Intent Classification on MASSIVE Dataset

Devin Muzzy dmuzzy@ucsd.edu
Aidan Bell abell@ucsd.edu
Isaiah Dailey idailey@ucsd.edu
Aditya Melkote avmelkote@ucsd.edu

Abstract

This paper explores the application of pre-trained transformer based models, specifically the BERT (bidirectional encoder representations from transformers) for the task of intent classification on the 60 intents from the MASSIVE dataset. Utilizing pretrained models (bert-base-encased) hosted on Huggingface along with several advanced techniques such as learning rate schedulers, warmup steps and contrastive learning along with hyperparameter tuning on the pretrained models, the authors demonstrate the effectiveness of each technique through examining the accuracy metric differences as well as examine how effective the combination of advanced techniques are for the task. The authors find that with hyperparameter tuning, the baseline model performs fairly well on intent classification, with significant increases when individually employing as well as combining techniques such as Linear Schedulers with warmup steps and Layer-Wise Learning Rate Decay. Unsupervised learning performs the worst out of all the models, while supervised contrastive learning results in a performance increase over directly simultaneously training the BERT encoder and classifier layer.

1 Introduction

In this report, we focus on application of intent classification, specifically on the MASSIVE dataset published by Amazon. This dataset is composed of an intent label for a given line of text, summing up to 60 unique intent classes and roughly 16.5k rows, split into train, validation and test sets. Specifically, the task we undertake is creating a model that can effectively perform intent classification on given texts.

Utilizing this dataset along with a pretrained BERT model (bert-base-encased) from Huggingface, we finetune the model with hyperparameter tuning, learning rate schedulers and warmup steps as well as contrastive learning, specifically with the SupContrast and SimCLR losses and demonstrate how models using each of these techniques as well as a combination of techniques perform in terms of effectiveness, given by the accuracy metric, in relation to each other. We hypothesize as to why these differences occur, as well as contrast between the techniques to understand how the differences in the underlying architecture choices have an impact on the accuracy.

2 Related Work

Amazon's MASSIVE intent dataset was created for various NLP tasks including intent classification, slot filling and virtual assistant evaluation. Along with releasing the dataset, Fitzgerald et al. ([2] released various fine tuned models, such as the XLM-R Base, a multilingual BERT based model fine tuned for intent classification for the 60 intents. The models were fine tuned with a myriad of metrics, specifically the exact match accuracy and the intent classification accuracy. Another model used was the mT5 for intent analysis, which was also finetuned on the MASSIVE dataset to predict intents from user inputs.

There has been further work done in the field of NLP using the BERT pretrained language model along with the MASSIVE dataset. Work by Labrak et al. ([5]) applies the BERT model for NLP in

the specific field of bio medicine and clinical applications. The authors utilized two different corpus's to train the model, where they used the MASSIVE dataset to train their own classifier to classify the French Healthcare Dataset by language, and retain the target language information. They then used a private healthcare dataset from their hospital, and performed organization, and fed the information into several different specialized BERT models, which they then performed tasks of named entity recognition, relation extraction and answering clinical queries.

3 Methods

For the purpose of intent classification with the MASSIVE dataset, we first loaded and tokenized the text field within the data to process the data for the input into the BERT model.

We loaded the pre-trained BERT model, specifically the Bert-base-encased from Huggingface, which is a pretrained model on the English language using a masked language modeling objective. We then developed the baseline IntentModel by passing the last hidden state's [CLS] token into a dropout layer, and then passed the output of the dropout layer into our classifier layer. Our training procedure iterated over batches (determined by our batch size) per each epoch.

We established the evaluation metric for our models as the accuracy, defined by Accuracy $= \frac{\text{total correct predictions}}{\text{total number of samples}}$.

We performed hyperparameter tuning as part of our fine-tuning procedure, and employed various different design choices as follows:

3.1 Baseline

Our design choices include using cross-entropy loss, utilizing an Adam Optimizer, as well as dropout rate of 0.1 along with 10 epochs for our training. Our learning rate was 1e-4. We selected this dropout rate as it is common for most datasets to pick a dropout rate between 0.1 and 0.2, leaning towards a smaller dropout rate the larger the dataset is.

We selected the Adam optimizer due to its adaptive learning rate property which allows for efficient convergence, along with its ease-of-use with its minimal hyperparameter tuning requirements.

3.2 Custom

The two custom finetuning techniques we chose were 1. linear scheduler with warmup steps, and 2. Layer-wise Learning Rate Decay (LLRD).

The linear scheduler with warmup steps requires an argument called warmup ratio, which sets the warmup period to a percentage of the total number of steps. We set this to 0.1, which is a common warmup ratio value for large datasets

For Layer-wise Learning Rate Decay (LLRD), we created three groupings of layers. Layers 0-3 had a learning rate of L, layers 4-7 had a learning rate of L*1.75, and the layers closest to output (layers 8-11) had a learning rate of L*3.5. Classification and pooling layers had a learning rate of L*3.6. Weights in all layers other than "bias", "LayerNorm.bias", and "LayerNorm.weight" had a weight decay value of .01. Our learning rate L was 1e-4.

For both finetuning techniques, the dropout rate chosen here was once again 0.1. We once again used the Adam Optimizer here.

3.3 Contrastive Learning

Next, we implemented contrastive learning training models, used to train and fine-tune an encoding network. We did this in two ways, following the strategies described in the papers [1] (SimCLR) and [4] (SupCon), respectively. These papers describe a training method for an encoding network which aims to teach the model to vectorize the inputs (images) in a way that is invariant under several standard transformations/augmentations (e.g., scaling, cropping, color shifting, sharpening, blurring, filtering, etc.). This is done by using a special loss function. The loss function first developed in [1] is self-supervised, and does not use or require any labels/outside information beyond the images

themselves. The paper [4] iterates on this work by introducing an updated loss function that can be used for reinforcement learning, which is useful when all the data is labelled. Since we are training on strings of words rather than images, rather than applying image transformations such as scaling, cropping, etc., we used dropout (as used in [3]), and trained the model to vectorize sequences of tokens in a way that is invariant under randomly masking some of the tokens. After training the encoder using contrastive learning, we trained a simple linear classifier on the embeddings to perform sentiment analysis.

Using both the SimCLR (unsupervised) and the SupCon (supervised) models, we fine-tuned a pre-trained BERT encoding model with a temperature of 0.07, a batch size of 100, a learning rate of 0.0001, a dropout rate of 0.1. We trained both the encoder and the classifier for 25 epochs. For the classifier training, we further used early stopping with a patience threshold of 4.

4 Results

We trained five different models for the task: The baseline model, two custom models with additional fine-tuning techniques (see the methods section above), and two contrastive models: a supervised model (SupCon) and an unsupervised model (SimCLR).[1] Each model was trained by fine-tuining a pre-trained BERT model. The best performing model was the model which used both of the fine-tuning techniques, while the worst-performing model was the SimCLR model (which was the only model which was trained via unsupervised learning). See Figure 1 for a collection of tables showing the training and validation accuracy by epoch for each of the models.

exp idx	exp	loss	accuracy
1	Test set before fine-tuning	4.1178	0.0098
2	Test set after fine-tuning	1.2379	0.78245
3	Test set with 1^{st} technique	0.9259	0.8624
4	Test set with 2^{nd} technique	1.1256	0.8533
5	Test set with 2 techniques	0.89742	0.8826
6	Test set with SupContrast	1.9584	0.8403
7	Test set with SimCLR	1.7630	0.5666

Table 1: Accuracy and Loss By Experiment

5 Discussion

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

A1: Our expected test accuracy is less than 1/60, because there are 60 possible outputs for our network, and before finetuning we are just randomly guessing. It is likely going to be lower than 1/60 because the categories in our training set are not randomly distributed, with some categories making up over 7% of the data, and some less than half of a percent. BERT pretrained weights are only for encoding, not for our feed forward classification layer.

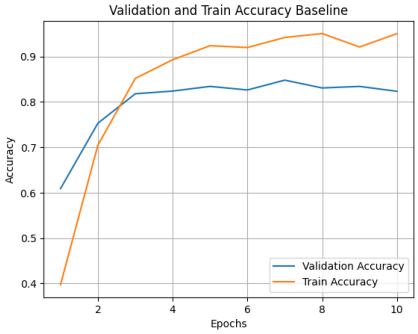
Q2: Do results match your expectation(1 sentence)? Why or why not ?

A2: Our results match our expectations with regards to the accuracy increasing greatly as well as the loss decreasing significantly, as we expected to increase our test accuracy from finetuning – and our initial accuracy of 0.0098 matched our prediction of an accuracy slightly < 1/60.

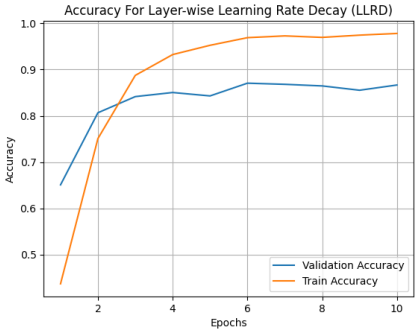
Q3: What could you do to further improve the performance?

A3: We could have performed even further fine tuning with regards to the hyperparameters, as well as employed techniques such as implementing a scheduler. Furthermore, we could stray from the given hyperparameter choice of 10 epochs by increasing the amount, which could potentially improve the test accuracy slightly.

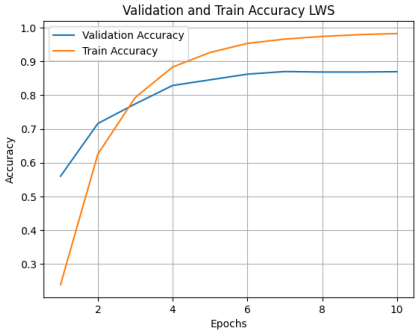
[1] More explicitly, for the contrastive models the encoding and classifier models were trained separately; SupCon used supervised reinforcement learning for both, while only the SimCLR classifier was trained using supervised reinforcement learning, the SimCLR encoder was trained using unsupervised learning.



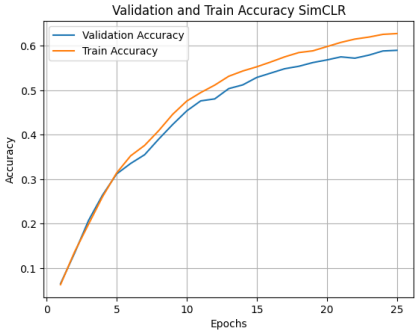
(a) baseline



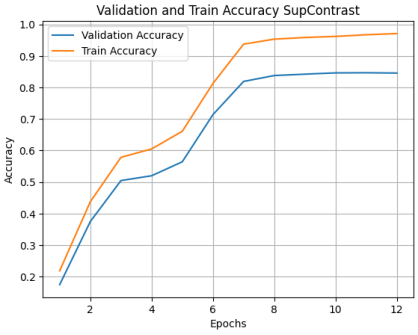
(b) Layer-wise Learning Rate Decay Accuracy



(c) linear scheduler with warm-up steps



(d) SimCLR



(e) SupCon

Figure 1: Comparison of Figures

Q4: Which techniques did you choose and why?

A4: The first technique that we chose was the linear scheduler with warm up steps. We chose a linear warm up scheduler to stabilize the training process, as large updates to the model parameters early in training causes instability. The linear warm up scheduler ensures that the learning rate is slowly ramped up during the warm up phase, then linear decays to 0. The second technique we chose was Layer-wise Learning Rate Decay (LLRD), which sets the learning rate of encoder layers closer to the output to higher values than encoder layers closer to the input. This encourages our model to leave lower-level encoder layers alone and focus on higher level layers that are likely handling much larger features. We chose this because we thought it would increase model accuracy without having to train for longer.

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

A5: We expect the results from the individual techniques to improve upon the baselines, and combined, we expect both to improve greater than if we had used only 1 of the techniques at a time.

Q6: Do results match your expectation? Why or why not?

A6: Our results match our expectations, as our expectation of each of the techniques as well as the combination of techniques to improve the accuracy, which was what occurred as a result of our experiment. This makes sense, as introducing a linear warm up scheduler adds stability in the training process, which results in an improved accuracy.

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

A7: To further improve performance, we could Re-initialize pre-trained layers, implement stochastic weight averaging (SWA), implement Frequent Evaluation, or tune hyperparameters such as learning rate. Increasing the number of epochs could also improve our performance.

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

A8: SimCLR and SupContrast are two examples of contrastive learning frameworks. They share a common architecture structure, featuring data augmentation, a base encoder, then a small network projection head with a contrastive loss function on top. The contrasting loss function is where they differ. The SimCLR uses normalized temperature-scaled cross-entropy loss, which is essentially cross-entropy loss with a temperature hyper parameter. However, SupContrast proposes a new loss function called self-supervised contrastive loss. This loss function uses labels on the inputs to ensure that all the positives in a batch contribute to the numerator. This allows SupContrast to generalize better to any number of positives as well as intrinsically perform hard positive/negative mining.

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

A9: SimCSE is a simple constrastive learning framework that moves the work done for contrastive models on images into the domain of NLP. In NLP there is no easy way to augment a sentence that preserves its original meaning. Therefore the authors of the paper turned to dropout to take traditional augmentations place. The point of augmentation is to generate two different embeddings that should be the same, and then use a loss function to attract them. When generating predictions on the same sentence through the same model with two different dropout masks applied, two different embeddings will be produced. Therefore since we have different embeddings for the same anchor sentence we are able to use a contrastive framework to learn the features.

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

A10: Overall, our results seemed to match expectations. As expected, the model using unsupervised learning performed the worst, achieving less than 50% test accuracy, while supervised contrastive learning resulted in a performance increase over directly simultaneously training the BERT encoder and classifier layer.

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

A11: One way we could further improve the performance of our model is by increasing the batch sizes and training steps. Empirical results from SimCLR along with many other self-supervised classifiers have seen large improvements by increasing the number of parameters in the model. Additionally the authors mentioned that increasing the batch size had a positive impact on the model's performance.

This may have less of an effect on our model since we have already finetuned the batch size, but it is likely that even finer-grained tuning could still improve performance.

6 Contributions

Devin: Pair programmed step 5 with Isaiah, did some fine tuning for supCon and simCLR, discussion Q8,9,11. Generated some graphs for results section.

Aidan: Implemented finetuning strategy 2: Layer-wise Learning Rate Decay (LLRD) and combined strategies 1 and 2. Contributed to write-up about LLRD in methods, results, & discussion.

Isaiah: Pair programmed step 5 with Devin and read SimCLR, SupCon, and SimCSE papers, helped write methods, results, and discussion sections. Added code support for Apple Silicon, ran code to generate graphs and helped to fine-tune the SimCLR and SimCon hyperparameters.

Aditya: Implemented initial data loading steps as well as baseline model. Implemented finetuning strategy #1: Linear Scheduler with Warmup Steps. Wrote the abstract, introduction, related work as well as parts of methods and discussion.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[2] Jack FitzGerald, Christopher Hench, Charith Peris, Scott Mackie, Kay Rottmann, Ana Sanchez, Aaron Nash, Liam Urbach, Vishesh Kakarala, Richa Singh, Swetha Ranganath, Laurie Crist, Misha Britan, Wouter Leeuwis, Gokhan Tur, and Prem Natarajan. Massive: A 1m-example multilingual natural language understanding dataset with 51 typologically-diverse languages, 2022.

[3] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings, 2022.

[4] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2021.

[5] Yanis Labrak, Adrien Bazoge, Richard Dufour, Mickael Rouvier, Emmanuel Morin, Béatrice Daille, and Pierre-Antoine Gourraud. Drbert: A robust pre-trained model in french for biomedical and clinical domains, 2023.