

---

# Custom Fine-Tuning and Contrastive Learning with Transformers

---

**Connor Gag**  
cgag@ucsd.edu

**Shivangi Karwa**  
skarwa@ucsd.edu

**Hansin Patwa**  
hpatwa@ucsd.edu

**Max Yang**  
guy006@ucsd.edu

## Abstract

In these experiments, we aim to understand the effects of different architectures and modeling techniques when using a pre-trained BERT encoder in our model to do scenario classification. We experiment with Stochastic Weight Averaging, Top Layer Reinitialization, Low-Rank Adaptation, and Contrastive Learning (SupCon and SimCLR). We found that using both Stochastic Weight Averaging and Top Layer Reinitialization resulted in the best validation accuracy of 91.15%. The accuracy for SupCon was very close and many of the other techniques were within 2% of this accuracy. SimCLR and LoRA performed the worst at about 81% accuracy.

## 1 Introduction

Transformers have been the benchmark in the field of natural language processing (NLP) by achieving state-of-the-art performance across various tasks, including text classification, sentiment analysis, and question answering. Among these models, BERT (Bidirectional Encoder Representations from Transformers) has been particularly influential due to its ability to capture deep contextual relationships within text. In this research, we aim to explore and fine-tune a pre-trained BERT model for scenario classification, a crucial task in NLP applications such as virtual assistants and intent recognition systems.

Scenario classification involves categorizing input text into predefined categories based on its meaning. In this study, we will leverage the Amazon MASSIVE dataset, which contains diverse text samples across 18 scenario categories. By training BERT on this dataset, we aim to develop a robust classification model that can accurately predict the scenario label based on the input text. For instance, given an input like "wake me up at nine am on Friday," the model should correctly classify it under the alarm scenario. This capability is essential for building intelligent systems that can understand user intent and improve human-computer interaction.

## 2 Related Work

Devlin et al. (2019) introduces the Bidirectional Encoder Representations from Transformers (BERT), which is already one of the fundamental models for Large language models nowadays. We use this framework as the baseline model. Hu et al. (2022) introduces Low-Rank Adaptation of Large Language Models (LORA), which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture. We use this framework as our alternative model in section 3.3. Khosla et al. (2021) introduces supervised contrastive learning by extending the self-supervised batch contrastive approach to the fully supervised setting and effectively leveraging label information. Gao et al. (2022) introduces the Simple Contrastive Learning of Sentence Embeddings (Simclr) framework and shows that it is better than the BERT and other baseline models by 4.2% and 2.2%. We borrow their framework as our customized model in section 3.4.

Many other papers use the same dataset. FitzGerald et al. (2022) presented the amazon massive scenario dataset that we use in these experiments. The intent of the dataset was to be used for slot-filling, intent classification, and virtual assistant evaluation. Muennighoff et al. (2023) introduces the Massive Text Embedding Benchmark (MTEB) method to apply the embeddings on semantic textual similarity on other tasks, such as clustering or reranking with the amazon MASSIVE dataset. They find that no particular text embedding method dominates all tasks. Zhang et al. (2023) introduces ClusterLLM, a novel text clustering framework that leverages feedback from an instruction-tuned large language model, such as ChatGPT, and empirically show that this framework is both effective for fine-tuning small embedder and cost-efficient to query ChatGPT, with the MASSIVE dataset.

### 3 Methods

#### 3.1 Baseline

The model is built on a BERT encoder (`bert-base-uncased`) with a projection layer that maps the hidden representations to a lower-dimensional space before classification. A dropout layer is applied after the encoder output to prevent overfitting. The classifier consists of two linear layers with a ReLU activation in between, mapping the projected embeddings to the target class logits.

For training, we used the Adam optimizer with a learning rate of  $1 \times 10^{-4}$  and an epsilon of  $1e-8$  to ensure numerical stability. A cosine annealing learning rate scheduler was applied to gradually decrease the learning rate over 10 epochs, reaching a minimum of  $1 \times 10^{-6}$ . We experimented with different learning rates, ranging from  $1 \times 10^{-5}$  to  $1 \times 10^{-2}$ , as well as different batch sizes such as 16, 32 and 64, and schedulers like CosineAnnealingLR and StepLR, but the existing setup yielded the best results i.e Learning rate of **1e-4**, batch size of **16** and scheduler as **CosineAnnealingLR**. The model was trained using cross-entropy loss, and we tracked both training and validation accuracy to monitor performance. We were able to achieve the test accuracy of **89.6099%**

#### 3.2 Custom Models

The methods we use for fine-tuning can be found in Chang (2021)

##### 3.2.1 Stochastic Weight Averaging

For our first custom fine-tuning approach, we implemented Stochastic Weight Averaging (SWA) to improve generalization. The training setup remained similar to the baseline, using the Adam optimizer with a learning rate of  $1 \times 10^{-4}$  and a cosine annealing learning rate scheduler, which gradually reduced the learning rate over 10 epochs to a minimum of  $1 \times 10^{-6}$ . SWA was applied in the final 25% of training epochs, where an averaged model maintained running updates of the weights. During this phase, the learning rate was set to three times the initial value using the SWA-specific scheduler. After training, batch normalization statistics were updated before final evaluation. We experimented with different SWA start points and learning rate scaling factors, but this configuration yielded the best results. The model was trained using cross-entropy loss, and validation accuracy was measured using both the standard and SWA-averaged models. Through this fine-tuning technique, we were able to achieve a test accuracy of **88.6684%**, which is very slightly better than the baseline model.

##### 3.2.2 Top Layer Reinitialization

The second custom configuration that we did was resetting the top layers of our pre-trained BERT encoder. The main parameters that we used here concern how we initialize the top n layers. We did this by filling in the weights for those layers with a normal distribution with a mean of 0 and a standard deviation of 0.2. We made this choice because these initializations are consistent with BERT’s initializations. Following this pattern, we reinitialized the bias terms to be zero and the layer normalization weights to one.

The top layers of the encoder contain more specific information about the data that it was trained on and the lower layers contain more general information. So by starting from scratch for the higher layers, we can start from scratch to encode more specific information about our data. We can alter how many of the top layers we change in order to see differences in accuracy. We ran this experiment

for 10 epochs for each different value of top layers to be reinitialized and found that re-initializing fewer layers resulted in the highest validation accuracy. It is important to note that the number of epochs is a limiting factor here because perhaps allowing more epochs would give the models with more reinitialized layers more time to catch up with those that had more pre-trained layers. Overall, re-initializing the top two layers had the best accuracy and is likely what we would go with in the future.

### 3.3 LoRA

We experimented with LoRA (Low-Rank Adaptation) with different rankings. The strategy allows us to have fewer trainable parameters in our model which makes training quicker and use less memory. The total number of parameters and total number of trainable parameters are shown below. As we can see, LoRA reduces the number of trainable parameters to less than 1% of the total parameters.

	Trainable Params	All Params	Trainable (%)
Before LoRA	109,482,240	109,482,240	100.00%
After LoRA	73,728	109,555,968	0.07%

Table 1: Trainable parameters before and after applying LoRA.

### 3.4 Contrastive Learning

The SupCon model using supCON loss is trained using specific hyperparameters. The model type and task are both set to supcon. The training process is divided into two phases: the first phase runs for 5 epochs with a learning rate of 0.0001 and a batch size of 128, while the second phase runs for 20 epochs with the same learning rate of 0.0001. The method is supcon. The model architecture consists of an encoder, from which the CLS token is extracted, followed by dropout and normalization. This processed token is then passed through a linear layer (single head). To train the model, a batch is passed through the network twice, effectively doubling its size. Since dropout is applied, identical data points in the batch generate different representations. The model is optimized using the SupCon loss, leveraging both features and labels. After training, all layers except the encoder are discarded, and a classifier is attached to the model. The encoder is then frozen, and the classifier is trained separately.

The SupCon model using simCLR loss is trained using specific hyperparameters. The model type and task are both set to supcon. The method is set to simCLR. The training process is divided into two phases: the first phase runs for 3 epochs with a learning rate of 0.00001 and a batch size of 128, while the second phase runs for 20 epochs with a learning rate of 0.001. The model architecture consists of an encoder, from which the CLS token is extracted, followed by dropout and normalization. This processed token is then passed through a linear layer (single head). To train the model, a batch is passed through the network twice, effectively doubling its size. Since dropout is applied, identical data points in the batch generate different representations. The model is optimized using the SupCon loss, leveraging both features and labels. After training, all layers except the encoder are discarded, and a classifier is attached to the model. The encoder is then frozen, and the classifier is trained separately.

## 4 Results

exp idx	exp	loss	accuracy
1	Test set before fine-tuning	3.2425	13.0127%
2	Test set after fine-tuning	0.6200	89.6099%
3	Test set with 1 <sup>st</sup> technique	0.7219	88.6684%
4	Test set with 2 <sup>nd</sup> technique	0.7097	89.7099%
5	Test set with 2 techniques	0.64730	90.5514%
6	Test set with SupContrast	307.0496	90.41%
7	Test set with SimCLR	475.251	81.81%

Table 2: Experiment Results

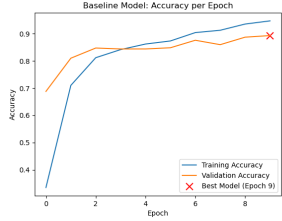

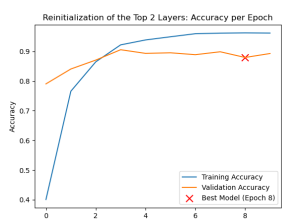
Model	Training Accuracy	Validation Accuracy	Loss Plot																																	
Baseline model	95.6487%	88.3423%	 <p>Baseline Model: Accuracy per Epoch</p> <table><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr><tr><td>0</td><td>0.35</td><td>0.75</td></tr><tr><td>1</td><td>0.75</td><td>0.85</td></tr><tr><td>2</td><td>0.82</td><td>0.88</td></tr><tr><td>3</td><td>0.85</td><td>0.88</td></tr><tr><td>4</td><td>0.88</td><td>0.88</td></tr><tr><td>5</td><td>0.90</td><td>0.88</td></tr><tr><td>6</td><td>0.92</td><td>0.88</td></tr><tr><td>7</td><td>0.93</td><td>0.88</td></tr><tr><td>8</td><td>0.94</td><td>0.88</td></tr><tr><td>9</td><td>0.95</td><td>0.88</td></tr></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.35	0.75	1	0.75	0.85	2	0.82	0.88	3	0.85	0.88	4	0.88	0.88	5	0.90	0.88	6	0.92	0.88	7	0.93	0.88	8	0.94	0.88	9	0.95	0.88
Epoch	Training Accuracy	Validation Accuracy																																		
0	0.35	0.75																																		
1	0.75	0.85																																		
2	0.82	0.88																																		
3	0.85	0.88																																		
4	0.88	0.88																																		
5	0.90	0.88																																		
6	0.92	0.88																																		
7	0.93	0.88																																		
8	0.94	0.88																																		
9	0.95	0.88																																		
First fine-tuning technique	97.5942%	89.7196%	 <p>SWA Model: Accuracy per Epoch</p> <table><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr><tr><td>0</td><td>0.35</td><td>0.75</td></tr><tr><td>1</td><td>0.75</td><td>0.85</td></tr><tr><td>2</td><td>0.82</td><td>0.88</td></tr><tr><td>3</td><td>0.85</td><td>0.88</td></tr><tr><td>4</td><td>0.88</td><td>0.88</td></tr><tr><td>5</td><td>0.90</td><td>0.88</td></tr><tr><td>6</td><td>0.92</td><td>0.88</td></tr><tr><td>7</td><td>0.94</td><td>0.88</td></tr><tr><td>8</td><td>0.96</td><td>0.88</td></tr><tr><td>9</td><td>0.98</td><td>0.90</td></tr></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.35	0.75	1	0.75	0.85	2	0.82	0.88	3	0.85	0.88	4	0.88	0.88	5	0.90	0.88	6	0.92	0.88	7	0.94	0.88	8	0.96	0.88	9	0.98	0.90
Epoch	Training Accuracy	Validation Accuracy																																		
0	0.35	0.75																																		
1	0.75	0.85																																		
2	0.82	0.88																																		
3	0.85	0.88																																		
4	0.88	0.88																																		
5	0.90	0.88																																		
6	0.92	0.88																																		
7	0.94	0.88																																		
8	0.96	0.88																																		
9	0.98	0.90																																		
Second fine-tuning technique: Reinitializing Top Layers	96.63019%	89.7688%	 <p>Reinitialization of the Top 2 Layers: Accuracy per Epoch</p> <table><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr><tr><td>0</td><td>0.35</td><td>0.75</td></tr><tr><td>1</td><td>0.75</td><td>0.85</td></tr><tr><td>2</td><td>0.82</td><td>0.88</td></tr><tr><td>3</td><td>0.85</td><td>0.88</td></tr><tr><td>4</td><td>0.88</td><td>0.88</td></tr><tr><td>5</td><td>0.90</td><td>0.88</td></tr><tr><td>6</td><td>0.92</td><td>0.88</td></tr><tr><td>7</td><td>0.94</td><td>0.88</td></tr><tr><td>8</td><td>0.96</td><td>0.88</td></tr><tr><td>9</td><td>0.97</td><td>0.90</td></tr></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.35	0.75	1	0.75	0.85	2	0.82	0.88	3	0.85	0.88	4	0.88	0.88	5	0.90	0.88	6	0.92	0.88	7	0.94	0.88	8	0.96	0.88	9	0.97	0.90
Epoch	Training Accuracy	Validation Accuracy																																		
0	0.35	0.75																																		
1	0.75	0.85																																		
2	0.82	0.88																																		
3	0.85	0.88																																		
4	0.88	0.88																																		
5	0.90	0.88																																		
6	0.92	0.88																																		
7	0.94	0.88																																		
8	0.96	0.88																																		
9	0.97	0.90																																		

Table 3: Model Performance Comparison

Model	Training Accuracy	Validation Accuracy	Loss Plot
First and Second Fine tuning techniques	97.6897%	91.1460%	<p>SWA Model with Reinitialization of the Top 2 Layers: Accuracy per Epoch</p>
SupContrast	89.85%	90.41%	<p>Supcon Model: Accuracy per Epoch</p>
SimCLR	78.93%	81.81%	<p>Supcon Model: Accuracy per Epoch</p>
LORA Rank 8	80.2327%	79.4392%	<p>Lora Model with Rank 8: Accuracy per Epoch</p>
LORA Rank 16	81.5703%	80.0787%	<p>Lora Model with Rank 16: Accuracy per Epoch</p>
LORA Rank 32	84.3061%	81.6527%	<p>Lora Model with Rank 32: Accuracy per Epoch</p>

Table 4: Model Performance Comparison

Top Layers Reinitialized	Training Accuracy (%)	Validation Accuracy (%)
2	96.6302	89.7688
4	95.2145	89.0802
6	94.2852	89.3261
8	84.1932	81.9970

Table 5: Training and Validation Accuracy for Different Numbers of Reinitialized BERT Layers

## 5 Discussion

### 5.1 Main Questions

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

A1: We expect to get about  $1/18 = 5.56\%$ , perhaps slightly better. This 5.56% is random guessing of the 18 classes, which is likely because the model has not been trained on our task. The model is pre-trained, but not on our task. It is possible that it will do slightly better, however, because of the pretraining that has been done that may translate to our problem.

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

A2: Yes, the results match our expectation. The accuracy on the test set before fine-tuning was 13.01%, which is not great but is better than random guessing. It is better than random guessing because it is pretrained on other data.

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

A3: There are many ways we can improve performance, such as training the model for longer, tuning the hyperparameters, testing out different architectures, etc. Many of the improvements are discussed and implemented in later sections, such as Stochastic Weight Averaging and Top Layer Reinitialization.

Q4: Which techniques did you choose and why?

A4: We chose Stochastic Weight Averaging (SWA) as a fine-tuning technique to improve generalization and stability during training. SWA averages multiple weight states from different training epochs, that helps smooth out fluctuations and prevents overfitting. This leads to better test accuracy compared to a standard model. We could adjust the number of epochs used for SWA to control how much averaging occurs.

We chose reinitializing the top layers because it is a relatively simple way to fine-tune a pretrained model to a specific task. This technique also gave us the flexibility to adjust the number of top layers to reinitialize.

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

A5: We expect that applying the combination of both the techniques i.e Stochastic Weight Averaging (SWA) and Reinitialization of the top layers will result in better text accuracy than applying individual techniques because SWA improves stability, and reinitialization helps the model learn better for our task. Together, they balance each other and improve test accuracy more than using them alone.

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

A6: Yes, the results matched our expectations as combining both techniques achieved a higher test accuracy of of **90.5%** as compared to **89%** when individual techniques were applied.

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

A7: We could improve performance by trying different optimizers and schedulers, such as adding warm-up steps for better training stability. We could also experiment with other advanced techniques such as Frequent Evaluation and include them if they improve performance.

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

A8: SimCLR and SupContrast are both contrastive learning methods, but they differ in their use of supervision and training objectives. SimCLR is self-supervised, learning representations by

maximizing agreement between augmented views of the same image without using labels. In contrast, SupContrast is supervised and leverages label information to group samples from the same class closer together while separating those from different classes.

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

A9: It passes the same sentence through a pre-trained language model twice, applying different dropout masks each time. This randomness in dropout produces slightly different embeddings for the same input, which are treated as positive pairs in contrastive learning

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

A10: supContrast does better than simCLR in our results and that makes sense. Because supContrast uses the labels to drive the same samples closer and opposite samples apart, it can learn a much better representation compared to simCLR, which is self-supervised.

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

A11: To further improve the model accuracy one thing that can be done is instead of a linear head layer we can have an MLP, another thing that will help for sure is increasing the batch size to as much as the GPU allows as the larger the batch size the better it gets.

## **5.2 Other Important Points**

### **5.2.1 LoRA Fine-Tuning**

We found that having a higher rank increased the training accuracy and the validation accuracy. This makes sense because as the rank increases, so does the number of trainable parameters in the model. And having more trainable parameters means that our model has the ability to learn more and perform better when it is time to do inference. We experimented with a rank of 8, 16, and 32 and found a consistent improvement as we increase the rank. The final validation accuracy at the 10th epoch for our LoRA models was about 80%, 8% less than our baseline.

However, training the model for more epochs would likely give us more insights on which rank is the best. None of the models converged in the 10 epochs that we gave them. In fact, they all continued to increase in accuracy at about the same rate even at the 10th epoch for both the training and validation sets, indicating that more learning could have been done. Overall, future experiments with higher epochs should be done to fully understand the impact of rank in LoRA for this data set, but we expect that accuracy will not be reduced by very much when the model is trained until convergence.

LoRA takes more epochs to converge, but this does not mean that it takes LoRA longer to converge overall. This is because it takes less time to run each epoch with LoRA. In our experiments, epoch 7 took 41 seconds for our baseline model and 23 seconds for LoRA. Each epoch took roughly half the time that our baseline took, and likely took much less memory. These benefits should be strongly considered when thinking about using LoRA in the future. The GPU we used for our tasks was NVIDIA A30 (MIG 2g.12gb) which gave us 12 gb of GPU memory.

### **5.2.2 Contrastive learning**

The main thing to note for our contrastive learning experiments, in particular simCLR experiments, when the simCLR loss was trained to its best value (somewhere in the high 200's) we were not able to train the classifier to achieve a high accuracy, no matter what batch size, learning rate, epoch number we set we were never able to increase the accuracy above 70. But surprisingly if we didn't train simCLR loss to that much but instead stopped at 450ish we were able to train our classifier model to achieve an accuracy of around 81%, if we ran for more epochs it surely looked like it would increase. In general the higher the batch size the faster we lowered the losses, and 0.0001 seemed to be the ideal learning rate. Any higher and it would start oscillating and any lower would take too long.

## **6 Authors' Contributions**

Connor: I helped with the setup of the baseline and fixed bugs for contrastive learning. I mainly focused on creating and experimenting with the fine-tuning technique of reinitializing the top n layers

and LoRA. I also wrote the report outline, abstract, many of the discussion questions, as well as the sections that correspond with the code I wrote (reinitialization and LoRA).

Shivangi: I worked on the baseline model training, before and after fine-tuning along with hyperparameter tuning. I also implemented the 1st fine tuning technique i.e Stochastic Weight Averaging and wrote the custom train method for it. I also ran and deduced results from implementing both the fine-tuning techniques together. With respect to the report, I worked on the parts I implemented, i.e Baseline model architecture explanation, 1st fine-tuning technique and answers for discussions related to baseline and custom model architecture.

Hansin: I wrote the code for the base model and then wrote the complete code for contrastive learning. Connor did help me debug the contrastive learning code multiple times. I also filled in all the parts of the report that were related to contrastive learning.

Max: Contribute to the introduction and related work.

## References

- Chang, P. (2021). Advanced Techniques for Fine-tuning Transformers.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs].
- FitzGerald, J., Hench, C., Peris, C., Mackie, S., Rottmann, K., Sanchez, A., Nash, A., Urbach, L., Kakarala, V., Singh, R., Ranganath, S., Crist, L., Britan, M., Leeuwis, W., Tur, G., and Natarajan, P. (2022). MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages. arXiv:2204.08582 [cs].
- Gao, T., Yao, X., and Chen, D. (2022). SimCSE: Simple Contrastive Learning of Sentence Embeddings. arXiv:2104.08821 [cs].
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.
- Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. (2021). Supervised Contrastive Learning. arXiv:2004.11362 [cs].
- Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2023). MTEB: Massive Text Embedding Benchmark. arXiv:2210.07316 [cs] version: 3.
- Zhang, Y., Wang, Z., and Shang, J. (2023). ClusterLLM: Large Language Models as a Guide for Text Clustering. arXiv:2305.14871 [cs] version: 2.