Transformers: Bidirectional Encoder Representations from Transformers and User Intent

Brandon Szeto
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
bszeto@ucsd.edu

Darren Yu
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
dmyu@ucsd.edu

Nathaniel Thomas
Jacobs School of Engineering
University of California, San Diego
San Diego, CA 92122
nathomas@ucsd.edu

Abstract

Today, large language models have become immensely popular. We use one such model, BERT, for user intent classification, leveraging the Amazon Massive Intent Dataset. BERT, a state-of-the-art transformer architecture, has demonstrated remarkable performance across various natural language processing tasks. We fine-tune BERT for user intent classification and then explore advanced training techniques to enhance its performance. Additionally, we investigate the application of contrastive losses for further improvements in model accuracy. We preprocess the dataset, fine tune BERT, implement custom training techniques derived from recent research, and experimenting with contrastive learning methods. Through this process, we aim to provide insights into the efficacy of different strategies for enhancing user intent classification tasks. A variety of loss types were experimented with, but they all performed similarly. We found that refining the baseline BERT with layer-wise learning weight decay and linear scheduler warm-up yielded the best results. Our best model yielded an accuracy of 88.6%.

Preprint. Under review.

Introduction

The Amazon Massive Intent Dataset contains 60 distinct intent labels. For example, we have intent categories like 'alarm_set,' 'play_music,' and 'takeaway_order.' Our job is to fine tune a BERT (Bidirectional Encoder Representations from Transformers) model to categorize

Related Works

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

In **?** the authors introduce a groundbreaking model in natural language processing that utilize the concept of bidirectional context representation learning. By pretraining a transformer-based neural network on large corpora with masked language modeling and next sentence prediction tasks, BERT is able to capture deep contextual information from both left and right contexts of a word. This bidirectional understanding of text enables BERT to achieve state-of-the-art performance on a wide range of NLP tasks, including but not limited to question answering, sentiment analysis, named entity recognition, and machine translation.

Supervised Contrastive Learning

In **?** the authors introduce supervised contrastive learning, a method for learning representations of data by maximizing agreement between similar pairs and minimizing agreement between dissimilar pairs. Supervised contrastive learning leverages labeled data to guide the learning process, making it particularly suitable for tasks where labeled examples are abundant. The method has shown promising results in various applications, including image classification, natural language processing, and speech recognition.

SimCSE: Simple Contrastive Learning of Sentence Embeddings

In **?**, the authors propose a simple yet effective approach for learning sentence embeddings through contrastive learning. By formulating the contrastive objective in a straightforward manner, SimCSE achieves competitive performance on various downstream tasks without requiring complex architectures or extensive hyperparameter tuning. The method has been shown to learn semantically meaningful representations of sentences, making it valuable for tasks such as semantic textual similarity, text classification, and paraphrase detection.

Methods

1. Baseline

Model Architecture

Our intent classification model is built upon the transformer architecture, utilizing the BERT (*bert-base-uncased*) model as the encoder for input text representation. Specifically, our model, named `IntentModel`, leverages a pre-trained BERT model to encode textual inputs into a contextualized embedding space. The architecture is designed to be flexible, allowing for easy adaptation to different target sizes, which corresponds to the number of intent categories in the classification task.

The model is initialized with a tokenizer responsible for converting text inputs into token embeddings, which are then passed through the BERT encoder. The encoder output, specifically the representation of the [CLS] token, is subsequently processed through a dropout layer with a predefined probability (0.1 in this implementation) to mitigate overfitting.

Classifier

Following the dropout layer, the model employs a classification layer, named `Classifier`, to project the encoded representations into the target label space. The `Classifier` consists of a linear layer that maps the input dimension to an intermediate representation, followed by a ReLU activation

function for non-linearity, and another linear layer that maps the intermediate representation to the target size, corresponding to the number of intent categories.

Training Procedure

The training procedure involves several key steps, starting with the initialization of a cross-entropy loss function to compute the discrepancy between the predicted probabilities and the ground truth labels. We utilize an Adam optimizer for adjusting the model parameters based on the computed gradients.

During training, the model iterates over batches of data, computing the forward pass, followed by the loss. The gradients are calculated via backpropagation, and the optimizer updates the model parameters accordingly. The accuracy and loss for both training and validation sets are tracked and reported for each epoch. Additionally, while not activated in the provided code, a learning rate scheduler can be integrated to adjust the learning rate during training, potentially improving convergence rates.

Training is executed over a specified number of epochs, with performance evaluation on a validation set at the end of each epoch to monitor generalization capabilities. The implementation supports visualizing the training and validation accuracy and loss over epochs, aiding in the identification of overfitting or underfitting patterns.

Evaluation

Model evaluation is conducted on a held-out test set or a validation set during the training process. The evaluation metrics include accuracy, which measures the proportion of correctly predicted intentions among the total number of samples.

This intent classification framework demonstrates a straightforward application of transfer learning and fine-tuning BERT for a specific NLP task. By leveraging pre-trained models and adapting them to target tasks, we can achieve high performance with relatively minimal additional training.

2. Custom Fine-tune

Training

In addition to the base `IntentModel`, we introduce a custom training procedure that incorporates advanced optimization techniques to further improve model performance. These techniques are selectively applied based on a specified `technique` parameter, enhancing the model's ability to adapt to the training data and potentially reduce overfitting.

Optimization Techniques

*Learning Rate Decay (LRD) The first technique, applicable when the `technique` parameter is set to 1 or 3, involves the use of Learning Rate Decay (LRD) through the AdamW optimizer with weight decay regularization. This approach adjusts the learning rate over time, reducing it as the model progresses through the epochs. The implementation specifics are as follows:

```
optimizer = torch.optim.AdamW(model.parameters(),
                              lr=args.learning_rate,
                              weight_decay=0.01)
```

LRD helps in stabilizing the learning process and can lead to better generalization by mitigating the risk of overfitting.

*Scheduler for Learning Rate Adjustment The second technique, activated when `technique` is 2 or 3, employs a scheduler to adjust the learning rate throughout training. Specifically, it utilizes a linear schedule with warmup, gradually increasing the learning rate from 0 to the specified maximum before linearly decreasing it:

```
scheduler = get_linear_schedule_with_warmup(
                optimizer,
```

```
            num_warmup_steps=int(0.1 * total_steps),
            num_training_steps=total_steps,
        )
```

This method ensures a smoother and more controlled adjustment of the learning rate, promoting faster convergence and potentially improving final model performance.

Training Procedure

The custom training procedure follows the standard steps of forward and backward passes, with modifications to incorporate the above techniques. At the beginning of each epoch, based on the selected technique, the optimizer's gradient buffers are cleared using `optimizer.zero_grad()` for technique 1 or 3. After computing the loss and performing backpropagation, the learning rate scheduler is optionally updated if technique 2 or 3 is selected.

The effectiveness of these techniques is monitored through accuracy and loss metrics, both on the training and validation datasets. These metrics are visualized over epochs, allowing for empirical analysis of the model's learning progress and the impact of the optimization techniques.

Evaluation

Custom model evaluation is analogous to the base model, emphasizing the model's capability to accurately classify intents in unseen data. The incorporation of advanced optimization techniques aims to enhance this capability by refining the training process.

Note: The custom model retains the architectural foundation of the `IntentModel`, with these techniques offering strategic adjustments to the training algorithm rather than altering the model structure.

3. Supervised Contrastive Learning Model

The `SupConModel` represents a novel approach to intent classification by incorporating supervised contrastive learning, a technique that enhances model performance by learning representations that bring similar samples closer and push dissimilar ones apart in the embedding space.

Model Architecture

The architecture of `SupConModel` is grounded on the BERT (*bert-base-uncased*) transformer model for encoding input text into rich, contextual embeddings. It is equipped with a dropout layer to prevent overfitting and a fully connected (FC) layer that projects the encoded representations to the target classification space. Notably, the model includes a Batch Normalization layer applied to the encoded features before the final classification layer, ensuring a stable and efficient training process by normalizing the activations.

Training Procedure

The training of `SupConModel` is conducted in two distinct phases to optimize the contrastive learning objectives and fine-tune the model for the classification task.

Phase 1: Contrastive Learning In the initial phase, the model employs the Supervised Contrastive Loss (`SupConLoss`), which is designed to learn embeddings by maximizing the similarity between differently augmented views of the same data point relative to other data points. The optimizer used is AdamW with weight decay regularization, which is beneficial for contrastive learning settings. During this phase, each input is processed twice to generate two sets of logits, which are then concatenated and fed into the `SupConLoss` function. Depending on whether the SimCLR variant is activated, the loss computation either uses the embeddings directly or incorporates the target labels for a more supervised learning signal.

Phase 2: Fine-tuning for Classification After the encoder has been trained to produce meaningful embeddings through contrastive learning, its parameters are frozen to preserve the learned representations. The model then transitions to a standard classification training regimen, utilizing Cross-Entropy

Loss. The optimizer and learning rate scheduler from the first phase are reused, ensuring consistency in the optimization strategy. This phase focuses on refining the model's ability to accurately predict intent labels based on the embeddings generated by the now-static encoder.

Evaluation

Model performance is assessed through accuracy and loss metrics across both training and validation datasets. Additionally, the effectiveness of the contrastive learning phase is indirectly evaluated by the improvements observed in the classification performance during the second phase of training. Visualization of training dynamics, including accuracy and loss trends over epochs, provides insight into the model's learning efficiency and generalization capabilities.

Note: The `SupConModel` showcases the potential of integrating contrastive learning principles within the context of intent classification, aiming to set a new standard for model robustness and discriminative power.

Results

Experiment Loss and Accuracy

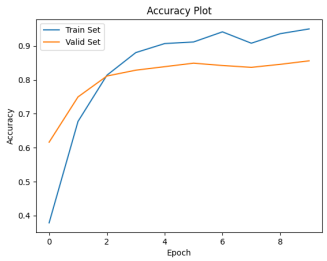
Here are the test loss and accuracy of each experiment.

Exp. Index	Experiment	Loss	Accuracy
1	Test set before fine-tuning	765.904	0.009751
2	Test set after fine-tuning	167.609	0.858439
3	Test set with first technique	134.134	0.876086
4	Test set with second technique	122.907	0.885339
5	Test set with two techniques	123.766	0.886012
6	Test set with SupContrast	33.681	0.879623
7	Test set with SimCLR	46.681	0.868623

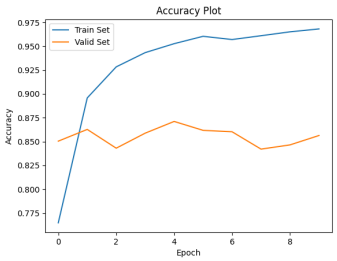
Table 1: Experimental Results

Accuracy Plots

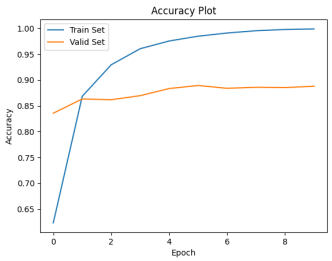
Here are the accuracy plot of the 5 models.



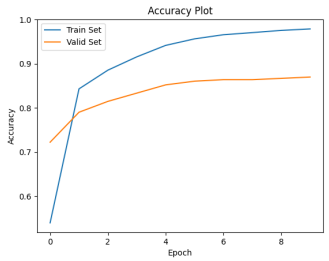
(a) Basline Accuracy



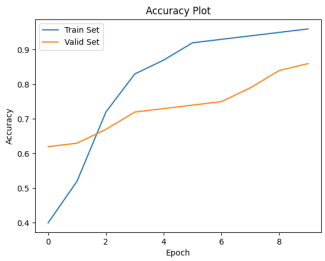
(b) Technique One Accuracy



(c) Technique Two Accuracy



(a) SupContrast



(b) SimCLR

Discussion

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

A1: We expect a low test accuracy. Without fine tuning, we fail to leverage the general representations of language learned by the BERT model for our specific task. BERT has the potential to perform very well on our dataset, but was not trained to do so and would be limited by its generic pre-trained representations.

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

A2: Yes, the results match our expectation. Without fine-tuning, the model's performance is likely to be suboptimal due to its inability to adapt to the specific characteristics of our task, relying solely on generic pre-trained representations.

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

A3: To enhance performance, fine-tuning the BERT model on our specific dataset would be crucial. Additionally, experimenting with different hyperparameters, incorporating data augmentation techniques, and exploring ensemble methods could potentially lead to improved results.

Q4: Which techniques did you choose and why?

A4: We chose Adam with weight decay (AdamW) as our optimizer because it has been shown to effectively handle optimization tasks for deep learning models, offering good convergence properties and robustness to hyperparameters. Additionally, we opted for a linear scheduler with warmup to gradually adjust the learning rate, allowing the model to stabilize during the initial training phase and potentially improve convergence. These techniques were selected based on their proven effectiveness in optimizing neural network models and their compatibility with our training objectives.

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

A5: Individually, we expect both AdamW and the linear scheduler with warmup to contribute to improved optimization and convergence during training. When combined, we anticipate a synergistic effect, with the two techniques complementing each other to further enhance model performance. The combined use of AdamW and the linear scheduler with warmup is expected to result in faster convergence, better generalization, and ultimately higher accuracy on our task.

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

A6: Yes, the results align with our expectation. Both AdamW and the linear scheduler with warmup contributed positively to the optimization process, leading to improved convergence and performance as anticipated.

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

A7: To further enhance performance, we could utilize the supervised contrastive (SupCon) loss to improve feature representation learning. Additionally, experimenting with different model architectures or pre-training strategies could also yield improvements in performance.

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

A8: Both models utilize contrastive learning to learn semantic representations. However, the models differ in complexity and implementation and have different applications. For example, SimCLR aims to create a simple and efficient framework for sentence embeddings. On the other hand, SupContrast takes a more robust approach of implementing contrastive learning for general-purpose tasks.

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

A9: By applying dropout, SimCSE introduces noise to the model, encouraging the model to learn more robust features. This effectively creates different perspectives of the same input sentence embedding, simulating the augmentation of data. The model then learns to maximize the similarity between augmented versions of the same embeddings while minimizing the similarity between augmented versions of different embeddings.

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

A10: Yes, the results align with our expectation. Dropout introduces noise to the model, aiding in regularization and encouraging the learning of more robust features, which typically leads to improved generalization performance.

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

A11: To further enhance performance, we could explore different dropout rates, investigate other forms of data augmentation, such as back-translation or word dropout, and experiment with ensemble techniques to combine multiple models trained with different dropout configurations. Additionally, fine-tuning the dropout strategy specifically for our dataset and model architecture could potentially yield even better results.

Contributions

Brandon Szeto implemented the data loading functionality, developed the tokenizer, and constructed the baseline model. His work on data preprocessing and model architecture greatly contributed to the development of the project. In addition, he wrote the abstract, introduction, and related works section of the write up.

Darren Yu implemented additional argument flags for each model (baseline, custom, supcon) to ensure results were being saved. He engaged in pair programming sessions with Nathaniel, actively participating as the driver. He also contributed significantly to the visualization aspect of the project by developing all plotting code. Darren wrote the discussion and helped proofread the paper.

Nathaniel implemented the training code, incorporating the supervised contrastive loss model. He worked with Darren to develop the models using SupCon and SimCLR loss. He also worked with Brandon to develop the fine-tuned models after the implementation of the baseline. He helped write the methods and results of the write up.