CS 224N: MinBERT and Downstream Tasks

Stanford CS224N Default Project

Rita Tlemcani
Department of Computer Science
Stanford University
ritabt@stanford.edu

Cole Sohn
Department of Computer Science
Stanford University
csohn@stanford.edu

Abstract

Generalizing Natural Language Processing (NLP) models to multiple tasks can provide advantages including robustness, improved real-world applicability, and greater data efficiency. This is because multi-task models are capable of utilizing diverse input data types during their training, and can develop a better understanding of patterns in language [1]. The BERT language embedding model achieves high accuracy in downstream language tasks, although separate fine-tuning is necessary for each individual task.[2]. In this paper, we conduct experiments to create a multi-task model that leverages BERT embeddings for various downstream tasks, demonstrating the benefits of multitask learning and achieving good performance. We focus on sentiment classification, paraphrase detection, and semantic textual similarity tasks through exploring different model architectures, loss functions, optimizers, and hyperparameters [Sec 3]. We present our results for the three downstream tasks [Tab 4].

1 Introduction:

Advances in NLP have been driven by the use of pre-trained language models such as BERT, which have shown remarkable performance on a wide range of NLP tasks [3][4].

An advantage of BERT is its ability to capture the semantic information present in text through its deep neural network architecture. However, the potential of BERT can only be realized when it is fine-tuned on a specific downstream task [5].

Multitask learning involves training a single model to perform multiple related tasks simultaneously, rather than using a separate model for separate tasks. The benefits of multitask models in NLP include improved data efficiency, robustness, and generalization [6]. Multitask learning utilizing the BERT model has the potential to produce models capable of performing a wide range of tasks [7].

In this paper, we investigate the performance of one multi-task deep neural network that leverages BERT to create embeddings and branches out into sentiment classification, paraphrase detection, and semantic textual similarity (STS) layers. We experiment with a pure transfer learning approach where we only train the additional task-specific layers added on top of Bert. We also experiment with model ensembling and weight averaging to produce a model with a single BERT stem.

2 Related Work:

Devlin et al. [2] introduced BERT, a pre-training approach for deep bidirectional transformers aimed at achieving high performance on transfer learning. By leveraging BERT as an embedding layer in neural networks tailored to specific natural language processing tasks, Devlin et al. began a trend in research to construct models that could leverage large-scale pre-training.

Expanding on this foundation, Sun et al. [8] proposed novel fine-tuning strategies for BERT on sentence-level tasks such as sentiment analysis, question classification, and topic classification, resulting in highly accurate models. Sun et al. introduced multitask training, a method proposed by Collobert et al. [1], which has shown potential for improving generalization and performance across multiple tasks.

To promote the development of models that perform well on multiple tasks, Wang et al. introduced GLUE [7], a set of evaluation metrics for sentence-level language tasks. The GLUE leaderboard fosters competition within the NLP research community and has encouraged research and development on multitask models.

Sentiment Analysis (SA) is a widely used machine learning task, useful in real-world applications such as movie review processing and political analysis. Hoang et al. [9] demonstrated the potential of fine-tuning BERT on SA and more complex tasks such as Aspect-based SA (ABSA). Sun et al. [10] proposed further work on fine-tuning BERT for SA by converting ABSA to a sentence-pair classification task.

Paraphrase detection is useful for tasks such as linking posts together in online databases such as Quora. Arase et al. [11] explored various methods of fine-tuning BERT for paraphrase detection tasks by modifying BERT to inject phrasal paraphrase relations, achieving promising results.

Semantic Textual Similarity (STS) involves measuring the relatedness of two pieces of text, and is critical to many NLP applications such as text classification, information retrieval, question answering, and machine translation. Ranasinghe et al. [12] explored transfer learning architectures such as siamese neural networks with RNN layers to process the output of BERT and finetune the model to achieve high accuracy on STS tasks.

3 Approach

For our approach we implement a minimal version of the BERT model. We then extend that BERT model with several task-specific transfer layers to produce outputs for multiple downstream sentence-level tasks.

3.1 Minimal Implementation of BERT

MinBert [13] is a minimal implementation of the BERT model [2] provided as start code by the Stanford CS224N teaching staff [14].

BERT is a deep neural network that consists of an embedding layer and 12 transformer encoder layers. The transformer layers utilize multi-head attention and a feed-forward layer, as defined by Vaswani et al. [15]. The model also includes dropout applied to the output of each sub-layer and the sums of the embeddings layer. MinBert is fine-tuned on next-sentence prediction.

Our provided implementation of MinBERT includes a tokenizer for data preprocessing, the embedding layer, the forward pass, and the fine-tuning layer. We implement the transformer layer and multiheaded self-attention components described in the original model's paper [2].

For the transformer layer, we calculate multiheaded attention following the methods described by Vaswani et al. [15]. We use linear layers to generate the key, value, and query for each token, and then apply Equation 1. The queries, keys, and values are packed together into Q, K, V respectively, and d_k is the dimension of the keys. We also apply an attention mask to the result of $QK^\top$ to mask out padding tokens.

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{1}$$

We also implement various sections of the full transformer layer such as the Add and Norm layer defined [15], and the forward pass for a layer.

To test our BERT implementation, we validate on a downstream sentiment analysis task. We use the SST [16] and CFIMDB [17] datasets to achieve a base accuracy on sentence-level sentiment classification [Tab 1]. We interpret the BERT pooler output with a dropout layer for a sentence. We

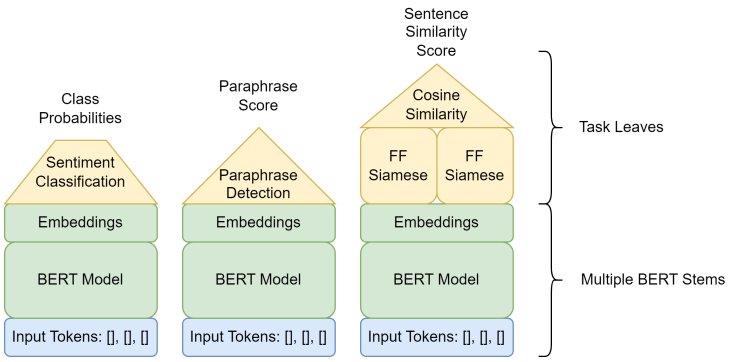

Figure 1: Ensembled Multitask BERT Model with Multiple Stems

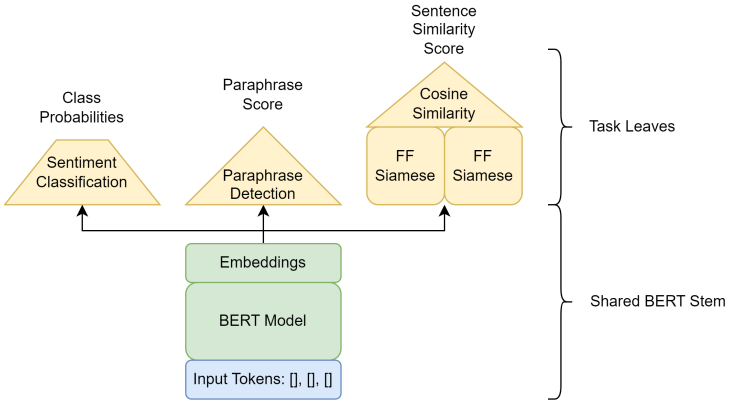

Figure 2: Multitask BERT Model with Single Stem

then project the embeddings to have length n for each sentiment category with a fully connected linear layer. We implement the step function of the Adam Optimizer by Kingma et al. [18] to train the sentiment classifier.

We evaluate the accuracy of our model for sentiment analysis on the the SST[16] dataset with both finetuning and no-finetuning of weights in the BERT stem. We achieve the accuracies shown in Table 1.

3.2 Multitask Learning

We will refer to the BERT embedding layers as the *BERT stem* and task-specific layers as *task branches*. In order to generalize BERT to multiple tasks, we fine-tune the model for each task individually with additional BERT pre-training [**Extension #1**]. This approach provides benefits such as allowing us to ensemble the model with multiple BERT stems [**Other Improvements #5**][Fig 1] as well as average the weights together to create a model with a single BERT Stem [Fig 2]. This approach can also help determine weight initialization and baseline hyper-parameters for additional multitask finetuning.

We implement extra features that improve training performance on all downstream task such as weight decay and grid-based hyper-parameter search. We also implement various improvements to training speed such as downsampling, single-task training, and calculating a training accuracy estimate from the output of each training epoch as opposed to re-evaluating our model on the training set.

When training our model on multiple tasks, we commonly encountered model overfitting. We combat overfitting by increasing dropout probability, increasing the weight decay of our model, and

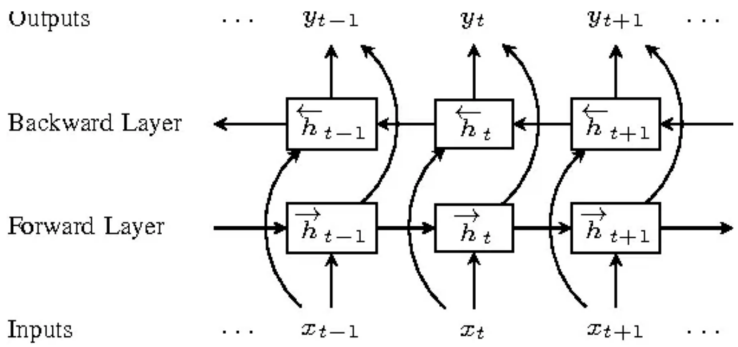

Figure 3: Visual of a Bi-Directional RNN from [20]

reducing layer sizes where applicable. We optimize hyperparameters [**Other Improvements #6**] using grid-search after finding good baseline hyperparameter values.

For our task-specific layers we experiment with multiple model architectures with feed-forward and RNN approaches as well as different optimizers and loss functions including Multiple Negatives Ranking Loss [**Extension #2**] for paraphrase detection, which we implement from scratch. We also experiment with Cosine Similarity [**Extension #3**] as a metric for trainig our STS task layers.

3.2.1 Sentiment Classification

We experiment with several approaches for additional training and fine-tuning BERT for Sentiment Classification. For each branch architecture in experiments, we take the hidden layer output of BERT [2] or the pooler output of BERT, and output a vector of size n, where n is the number of sentiment categories. In the case of the Stanford SST dataset [16], $n = 5$ with values corresponding to the probability of each category. We set loss to be the cross entropy 2 of this ouput with labels t being one-hot vectors representing the correct category index for a given sample, and $p(i)$ being the softmax probability output from our model of the i^{th} class. We use the AdamW optimizer.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log p(t) \tag{2}$$

We train on the Stanford SST Dataset [16] with various branch architectures, hyperparameters, and fine-tuning the BERT Stem.

For our baseline, we implement a dropout layer followed by a single linear layer. We then experiment with additional linear layers of varying hidden sizes with the goal of introducing more complexity to our model. This approach, however, was prone to overfitting due to the increased number of parameters.

Following the work of Qiang et al. [19], we experiment with replacing our output linear layer with Bidirectional GRU (BiGRU). BiGRU uses two separate GRU layers, one to process the input sequence forward and one to process the input sequence backward. Each GRU layer contains a series of nodes or neurons that maintain a hidden state, which is updated at each time step based on the input and the previous hidden state. This allows the model to capture past and future context. The figure 3 from [20] illustrates the concept of a Bi-Directional RNN.

3.2.2 Paraphrase Detection

For paraphrase detection, we take as input two sentence representations and output a binary digit that encodes whether the two sentences are paraphrases of one another. For all our experiments on paraphrase detection, we use pooled outputs from BERT as our input, and we use the AdamW optimizer. We experiment with multiple model architectures and loss functions.

Our first baseline approach is to concatenate sentence pairs' BERT pooler outputs of dimension d into a single vector, and apply a dropout layer and a linear layer of size $2d \times 1$. With this setup and Binary Cross Entropy Loss, we achieve the results shown [Tab 2].

Binary Cross Entropy Loss (BCE Loss) measures the difference between a predicted probability distribution and the true distribution for binary classification. The equation is shown 3 where N is the total number of samples, y_i is the true binary label of the i^{th} sample, and p_i is the predicted probability of the i^{th} sample being in class 1.

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{3}$$

We experiment with multiple linear layers, but saw similar over-fitting problems to sentiment classification. 3.2.1

We experiment using Multiple Negatives Ranking (MNR) Loss [**Extension #2**] but saw decreased accuracy [Tab 2]. Multiple Negatives Ranking Loss is used when the goal is to learn a ranking function that can distinguish between different levels of similarity between pairs of sentences, but for this task our goal is binary classification, so Binary Cross Entropy was sufficient as it is a simpler loss function that is easier to optimize.

$$L_{MNR} = -\frac{1}{K}\sum_{i=1}^{K}[S(x_i, y_i) - \log \sum_{j=1}^{K} e^{S(x_i, y_j)}], i \neq j \tag{4}$$

MNR Loss is shown 4 where $S(x_i, y_i)$ is the score of a similar pair and $S(x_i, y_j)$ is the score of a dissimilar pair. We implement MNR Loss from scratch making use of Pytorch [21] tensors. For each batch, we select pairs with true labels from the Stanford SST dataset [16]. We set K to be the number of true pairs found in our current batch. Next, we calculate scores for similar pairs and dissimilar pairs by matching each first sentence with every other second sentence and running all pairs through our model to calculate $S(x_i, y_i)$ and $S(x_i, y_j)$. We use these scores with our vectorized implementation of MNR Loss [Eq 4].

3.3 Semantic Textual Similarity

For the Semantic Textual Similarity (STS) task, we are provided two sentences and output a continuous score between 0-5 that expresses how similar sentences are and what semantic similarity categories they fall into [22].

Our task layers for STS take two sentence pooled embeddings from BERT as input, and output a single logit between 0-5.

For each of our experiments for the STS task, we use mean squared error loss 5. Where y_i is the true value of the target variable for the i-th instance in the dataset, $\hat{y}_i$ is the predicted value for the same instance, and n is the total number of instances in the batch.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

.

Our baseline approach is similar to paraphrase, where we concatenate pooler outputs for both sentences, and predict the digit with a single linear layer. Our baseline achieves low accuracy due to the high-complexity of STS tasks.

For our second approach, we follow the implementation from [23]. We implement Siamese Linear Layers on the output of BERT for each embedding, where the input and output size is the size of the embedding. Next, we use Cosine Similarity [**Extension #3**] [Eq 6] to calculate the similarity between the output of the embedding pairs. Cosine Similarity outputs a score betweeen [-1, 1], so we normalize and multiple our result by 5 to correspond with our training labels. Cosine Similarity is shown 6 where A and B represent a pair of feature vectors, and ϵ is a small number to avoid

divide-by-zero errors. We use the MSE loss function [Eq 5] on the cosine similarity scores between the pair of vectors in each batch.

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\max(|\mathbf{A}||\mathbf{B}|, \epsilon)} \tag{6}$$

We also experimented with replacing our linear layers with bi-directional GRU [23], but saw significant overfitting to our training data. This is because the number of parameters in bi-directional GRU is great enough to memorize our training data for STS. Note that we train on only 6,041 examples from the STS benchmark dataset. Due to this relatively low sample-size, overfitting is a concern.

We experiment with additional datasets [**Extension #9**] through data augmentation by reformatting the QQP training data to train our STS model. We assign 1 QQP labels a random STS value between 4-5 and we assign 0 labels a random STS value between 0-4. While this approach is ad-hoc, it works to improve both the generalization and accuracy of our model.

4 Experiments

4.1 Data

For sentiment classification, we train and validate on the Stanford Sentiment Treebank Dataset (SST) [16], which consists of movie reviews parsed by the Stanford Parser [24] annotated by human judges. For this paper, we use 8,544 examples to train, 1,101 examples for dev validation and 2,210 examples for our test set.

For initial training and validation of the sentiment classification task, we train and validate on the CFIMDB dataset which contains 1,701 test examples, 245 dev examples, and 488 test examples [17]. The CFIMDB dataset consists of binary labels while we want our model to perform well on four output sentiment classification categories so we only use the CFIMDB dataset for initial validation of our minBERT implementation.

For paraphrase detection, we train and validate on the Quora Question Pairs Dataset (QQP) [25], which contains pairs of sentences and a binary label indicating whether or not they are paraphrases. We train on 141,506 examples from QQP, use 20,215 examples for dev validation, and 40,431 examples for test validation.

For semantic textual similarity, we use the SemEval STS Benchmark Dataset (STS-B) [22]. This dataset consists of pairs of sentences and a continuous score between 0-5 that represent sentence similarity. These scores can be interpreted as categories of similarity: Not Equivalent (0), Not Equivalent but Same Topic (1), Not Equivalent but Share Some Details (2), Roughly Equivalent but Some Details Differ (3), Mostly Equivalent (4), Completely Equivalent (5). We train on 6,041 examples from STS-B, and use 864 examples for dev validation and 1,726 examples for test validation.

4.2 Evaluation Method

To evaluate our model on sentiment classification, we compute the fraction of correctly classified examples on our dev and test SST datasets.

To evaluate our model on paraphrase detection, we compute the sigmoid of the outputs of our model to ensure outputs are between 0 and 1. We then round to the nearest integer, so any outputs less than 0.5 will be categorized as false and any greater than or equal to 0.5 will be categorized as true. We then take the ratio of correctly predicted labels to incorrectly predicted labels as our accuracy.

To evaluate our model on semantic textual similarity, we use Pearson Correlation between our outputs and the ground-truth labels. Pearson Correlation measures the strength of a linear relationship between two variables. This means, if datapoints are scattered around a straight line, they will have a high correlation. We use this for calculating STS accuracy as we want our outputs to have a linear relationship with the ground-truth labels, so they do not have to be normalized. This also allows our outputs and labels to be continuous when compared to the more simple metrics used for classification accuracy. We calculate the Pearson Correlation using the Numpy library [26].

Model	SST Train	SST Dev
Single Layer FF$_{noFT}$	0.270	0.350
Single Layer FF$_{FT}$	0.463	0.510
Multi-Layer FF$_{noFT}$	0.279	0.389
Multi-Layer FF$_{FT}$	0.664	**0.530**
BiGRU$_{noFT}$	0.517	0.476
BiGRU$_{FT}$	0.796	0.508

Table 1: SST accuracy for various model architectures

Model	QQP Train	QQP Dev
CE FF$_{noFT}$	0.615	0.645
CE FF$_{FT}$-CE	0.812	**0.787**
MNR FF$_{noFT}$-MNR	0.271	0.624
MNR FF$_{FT}$-MNR	0.273	0.624
Siamese-FF$_{noFT}$	0.593	0.624
Siamese-FF$_{FT}$-CE	0.629	0.625

Table 2: Paraphrase Detection accuracy for various model architectures

4.3 Experimental Details

For training on SST, we use a learning rate of $2e-05$, a dropout probability of 0.5, and a weight decay of 0.01. For training on QQP, we use a learning rate of $1e-05$, a dropout probability of 0.5, and a weight decay of 0.01. For training on STS, we use a learning rate of $1e-05$, a dropout probability of 0.5, and a weight decay of 0.1. We adjust hyper-parameters as necessary to improve performance on each task, and further optimize through grid-based search.

The architectures of each model used are explained in detail in Section 3. We perform training on an NVIDIA GTX 1080 video card on a local machine running Windows 11. Training times for SST were roughly 1 minute/epoch, for QQP 40 minutes/epoch, and for STS 1 minute 45 seconds/epoch. For every experiment, we train for 10 epochs and store the model that reached the highest dev accuracy during training.

4.4 Results

We provide the results to our experiments on sentiment classification 1, paraphrase detection s2, and semantic textual similarity 3. We provide the results of our final models 4 achieved by ensembling our three task specific models with multiple BERT branches and one BERT branch with averaged weights.

5 Analysis

We experiment with different model architectures and found that the more complex architectures (such as multiple linear layers and Bi-Directional GRU) improved accuracy on our training set but not our dev set predictions, leading to increased overfitting and “memorization" of the training data.

Approaches such as weight decay, using simpler models, and increasing drop out rate reduced over-fitting but they persisted. These problems likely persist because we fine-tune our model on each task individually to improve performance. Investigating methods to improve performance when fine-tuning our model on all tasks at once such as gradient surgery (Yu et al. [27]) or using Mixout (Lee et al. [28]) may improve generalizability of our model.

We also achieve higher baseline accuracies for paraphrase detection than either classification task as paraphrase detection produces a binary label with a much higher probability of guessing the answer at random correctly.

Model	STS-B Train	STS-B Dev
FF_{noFT}	0.126	0.253
FF_{FT}	0.586	0.398
Siamese-FF_{noFT}	0.206	0.232
Siamese-FF_{FT}	0.813	**0.514**
Siamese-$BiGRU_{noFT}$	0.228	0.218
Siamese-$BiGRU_{FT}$	0.740	0.369

Table 3: STS Pearson correlation scores for various model architectures

Model	SST	QQP	STS
Ensembled Multi-BERT Stem Dev	0.530	0.787	0.514
Ensembled Multi-BERT Stem Test	0.535	0.789	0.444
Ensembled Avg Weight Single BERT Dev	0.381	0.706	0.357

Table 4: Final Model dev set results for various model architectures using evaluation metrics discussed in 4.2

6 Conclusion

Our experiments with extending BERT to multiple tasks have shown promising results and directions for further experimentation. By using various model architectures and hyperparameters, we were able to achieve better-than-random accuracy on three distinct NLP tasks: SST, Paraphrase, and STS.

Our approach of using feed-forward and recurrent neural networks on BERT outputs allowed us to leverage the power of pre-trained language models while adapting them to specific downstream tasks. We found that tuning hyperparameters such as learning rate, dropout probability, and weight decay was critical in achieving our best performance on each task.

Overall, our experiments demonstrate the effectiveness of fine-tuning BERT for multiple NLP tasks and the importance of carefully tuning hyperparameters. Our experiments also show the problems with fine-tuning BERT on individual tasks, such as overfitting. As future work, we hope to experiment with methods for fine-tuning BERT on multiple tasks simultaneously to acheive greater generalizability.

References

[1] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, sep 2019.

[4] Wenliang Dai, Samuel Cahyawijaya, Zihan Liu, and Pascale Fung. Multimodal end-to-end sparse model for emotion recognition, 2021.

[5] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese Computational Linguistics (CCL)*, 2019.

[6] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy, July 2019. Association for Computational Linguistics.

[7] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

[8] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2020.

[9] Mickel Hoang, Oskar Alija Bihorac, and Jacobo Rouces. Aspect-based sentiment analysis using BERT. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 187–196, Turku, Finland, September–October 2019. Linköping University Electronic Press.

[10] Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. *CoRR*, abs/1903.09588, 2019.

[11] Yuki Arase and Jun'ichi Tsujii. Transfer fine-tuning: A BERT case study. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019.

[12] Tharindu Ranasinghe, Constantin Orasan, and Ruslan Mitkov. Semantic textual similarity with Siamese neural networks. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1004–1011, Varna, Bulgaria, September 2019. INCOMA Ltd.

[13] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes, 2020.

[14] Cs224n: Natural language processing with deep learning.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[16] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[17] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[19] Qiang Lu, Zhenfang Zhu, Fuyong Xu, Dianyuan Zhang, Wenqing Wu, and Qiangqiang Guo. Bi-gru sentiment classification for chinese based on grammar rules and bert. *International Journal of Computational Intelligence Systems*, 13:538–548, 2020.

[20] Madhu Ramiah. Bi-directional rnn amp; basics of lstm and gru, Jan 2021.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[22] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.

[23] Tharindu Ranasinghe, Constantin Orasan, and Ruslan Mitkov. Semantic textual similarity with Siamese neural networks. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1004–1011, Varna, Bulgaria, September 2019. INCOMA Ltd.

[24] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics.

[25] SambitSekhar. First quora dataset release: Question pairs, Feb 2017.

[26] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[27] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning, 2020.

[28] Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. Mixout: Effective regularization to finetune large-scale pretrained language models, 2020.