Question and Answering on SQuAD 2.0: BERT Is All You Need

Sam Schwager
Department of Computer Science
Stanford University
Stanford, CA 94305
sams95@stanford.edu

John Solitario
Department of Computer Science
Stanford University
Stanford, CA 94305
johnny18@stanford.edu

Abstract

In this paper, we present a series of experiments using the Huggingface Pytorch BERT implementation for questions and answering on the Stanford Question Answering Dataset (SQuAD). We find that dropout and applying clever weighting schemes to the loss function leads to impressive performance. More specifically, we ensemble across three models: one leveraging dropout, another weighting questions with answers more heavily than those without, and finally, one weighting the answer span start more heavily than the end. This leads to a test set **F1 score of 76.545 and a test set EM score of 73.609.**

1 Introduction

Question answering systems have captured the minds of budding computer scientists since the early 1960s due to their evident usefulness in a variety of domain-specific tasks (5). In general, question answering covers a wide field of systems that automatically answer questions posed in a natural language. A subset of question answering, known as machine reading comprehension, focuses on finding an answer to a given question in a specific paragraph or document, and the majority of early natural language processing (NLP) research specifically focused on machine reading comprehension (16). However, systems had little success with machine reading comprehension until the production of large-scale, labeled datasets, which allowed researchers to build supervised neural systems. The Stanford Question Answering Dataset (SQuAD) is a prime example of one of these large-scale, labeled datasets for reading comprehension.

The original version of SQuAD, SQuAD 1.1, only contains questions that are guaranteed to have answers in the associated document (12). Furthermore, various systems have surpassed human-level performance on SQuAD 1.1. However, in the context of SQuAD 1.1, these models only have to find the answer span that seems most related to the given question, which reduces model robustness. Thus, despite human-level performance, models built for SQuAD 1.1 lack true language understanding (7) (18). In order to remedy the issues with SQuAD 1.1, Rajpurkar et al. develop SQuAD 2.0, which combines answerable questions from SQuAD 1.1 with new, unanswerable questions about the same paragraphs (11). For the scope of this project, we focus on the more advanced SQuAD 2.0, which was released in mid-2018.

Early models built for SQuAD 2.0 substantially underperformed human-level performance until the release of BERT by Devlin et al.(4). Now, as of March 2019, nineteen out of the top twenty submissions on the SQuAD 2.0 leaderboard leverage BERT in some capacity, and, by relying heavily on BERT, the top submissions have almost achieved human-level performance on SQuAD 2.0. Thus, BERT has become foundational to state-of-the-art machine reading comprehension systems.

In this paper, we fully explore the power of BERT. First, we finetune the original BERT model by experimenting with a variety of parameters, including learning rate, dropout, batch size, and training epochs. Second, we experiment with a bespoke output layers to bolster BERT's shallow output layer. Third, we tweak the loss function in a handful of ways in an attempt to improve BERT without altering the underlying model architecture. Finally, throughout our experimentation we develop a variety of BERT models and certain models perform better on certain tasks. Therefore, we leverage an intelligent version of ensembling in our final model, which achieves an **F1 score of 76.545** and an **EM score of 73.609** on the test set.

2 Related Works

Introduced in October 2018, **BERT** stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Devin et al. design BERT to pre-train deep bidirectional language representations by jointly training on both left and right contexts of a given word in all layers of their model. Underlying BERT's model architecture is a multi-layer bidirectional Transformer encoder, originally implemented by Vaswani et al. (21). Transformers dispense entirely with recurrence and convolution mechanisms and rely solely on attention mechanisms, which significantly decreases training time.

Furthermore, the work of Devin et al. follows from a long history of pre-training general language representations. Specifically, we note GLoVE embeddings developed by Pennington et al., which rely on word-to-word co-occurrence statistics, and ELMo embeddings developed by Peters et al., which generalize traditional word embeddings by integrating context-sensitive features from language models (9) (10). Nonetheless, pre-trained embeddings have become integral to NLP systems, offering significant improvements over embeddings learned from scratch (17).

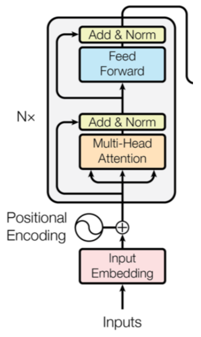
Moving forward, Devin et al. achieve state-of-the-art results on a wide range of NLP tasks by fine-tuning BERT. Typically, fine-tuning approaches involve pre-training some model architecture on a language model objective before fine-tuning that same model for a supervised downstream task (6). With a fine-tuning approach, models only need to learn a few parameters from scratch. Additionally, there has been work showing the effectiveness of transfer learning from supervised tasks with large-scale datasets, such as natural language inference and machine translation (3) (8).

3 Approach

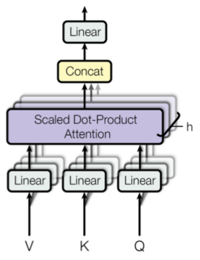
3.1 Baseline

We leverage an adjusted version of the BiDAF model laid out in Seo et al. as our baseline model (13). However, in contrast to Seo et al., our BiDAF model does not include a character-level embedding layer. Our baseline BiDAF model has five primary layers, which follows from the high-level structure commonly found in SQuAD models. As the default, our model was trained with a hidden size of 100.

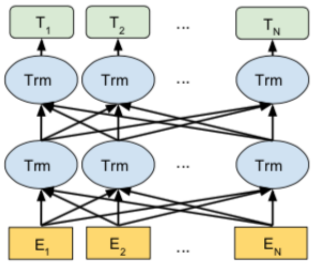
1. **Embedding Layer:** Embedding look-up to convert indices into GLoVe word embedding vectors. 2,196,017 GloVe vectors each with an embedding size of 300.
2. **Encoder Layer:** Two-layer bidirectional LSTM to encode the embed sequence (15).
3. **Attention Layer:** BiDAF layer that allows information to flow from the context to the question and from the question to the context.
4. **Model Encoder Layer:** One-layer bidirectional LSTM to refine the sequence of vectors after the attention layer.
5. **Output Layer:** Produces a vector of probabilities corresponding to start and end positions and then uses a softmax to output start and end pointers for the answer span.



(a) Transformer Encoder



(b) Multi-Head Attention



(c) BERT Architecture

Figure 1: Key Components for BERT Architecture (4) (21)

3.2 BERT Model

3.2.1 Model Overview

As noted in **Section 2**, BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (see **Figure (1)(b)**) (21). BERT relies on several layers of Transformer blocks (see **Figure (1)(a)**, note positional embeddings are not used in BERT). Each Transformer block consists of two sub-layers, a multi-head self-attention mechanism followed by a simple, position-wise fully connected feed-forward network. Residual connections exist around each of the two sub-layers, and dropout, following after each sub-layer, provides layer normalization.

• **Multi-Head Attention:** We can describe an attention function as mapping a query (Q) and a set of key-value pairs (K, V) to an output vector (O), where the output is a weighted sum of the values. The weight assigned to each value is computed by a compatibility function of query (Q_i) with the corresponding key (K_i). Within BERT, the input matrix X, which either comes from the input embeddings or the previous hidden layer, is separately projected using the following weight matrices:

$$XW_i^Q = Q_i \in \mathbb{R}^{input \times d_q};\;\; XW_i^K = K_i \in \mathbb{R}^{input \times d_k};\;\; XW_i^V = V_i \in \mathbb{R}^{input \times d_v}$$

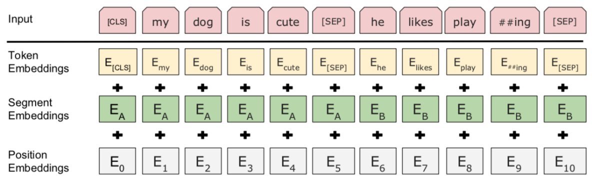
Instead of performing a single attention function, the multi-head attention mechanism linearly projects the queries, keys, and values h times with different, learned linear projections to d_v, d_k and d_q dimensions, respectively. On each of these projected versions of queries, keys, and values, the multi-head attention mechanism performs the attention function in parallel. The output vectors are then concatenated and once again projected, resulting in an output vector (O). Multi-head attention allows the model to jointly attend to information from different subspaces at different positions which bolsters the effectiveness of the attention (see **Figure (1)(b)**).

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O;\; head_i = Atten(Q_i, K_i, V_i)$$

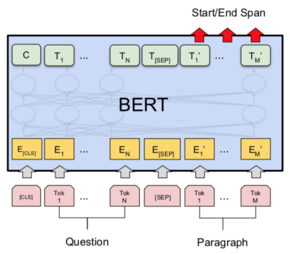
• **Feed Forward:** The feed forward network, which comes after the multi-head attention mechanism, consists of two linear transformations with ReLU activitation in between.

$$FFN(O) = ReLU(OW_1 + b_1)W_2 + b_2$$

• **Input Representation:** BERT can handle single sentences or pairs of sentences with a one token sequence. The input representation for a given token is constructed by summing over three different embeddings, corresponding to the token, segment, and position (see **Figure (2)(a)**).



(a) Input Embeddings



(b) Adjustments for QA

Figure 2: BERT for Question Answering (4) (21)

1. **Token Embeddings:** drawn from WordPiece embeddings (19).
2. **Segment Embeddings:** first token of every sequence is denoted with $[CLS]$. Sentence pairs separated with $[SEP]$. Learned sentence A embedding for every token of the first sentence and a sentence B embedding for every token of the second sentence.
3. **Position Embeddings:** learned and support sequence lengths up to 512 tokens.

Devin et. al create two versions of the underlying BERT model, BERT$_{BASE}$. and BERT$_{LARGE}$. **For the scope of this project, we only leverage BERT**$_{BASE}$, where the Transformer consists of 12 transformer blocks and each transformer block has 12 attention heads. Furthermore, BERT$_{BASE}$ is trained with a hidden size of $h = 768$ and a max sequence length of 512. Therefore, the overall model has approximately 110 million parameters.

• **Pre-training Procedure:** Before fine-tuning BERT, Devin et. al pre-train BERT on two novel unsupervised prediction tasks: masked language model (LM) and next sentence prediction. The masked LM task involves masking some percentage of input tokens at random, and then predicting only those masked tokens. Furthermore, the next sentence prediction task is specifically designed to bolster question answering ability, as developing an understanding of the relationship between two sentences is not directly captured by language modeling. In the prediction task, the model determines if a given sentence B proceeds the sentence A. Specifically, when choosing the sentences A and B for each pre-training example, 50% of the time B is the actual next sentence that follows A, and 50% of the time it is a random sentence from the corpus. In this project, we fine-tune each of our models by starting with the pre-trained BERT model.

3.2.2 Fine-Tuned Model

• **Shallow Output Layer:** In order to fine-tune BERT$_{BASE}$ we follow the same approach of Devin at al. We train two new sets of parameters for the SQuAD 2.0 task, a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$, where H is the hidden size of 768. We train S and E by passing the BERT output through a Linear Layer. More specifically, given $T_i \in \mathbb{R}^H$ (final hidden vector from BERT for the i^{th} input token), the probability of word i being the start of the answer span is computed as a dot product between T_i and S, followed by a softmax over all of the words in the paragraph. The end of the answer span is similarly calculated, and the maximum scoring span is used as the prediction (see equations below and **Figure (2)(b)**). We leverage the power of BERT by using the $BERT_{SMALL}$ model implemented HuggingFace (1).

$$P_i^S = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}} \; ; \; P_i^E = \frac{e^{E \cdot T_i}}{\sum_j e^{E \cdot T_j}}$$

• **Bespoke Output Layer:** In an attempt to bolster the original BERT model, we decide to build a model with a bespoke output layer. The original shallow output layer projects directly from a vector of size 768 to an output vector of size 2, so the output layer can only learn a linear transformation. Therefore, we replace the shallow output layer with a single hidden layer with ReLU activation:

$$\mathbf{x}_{output} = ReLU(\mathbf{W}_1 \mathbf{T} + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

where $\mathbf{W}_1 \in \mathbb{R}^{256\times768}$, $\mathbf{W}_2 \in \mathbb{R}^{2\times256}$, $\mathbf{b}_1 \in \mathbb{R}^{256}$, and $\mathbf{b}_2 \in \mathbb{R}^2$. Furthermore, when compared to the shallow output layer, $(\mathbf{x}_{output})_0 = P_i^S$ and $(\mathbf{x}_{output})_1 = P_i^E$ for the i^{th} input token.

• **Loss Function Adjustments:** In the original BERT model, the loss from start and end logits are weighted equally in the loss function. However, we first note that the start loss is probably more important than the end loss when finding the correct answer span. Second, we note that the start and end loss are probably more important when a question has an answer than when a question does not have an answer. We explore both of these claims in **Section 4.3**. Thus, we build three models, each with a different loss function. Each of these models leverage the original BERT model.

1. Start loss weighted twice as much as the end loss (**Note:** we did not try weighting the end loss twice as much as the start loss due to initially poor results).
2. Loss for questions **with** answers weighted twice as much as questions without answers.
3. Loss for questions **without** answers weighted twice as much as questions without answers (for experimental robustness).

• **Intelligent Ensembling:** Since we create a variety of BERT models, we decide to ensemble across three different models in our final model. For a given question, our ensemble model analyzes the output of each BERT model and then chooses the answer with the relative, highest probability. However, if any of the models predict that the question does not have an answer, our ensemble model will also predict no answer. Ensembling across different versions of the same model has been proven successful for question and answering tasks per Wang et al. (20).

4 Experiments

4.1 Data

Briefly described in **Section 1**, we use SQuAD 2.0, which is a reading comprehension dataset consisting of questions posed by crowdworkers on a set of Wikipedia articles. Furthermore, the answer to every question is a segment of text from the corresponding reading passage or the question is unanswerable. For the scope of our project, we use the official SQuAD 2.0 train and dev sets. We leverage the two datasets in the following way:

- `train` (129,941 examples): All taken from the official SQuAD 2.0 training set.
- `dev` (6,078 examples): Half of the official dev set, randomly selected.
- `test` (5,921 examples): The rest of the official dev set, plus hand-labeled examples.

4.2 Evaluation Method

To evaluate our models we use the standard SQuAD performance metrics: Exact Match (EM) score and F1 score. For our project, we focus on the EM and F1 scores with respect to the dev set.

- **Exact Match:** A binary measure of whether the system output matches the ground truth answer exactly.
- **F1:** Harmonic mean of precision and recall, where precision = (true positives) / (true positives + false positives) and recall = true positives / (false negatives + true positives). F1 score = (2×prediction×recall) / (precision + recall). (**Note:** In SQuAD, precision measures to what extent the predicted span is contained in the ground truth span, and recall measures to what extent the ground truth span is contained in the predicted span.)

4.3 Experimental Details

We train all of our models on Azure Virtual Machines each equipped with an Nividia NV6 GPU. Additionally, due to memory constraints on the virtual mahcines, we use a batch size of 6 for all of our experiments using BERT. As a baseline, we trained the given BiDAF model using dropout with a learning rate of 0.5 for 30 epochs. Afterward, we started experimenting with the Huggingface PyTorch implementation of the BERT base model. We originally used a learning rate of $5\text{e}{-5}$, which led to results significantly worse than those detailed in the original BERT paper. However, after reviewing other BERT implementations and experimenting with the hyperparameters, we found that a learning rate of $3e{-}5$ led to significantly better results. Upon further review, we found that the creators of Huggingface use this learning rate to achieve their best results, and they also emphasize the importance of the learning rate for model performance.

After settling on a learning rate of $3e{-}5$, we decided to experiment with using dropout before the output layer, which the Huggingface implementation had notably decided not to use. We used a dropout probability of 0.1 and trained models for both two and three epochs. We noticed that dropout did very little to reduce overfitting and actually led to worse performance in general. Nonetheless, we found that the models trained with dropout were generally better at detecting the correct start and end positions for dev examples with answers (and significantly worse at detecting those with no answer). This phenomenon is further explored in **Section 5**.

We then decided to make more drastic changes to the BERT model by overhauling the structure of the output layer as detailed in the **Section 3.2.2**. Once again, to train this model we used a learning of $3e{-}5$, and despite spotty results in our previous experiments, decided to use dropout for this modified model as a regularization technique to curb the effects of the added complexity.

Finally, unsatisfied with the results of altering the structure of the output layer, we decided to focus our efforts on modifying the model's loss function. First, we decided to test the hypothesis that weighting the loss associated with the model's choice of start position more heavily than the loss associated with the chosen end position would lead to better performance. Intuitively, since English is read left-to-right, we felt that the information carried forward from the start position onward may be more valuable than the information carried backward from the end position. As such, we modified the model's loss to weight the start loss twice as much as the end loss and trained again for two epochs using the same learning rate but without dropout.

We were relatively satisfied with the results of weighting the start loss more heavily than end loss, but we decided to go back to the drawing board to see if we had overlooked anything crucial. after thinking more deeply about the fact that the Huggingface implementation was designed to work with SQuAD 1.1 and 2.0, we decided that we had not accounted for key difference between the two datasets. Since the difference between versions 2.0 and 1.1 is the presence of questions with no answers and the lack thereof respectively, we decided to again see if there was a way to tweak the model's loss. Analyzing the results of our previous experiments, we noticed that the best performing models with respect to detecting questions with no answer often struggled to detect correct answers in questions with answers and vice versa. To address this tradeoff, we decided to train one model that would bias toward detecting questions with no answer and another that would bias toward detecting the correct start and end positions in questions with answers. Specifically, we trained one model with a loss function that weighed the loss for questions with no answer twice as heavily as the loss for questions with answers and another that did the reverse. We trained both of these models using the same learning rate as before and, again, did not use dropout.

5 Results and Analysis

To measure the performance of our models, we considered the overall F1 and EM scores along with the F1 and EM scores on the subset of the data containing questions with answers and the subset containing questions without answers. We found that the BERT base model with a learning rate of $5e{-}5$ gave significant improvement over the BiDAF baseline model (approx. +7 F1 and +7 EM). However, using a learning rate of $3e{-}5$ led to even more drastic improvements over the baseline (approx. +16 F1 and +16 EM). See **Figure 3** for a detailed breakdown of our results.

Furthermore, after training for two epochs with dropout, using a dropout probability of 0.1, we found that overall performance slightly worsened, but our model did slightly better on both scoring metrics for questions with answers. As such, we decided to train for another epoch, 3 in total, to see if the gain for questions with answers could be amplified. Indeed it was, as we achieved an increase of approximately 1.5 points for both F1 and EM for questions with answers. Finally, to combine the expressive power of the model without dropout with the regularization benefits

Model ID	Model Description	Dropout	Learning Rate	Epochs Trained	No Answer F1 / EM	Has Answer F1	Has Answer EM	F1	EM
1	*BiDAF Baseline*	X	0.5	30	-	-	-	59.81	56.7
2	*BERT Small*		0.0005	3	53.0934	81.9576	74.5704	66.9129	63.3761
3	*BERT Small*		0.0005	2	53.8826	82.8881	75.7388	67.7697	64.3468
4	*BERT Small*	X	0.0003	3	61.9003	81.3521	74.7423	71.2133	68.0487
5	*BERT Small*	Last Epoch	0.0003	3	62.2159	82.0746	75.3265	71.7238	68.4929
6	*BERT Small*	X	0.0003	2	71.3068	80.258	73.7801	75.5924	72.491
7	*BERT Small*		0.0003	3	73.5795	77.7345	71.3402	75.5689	72.5074
8	*BERT Small*		0.0003	2	72.6326	79.8446	73.4364	76.0866	73.0174
9	*BERT - Bespoke Output Layer*	X	0.0003	2	66.9508	77.3443	71.5464	71.9269	69.151
10	*BERT - Adjusted Loss Function: start loss weighted twice end loss*	X	0.0003	2	71.2437	78.7914	71.6495	74.8574	71.438
11	*BERT - Adjusted Loss Function: loss weighted twice when no answer*		0.0003	2	59.5013	81.2957	74.1237	69.9359	66.5021
12	*BERT - Adjusted Loss Function: loss weighted twice when has answer*		0.0003	2	72.6641	80.6745	74.4674	**76.4993**	**73.5275**

Figure 3: Experimental details and single-model results

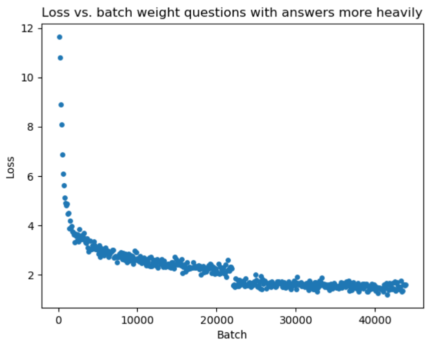
of dropout, we decided to train our previous best model for an additional epoch with dropout. Specifically, this was the model we had trained with a learning rate of 3e−5 for two epochs, and had found that we overfit when we had previously trained for an additional epoch without dropout. Indeed, the trend in dropout leading to higher performance on questions with answers continued. we achieved a gain of approximately 2 F1 and EM points over our previous best for questions with answers.

Next, after modifying the output layer to contain a hidden layer with ReLU activation as described in **Section 3.2.2**, we found a significant decrease in performance. In particular, we trained this modified model for two epochs with dropout (again with probability 0.1) and a learning rate of 3e−5. Overall, saw a decrease of approximately 4 F1 and EM points compared to our previous best.

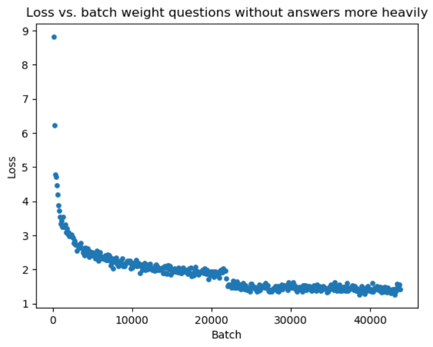
Moving forward, the results of our modified loss function experiments indicated that weighting the start loss more heavily than the end loss, while enticing, led to only slightly different results. On the other hand, we found much more significant changes after performing experiments where we weighted the loss for questions without answers differently from that for questions with answers. Specifically, weighting the loss for questions with answers twice as much as that for questions without answers led to some improvement across the board. However, the reverse led to a significant decrease in performance. Intuitively, weighing the loss for questions that have answers more heavily than that for questions without answers may give the model more power to learn the complex relationship between the start and end positions of answers of variable lengths across the data set, even if that trades off in the form of slightly decreased performance on questions without answers.

On the other hand, it makes sense that weighting the performance on questions without answers over the performance on questions with answers could lead to issues, since modeling answers is inherently more complicated in many ways than modeling the lack thereof. After all, a question without an answer always has the same start and end position (namely -1 and -1), while the start and end positions for questions with answers vary drastically from example to example. **Figure 4** shows how our adjusted loss decreases over the training periods.

Finally, in order to combine the best aspects of the various models resulting from our experiments, we decide to use ensembling, as detailed in **Section 3.2.2**. We considered various ensembling strategies, but we ultimately decided to choose the simplest set of models that would emphasize the most powerful aspects of the various models we had created. Specifically, we used three models in our ensemble: basic model with dropout, weighting questions with answers more



(a) Questions with answers weighted more heavily



(b) Questions without answers weighted more heavily

Figure 4: Change in loss with respect to the the number of batches (training periods)

heavily, and weighting the start of the answer span more heavily than the end. This led us to choose **models 6, 10, and 12**. **This led to a test F1 score of 76.545 and a test EM score of 73.609.**

6 Conclusion

Overall, in this paper, we explore the full power of the basic BERT model by experimenting with a variety of hyperparameters, including the learning rate, dropout, and training epochs. Furthermore, we experiment with a bespoke output, layer built on top of BERT, that replaces the shallow output layer in the original model. We also extensively explore tweaking the BERT loss function. In particular, we try weighting the start loss twice as much as the end loss, and we test weighting questions with answers more than questions without answers and vice versa. Finally, we ensemble across three models that collectively emphasize the best findings of our various experiments to achieve very competitive test set F1 and EM scores. For future work, we note that using our same strategies with respect to BERT$_{LARGE}$ instead of BERT$_{BASE}$ would almost certainly achieve significantly better results. However, we leave this as an exercise to the wealthy reader.

References

[1] J. Chaumond, P. Cistac, T. Wolf, and V. Sanh (Hugging Face). pytorch-pretrained-bert source code. https://github.com/huggingface/pytorch-pretrained-BERT

[2] D. A. Chen, J. W. Fisch, and A. Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. *ArXiv e-prints.*

[3] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* pages 670–680.

[4] J. Devlin , M.W. Chang, K. Lee, and K. Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv e-prints.*

[5] B. F. Green Jr., A. K. Wolf, C. Chomsky, and K. Laughery. 1961. Baseball: An Automatic Question-Answerer. *Western Joint IRE-AIEE-ACM Computer Conference,* pages 219–224.

[6] J. Howard and S. Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. *In ACL. Association for Computational Linguistics.*

[7] R. Jia and P. Liang. 2017. Adversarial Axamples for Evaluating Reading Comprehension Systems. *In Empirical Methods in Natural Language Processing (EMNLP).*

[8] B. McCann, J. Bradbury, C. Xiong, and R. Socher. 2017. Learned in translation: Contextualized word vectors. *In NIPS.*

[9] J. Pennington, R. Socher, and C. D. Manning. 2014. Glove: Global vectors for word representation. *In Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532– 1543.

[10] M. Peters, W. Ammar, C. Bhagavatula, and R. Power. 2017. Semi-supervised sequence tagging with bidirectional language models. *In ACL.*

[11] P. Rajpurkar, R. Jia, P. Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. *ArXiv:1806.038221v1*

[12] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv:1606.05250v3*

[13] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *ArXiv e-prints.*

[14] O. Vinyals, M. Fortunato, and N. Jaitly. 2015. Pointer networks. *In Advances in Neural Information Processing Systems,* pages 2692–2700.

[15] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation 9(8)*, pages 1735–1780.

[16] S. Slade. 1987. The Yale Atrifical Intelligence Project: A Brief History. *AI Magazine Volume 8 Number 4*, pages 67-76.

[17] J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. *In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394.

[18] D. Weissenborn, G. Wiese, and L. Seiffe. 2017. Making neural QA as simple as possible but not simpler. *In Computational Natural Language Learning (CoNLL).*

[19] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, and K. Macherey. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144.*

[20] W. Wang, M. Yan, and C. Wu. Multi-Granularity Hierarchical Attention Fusion Networks for Reading Comprehension and Question Answering. 2018. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. `www.aclweb.org/anthology/P18-1158`

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *In Advances in Neural Information Processing Systems*, pages 6000–6010.