Applying Ensembling Methods to BERT to Boost Model Performance

Charlie Xu, Solomon Barth, Zoe Solis
Department of Computer Science
Stanford University
`cxu2, sbarth, zoesolis @stanford.edu`

Abstract

Due to its wide range of applications and difficulty to solve, Machine Comprehension has become a significant point of focus in AI research. Using the SQuAD 2.0 dataset as a metric, BERT models have been able to achieve superior performance to most other models. Ensembling BERT with other models has yielded even greater results, since these ensemble models are able to utilize the relative strengths and adjust for the relative weaknesses of each of the individual submodels. In this project, we present various ensemble models that leveraged BiDAF and multiple different BERT models to achieve superior performance. We also studied the effects how dataset manipulations and different ensembling methods can affect the performance of our models. Lastly, we attempted to develop the Synthetic Self-Training Question Answering model. In the end, our best performing multi-BERT ensemble obtained an EM score of 73.576 and an F1 score of 76.721.

1 Introduction

Machine Comprehension (MC) is a complex task in NLP that aims to understand written language. Question Answering (QA) is one of the major tasks within MC, requiring a model to provide an answer, given a contextual text passage and question. It has a wide variety of applications, including search engines and voice assistants, making it a popular problem for NLP researchers.

The Stanford Question Answering Dataset (SQuAD) is a very popular means for training, tuning, testing, and comparing question answering systems. It is a large, diverse database of high-quality Wikipedia passages, reading comprehension questions, and accepted answers compiled by Stanford researchers. It uses the Exact Match (EM) and harmonic mean (F1) scores as metrics and maintains a leaderboard to see how the highest performing models compare against one another and against human performance.

In this paper, we describe our implementation of question answering systems that use a multi-BERT model ensemble to achieve superior performance as well as our attempt to implement a synthetic self-training question answering (SSTQA) model. The ensemble was created by manipulating how the BERT models were generated to achieve differences between models and then using various ensembling methods that sought to leverage each individual model's strengths. For SSTQA, we set out to implement both a question answering and a question generation (QG) model so that we could create additional labeled data through question generation and train the QA model on the data, increasing its ability to identify and handle edges cases that cannot be captured generally. However, we were not able to create a working version of the model, so all of our results and analysis stem from our work with ensemble models.

2 Related Work

2.1 Synthetic Self-Training Question Answering

In [1], Sachan and Xing develop a Synthetic Self-Training Question Answering model that addresses the problem of labeled text generation as human labeling is both slow and expensive. Their model generates its own labeled data from unlabeled data on the internet by jointly training Question Answering and Question Generation models. On its own this model does not performs extremely well; when using word counts, it achieves a Mean Average Precision (MAP) score of 0.396, a Mean Reciprocal Rank (MRR) score of 0.401, and a Precision@1 score of 0.179. But as we saw with the top-ranking question answering model on the leaderboard, it can lead to impressive improvements when ensembled with high-performing models. Our project draws inspiration from this original work and from the SQuAD leaderboard for which BERT and SSTQA ensembles are some of the top submissions. We attempted to implement and ensemble SSTQA with other existing QA models, namely BERT and the BiDAF model, to achieve better performance.

2.2 BERT Ensembling

There are a number of papers that generally relate to our work regarding BERT ensembling. The original paper that first introduced BERT mentions using an ensemble of 7 different BERT models that achieves superior performance on QA tasks. [2] Many submissions on the SQuAD leaderboard use models that are either an ensemble of BERT with other models or an ensemble of different BERT models [3]. Moreover, the idea of using multi-BERT ensembles has been applied to tasks other than QA. Cakaloglu et al. used a BERT ensemble to find documents that might be useful for answering a question [4], while Liu et al. used an ensemble of 25 different BERT models within a larger classification model in order to detect fake news.[5]

3 Approach

3.1 BiDAF

[6] We used the BiDAF model, as described in the DFP handout, as a baseline for our model performance. We did not focus our project on the improvement of BiDAF, and we only used it in 2 of our ensembles. Thus, for more details on BiDAF, please refer to the DFP handout or the cited paper.

3.2 BERT

[2] The BERT model was used heavily in our ensemble models and analyses. BERT's superior performance (and our interest in it) stems not only from its model architecture, which leverages a multi-layer bidirectional Transformer encoder, but also its core innovation of pre-training tasks, both of which we will briefly discuss below.

Inputs: As input, BERT takes standard token embeddings and other embeddings that provide important metadata. Since Transformers cannot directly consider the order of inputs, BERT takes positional embeddings as inputs to learn the positions of words in a sentence. Moreover, BERT takes segment embeddings, which enable it to differentiate between sentences in a sentence pair.

Pre-Training: BERT employs two pre-training tasks, Masked Language Model (MLM) and Next Sentence Prediction (NSP). With MLM, a set percentage of words are masked, while a set percentage of words are replaced with a random word. BERT must then correctly predict the masked/replaced word. Through this process, BERT develops a deep bidirectional representation of the data, allowing it to develop a general understanding of the language itself. With NSP, BERT is trained to predict whether the second sentence in a sentence pair is random or not, based on the first sentence. This process allows the model to understand the relationship between two sentences, something highly useful for QA.

Fine-Tuning: After pretraining, BERT produces a sequence of hidden states that are fed into an additional transformer layer for fine-tuning the BERT model to fit a specific task (in our case, QA). This separation of training into pre-training and fine-tuning allows BERT to be used for a variety of NLP tasks without substantial architectural changes.

We used the HuggingFace implementation of BERT [7], albeit with small adjustments discussed later in the paper. For more details on BERT, please refer to the DFP handout or the original paper. [2]

3.3 Synthetic Self-Training QA Model

The model we intended to implement ourselves is the Synthetic Self-Training QA (SSTQA) model, using the Self-Training for Jointly Learning to Ask and Answer Questions paper [1] as a guide. This paper combines a QA model with a question generating (QG) model to generate a model that can train itself on self-generated data. **Question Answering**: The QA model in the paper is inspired by a neural model from the *Attentive Reader* framework from Hermann et al. (2015) [8] and Chen et al. (2016) [9]. It uses two bidirectional LSTMs with dropout regularization and attention to answer a question q for a passage p.

To do this, it first maps all words in the vocabulary to word embeddings. Then, it uses the first bi-directional LSTM with dropout regularization to create encodings of the contextual representations of each word in the passage, $\mathbf{h_t}$, and it uses the second LSTM to do the same for each word in the question. Next, it computes the alignment distribution a based on relevance, and uses that to produce a final output vector through s a weighted combination of the contextual embeddings: $o = \sum_i a_i h_i$. Finally, it predicts the correct answer using the argmax over the output vector's candidate answers.

Question Generation: Due to the similarity of this problem with our seq2seq with multiplicative attention model from the Neural Machine Translation assignments [10], we reused much of the code and encoder/decoder structure for the QG model. Rather than translate from English to Spanish, our model transduces an input statement into an output question.

The QG model uses a seq2seq model with soft attention. Given a sentence from a passage represented as the input sequence $\mathbf{x}$, and a generated question represented as the output sequence $\mathbf{y}$, the goal of the seq2seq model is to find a $\hat{\mathbf{y}} \in \mathcal{Y}$, the space of all possible generated questions, such that $\hat{y} = \text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$. We use token-level predictions to generate the conditional probability, where each token probability y_t is generated using the previous sequence tokens $y_{<t}$ and the input $\mathbf{x}$:

$$P(y_t|y_{<t}, \mathbf{x}) = softmax(Wtanh(W_t[h_t^{(d)}; c_t]))$$

$\mathbf{W}$ and $\mathbf{W_t}$ are both learned weight vectors; $h_t^{(d)}$ is the decoder RNN state at step t; and c_t is the weighted average over token representations from encoder RNN states $h_i^{(e)}$:

$$c_t^{(d)} = \sum_{i=1}^{|x|} a_{it} h_i^{(e)}$$

For the weights, we use soft attention and bilinear scoring followed by softmax normalization:

$$a_{ij} = \frac{exp(h_j^{(e)T} W h_i^{(e)T})}{\sum_{i'} exp(h_j^{(e)T} W h_{i'}^{(e)T})}$$

Training: To run the self-training model, each model is first trained separately on a labeled corpus data set. The QG model then generates more questions from an unlabeled text corpus, which are selected by an oracle and are then answered using the QA model. Finally, the original questions and new questions that are selected by a second oracle are used to stochastically update both models.

3.3.1 Challenges

We experienced some challenges with implementing SSTQA. The QA model that was used, Attentive Reader, was initially built to work on the CNN/Daily Mail data set (mainly designed for extractive text summarization), which is somewhat different from the SQuAD data set. The SSTQA paper went over the original structure of the Attentive Reader, but did not discuss how the Attentive Reader model was adapted to fit the SQuAD data set. Moreover, based on our BiDAF and BERT EM and F1 scores, we felt that both BiDAF or BERT would perform better than the Attentive Reader Model, and that it would make more sense to simply combine the QG model with either BiDAF or BERT for synthetic self-training. There were additional challenges with implementing the QG model.

3.4 Dataset Manipulation + Ensembling

Given the challenges with SSTQA, we decided to explore ways of mimicking the impacts of SST. We also wanted to combine the idea of SST with the concept of ensembling to create a theoretically better ensemble model. To this end, we developed the idea of manipulating the dataset that a BERT model would be trained on to generate various different models and then ensembling them together to create superior models. The details of this process are discussed further in the Experiments section.

Lastly, we tried various methods for ensembling. For regular models, we multiplied the start index probability p_s and the end index probability p_e to generate a joint probability $p_s p_e$ for each prediction. Then, for each question, we would pick the prediction that was associated with the highest joint probability. We modified both the BiDAF implementation code and BERT implementation code to provide us with these probabilities. For ensembling specialized models, we used a model trained generally to choose what questions the predictions from the specialized model would be used for (explained in detail in the Experiments section).

4 Experiments

4.1 Datasets

We used the SQUAD 2.0 reading comprehension dataset, which has 150,000 paragraphs and associated questions, roughly half of which cannot be answered from the given paragraph. We also used the SQuAD 1.1 reading comprehension dataset, which has 100,000+ paragraphs and associated questions, all of which can be answered from the given paragraph. Since both datasets were designed by researchers, we did not need to do any pre-processing.

4.2 Evaluation Method

To evaluate our results, we will use the metrics of Exact Match (EM) and F1, as specified in the default final project handout. We will be comparing our scores not only to the scores of the BiDAF model in last place on the SQUAD leaderboard, but also the scores of the BERT single-model scores listed on the SQUAD leaderboard. Furthermore, to determine how well our models are performing, we will officially use the BiDAF model, as listed in the default project handout, as our baseline. This baseline is fitting, since the overall goal of the project is to produce a complex neural model that performs significantly better than BİDAF. However, since we are seeking to improve upon BERT's performance via ensembling and/or dataset manipulation, our true baseline will be the BERT single-model score that we are able to achieve with a standard configuration and dataset.

4.3 Experimental Details

4.3.1 BERT Hyperparameter Tuning

We first wanted to develop the best possible standard BERT model that we could and use that for the basis of all of our other experiments. To accomplish this goal, We decided to focus on optimizing batch size because we hypothesized that, unlike the other hyperparameters, increasing batch size would not only increase EM and F1 performance, but would also decrease the training time (fewer batches to run through the model each iteration). We initially started off with the standard BERT configuration posed by HuggingFace, which used a batch size of 12, and then increased the batch size from there. The results of our experimentation are listed in the table below.

Table 1: Bert Model Hyperparameter Test Results

BERT Model Type	Batch Size	Learning Rate	# Train Epochs	Dev EM Score	Dev F1 Score
bert-base	12	0.00003	2	70.715	74.301
bert-base	20	0.00003	2	71.438	74.999
bert-base	22	0.00003	2	69.371	72.836
bert-base	24	0.00003	2	N/A	N/A

Batch size did generally increase EM and F1 performance, so long as the batch was divisible by 4. Moreover, we also observed a considerable decrease in training time, with the model of batch size 12 requiring about 10 hours to train and the model of batch size 20 requiring only about 8 hours to train. This decrease in training time was especially attractive, given both our time and resource constraints.

However, when the batch size was 22, we experienced a significant decrease in EM and F1 performance below even that of when the batch size was 12. There was also no significant change in training time. We are not entirely certain as to why this phenomenon occurred, although from the papers that we have seen, batch sizes are always divisible by 4 (e.g. 32, 64, 128), so divisibility by 4 may bear some importance. Lastly, we were not able to proceed to a batch size of 24, since we experienced an out-of-memory error at a batch size of 24.

We opted to not experiment with any other hyperparameters, since adjusting all other hyperparameters listed (with the exception of training rate) would likely lead to longer training times. However, we hypothesize that increasing the number of training epochs and switching to BERT-large would likely have increased EM and F1 performance, but also required more training time.

4.3.2 Dataset Manipulation

We wanted to explore how adjusting the ratio of impossible to possible questions by manipulating the SQuAD 2.0 dataset would affect our BERT model performance. We carried out this process in two ways. We first created a trimmed dataset that only included the possible questions from SQuAD 2.0. While the model trained on this dataset would have fewer training examples to work with, it would be optimized toward answering possible questions, likely increasing its performance on those questions.

We then obtained the SQuAD 1.1 dataset, and, after making some changes, merged it with both the main SQuAD 2.0 dataset and our trimmed dataset. For the full combined dataset, we wanted to observe how the model would perform with the addition data that would likely be created using SST, providing us with a better understanding of how SST might have impacted our model. Furthermore, we wanted to observe how tweaking the percentage of possible questions from 50% to 25% would impact our model. We hypothesized that the model would likely do better overall than a standard BERT model (due to the additional data), albeit with performance on impossible questions somewhat suffering. For the trim combined dataset, we hypothesized that this data would allow us to create a specialized model that would be even better than our model that used the trimmed dataset.

4.3.3 Different Ensembling Methods

As mentioned above, we used different ensembling methods to generate various ensemble models. For non-specialized models, we used a standard method of calculating the joint probability associated with a prediction and then, for each probability, picking the prediction with the highest joint probability.

However, for incorporating the specialized models that were trained solely on questions with possible answers, we adopted a classifying approach, where we would use a general model to determine under the questions for which we would use the general model predictions and the questions for which you would use the specialized model predictions.

Within this classifying approach, we developed two methods. With the first method (Classify-Verify), if the general model predicted a possible answer, we would compare the prediction probability of the general model with that of the specialized model and then pick the prediction with the highest probability. If the general model predicted "No Answer", we would use the "No Answer" prediction.

With the second method (Classify-Trust), if the general model predicted a possible answer, we would automatically use the prediction generated by the specialized model. As with the first method, if the general model predicted "No Answer", we would use the "No Answer" prediction.

4.4 Results

We would first like to clarify various abbreviations used to reference different models:

Table 2: Model Abbreviation Explanations

Abbreviation	Explanation
Bi	Baseline BiDAF model trained on the standard SQuAD 2.0 dataset
BERT	BERT model trained on the standard SQuAD 2.0 dataset
BeFC	BERT model trained on the full combined dataset (SQuAD 1.1 + 2.0)
BeT	BERT model trained on the trimmed dataset (only answerable questions)
BeTC	BERT model trained on the trim combined dataset (SQuAD 1.1 + trimmed 2.0)
-V	used the Classify-Verify ensembling method
-T	used the Classify-Trust ensembling method

4.4.1 Dev Results

Table 3: Model Dev Set Results

Model	Dev EM Score	Dev F1 Score
BiDAF (Baseline)	56.50	60.16
BERT	71.438	74.999
BeFC	72.886	76.259
BeT	38.236	42.086
BeTC	38.105	42.049
Bi+Be	72.639	75.71
Bi+BeFC	72.869	76.153
Be+BeFC	73.593	76.689
Be+BeFC+BeT-V	73.61	76.706
Bi+BeFC+BeT-T	73.873	76.834
Bi+BeFC+BeTC-V	73.61	76.706
Bi+BeFC+BeTC-T	73.873	76.834

As can be seen, our results are in line with what we would expect. All regular BERT models performed significantly better than the baseline model, likely due to BERT's incorporation of both pre-training and fine tuning as well as heavy amounts of attention. In fact, the BeFC model performs better than our worst-performing ensemble model, Bi+Be! Lastly, BeT and BeTC performed worse than baseline because they were designed to only answer questions with possible answers, which accounted for half of the dev dataset.

All of the ensemble models, except for Bi+Be, performed better than any of the individual models. Moreover, as expected, ensembling more models generally increased performance. Interestingly enough, despite having different EM and F1 scores individually, both models, when ensembled with other models, increased performance by the same amount. This result likely means that the difference in their performance stemmed from their answers on unanswerable questions. Moreover, it suggests that BERT may not need extreme amounts of additional data to predict answerable questions well. Lastly, ensemble models created with Classify-Trust beat those created with Classify-Verify, indicating that sometimes the general model might be incorrectly overconfident about its predictions.

4.4.2 Test Results

Table 4: Model Test Set Results

Model	Test EM Score	Test F1 Score
Be+BeFC	73.576	76.721
Be+BeFC+BeTC-T	73.221	76.454
Be+BeFC+BeT-T	73.221	76.454
Be+BeFC+BeTC-V	73.576	76.721

Note: Since we could only make 3 submissions to the test leaderboard, we did not submit Be+BeFC+BeTC-V. However, we know its EM and F1 score because we compared its predictions with that of Be+BeFC and found that they generated the same exact predictions.

As with our dev set, Be+BeFC+BeTC-T and Be+BeFC+BeTC-T models achieved the same performance. However, our Be+BeFC model actually performed better than both Be+BeFC+BeT-T and Be+BeFC+BeTC-T, which wildly differed from our dev results. Moreover, it seems that ensemble models created with Classify-Verify performed better than ensemble models created with Classify-Trust, although we later determined that the predictions of ensemble models created with Classify-Verify ended up being the same as the predictions of Be+BeFC.

We are not fully sure why Be+BeFC+BeTC-T performs worse than Be+BeFC, suggesting that the inclusion of the classifying model actually seems to decrease performance. After all, this phenomenon was not observed with the dev set. However, from basic visual examination, it appears that on the questions that the two models differ, the predictions of Be+BeFC+BeTC-T tend to include the predictions of Be+BEFC as well as considerably more context words (e.g. if Be+BeFC predicted "numbers", Be+BeFC+BeTC-T might predict "the set of numbers"). The inclusion of these extraneous words would contribute to a higher FP rate for Be+BeFC+BeTC-T predictions, ultimately resulting in a lower EM and F1 score. However, more extensive testing would be required to verify this.

5 Analysis

5.1 Possible Answer vs. Impossible Answer Analysis

Figure 1: Model Average F1 Scores by Answer Type

Model	Possible	Impossible
BERT	0.7923	0.7229
BeFC	0.8083	0.7206
BeT	0.8777	0.0013
BeTC	0.8776	0.0013
Bi+Be	0.8002	0.7175
Bi+BeFC	0.8176	0.7175
Be+BeFC	0.8064	0.7383
Be+BeFC+BeT-V	0.8064	0.7383
Be+BeFC+BeT-T	0.7866	0.7592
Be+BeFC+BeTC-V	0.8064	0.7383
Be+BeFC+BeTC-T	0.7866	0.7592

Overall, our individual models perform consistently better on answerable questions than unanswerable questions, with this observation being especially true for BeT and BeTC. Interestingly enough, while ensemble models do appear to raise performance in both categories, the primary improvements in model performance stems from the improvement of the models' ability to correctly identify unanswerable questions. Lastly, Classify-Verify ensemble models appear to perform better with answerable questions, but worse with unanswerable questions when compared to Classify-Trust ensemble models.

5.2 Question Type Analysis

Figure 2: Model Average F1 Scores by Question Type

Model	How	Other	What	When	Where	Which	Who	Why
BERT	0.7396	0.6158	0.7613	0.7574	0.6791	0.7630	0.7868	0.6915
BeFC	0.7348	0.6002	0.7695	0.7635	0.6887	0.7874	0.7880	0.7027
BeT	0.4054	0.3275	0.4121	0.4714	0.4369	0.5384	0.4186	0.3590
BeTC	0.4109	0.3130	0.4100	0.4786	0.4392	0.5358	0.4211	0.3620
Bi+Be	0.7398	0.6110	0.7606	0.7889	0.6799	0.7587	0.7804	0.6915
Bi+BeFC	0.7378	0.6002	0.7707	0.7978	0.6848	0.7808	0.7848	0.7027
Be+BeFC	0.7569	0.5946	0.7744	0.8104	0.6942	0.7801	0.7870	0.7022
Be+BeFC+BeT-V	0.7569	0.5946	0.7744	0.8104	0.6942	0.7801	0.7870	0.7022
Be+BeFC+BeT-T	0.7623	0.6328	0.7753	0.8106	0.6870	0.7656	0.7915	0.7164
Be+BeFC+BeTC-V	0.7569	0.5946	0.7744	0.8104	0.6942	0.7801	0.7870	0.7022
Be+BeFC+BeTC-T	0.7623	0.6328	0.7753	0.8106	0.6870	0.7656	0.7915	0.7164

Overall, the models tend to be fairly similar in terms of their performance across categories. All models generally appear to do very well on "What, When", "Who", and "Which" type questions and less well on "How", "Other", "Where", and "Why" type questions. BeFC performs better than BERT across all categories, but especially "Which" questions, demonstrating the benefit of the additional data. Moreover, we can observe slight differences in performance between BeT and BeTC, with BeT performing better on "Other", "What", and "Which" questions and BeTC performing better on the rest. Moreover, ensembling models increases performance across the board, with the majority of improvement stemming from better performance on "When" and "How" questions.

5.3 Selected Examples Error Analysis

Lastly, we wanted to get a better sense of what types of errors our models were making, motivating us to select and analyze the following questions, which our most powerful models all got wrong.

5.3.1 Incorrect Inference

Context:	“Norman mercenaries were first encouraged to come to the South by the Lombards to act against the Byzantines”
Question:	Who did the Normans encourage to come to the South?
True Answer:	<No Answer>
Predicted:	Norman Mercenaries

In this question, the model likely attends strongly to the word "encouraged" and predicts that the subject is doing the encouraging, since most sentences are constructed with the subject doing the encouraging. However the model likely does not pick up on the fact that the inclusion of "were" before "encouraged" means that the direct object of the sentence is in fact doing the encouraging.

5.3.2 Lack of Contextual Knowledge

Context:	“The descendants of Rollo’s Vikings and their Frankish wives would replace the Norse religion and Old Norse language with Catholicism (Christianity) and the Gallo-Romance language of the local people”
Question:	What was the Norman religion?
True Answer:	Catholicism
Predicted:	<No Answer>

For this question, "religion" appears next to "Norse", and later on in the context (not shown), "Norman" appears next to "language". Since "religion" and "Norman" do not appear together and are used in different situations, the model, while trying to attend to either word, likely determined that the surrounding words did not answer the question and thereby predicted No Answer. However, humans would possess the contextual knowledge that Catholicism is a religion, and that the paragraph discusses the creation of Norman culture, allowing them to answer it with ease.

6 Conclusion

In this paper, we explored how the use of multiple different BERT models and multiple different ensemble methods could generate models with superior performance. Our experiments not only demonstrated superior performance over the baseline, with our ensemble models performing the best. We were also able to determine that manipulating the training dataset used would lead to significant differentiation between the strengths and weaknesses of different models, which we were able to leverage through ensembling. Moreover, we also demonstrated that using a generally trained model as a classifier of sorts for determining when to use a specialized model could lead to significant improvements in performance.

Lastly, there were a variety of things that could be done to improve upon our work. Since we appeared to hit a limit on the performance of our models focused on predicting possible answers, we would also like to explore adjusting the training dataset to generate models that would be focused on predicting impossible answers. Ensembling this with a generalized model and/or the specialized possible answers model might lead to improvements in performance. Moreover, it would also be interesting to explicitly train a NN classifer for determining when a question is answerable or not, instead of using a generalized BERT model. We would also be interested in exploring whether we could specialize BERT models to answer different question types (e.g. "How" questions) and see if that would improve performance. Lastly, it would be interesting to see how using of other ensembling methods (e.g. majority vote) or other models would affect model performance.

References

[1] Mrinmaya Sachan and Eric P. Xing. Self-training for jointly learning to ask and answer questions. *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:629–640, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Stanford NLP Group. Squad2.0 - The Stanford Question Answering Dataset. Stanford University. https://rajpurkar.github.io/SQuAD-explorer/.

[4] Tolgahan Cakaloglu and Xiaowei Xu. A multi-resolution word embedding for document retrieval from large unstructured knowledge bases. *CoRR*, abs/1902.00663, 2019.

[5] Shuaipeng Liu, Shuo Liu, and Lei Ren. Trust or suspect? an empirical ensemble framework for fake news classification. *WSDM Cup 2019 Fake News Classification Challenge*, 2019.

[6] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[7] Huggingface. Huggingface/pytorch-pretrained-bert. Github.

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in Neural Information Processing Systems*, page 1693–1701, 2015.

[9] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*, 2016.

[10] Christopher D Manning. Cs224n: Assignment 4.