training, and take other sentences in the same set as negatives.

Question 10: If we do not fine-tune the model, how do the embeddings of the [CLS] tokens look like (1 sentence)?
Answer for question 10: (1 sentence) If we don't fine-tune the model, we expect that the embeddings will be randomly distributed just as the reason in question 1.

Question 11: What can be the differences between the the embeddings of the [CLS] tokens fine-tuned through cross-entropy, SupContrast, and SimCLR (1-2 sentences)?
Answer for question 11: (2 sentence) We expect that the plot of cross-entropy and SupContrast will be like some clusters, but we think the clusters in plot of cross-entropy are closer to the center while cluster in plot of SupContrast are closer to the edge. For the plot of SimCLR, we don't think each class of embeddings will form a cluster.
Other comments about question 11: We make such assumption because SupContrast will make the embeddings from different classes as far as possible. So each cluster of embedding will be more likely placed on the edge of the plot. While cross-entropy don't do the contrast part, we expect those clusters will not be too far apart from each other. However since we feed the embeddings to classifier when using cross-entropy, it's still supervise learning, so we expect embeddings will still form clusters by classes. For SimCLR, since it doesn't use label during training, it only learn about the features of individual sample, then the embeddings will not be grouped by classes. Thus, we expect embeddings will be more dispersed. However, we think although it doesn't form clusters, the embeddings belong to the same group will still be closer than those belong to different groups because samples in same class are more like to have similar feature.

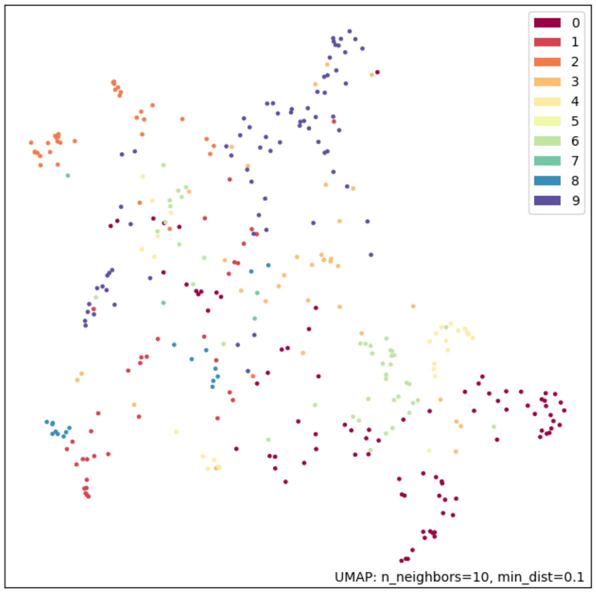

Figure 6: Plot of embedding of the model without fine-tuning

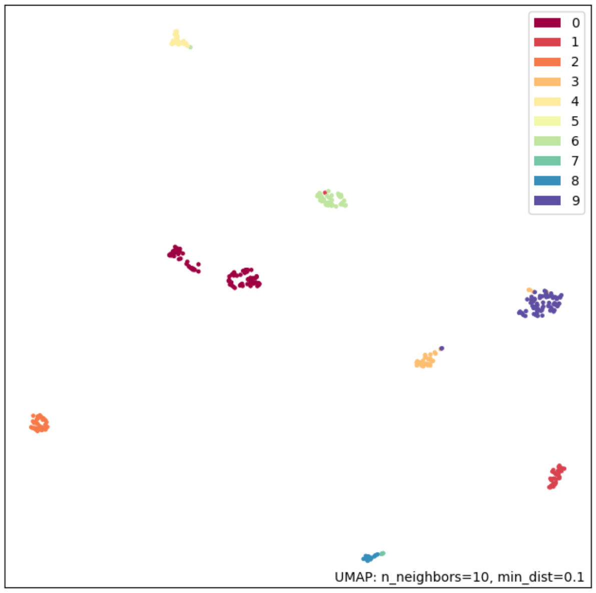

Figure 7: Plot of embedding of baseline model with final setting

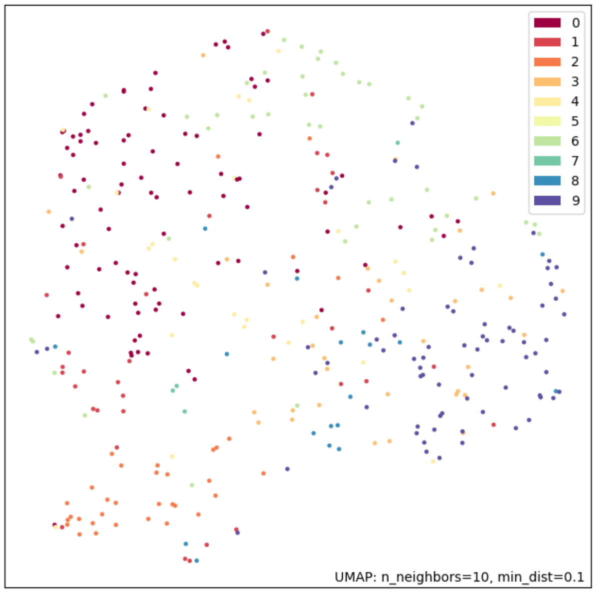

Figure 8: Plot of embedding of SimCLR loss function with final setting for SimCLR

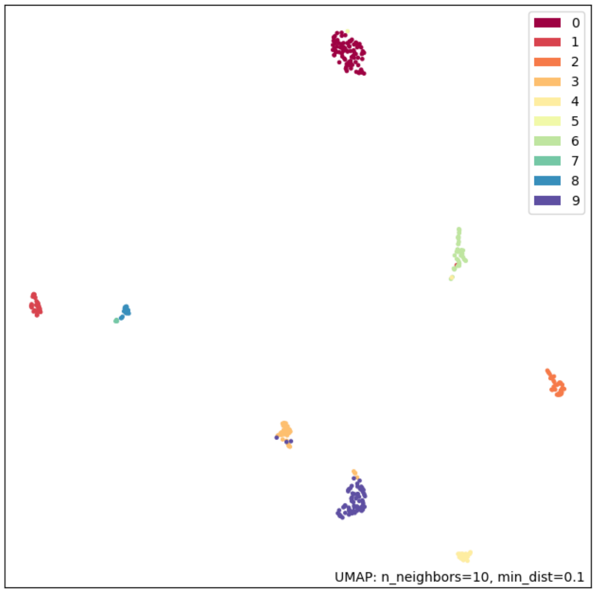

Figure 9: Plot of embedding of SupCon loss function with final setting for SupCon

Question 12: Do the results match your expectation(1 sentence)? Why or why not (1-2 sentences)?
Answer for question 12: (3 sentences) The result matches our expectation because we expect cross entropy to correctly classify but with no spatial mapping for embedding since it doesn't contrast the features through data augmentation, supcontrast achieves the mapping with spatial mapping and high test accuracy, and SimCLR has medium accuracy of classification with no cluster in the plot since it doesn't learning the correlation of features in same class. It's also worth noting that after chaging the batch size, we find smaller batch work better (which is opposite to our expectation), which makes sense because our dataset is relatively small and somehow redundant. By partitioning the data into smaller minibatches, we avoid redundancy and hence improve generalization.
Other comment for question 12: It's worth to mention that comparing to figure 6 which is the plot of embedding of the model without fine-tuning, we can find that the embedding of SimCLR (figure 9) is not randomly distributed although it's dispersed. We can still find the elements belong to the same group are closer to each other (i.e. most red points are at upper left part) than those not belong to same class. We think it's because the samples from the same class are more like to have some similar features, thus by the structure of contrastive learning, they will have closer distance. By comparing figure 7 and 9, we can find the plots of Cross entropy and SupContrast look similar, one major differece is that what we mention above in question 11 and clusters in the plot of SupContrast looks more compact, especially the cluster of class label 0.

Question 13: What could you do to further improve the performance (1 sentence)?
Answer for question 13: Since our training set is relatively small, we can augment the training data to enlarge our training set to get better training performance, and also explore other possible better hyper-parameter combination to improve performance.

5 Contribution

Xiaoyan He, Dongze Li, and Yunfan Long equally distributed all the work in modifying the given Bert Architechture, working on Supcontrast and SimCLR network, and carrying out experiments, tuning hyperparameter, and debugging. We took turns that one wrote a test and the other made the test pass in the process of completing this project. During this process, we met together in library, took turn to serve as the primary programmer when working on Supcontrast and SimCLR network, transfering the learning from pre-trained Bert, and conduct all the experiments together. During hyperparameter tuning, we all ran same amount of load during the two weeks.

Beyond the equally distributed when working on and fine-tuning Supcontrast and SimCLR network, every of us undertake some extra work in different areas:

Dongze Li: hyperparameter tuning of the learning rate and number of epochs for Supcontrast Model and implementing code for loss calculation.

Xiaoyan He: hyperparameter tuning of the batch size and hidden units for Supcontrast Model and implementing code for visualization.

Yunfan Long: hyperparameter tuning of the batch size, temperature, and drop-out for SimCLR and implementing code for management/deployment of experiment result.

Note that we collaborate and help each other with debugging and testing in every aspect of this project along the way.

All of us contributed equally on writing the report.

6 Reference

[1] Emanuele Bastianelli, Andrea Vanzo, Pawel Swietojanski, and Verena Rieser. 2020. *SLURP: A Spoken Language Understanding Resource Package.* In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 7252–7262, Online. Association for Computational Linguistics.

[2] FitzGerald, Jack G. M. et al. *"MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages."* ArXiv abs/2204.08582 (2022): n. pag.

[3] G. Cottrell. *Lecture 10: Transformer Networks*, lecture notes, Department of Computer Science, University of California San Diego, Nov 2022.

[4] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia, *"Attention is all you need"*, 2017.

[5] Alexander. Rush. *"The Annotated Transformer"*, Harvard NLP group, 2018.

[6] T. Zhang, F. Wu, A. Katiyar, K. Weinberger, and Y. Artzi, *Revisiting Few-sample BERT Fine-tuning*, 2021

[7] C. Sun, X. Qiu, Y. Xu, and X. Huang, *How to Fine-Tune BERT for Text Classification?* 2020

[8] J. Howard and S. Ruder, *Universal Language Model Fine-tuning for Text Classification*, 2018

[9] A. Zhang, Z.C. Lipton, M. Li, A.J. Smola. Dive into Deep Learning. *arXiv:2106.11342 [cs.LG], Jul 2022*

[10] A. Paszke. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019.