Bert++
Default CS224N Project

Sophia Barton
sophiapb@stanford.edu

Juliet Okwara
jokwara@stanford.edu

Lea Jabbour
ljabbour@stanford.edu

Abstract

Our main goal is to produce a question answering (QA) system that performs well on SQuAD 2.0 and improves upon the BiDAF baseline, through use of the BERT model. To do so, we experiment with fine-tuning BERT, and with adding two different attention mechanisms to BERT: Attention-over-Attention (AoA) and Dynamic Coattention Network (DCN). The fine-tuned BERT model achieved the highest scores: EM score of 73.69 and and F1 score of 76.98 on the test dataset. We believe our attention mechanisms hold promise to improve upon this score, but require additional fine-tuning and experimentation.

1 Introduction

Our task is to create a question answering (QA) system that performs well on the Stanford Question Answering Dataset (SQuAD), particularly on SQuAD 2.0. QA is a very important task in NLP, as it holds great promise to automate reading comprehension, and to ultimately help us extract useful information from massive amounts of text data. Open-domain QA dates back to the 1960s, and traditional methods relied on linguistics techniques, such as Named Entity Recognition (NER), Parsing, and Part of Speech (POS) tagging. More recently, deep learning models have shown great promise for the QA task, yielding significantly better results.

The SQuAD 2.0 dataset differs from SQuAD 1.0 in that it contains unanswerable questions as well as answerable questions - whose answers are spans of text in the corresponding document. Therefore the SQuAD 2.0 task is difficult for two reasons: (1) systems must determine when no answer is available, (2) systems must correctly return the span of text which answers the question when possible. In this project, we investigate Google's revolutionary model, Bidirectional Encoder Representations from Transformers (BERT) [2], which was released at the end of 2018. BERT is a pre-trained, general purpose language representation model, which can be fine-tuned on NLP tasks such as QA. In addition to fine-tuning BERT, we also implement our own versions of Attention-over-Attention (AoA) [1] and Dynamic Coattention Network (DCN) [6] on top of BERT. Out of all our models, the BERT fine-tuned model (with no additional attention) performs best on the SQuAD 2.0 dataset, achieving an F1 score of 77.72 on the dev set and an F1 score of 76.98 on the test set. We believe our additional attention mechanisms have potential to outperform this model, but require more hyperparameter fine-tuning to achieve better results.

2 Related Work

There are two main classes of models used to solve QA tasks: pretrained contextual embeddings (PCE) models, and non-PCE models. PCE methods, such as Embeddings from Language Models (ELMo, 2018) [4] and Bidirectional Encoder Representations from Transformers (BERT, 2018) [2], are newer and achieve state-of-the-art results. The novelty of ELMo and BERT is that word embeddings are based on the context that a word appears in, as opposed to using fixed embeddings for words. Both methods pretrain weights on a large-scale language modeling dataset, which can be

loaded as initial layers of other models. Non-PCE methods do not use pretrained weights to generate predictions. Non-PCE model types include character-level embeddings (as used in Bidirectional Attention Flow for Machine Comprehension (BiDAF)[1] [5], self-attention (as used in R-Net) [3], and Transformers (as used in QANet) [7].

We explore PCE models, given their impressive performance over non-PCE methods. In particular, we use BERT instead of ELMo, because it uses Transformers, which have become the dominant model family for deep learning in NLP. BERT trains deep Transformers on a bidirectional language modeling task, and achieves state-of-the-art results on many NLP tasks, including SQuAD v1.0 and v2.0. Indeed, the top twenty models on the SQuAD 2.0 leaderboard all use BERT in some way.[2] BERT's most important technical innovation is the bidirectional nature of the model. The major limitation of other language models (LMs) is their unidirectionality, because tokens can only attend to previous tokens in the self-attention layers. This is restrictive for QA, where it is very important to use context from both the left and right of each token.

For further inspiration, we examine the models used in conjunction with BERT on the SQuAD 2.0 leaderboard. The top submissions are all quite recent, so many lack published papers. We did find papers on Attention over Attention (AoA, 2017) [1] and Dynamic Coattention Network (DCN, 2018) [6], which are used in the models in 11^{th} and 26^{th} place, respectively.

AoA's novelty is that it places a second attention mechanism over the document-level attention to produce "attended attention." AoA was designed for the cloze-style reading comprehension problem. The cloze-style task is very similar to the QA task, although it traditionally requires predicting a single word, rather than a start and end word to produce an answer span from the text. Therefore the prediction portion requires adaptation for SQuAD.

DCN is designed for the QA task, and has three main contributions. First, DCN has a Coattention Layer, which provides two-way attention (context to question, and question to context). Second, DCN also attends over representations that are themselves attention outputs, thus forming a second-level attention mechanism. Third, DCN introduces a dynamic pointing decoder that iterates over potential answer spans. This iterative procedure allows the model to escape local maxima corresponding to incorrect answers.

3 Approach

3.1 Overview

We use three models, all of which include open-source BERT code. The first is a fine-tuned version of the BERT model adapted for SQuAD, the second is BERT + AoA, and the third is BERT + DCN.

We use the following BERT implementation and pretrained weights, adapted for PyTorch: `https://github.com/huggingface/pytorch-pretrained-BERT`.

3.2 BERT Model

Google's BERT model is a multi-layer bidirectional Transformer encoder designed to pretrain deep bidirectional representations by conditioning on the left and right context of all layers jointly. The encoder incorporates six identical layers, each with two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. There is a residual connection around the sub-layers. This is followed with layer normalization.

BERT pre-trains on the concatenation of BooksCorpus (800M words, Zhu et al., 2015), and English Wikipedia (2,500M words). Each input sequence is generated by sampling two spans of text, the first of which receives the sentence A embedding, and the second of which receives the sentence B embedding. The input representation sums up token embeddings (WordPiece embeddings), segment embeddings (representing whether the word belongs to sentence A or B), and position embeddings (see Figure 1).

[1] BiDAF is used as the default project baseline, however, without character-level embeddings.
[2] https://rajpurkar.github.io/SQuAD-explorer/

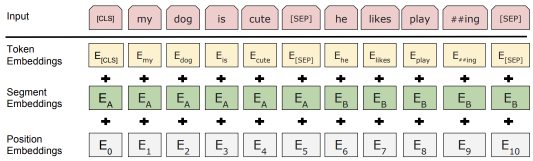

Figure 1: BERT input embeddings

BERT is pre-trained using two unsupervised prediction tasks. The first is Masked Language Modeling (MLM), and is necessary to create a 'bidirectional' model. The authors use MLM to mask 15% of tokens at random in each sequence and then predict only those tokens. The second task is Next Sentence Prediction (NSP), which is critical for QA, because it allows for the model to learn the relationship between pairs of sentences. When selecting sentences A and B for each pre-training example, 50% of the time B is the true next sentence, and 50% of the time it is a randomly selected sentence. Because MLM only predicts 15% of words in each batch, BERT converges more slowly than left-to-right approaches, but outperforms them after a small number of pretraining steps.

BERT can be easily adapted for many tasks, including QA on the SQuAD dataset. Each input question and paragraph is packed into a single sequence. Questions receive the A embedding, and paragraphs receive the B embedding. During the fine-tuning process, the only new parameters learned are: a start vector $S \in \mathbb{R}^h$, and end vector $E \in \mathbb{R}^h$, where h is the hidden size. The probability of word i being the start of the answer span is the dot product between the final hidden vector for the i^{th} input token, T_i, and S, softmaxed over the words in the paragraph:

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

A similar process is done to find the end of the answer span. At inference time, we must impose that the end of an answer span must come after the start of the answer span.

The huggingface repository provides an example, `run_squad.py`, that allows us to fine-tune an instance of `BertForQuestionAnswering`, the BERT model adapted for SQuAD. `BertForQuestionAnswering` is a BERT Transformer with a token classification head on top, and has a `from_pretrained` class method that allows us to load BERT weights that are pretrained specifically for the SQuAD task. This runs the output from the generic `BertModel` through a Linear Layer to obtain start and end logits. Cross Entropy Loss is used during training, and softmax is used in the `write_predictions` function to select the most likely start and end positions for each answer span. To fine-tune BERT, we experiment with different hyperparameters, and run `run_squad.py`.

3.3 Attention over Attention

For our second model, we add an AoA architecture to the `BertModel` output. We edit the `BertForQuestionAnswering` class in the `modeling.py` file. First, we split the output of the `BertModel`, into queries (Q) and contexts (C), each padded to a maximum length. We then have $Q \in \mathbb{R}^{h \times n}$ and $C \in \mathbb{R}^{h \times m}$, where h is the hidden size of the BERT output ($h = 768$), n is the length of the padded queries ($n = 66$), and m is the length of the padded contexts ($m = 382$).[3] We calculate M, the pair-wise matching matrix containing similarity scores of all document and query word pairs:

$$M = D^T Q \in \mathbb{R}^{m \times n}$$

We apply a column-wise softmax on M, to calculate query-to-context (Q2C) attention at time t ($\alpha(t) \in \mathbb{R}^m$), and a row-wise softmax to calculate context-to-question (C2Q) attention ($\beta(t) \in \mathbb{R}^n$):

$$\alpha(t) = \text{softmax}(M(1,t), ..., M(m,t))$$
$$\beta(t) = \text{softmax}(M(t,1), ..., M(t,n))$$

Because QA only relies on Q2C attention, we average all $\beta(t)$ to produce β, an averaged query-level attention.

[3]Please note that we did not remove the `CLS` or `SEP` tokens

We calculate the 'attended document-level-attention', $s \in R^m$, as the dot product of α and β. This essentially is a weighted sum of each $\alpha(t)$ in regards to a query word at time t:

$$s = \alpha^T \beta$$

We append s to the context hidden states produced by BERT, and feed the result into a Linear Layer, to obtain start and end logits. We believe s provides valuable information and should improve predictions, since document-level attention incorporates the importance of each query word.

3.4 Dynamic Coattention Network

Our third model adds a variation of DCN to the BERT output. As described in section 3.4, we obtain queries (Q) and contexts (C), each padded to a maximum length. Note that DCN requires appending sentinel vectors to the query and contexts, because these allow the model to not attend to any word. These sentinel vectors are trainable parameters in the DCN embedding layer. Since we replace this embedding layer with the BERT outputs, we preserve the `SEP` tokens[4] at the end of the queries and contexts to act as proxies for the sentinel vectors.[5] We then calculate $L = D^T Q \in \mathbb{R}^{m \times n}$, the affinity matrix containing scores corresponding to all pairs of document and question words.[6]

We use L to compute attention outputs for both directions.[7] We take a row-wise softmax of L to obtain the matrix α, which is used to calculate C2Q Attention outputs ($a \in \mathbb{R}^{m \times h}$). We take a column-wise softmax of L to obtain the matrix β, which is used to calculate Q2C Attention outputs ($b \in \mathbb{R}^{n \times h}$)[8]:

$$a = \alpha\, Q^T,\ b = \beta^T D^T$$

Next, we use the the C2Q attention distribution, α, to take weighted sums of the Q2C attention output, b, which gives us the second-level attention outputs, s:

$$s = \alpha\, b \in \mathbb{R}^{m \times h}$$

We then concatenate the second-level attention output s with the C2Q attention output a, and feed this sequence through a bidirectional LSTM. The concatenated forward and backward hidden states form our coattention encodings:

$$\{u_1, ..., u_N\} = \text{BiLSTM}\ (\{[s_1; a_1], ..., [s_N; a_N]\})$$

The DCN paper describes a dynamic decoder to avoid getting stuck in local maxima when making predictions, but due to time constraints for this project, we implement a similar decoder to that of BiDAF. For our first version of our BERT + DCN model (v1), we calculate logits by summing the projection of attention through a Linear Layer with output size 2, and the projection of the bidirectional LSTM output using a Linear Layer with output size 2. We then split the result into the start and end logits, and use the Cross Entropy Loss that `BertForQuestionAnswering` uses. For our second version of our BERT + DCN model (v2), we use Linear Layers with output size 1 to predict the start logit distribution. We then run the output of the bidirectional LSTM through another bidirectional LSTM, and sum the linear projection of those outputs with the linear projection of the attention layer to obtain the end logit distribution.

3.5 Baseline

For our baseline we refer readers to the BiDAF model described in the default project specification.[9]

[4]These parameters are learned during BERT fine-tuning

[5]We remove the `CLS` token at the beginning of each query. Thus, the padded length of queries is now $n = 64$.

[6]Note that this is equivalent to the computation of M in AoA.

[7]Please note that the naming conventions for α and β in DCN and AoA are reversed

[8]Please note that both the row-wise and column-wise softmaxes are masked softmaxes, so padding receives a probability of 0. We use similar functions as those used in the baseline code provided by Chris Chute.

[9]http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf#page=7

4 Experiments

4.1 Data

We are using the custom SQuAD dataset provided. Our dataset split is as follows:

- train (129,941 examples): All taken from the official SQuAD 2.0 training set
- dev (6078 examples): Roughly half of the official dev set, randomly selected
- test (5921 examples): Remaining examples from the official dev set, and hand-labeled ones

To run the BiDAF baseline model, as well as our BERT models, we use the train and dev datasets. For our highest performing model (fine-tuned BERT) we also make final predictions on the test dataset.

4.2 Evaluation Method

To evaluate the results of our model, we convert the predictions file from a `json` to a `csv` format to submit to the leaderboard and calculate EM and F1 scores. For questions with no answer (NA), both the F1 and EM score are simply 1 if our model correctly predicts NA and 0 otherwise. For questions with answers, the maximum F1 and EM scores across the three human-provided answers are taken. In the end, our EM and F1 scores are averaged across the evaluation dataset to get the reported scores.

4.3 Experimental Details

First, we run the baseline model using the command from the default project specification.[9] This model takes roughly 9 hours to train on our NV6 VM.

Second, to run the BERT model for SQuAD, we utilized the version 2, bert-base-uncased model with the following parameters: a learning rate of $3e^{-5}$, 3.0 training epochs, a max sequence length of 384, and a document stride of 128 (See Appendix 2 for full command).[10]

Fine-tuning this BERT model takes approximately 9 hours on an NV12 (batch size of 12), and approximately 12 hours on an NV6 (batch size of 6).[11] BERT + AoA takes approximately 14 hours to train on an NV6 (batch size of 6). BERT + DCN (v1) takes approximately 16 hours to train on an NV12 (batch size of 6)[12], and we estimate BERT + DCN (v2) will take roughly 24 hours to train on an NV12 (batch size of 6).[13]

4.4 Experimentation with BERT Query & Context Splitting

We attempt two methods for splitting BERT's output into query and context, which is necessary to calculate attention. In the first method, we use the `token_type_id_masks`, which assign 0 to query (and padding) tokens, and 1 to context tokens, to partition the BERT hidden output tensor within `modeling.py`. In the second method, we pad queries to the maximum question length of 64 in `run_squad.py`. This allows us to know exactly where to partition the BERT hidden state output tensor (unlike the previous method, in which query lengths are variable). This standardizes splitting, but may also reduce the length of certain context sequences, because the maximum total sequence length is retained at 384.[14] Unfortunately, an error appears in the second method after training.[15] Since this error only surfaces after many hours of training, we did not find it fruitful to continue debugging given time constraints. Thus, we use the first method for our BERT + AoA and BERT + DCN models.

[10] `train_batch_size` requires modification depending on the model and VM, due to memory constraints.

[11] To obtain final predictions for the test leaderboard, we use the same command on our NV12, but with the `test-v2.0.json` dataset as the prediction file.

[12] Note that we reduce the batch size even though we use a NV12, because the additional infrastructure (i.e. LSTM layers) requires more memory.

[13] This model has an additional bidirectional LSTM layer to predict end logits, and is still training.

[14] The length of an input sequence is equal to the length of the query (capped at 64), plus the length of the context (capped at $384 -$ len(query) - 3), plus 3 (for the `CLS, SEP, SEP` tokens)

[15] `AttributeError: 'NoneType' object has no attribute 'start_logit'`

4.5 Results

Table 1 shows our quantitative results from the baseline and BERT models:

Table 1: Dev Scores

Model	EM	F1
BiDAF	57.32	61.06
BERT	74.48	77.72
BERT + AoA	35.36	46.91
BERT + DCN (v1)	42.83	44.85

The EM and F1 scores for our models are generated by our submissions to the dev PCE leaderboard. The results from running the BiDAF baseline are slightly higher than the default project handout; this is likely due to random variation.

The results from running BERT are what we expect, per comparison with other students. It took us quite some time to discover why our earlier scores generated from BERT were roughly 10 points lower than expected (refer to Appendix 1 for further explanation).

Our results from BERT + AoA and BERT + DCN are much lower than expected.

We expected BERT + AoA to improve BERT's scores because it contains additional information from the bidirectional attention mechanisms. We did not expect a very large improvement from this method (i.e. we expected to improve BERT's scores by one or two points), because we only concatenate a single scalar to each of the context hidden states. Indeed, reducing Q2C and C2Q information to a single scalar per context word is an information bottleneck, and we believe that concatenating the dot product of context and attention to the hidden states would have been more helpful. In any case, our scores did not improve over BERT, but rather dropped significantly compared to BERT. We have three hypotheses to explain these low scores: (1) our method of splitting queries and contexts is incorrect, (2) our models require more hyperparameter fine-tuning, particularly in adjusting the `null_score_diff_threshold` parameter, because our probability distributions may be spikier without the query tokens,[16] (3) padding tokens should have been masked in the Cross Entropy Loss (which computes a softmax) to ensure they do not get selected for the answer span. Another possible source of error is that we included the `CLS` and `SEP` tokens from the BERT model output. However, we do not think this is a great source of error, since the final Linear Layer should learn very negative scores for these tokens, giving them low probabilities after softmax function.

We expected DCN to perform well above the original BERT code, so we were surprised it received the lowest F1 score. We hypothesize that similar problems that arose in AoA contributed to our low scores, particularly hypotheses (1) and (2). Furthermore, BERT + AoA may be performing better than DCN because AoA's final Linear Layer can more easily learn negative weights for padding tokens, but the LSTMs used in DCN still place some importance on them. Furthermore, in DCN v1, we do not use an additional RNN to calculate end logits, which we believe lowers our scores significantly. Unfortunately our BERT + DCN v2 (which uses a second bidirectional LSTM to calculate end logits) is still training, so we do not have the results yet. We will be sure to include these results and further analysis in our poster.

Lastly, since BERT performs the highest out of our models on the dev leaderboard, we use BERT to make final predictions on the test dataset. On the test dataset, BERT achieves an EM score of 73.69 and and F1 score of 76.98 (BERT++ on leaderboard).

5 Analysis

5.1 Qualitative Analysis of Results

We divide our models' incorrect answers into three categories: incorrect answer, incorrectly reporting no answer (NA), and incorrectly reporting an answer.

[16]This is described more in section 5.2

5.1.1 BERT

BERT's incorrect answers at times consist of answers that are partially correct. An example of this is ID `fd4619c4ba43dcc17afe7bb0d`: 'Upon learning of a French scouting party in the area, what did Washington do?'. The dev set answers include: 'killed many of the Canadians' and 'surprised the Canadians on May 28.' However, BERT answers with 'surprised the Canadians.' Although a human grader would likely deem this answer correct, EM does not count the answer and F1 gives partial credit to the answer. Repetitive near-misses could have cost BERT greatly.

A trend we observe with BERT (and all our models) is their inability to locate specific names and locations. An example of this is ID `fc2bcb90ce3cc22d6ea2e6cda`: 'Who was a prominent Huguenot in Holland?.' The dev set answers are 'Pierre Bayle,' but BERT predicts NA.

BERT's largest shortcoming is its inability to handle NA questions. Often, BERT provides an answer when it should be NA. Consider dev example ID `f0229ad5e7385b8deb7aeedde`: 'What is the full official city name of Miasto?.' The correct response is NA, but BERT provides the answer 'miasto stołeczne Warszawa.' This is likely because the phrase 'official city name' appears in the context close to 'miasto stołeczne Warszawa.' However, BERT does not understand that the city referenced in the context is Warsaw, not Miasto. BERT can easily be confused by modifiers and contexts that contain phrasing similar to the question but do not have the answer the question is searching for.

5.1.2 BERT + AoA

BERT + AoA has a tendency to cut off early despite being on the right track. The best example of this is with ID `9dbc679fb46ba6e0d67bdade9`, 'What are cancerous tumors of the skin known as?.' The dev answer is 'melanomas'; however, BERT + AoA predicts 'tumors called mel,' cutting the word melanomas short.

This characteristic is even evident in NA questions like that of ID `fa6f8ae4e32f8bffaff4c3cee`: 'In Sept 1760 who negotiated a war from Montreal?.' BERT + AoA answers with 'Governor Vaudre,' an incorrect answer, but more importantly, an answer truncated from the context phrase 'Governor Vaudreuil.' These tendencies are likely the reason for BERT + AoA's low F1 and particularly low EM scores (which does not credit partial answers). BERT + AoA struggles in a similar manner to BERT in regards to NA questions.

5.1.3 BERT + DCN (v1)

BERT + DCN (v1)'s incorrect answers are strikingly different than those generated by BERT and BERT + AoA. BERT + DCN often produces single word or even partial word responses. An example of this is dev ID `eb7fc2fa24cf4c3ded5face6e`: 'Gamma delta T cells share the characteristics of what other types of T cells?.' A correct answer is 'helper T cells'; however, BERT + DCN predicts 'er.' This is likely a major truncation of 'helper.' We believe BERT + DCN (v2) will do a much better job at this, because the end logits will be predicted more accurately. BERT + DCN (v1) seems to pay too much attention to parts of words rather than analyzing the entire word. This odd behavior may be due in part to improper hidden state splitting techniques, but we have been unable to confirm this. BERT + AoA does not have this problem to the extent of BERT + DCN (v1). The additional attention layer involved in BERT + DCN (v1) could be causing these errors. Also, BERT + DCN's tendency to focus only on certain parts of the question, while ignoring parts that should be more relevant, indicates that another RNN layer may be helpful in order to successfully encode the context.

Like BERT, BERT + DCN (v1) often returns NA when an answer is present, especially in the case of specific names or locations. In particular, BERT + DCN (v1) is prone to producing an answer simply because a question word appears close to a context token. For example, ID `b1aaddedcbdefbc3d6c56f70c`: 'What type of electric engine produces most electricity in the world today?.' The correct response is NA, however BERT + DCN replies with a quite random response of 'continuing.' BERT + DCN is likely paying additional attention to aspects of context it would likely be better off without. This could also be reflective of BERT + DCN possibly taking padding tokens into consideration. This may have resulted in an odd distribution of scores, leading to inaccurate and nonsensical predictions.

5.2 The 'No Answer' Problem

All three of our models struggle with predicting NA appropriately. To further analyze this, we observe true positives (TP), false positives (FP), and false negatives (FN). Results are summarized in Table 2.

Table 2: NA Analysis

Model	TP Rate	FP Rate	FN Rate
BERT	38.23%	5.3%	13.9%
BERT + AoA	33.9%	8.0%	18.1%
BERT + DCN (v1)	39.6%	19.3%	12.5%

About 52% of the dev set has NA, but BERT predicts NA on 37% of examples, BERT + AOA predicts NA on 41% of examples, and BERT + DCN predicts NA on 59% of examples. Therefore, we can see that BERT is under-predicting the percentage of NA, but it has a high TP rate, and the lowest FP and FN rate of the four models. BERT + AOA seems to predict a more reasonable amount of NAs, but we can see that the FP and FN rates are quite high, so it is not predicting NA for the correct questions. BERT + DCN over-predicts NA, and has a very high FP rate. This is likely the cause of the very low F1 score, since it predicts NA instead of even a partial answer for many questions. We believe BERT + DCN (v2) will have a higher F1 score, because it should capture more of the answer span.[17]

6 Conclusion

Overall, we have learned a great deal about advantages and disadvantages of using PCE models such as BERT for QA, particularly for SQuAD 2.0. We have also learned pitfalls of implementing attention mechanisms in addition to BERT, and have reasoned through hypotheses as to why our attempts are not as fruitful as we initially had hoped for.

With regards to future work, we believe additional hyperparameter fine-tuning could improve the NA problems that our models each face in unique ways. In particular, we would adjust the value of the `null_score_diff_threshold`. This flag is responsible for determining the threshold at which the model predicts no answer, as opposed to predicting the start and end logits with the highest probabilities.

Furthermore, we would like to experiment with the bert-large-uncased model, which consists of 24-layer, 1024-hidden, 16-heads, 340M parameters. This model would have taken too long to train given our project timeline, but would allow us to achieve much higher BERT results (i.e. an F1 > 80).

In addition, an ablative analysis, in which we run our models with only one aspect changed and all others held constant, would be very helpful to pinpoint issues within our models. In particular, we would like to rerun the BERT model, while only passing in the *context* hidden states through the final output layer, as opposed to the entire hidden state tensor. This would allow us to verify if our splitting is correct, and fine-tune to obtain similar EM and F1 scores as the original BERT model itself.

Lastly, we are interested in ensembling. We would obtain logits across different models, apply a softmax function, and then take the weighted average of these different models (weighted by F1 to place higher weight on the models which perform better). Since our models have different failure modes - particularly regarding the NA problem - ensembling would be beneficial to help us leverage the best aspects of each model.

7 Acknowledgments

We would like to acknowledge the teaching staff for their support, particularly Saahil Agrawal and Chris Chute. We also want to acknowledge Kevin Tran and Danny Takeuchi, who worked with us to debug our second method of splitting queries and contexts, and Tegiri Sido, who worked with us to decipher why the huggingface BERT code was initially giving us low scores. Finally, we would like to thank the owner of the huggingface repository and Google for the open-source BERT code.

[17] Please note that this model is still running, but we will report the results at the poster session.

References

[1] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *CoRR*, abs/1607.04423, 2016.

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

[3] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

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

[6] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[7] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.

Appendix

Appendix 1

Our BERT scores reported in our milestone were lower (EM: 62.41; F1: 65.72), because we had cloned the huggingface repository and used the command `pip install -r requirements.txt` as opposed to `pip install -editable .`. Thus, we hypothesize that somehow the pretrained model weights adapted for SQuAD were not being used, even though it seems unlikely that *no* pretrained weights were used, because our model did achieve scores of 62.41 and 65.72 for EM and F1, respectively, which are not plausible if BERT was training entirely from scratch for only 3 epochs. We have posted this as an issue to huggingface to see what exactly occurred and what weights our previous BERT model was using.

Appendix 2

```
python run_squad.py \
--bert_model bert-base-uncased \
--do_train \
--do_lower_case \
--do_predict \
--train_file $SQUAD_DIR/train-v2.0.json \
--predict_file $SQUAD_DIR/dev-v2.0.json \
--train_batch_size 12 \
--learning_rate 3e-5 \
--num_train_epochs 3 \
--max_seq_length 384 \
--doc_stride 128 \
--output_dir /tmp/debug_squad/ \
--version_2_with_negative
```