Large Language Models and Com2Sense: Training to Maximize Standard and Pairwise Accuracy

Drew Letvin, Rishab Abdulvahid, Abhijat Gupta, Jaron Shen

University of California, Los Angeles

1 Introduction

In this project, we took advantage of publicly available large language models to create a machine learning model that would determine whether or not a given statement made sense. Today, thanks to the existence of pretrained large language models, we have access to powerful tools that provide a strong head start to tackling natural language processing tasks. The language models that we decided to try out in our application were BERT and two large language models built off the base BERT model, RoBERTa and DeBERTa.

The Com2Sense dataset was used to train and evaluate the performance of our implemented model. This dataset is a challenging datset that features sentence pairs, each with their own label. The sentences are complementary and nearly identical except for a switched word that creates their complementary nature. This unique characteristic was used to define a second test metric of pairwise accuracy which we optimized our language models around. This metric as well as how we optimized our parameters can be found in the Methods section below.

2 Methods

The goal was to achieve the maximum pairwise accuracy and standard accuracy without sacrificing either metric in the process. Standard accuracy is defined simply as the total number of correct predictions or the number of predictions that matched their associated label. Pairwise accuracy is a less common metric that is unique to the Com2Sense dataset. Since Com2Sense data items are pairs of sentences, each with their own label, we define pairwise accuracy to be the number of sentence pairs for which both sentences in a pair are predicted correctly. While higher standard accuracy would intuitively seem to also result in an increase in pairwise accuracy, this does not always hold true and both metrics must be observed and accounted for in hyperparameter tuning.

We began first by examining three different learning models: BERT, RoBERTa, and DeBERTa. Our initial tests of these models were meant to guague which model we would attempt to fine-tune going forward. With a batch size of 4 and learning rate of 1e-5, each model was trained up to 4000 steps. It quickly became clear that the base BERT model was far inferior when examining the pairwise accuracy metric. Despite all three models attaining similar standard accuracies within the ranges of 50-60%, BERT had trouble even reaching double digit pairwise accuracy on our dev split. RoBERTa and DeBERTa faired far better posting pairwise accuracies closer to the 15-20% range. DeBERTa appeared a more well rounded model throughout multiple checkpoints and had the highest pairwise accuracy of the three making it our choice for further tuning and development.

Following the selection of a learning model we moved on to experimenting with different numbers of steps, batch sizes and learning rates. We initially tested step sizes from 1000 to 4000 with checkpoints every 1000 steps. This large granularity helped us to get a sense of where in the model we may find a sweet spot between over and underfitting. The DeBERTa model trained with a batch size of 4 showed promise between 2000 and 4000 steps so we chose to focus here. Next we iteratively tested batch sizes of 4, 8, 16, and 32 while holding the learning rate constant at 1e-5.

A major problem we were experiencing at this stage was the time to train each model as training 4000 steps would take roughly 3 hours each time we wanted to try a new set of parameters due to the

limiting factor of only having one training GPU. Because of this we decided to try and raise the learning rate while lowering the number of training steps in a hope they may offset one another and lead to better predictions in less time. We tested learning rates of 1e-5, 1e-4, and 5e-4 with a max step size of 1500. Despite evaluating checkpoints starting in 300 step intervals the learning rates proved to be too high even at low step numbers and led to the model predicting only 1's for each of the higher learning rates.

3 Results

Table 1, found below, shows our best parameters for the DeBERTa model. Its worth noting here that even though we tested up to 4000 steps we chose to use the 3000 step checkpoint due to apparent overfitting. We saw a drop in both standard and pairwise accuracy at the following checkpoints and thus decided to stop the training early.

Parameter	Value
Number of Steps	3000
Learning Rate	$1 * e^{-5}$
Batch Size	16

Table 1: Best Parameters

Then in table 2 our reported metrics for this model on both the dev and test splits of our data can be found. The Dev split was the same split used to evaluate our training at different checkpoints. The Test set was a separate set in which the labels were held out entirely until after predictions had been made.

Eval Dataset	Standard Accuracy
Dev	55%
Test	54.337%
Eval Dataset	**Pairwise Accuracy**
Dev	24%
Test	27.849%

Table 2: Accuracy of Predictions

We can see that our model performed similarly on both the Test and Dev splits despite only the Dev data being used to select our model parameters. This is a good sign that the model was not overfit to the training data. Furthermore, despite still having a standard accuracy of roughly 55%, not much different from that of the other learning models and parameters we tested, we do see a large jump in pairwise accuracy. This is a positive sign that these parameters help the large language model handle the added challenge of the Com2Sense dataset.

4 Conclusions/Discussion

Ultimately, we were satisfied with the performance that our optimized models achieved after some tuning of hyperparameters. An interesting point to note is the large difference in pairwise accuracy of the DeBERTa/RoBERTa models with the base BERT model. These models are all transformer models, but the DeBERTa and RoBERTa make important adjustments to the base formula. The RoBERTa model uses dynamic masking rather than static masking, which helps the model to generalize better to new combinations of words and tokens (Tilbe, 2022) . Additionally, RoBERTa tokenizes between words, which helps it to tackle new words. DeBERTa further improves on RoBERTa using a "disentangled" mechanism for computing attention, which should make the model more efficient (Tilbe, 2022) .

A batch size of 16 seems to be the best size for us to use for training our model. At sizes lower than that, our GPU compute units were not being exercised to the fullest and, perhaps, we were too far from a well-performing optima. However, 16 seemed to be our sweet spot. Additionally, at a step size of 4000, our accuracy score took a hit, likely because our model was overfitting and not generalizing well. A learning rate of 1e-5 worked well with these parameters and, in the long run, had little impact on accuracy. The most significant jumps in our tuning stage came from switching to models like DeBERTa and RoBERTa as well as tweaking the batch size to locate the "sweet spot" for fitting the selected model.

The tuning of hyperparameters can only take this task so far. The Com2Sense data set is a rather challenging data set and striking the right balance necessary to improve both standard and pairwise accuracy creates an extra layer of difficulty (Singh et al., 2021) . Perhaps in the future implementing techniques common in traditional machine learning such as boosting and ensemble modeling could see further improvement. That being said these would be computationally expensive and thus would likely not be very feasible on the set up used to obtain our results. Additionally, we believe pretraining our

model on different datasets may help to provide performance gains on the Com2Sense task. At the very least It would be interesting to see how different corpi of data affect the behavior of our model.

References

Shikhar Singh, Nuan Wen, Yu Hou, Pegah Alipoormolabashi, Te-Lin Wu, Xuezhe Ma, and Nanyun Peng. 2021. Com2sense: A commonsense reasoning benchmark with complementary sentences.

Anil Tilbe. 2022. Siebert, roberta, and bert: Which one to implement in 2022?

A Appendices

A.1 Implementation Details

Transformer-based models BERT, DeBERTa, and RoBERTa were implemented using the HuggingFace PyTorch API. These models were trained using one Nvidia T4 GPU on Google Cloud's compute engine service. The OS used for training was Linux Debian-10 on a nN1-standard-2 chip. Ideally 3 more T4 GPUs would have been utilized to train in parallel and reduce time for training, validation, and prediction. With the single T4 set up, training 4000 steps with DeBERTA to assertain our best model took between 3 and 4 hours.