User Intent Classification with Transformers and Contrastive Learning

Lian Gan
University of California San Diego
lgan@ucsd.edu

Jessica Song
University of California San Diego
jwsong@ucsd.edu

Zhirui (Raymond) Xia
University of California San Diego
z4xia@ucsd.edu

Boqi (Bobby) Zhu
University of California San Diego
b2zhu@ucsd.edu

Abstract

In this study, we explore the application of transformers, specifically a fine-tuned BERT model, and contrastive learning techniques, including SimCLR and SupCon, to the task of user intent classification using the Amazon Massive Intent dataset. Our approach involves three key phases: establishing a baseline with a fine-tuned BERT model, implementing custom fine-tuning strategies such as Layer-wise Learning Rate Decay (LLRD) and Stochastic Weighted Average (SWA), and applying contrastive learning methods to enhance representation learning. Our baseline model achieved an accuracy of 88.4%. With further experiments, our findings showed that our fine-tuned BERT model achieved a commendable accuracy of 88.2%, the incorporation of contrastive learning techniques yielded slightly lower accuracies, with 80% for embedding SimCLR Loss and 88.07% for embedding with SupCon loss. Our results highlight the challenges and potential of leveraging contrastive learning in the NLP domain, especially for tasks centered around intent classification.

1 Introduction

The automatic classification of user intent from textual data is a critical task in natural language processing (NLP), with wide-ranging applications in chatbots, virtual assistants, and customer service automation. Recent advancements in deep learning, particularly the advent of transformer models like BERT, have significantly pushed the boundaries of what's possible in understanding and classifying textual data. In this study, we delve into the domain of user intent classification by leveraging a fine-tuned BERT model as our baseline, given its proven efficacy in capturing contextual representations of text. Specifically, we experiment with LLRD and SWA, examining the effects of fine-tuning learning rate change throughout the training process and utilizing weighted average of gradients.

However, the quest for improved performance and more nuanced understanding of text has led to the exploration of contrastive learning techniques, such as SimCLR and SupCon, which have shown promise in the field of computer vision and are increasingly being adapted for NLP tasks. Contrastive learning focuses on learning representations by contrasting positive pairs against negative pairs, which can potentially lead to more robust feature extraction in the context of user intent classification.

Our work is inspired by three papers that seeks to combine the representational power of transformers with the nuanced learning capabilities of contrastive learning methods with their newly designed loss function. In [1], we study the fundamental idea of contrastive learning. Through [3], we examine the power of supervised contrastive learning and implement it in our model. Ultimately, we studied [2] to learn about an upgrade to both Chen et al's SimCLR and Khosla et al's SupCon Loss – SimCSE, one that incorporates both supervised and unsupervised contrastive learning. In this paper, we extend these concepts to the realm of NLP, specifically targeting the challenging task of classifying a wide array of user intents in the Amazon Massive Intent dataset. Our objective is not only to benchmark the performance of these advanced techniques against a strong baseline but also to uncover insights that could guide future research in the fusion of transformers and contrastive learning for NLP applications.

2 Related Work

In this paper, we made references to three papers. In the following section, I will briefly introduce each of them.

Chen et al's "A Simple Framework for Contrastive Learning of Visual Representations" introduces the idea of SimCLR Loss, a streamlined framework for contrastive learning of visual representations. The paper shows that data augmentation plays a critical role in creating effective predictive tasks, and demonstrates how a learnable nonlinear transformation prior to the contrastive loss can significantly enhance the quality of learned representations, specifically on image datasets. Moreover, it presents findings that larger batch sizes and extended training durations yield substantial benefits in contrastive learning compared to traditional supervised methods. In the end, utilizing SimCLR led to notable advancements over prior self-supervised and semi-supervised learning benchmarks on ImageNet.

In Khosla et al's work, "Supervised Contrastive Learning," the authors extend batch contrastive learning approaches to a fully-supervised setting to enhance the training of deep image models. In contrast to SimCLR, SupCon incorporates label information to define positive and negative pairs more explicitly. In this framework, positive pairs are not only different augmentations of the same input but also different inputs that share the same class label. This means that all samples belonging to the same class are pulled closer together in the embedding space. Simultaneously, samples from different classes (negative pairs) are pushed apart, leading to well-separated clusters in the embedding space. By leveraging label information, the proposed supervised contrastive (SupCon) loss effectively clusters embeddings of the same class closer together, demonstrating superior performance on the ImageNet dataset. Additionally, the SupCon loss contributes to model robustness and stability across different hyperparameters, offering a simple yet effective approach for enhancing deep learning models, showing its advantages as a contrastive learning loss.

Both methods mentioned above were designed to serve image datasets. However, in this paper, we wish to apply contrastive learning to text classification. Therefore, we study the design of SimCSE, a contrastive learning loss on NLP tasks. We learned from its ideas and incorporated them in our discussion. SimCSE introduces a novel contrastive learning framework for enhancing sentence embeddings, employing a straightforward unsupervised method that uses standard dropout as a form of minimal data augmentation to prevent representation collapse by predicting the input sentence against itself. This approach is akin to a NLP version of SimCLR. Building on this, SimCSE integrates a supervised approach by leveraging annotated "entailment" and "contradiction" pairs from natural language inference datasets as positive and hard negative pairs, respectively, within the contrastive learning objective. This method achieves significant improvements in semantic textual similarity tasks, with the unsupervised and supervised models outperforming previous benchmarks. Furthermore, SimCSE utilizes regularization of the anisotropic space of pre-trained embeddings, making it more uniform and better aligning positive pairs when supervised data is available.

3 Methods/Experiments

3.1 Baseline

3.1.1 Final Experiment Settings

- Batch Size: 64
- Learning Rate: 0.0001
- Hidden Dimension: 250
- Dropout Rate: 0.5

- Embedding Dimension: 768
- Adam Epsilon: 1×10^{-8}
- Number of Epochs: 10
- Maximum Length: 20

3.1.2 Optimizer and Loss Function

- Optimizer: Adam Optimizer
- Loss Function: Cross Entropy Loss

In this section, we describe the process of setting up the baseline model for our intent classification task. The process is as follows:

3.1.3 Best Performance Metrics

- Test Accuracy: 0.884
- Average Test Loss: 0.963
- Validation Accuracy: 0.885
- Average Validation Loss: 0.968

A pre-trained BERT model "bert-base-uncased" will be fine-tuned on the specified dataset. The BERT model is leveraged for its powerful language understanding capabilities, which is crucial for the intent classification task.

The `baseline_train` function within `main.py` is utilized to train the baseline model. This function orchestrates the training process by setting up the data, model, and training loop.

The `IntentModel` class from `model.py` is used to load the pre-trained BERT model through the `BertModel.from_pretrained(...)` method. This pre-trained BERT model serves as the encoder in our setup, specifically utilizing the `bert-base-uncased` variant.

After the input is processed by the encoder, the last hidden state's `[CLS]` token is extracted. This token is passed through a dropout layer, configured with a predefined dropout rate. The resulting output is then fed into a classifier. The `[CLS]` token is chosen as the sentence embedding because it represents the aggregate understanding of the input sentence, which is beneficial for classification tasks.

3.2 Custom Model

3.2.1 Final Experiment Settings

LLRD technique:

- Batch Size: 16
- Learning Rate: 0.0001
- Hidden Dimension: 300
- Dropout Rate: 0.6
- Embedding Dimension: 768
- Adam Epsilon: 1×10^{-8}
- Number of Epochs: 10
- Maximum Length: 20
- lr-mult:0.83

SWA technique:

- Batch Size: 16
- Learning Rate: 0.0001

- Hidden Dimension: 300
- Dropout Rate: 0.7
- Embedding Dimension: 768
- Adam Epsilon: 1×10^{-8}
- swa-lr: 1e-4
- swa-start: 1
- Number of Epochs: 10
- Maximum Length: 20

LLD and SWA technique:

- Batch Size: 16
- Learning Rate: 0.0001
- Hidden Dimension: 300
- Dropout Rate: 0.6
- Embedding Dimension: 768
- Adam Epsilon: 1×10^{-8}
- Number of Epochs: 10
- Maximum Length: 20
- swa-lr: 1e-4
- swa-start: 1
- lr-mult:0.83

3.2.2 Optimizer and Loss Function

LLRD technique:

- Optimizer: Adam Optimizer
- Scheduler: get-cosine-schedule-with-warmup
- Loss Function: Cross Entropy Loss

SWA technique:

- Optimizer: Adam Optimizer
- Scheduler: get-cosine-schedule-with-warmup (Before SWA)
- Scheduler: SWALR (After SWA)
- Loss Function: Cross Entropy Loss

LLRD and SWA technique:

- Optimizer: Adam Optimizer
- Scheduler: get-cosine-schedule-with-warmup (Before SWA)
- Scheduler: SWALR (After SWA)
- Loss Function: Cross Entropy Loss

3.2.3 Best Results

LLRD technique:

- Test Accuracy: 0.885
- Validation Accuracy: 0.890
- Average Validation Loss: 0.741

SWA technique:

- Test Accuracy: 0.867
- Validation Accuracy: 0.875
- Average Validation Loss: 0.620

LLD and SWA technique:

- Test Accuracy: 0.882
- Validation Accuracy: 0.888
- Average Validation Loss: 0.856

1st Technique:

Layer-wise Learning Rate Decay which is a technique designed to adjust the learning rate applied to the parameters of a neural network differently across various layers. This approach is predicated on the hypothesis that different layers of a neural network learn at different speeds and, more importantly, require varying levels of adjustment to their weights during the training process. The core idea is that layers closer to the input (shallower layers) may not need as much fine-tuning as layers closer to the output (deeper layers), since the initial layers often capture more general features while the deeper layers adapt to more specific features of the task.

In implementing this technique, the first step involves enumerating all parameters of the model and organizing them into a list. This list is then reversed, ensuring that parameters from deeper layers are positioned at the beginning of the list. A factor termed lr-mult (learning rate multiplier) is applied across different layers to modulate the learning rate. It's crucial to maintain consistency in the learning rate within the same layer, which necessitates using the names of parameters as a constraint. This ensures that adjustments in the learning rate only occur with a change in layers. Then, these new parameters are passed to Adam Optimizer to replace the default parameter.

By adopting this methodology, each layer is assigned a distinct learning rate, with deeper layers receiving a higher rate compared to shallower ones. This differentiation allows deeper layers to adapt more rapidly to the specifics of the task, while shallower layers undergo minimal changes, preserving their capability to capture general features of the data. This stratified approach to adjusting learning rates can lead to improved model performance by balancing the need for stability in early layers with the flexibility required in deeper layers for task-specific learning.

2nd Technique:

Stochastic Weight Averaging (SWA), a powerful method aimed at enhancing the generalization of deep learning models. SWA works by averaging the weights of the model across different points in the training phase, specifically targeting the later stages when the model is close to convergence. This approach is based on the observation that averaging the weights helps in finding flatter minima in the loss landscape, which are associated with better generalization properties compared to sharper minima.

To apply SWA in PyTorch, we begin by wrapping our custom model with the AveragedModel provided by the PyTorch library. This wrapper facilitates the recording and calculation of the average of the model's weights after the specified start point for SWA (referred to as swa-start). For our model, which converges relatively quickly, we decide to set swa-start to 1 epoch after a careful tuning process. This early start is chosen to maximize the period during which weight averaging can occur, based on the specific convergence behavior of our model. After reaching the swa-start epoch, the model switch the learning rate scheduler to SWALR (Stochastic Weight Averaging Learning Rate), which is designed to work in tandem with SWA by adjusting the learning rate in a manner that complements the weight averaging process. Post-swa-start, the weights of the model are periodically recorded for averaging. Concurrently, a constant learning rate 1e-4(denoted as swa-lr) is applied after careful tuning. This specific learning rate encourages further exploration of the weight space, ensuring that the averaged weights represent a broad and effective search of the loss landscape, rather than being confined to a narrow region. After completing the specified duration of SWA training, the model undergoes a final averaging step to consolidate the recorded weights into a single, averaged model. This model, enhanced by the SWA process, is expected to exhibit improved generalization on unseen data.

Combined Techniques:

We implement SWA and LLRD simultaneously. In this approach, the model applies LLRD from the start. This entails assigning different learning rates to various layers of the model, typically allocating lower learning rates to shallower layers and higher rates to deeper layers. After surpassing the swa-start point, the model transitions to SWA. Upon completing the designated SWA training epochs, we perform one final averaging of the weights to finalize the SWA-enhanced model.

3.3 Contrastive Learning

3.3.1 Final experimental settings

- Batch Size: 128 (SupCon) and 32 (SimCLR)
- Learning Rate: 0.00001 (SupCon) and 0.000005 (SimCLR)
- Classifier Learning Rate: 0.0001
- Hidden Dimension: 150
- Hidden Dimension in MLP: 768 (SupCon) and 512 (SimCLR)
- Dropout Rate: 0.2
- MLP Output Dimension: 128
- Adam Epsilon: 1×10^{-8}
- Number of Epochs: 15 (SupCon) and 5 (SimCLR)
- Number of Epochs for classifier: 50 (SupCon) and 05 (SimCLR)
- Maximum Length: 20
- Temperature: 0.7

3.3.2 Optimizer and Loss Function

- Optimizer: Adam Optimizer
- Loss Function: SupConLoss (for encoder and projection), Cross Entropy Loss (for Classifier)

3.3.3 Performance Metrics

- Test Accuracy: 0.88077 (SupCon) and 0.80061 (SimCLR)
- Validation Accuracy: 0.87864 (SupCon) and 0.805059(SimCLR)
- Average Validation Loss: 0.55282 (SupCon) and 0.73450 (SimCLR)

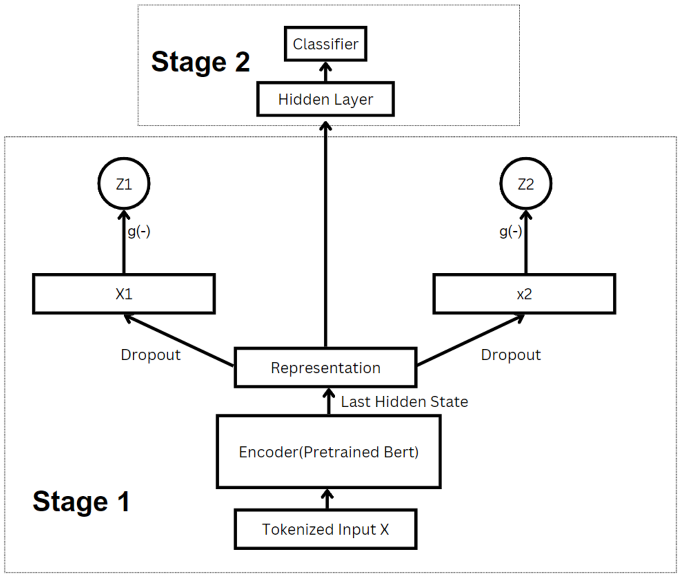

Figure 1: Contrastive Learning Architecture

In this section, we will talk about the method and implement of contrastive learning, SupCon and SimCLR. SupCon is supervised learning stand for supervised contrastive learning, and SimCLR is unsupervised learning stand for simple contrastive learning of representation. Figure 1 shows the architecture of contrastive learning model. We feed the input to the encoder, here we use the pretrained BERT model. Data augmentation is different in image task and NLP task. In this experiemnt, we use a dropout layer as a process of data augmentation. We feed the last hidden state from encoder into dropout layer twice. Thus, we now have two different representation from one input. g(-) is nonlinear multi-layer projection in our experiment, which gives us a normalized embedding z, with length 1. During training process, we need to minimize distance on positive pairs z, and maximize distance on negative pairs. Both Supcon and SimCLR use the same model architecture, the only different is their loss function.

In contrastive learning we use a new loss function, SupConLoss, referring to Khosla et al [3], in 7 and Chen et al [1] in 7. To summarize this loss function, to decrease the loss, we need to increase the cosine similarity between positive pairs and decreases cosine similarity between negative pairs. In SimCLR, there is only one positive pair. Since SimCLR is unsupervised learning, which we don't have the true label while training, there exist false negative pairs. Those false negative pairs will negatively affect our training process. While in SupCon, we have the true label while training, so all positive and negative pairs are true, and we will have more than one positive pairs. Thus, theoretically, SupCon will give better result than SimCLR.

There are two training stages. In the first training stage, we first train the encoder and embedding layer. The positive pairs and negatives are generated among each batch. For each batch, we calculate the SupConLoss and update the weights. In the second training stage, we freeze the encoder and then feed the representation from encoder to a classifier. In our experiment, our classifier include a hidden layer with 256 hidden units and then output the result for 60 categories, which used to calculate Softmax and cross-entropy loss. In this stage, we train the classifier layer only. And during the inference process, we also use this classifier to predict the result, hence we throw away embedding here.

In both SupCon and SimCLR, we use a multi layer non-linear projection (mlp). Our mlp map 768 to 128, with a hidden layer of size 768 (SupCon) and 512 (SimCLR) and ReLu activation function. We found that when throw a away normalization with high temperature, it gives better performance. Thus, in our experiment, we feed cls token into projection without normalization. Other hyperparameters, can be find above.

When fine tuning stage 1 (encoder + projection), we assume that models are similar if their hyperparameters are similar. Thus, at the beginning, we start with default hyperparameters and train the stage 1 for 10 epochs and test the accuracy. We save the model with torch.save(), which gives a pre-trained stage 1. Next, we change a hyperparameter with a small value, for example change temperature from 0.7 to 0.75, load the pre-trained model and train it for 3 epochs. Then we test the accuracy on new stage 1 again. If the accuracy increase, we either train it for more epochs or change the hyperparameter in the same direction. If accuracy decrease, we know that the previous modification in hyperparameters has negative effect. By redoing this procedure, we can roughly get the local best hyperparameters. Finally, we use that group of hyperparameters and a small learning rate, which is 0.00005 in our experiment, to find the best stage 1 model. This strategy significantly save a lot of time for hyper-tuning and it can also be applied on stage 2 (classifier).

4 Results

4.1 Baseline Model Performance

exp idx	exp	loss	accuracy
1	Test set before fine-tuning	2.32201554	0.44783465
2	Test set after fine-tuning	0.96259818	0.88417119

In the baseline model performance, we see that the fine-tuning was able to create a huge impact.

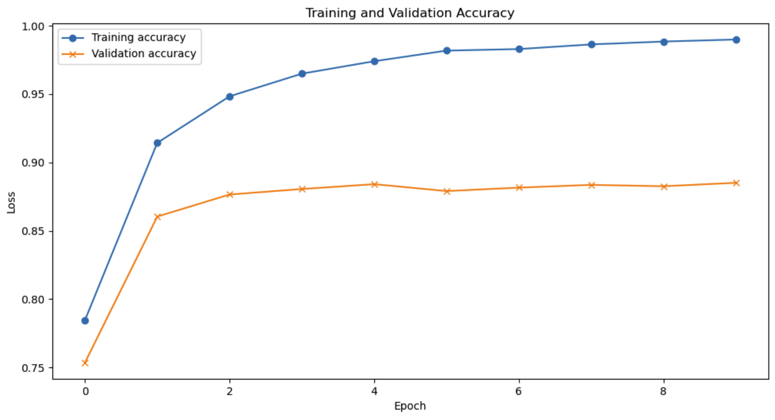

Figure 2: Baseline result after tuning: Accuracy VS. Epochs

4.2 Custome Fine-tuning Strategies

exp idx	exp	loss	accuracy
3	Test set with 1^{st} technique	0.78462094	0.88513513
4	Test set with 2^{nd} technique	0.68924987	0.86722972
5	Test set with 2 techniques	0.87979962	0.88209459

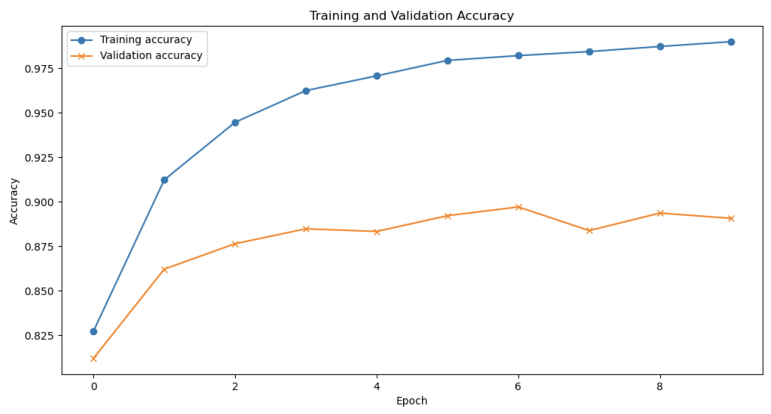

Figure 3: Layer-wise Learning Rate Decay: Accuracy vs epochs

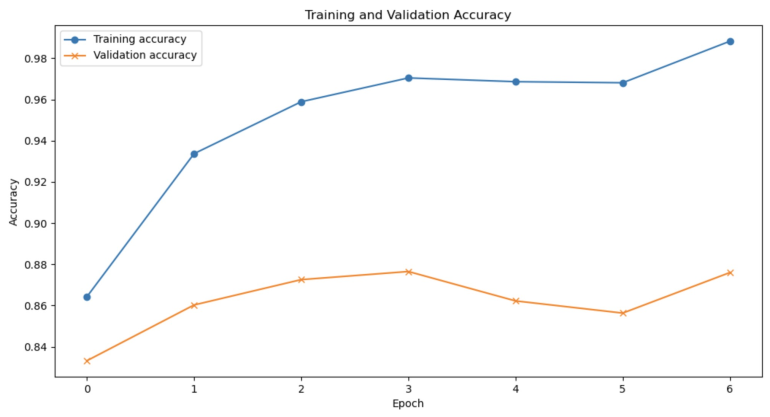

Figure 4: Stochastic Weight Averaging: Accuracy vs epochs

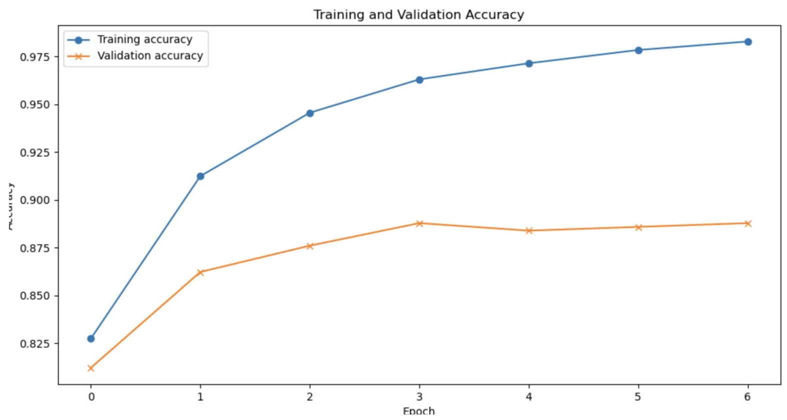

Figure 5: LLRD and SWA: Accuracy vs epochs

Here, we see that our custom finetuning strategies were able to achieve a pretty good accuracy around 0.88 as well. Our results showed that LLRD seems to be more effective for our specific dataset and network structure.

4.3 Contrastive Learning

exp idx	exp	loss	accuracy
5	Test set with SupContrast	0.50154748	0.88077445
6	Test set with SimCLR	0.74207450	0.80061141

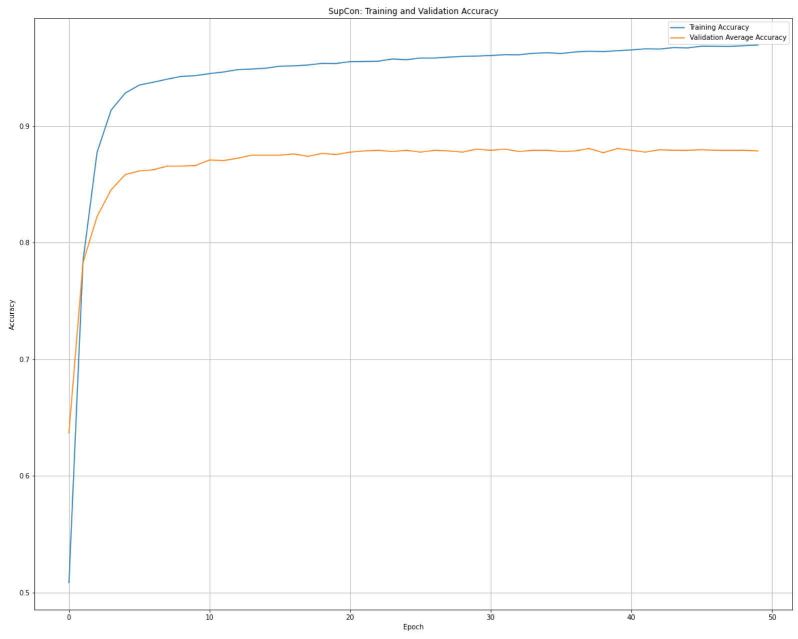

Figure 6: SupCon: Accuracy vs epochs

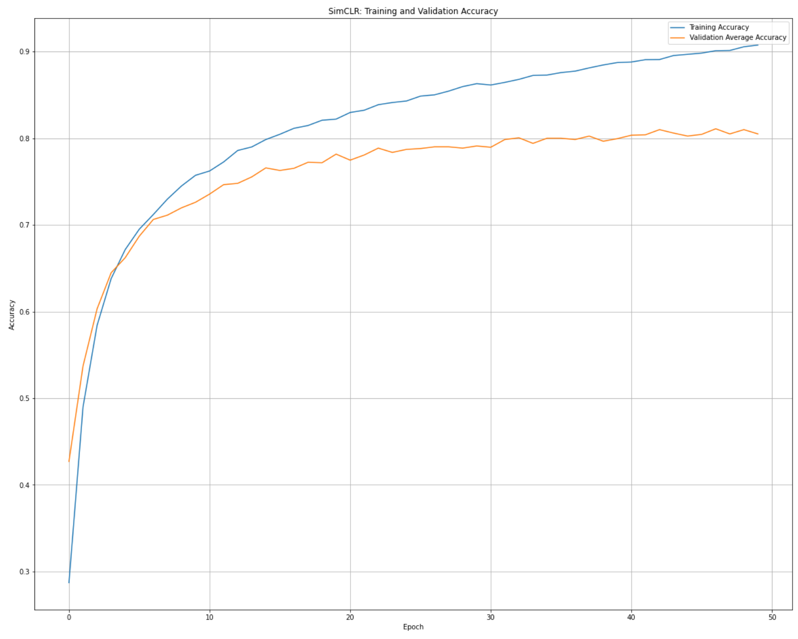

Figure 7: SimCLR: Accuracy vs epochs

Lastly, even though the plot shows a lot more training epochs, our model actually still converges to a relatively steady increase in terms of accuracy in around 5 epochs, with SupCon performing much better than SimCLR.

5 Discussion

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

A1: Without fine-tuning, the expected test accuracy of the model is very low. Pre-trained models like BERT are trained on general tasks and have not been adapted to our specific task at hand, user intent classification. Not knowing how well the pre-trained BERT fit to our specific domain and content of user intent, and how well it can distinguish the nuances between 60 classes, we can't expect a specific numerical test accuracy. We simply know that it's very likely to be a very poor performing accuracy.

Q2: Do results match your expectation(1 sentence)? Why or why not ? Our results showed that the model without finetuning had an accuracy of 0.45%, matching our expectations that we will get a low accuracy without fine-tuning. The model's parameters are not optimized for the nuances and specific features of the new dataset, leading to less-than-optimal performance.

Q3: What could you do to further improve the performance ?
After finetuning the model with selectively choosing cross entropy loss, Adam optimizer, and no learning rate scheduler, we can add further fine-tuning techniques to the model to adjust different aspects of the model. We can add techniques such as Layer-wise Learning Rate Decay (LLRD), warm-up steps, re-initializing pre-trained layers, stochastic weight averaging (SWA) and frequent evaluation.

Q4: Which techniques did you choose and why?

We chose Layer-wise Learning Rate Decay and Stochastic Weight Averaging.

Layer-wise Learning Rate Decay: BERT is a deeply pre-trained model with layers fine-tuned on different adapted tasks of understanding language. The first few layers capture the language-general features, and the deepest layers capture features more generic to tasks. These initial layers are generic to understand language and deeper layers are used to capture more task-specific features. In this case, we can give each layer a custom learning rate to fine-tune them. The deeper layer having a higher learning rate can adapt to specific tasks better while the shallow layer preserves the general language understanding.

Stochastic Weight Averaging: SWA can also help in reducing the risk of overfitting. By averaging weights over different epochs, it effectively uses a form of ensemble learning within a single model, which usually leads to a model that performs better on unseen data. Also, averaging multiple points in the parameter space, it would result in a smoother optimization landscape and a more robust final model. For a task like intent classification, where the model needs to generalize well across a wide range of user intents, SWA can be particularly beneficial.

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

For LLRD, we expect an enhanced test accuracy. By applying a higher learning rate to the upper layers and a lower one to the lower layers, LLRD should be able to adjust the pre-trained model to the specificities of the new task and obtain a minimal loss of the general language understanding embedded in the model. By fine-tuning the model in such a layer-wise manner, LLRD is expected to minimize the discrepancy between the learned representations and those necessary for the new task, leading to improved test accuracy.

For SWA, we expect to improve the generalization of our model. By averaging models across different points in the parameter space, SWA should be able to find a more robust solution that could perform better on unseen data. This strategy not only aids in finding a more robust solution that generalizes better to unseen data but also mitigates the risk of overfitting. SWA's model averaging effect smooths out the fluctuations in training, contributing to a more stable and reliable model performance across different datasets.

For both methods applied, the combined approach is anticipated to yield a model that not only fine-tunes more effectively across different layers but also achieves superior generalization capabilities.

Therefore, while combining SWA with LLRD may show a drop in the training accuracy compared to LLRD, the model with LLRD and SWA combined should have overall better generalizability and perhaps a better performance on the test set. The advantages of using both approaches are mentioned above, so the benefits of using both methods are expected here as well.

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

LLRD results matched our expectations, while SWA and the combined model didn't. In the next few paragraphs, we explain how our results have matched our expectations in the LLRD method, SWA method, and the two combined.

Layer-wise Learning Rate Decay (LLRD) has been observed to improve test accuracy from 0.876 to 0.885 in our case. This improvement aligns with expectations, as fine-tuning each layer's learning rate allows the model to maintain foundational language understanding at shallower layers while enhancing its ability to perform specific tasks at deeper layers effectively. A well-tuned learning rate multiplier (lr-mult) ensures the model is better optimized for unseen data, highlighting the effectiveness of LLRD in achieving nuanced model adjustments that directly contribute to improved generalization.

In contrast, applying Stochastic Weight Averaging (SWA) led to a slight decrease in test accuracy, from 0.876 to 0.867. This outcome surprised us, however, it might stem from a couple of key factors. First, for tasks involving complex data distributions, the process of averaging weights with SWA may not effectively capture the critical decision boundaries necessary for optimal performance. This is because the averaging process could blur the specificities learned during training, which are crucial for handling nuanced or intricate data patterns. Second, SWA introduces an additional layer of regularization by averaging weights across different states. Since our model already incorporates substantial regularization mechanisms, such as dropout, the additional regularization from SWA might result in over-regularization. This could overly simplify the model, hindering its ability to capture and act upon the complexities within the data distribution.

When combining LLRD with SWA, there was an observed increase in test accuracy, but not to the extent achieved with LLRD alone. This is also to our surprise as we thought that SWA would have generalized the model to perform better on the test set. The suboptimal performance observed when combining Layer-wise Learning Rate Decay (LLRD) with Stochastic Weight Averaging (SWA) can be attributed to conflicting optimization dynamics. LLRD fine-tunes each layer's learning rate to enhance task-specific performance, while SWA averages weights across different training states, aiming for generalization. This averaging may dilute the precise, layer-specific adjustments made by LLRD, potentially leading to a compromise in the model's ability to effectively capture and generalize from the nuanced patterns in the data.

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

We can try to do more data argumentation to increase the size of training data. For example, we can consider techniques like synonym replacement and sentence shuffling so that the model would have more data to train on, which generally lead to better performance. Also, we can inject noise to different phase of training stage. For example, we can add random noise to the input data. It could be adding, removing, or replacing words in a sentence randomly. In this case, model become more robust to variations in input data, improving its ability to understand and process text that it has not seen before. Also, we can add noise to the weights of the neural network during training. This approach simulates a form of regularization, pushing the model to find more stable solutions. Also, as mentioned in the PA, there are 5 advanced techniques, we can try to implement rest of the techniques to see if there is any improvement, and tries to combine them to get even better performance.

Q8: Compare the SimCLR with SupContrast. What are the similarities and differences?
SimCLR and SupContrast are both contrastive learning frameworks. They are both designed to learn more effective representations by maximizing agreement between differently augmented views of the same data input. However, they differ in their approaches. SimCLR is an unsupervised contrastive loss, and it relies heavily on data augmentation strategies to create positive pairs. It marks one input as the anchor, creates a positive pair through augmentation, and mark all other input as negatives. Meanwhile, SupContrast is a supervised contrastive loss. It incorporates label information to this definition of positive and negative pairs and relies on more than one positive sample, leveraging

the class labels to form positive and negative pairs and potentially leading to more meaningful representations.

Q9: How does SimCSE apply dropout to achieve data augmentation for NLP tasks? SimCSE uses standard dropout to create two embeddings of the anchor sentence as a positive pair. It applies standard dropout twice with different masks to the same input sentence during training. This approach effectively creates 'augmented' versions of the data, as the randomness introduced by dropout alters the activations within the network, leading to variations in the generated embeddings, which the model then tries to bring closer in the embedding space.

Since dropout randomly zeroes out a subset of activations, each pass through the model results in a slightly different representation of the same input sentence due to the different sets of neurons being active. This process effectively creates two "augmented" versions of the same sentence.

Q10: Do the results match your expectation? Why or why not? We expected our model with contrastive learning loss function to perform relatively better than our baseline model. Since contrastive learning is specifically designed to draw similar representations together and maximize the distance between two sentences without the same meaning, we believed that this more sophisticated loss can help us cluster and distance the differences between each user input better. Within the realm of applying contrastive learning losses, we also expected SupCon to perform better than SimCLR. This is because SimCLR is a unsupervised loss, while SupCon is a supervised loss. In some cases, user input with similar semantics and words may have very different intent. In these cases, the labels in SupCon help us to learn the user intent better than when we allow SimCLR to help classify solely with the help of sentence embedding similarities.

Our performance didn't really match our expectations. With using supervised contrastive learning loss on the test set, we were able to get an accuracy of 0.88 compared to the baseline model of 0.87. While the performance did get better, this change in accuracy was fairly insignificant. Moreover, SimCLR performed worse than SupCon, arriving at a test accuracy of 0.8. This matched our expectations in that it performed worse than SupCon; however, it's much lower than our baseline model.

The discrepancy might arise from differences in dataset complexity, model architecture, hyperparameter settings, or the extent of pre-training, especially in contrast to image datasets. Additionally, the unique characteristics of the NLP task at hand and the data distribution might affect the efficacy of contrastive learning approaches.

Q11: What could you do to further improve the performance ? There are a lot of data augmentation, model fine-tuning, and change of loss and optimizer experiments that we can perform in order to further improve the performance of our model.

In our classification, with contrastive learning, the performance of the SimCLR and SupCon losses did not match our expectations. Our hypothesis is that these losses may be you were not able to capture the deeper semantic meaning behind our dataset. Therefore, we can try other contrastive learning losses, such as triplet loss, N-pair loss and InfoNCE loss.

Moreover, we can try advanced data augmentation. Due to the complexity and computational limitations in this PA, we only tried the dropouts data augmentation. However, we can also try other popular data augmentation techniques in NLP, such as paraphrasing, back-translation, or using adversarial examples, to generate more diverse and meaningful positive pairs for contrastive learning.

We can also perform model architecture tweaks specific to our task. We can vary the size of the projection head in contrastive learning or add task-specific layers on top of the pre-trained model before fine-tuning.

Lastly, we can also combine the predictions from the baseline, SimCLR, and SupCon models using ensemble techniques. This could leverage the strengths of each approach and potentially lead to better overall performance.

6 Authors' Contributions and References

Jessica Song: I helped Zhirui fix the baseline model, implemented the majority of SupCon model and train, and contributed to integrating the SupCon Loss to our model through fixing the dimension errors. In the report, I wrote most of the discussion on contrastive learning, abstract, introduction, and related work.

Lian Gan: Base on dataloader and bert implemented by groupmates, I reporoduce the baseline model by myself before working on contrastive learning. Then I pair programed with Jessica, and implement the Supcon model, training process, write CustomSupConLoss function, rewrite evaluation function for CL and fix bug when merging codes. I write method/experiments and result part for contrastive learning in this report.

Bobby Zhu: I helped Zhirui fix the baseline model, and I implemented the advanced techqiues such LLRD (alone) and SWA (with Zhirui Xia). Also, I developed the Custom Model class and custom training. I also create early stop and visualiation code for tuning and report. For report, I wrote method/experiments and result part for custom model.

Zhirui Xia: I wrote the baseline, including writing a pipeline for outputting the results a folder for experimentations and reports. I helped fix bugs of everything, including fixing the SWA techniques with Bobby, developed and debug the SupCon model, and merged all branches and resolved all conflicts. For the report, I wrote method/experiments and result part for the baseline.

7 References

Tianyu Gao, Xingcheng Yao, Danqi Chen. SimCSE: Simple Contrastive Learning of Sentence Embeddings. arXiv preprint arXiv:2104.08821

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, Dilip Krishnan. Supervised Contrastive Learning. arXiv preprint arXiv:2004.11362

Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. arXiv preprint arXiv:2002.05709