Extending Answer Prediction for Deep Bi-directional Transformers

Peter Dun
Stanford University
bodun@stanford.edu

Lauren Zhu
Stanford University
laurenz@stanford.edu

David Zhao
Stanford University
dzhao0@stanford.edu

Abstract

The current state-of-the-art technique used to tackle the Stanford Question Answering Dataset (SQuAD) makes use of Bidirectional Encoder Representations from Transformers, or BERT. After pre-training a deep bidirectional representation conditioned on both the left and right contexts, BERT can be simply fine-tuned for the SQuAD task with a linear layer that outputs the predicted answer span. We investigate alternative ways to interpret and process BERT encoding outputs, including Pointer-Net, Dynamic Pointing Decoder, and Dynamic Chunk Reader. The best-performing model, the Dynamic Decoder, uses pre-trained BERT encodings and improves F1 from the baseline BiDAF on the test set from 59.920% to 75.035% and the EM score from 56.298% to 71.784%.

1 Introduction

Question answering is a critical component of communication and language comprehension, which is why datasets have been built to specifically address this challenge. In particular, the Stanford Question Answering Dataset (SQuAD) is one of the most comprehensive datasets available. Compared with previous datasets, SQuAD answers have variable lengths and have a sizeable subset of candidate answers. For SQuAD 1.0 and 1.1, models have already been built that can exceed human-level performance. In this work, we approach SQuAD 2.0, which is an extension of SQuAD 1.1 that also contains unanswerable questions [1], in which the model must predict "no answer" if the question does not have an answer in the passage.

A core component of most state-of-the-art models on SQuAD 2.0 is Bidirectional Encoder Representations for Transformers, or BERT [2]. To handle SQuAD 1.0 AND 1.1, the vanilla method of using BERT concatenates it with a single linear layer that predicts the resulting answer span. Because the encoder representation of BERT contains complex information that a single dense layer cannot interpret intelligently, we investigate extensions to BERT with various interpretation models including Pointer-Net, Dynamic Pointing Decoder (DD), and Dynamic Chunk Reader (DCR).

The Pointer-Net [3] is an attention-based RNN that generates dynamically-sized outputs based on tokens from the input text. It predicts the start and end tokens, even though there may be several intuitive answer spans within the document. The Dynamic Pointing Decoder [4] aims to solve this problem with an LSTM that produces new estimates of the start and end positions for every iteration via a multilayer neural network. The Dynamic Chunk Reader [5] also alleviates this problem by extracting and ranking a set of answer candidates from the document, and then selecting the answer that maximizes cosine similarity between the span and the question.

We find that the Dynamic Decoder architecture significantly outperforms the BiDAF baseline on SQuAD 2.0 in both F1 and EM metrics on both dev and test sets. Our other approaches, the hybrid Answer Pointer-BERT and Answer Chunk Ranker-BERT models, underperformed the baseline but represent interesting future research directions in leveraging information in BERT contextual representations.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

2 Related Work

2.1 BiDAF

The baseline model [6] is based on Bidirectional Attention Flow (BiDAF) [7], but does not include a character-level embedding model. Specifically, the model begins with a word-level embedding layer and a bidirectional LSTM encoder. Then, there is a key bidirectional attention flow layer that allows attention to flow both from context to question and question to context. Next is a modeling layer that refines the sequence of vectors after the BiDAF layer with another LSTM. The output layer utilizes another bidirectional LSTM on the modeling layer to produce a vector of probabilities corresponding to each position in the context: p_{start} and p_{end}. While we do not make architectural changes or extensions from this BiDAF baseline, it serves as a good metric to compare with the performance of vanilla BERT and the performance of our BERT extensions.

2.2 BERT

Unlike previous work for pre-training contextual representations, BERT is the first to pre-train deep bidirectional representations with a plain text corpus, namely Wikipedia. BERT jointly conditions on both left and right contexts for all layers [2], using masking to allow for proper bidirectional attention. The strong advantage of BERT is that it achieves high performance for a variety of tasks with a simple fine-tune to the task and data. The BERT Base model that we train as our own baseline and further extend in our work has 12 layers, a hidden size of 768, and 12 self-attention heads [8]. While the BERT Large model has better performance, we extend from BERT Base with our Dynamic Pointing Decoder, Pointer Net, and Dynamic Chunk Reader in order to save computational time and expense.

2.3 Match-LSTM and Answer Pointer

Our first experimental output architecture is the Pointer-Net from Wang and Jiang's [3] Match-LSTM and Answer Pointer. The end-to-end model uses two LSTM Preprocessing Layers to separately incorporate contextual information of individual tokens in the passage and the question. Then, a bi-directional Match-LSTM layer is applied to sequentially aggregate the matching of the attention-weighted question to each token of the passage. To generate a prediction, the Pointer-Net is adapted to a sequence model that predicts individual tokens, and a boundary model that predicts a start and end token, creating a span of tokens as the answer. Because pre-trained BERT encodings potentially provide very contextual information for a Pointer-Net architecture, we replace the interim output of LSTM Preprocessing Layers and Match-LSTM with BERT as input to a boundary model of Pointer-Net to predict a better answer span.

2.4 Dynamic Coattention Networks

Our second experimental output network is from the Dynamic Coattention Networks paper by Xiong, et. al [4]. The full end-to-end model first encodes the passage and question separately using an LSTM. Then a coattentive encoder captures interactions between the question and passage by creating a co-dependent representation of the two together and feeding this representation to a bi-directional LSTM.

Lastly, the Dynamic Pointing Decoder (DD) iteratively selects and improves an answer span based on this encoding. To do this, it passes previous estimates of start and end token indices along with the previous hidden state as input to the LSTM. The current hidden state of the LSTM is passed to two Hidden Maxout Networks (see Appendix 7.1) where one updates the start token prediction and the other end token prediction. These are then passed to the LSTM to compute new estimates of the span. In our experiments, we use the pre-trained BERT encodings to replace the coattention encoding from this paper and maintain the structure of the DD for the SQuAD 2.0 task.

2.5 Answer Chunk Extraction and Ranking

The end-to-end Answer Chunk Extraction and Ranking model from Yu et. al [5] proposes a Dynamic Chunk Reader (DCR) that encodes the document and question with RNNs, and then applies a word-by-word attention mechanism to acquire question-aware representations for the document. Lastly, it generates chunk representations and a ranking module to select the top-ranked chunk as the answer.

Like Pointer-Net and the DD, we apply the chunk extraction and ranking architecture to contextual BERT hidden states to see if we can improve from vanilla BERT's single linear layer output.

3 Approach

3.1 Baseline

For the default class-provided baseline BiDAF that we trained, refer to Seo et. al's work on machine comprehension and the course-provided handout [6] [7]. Beyond the default baseline, our team chose to also fine-tune a BERT baseline based on the HuggingFace Github repository. Our baseline model is BERT followed by a single linear layer with an output of dimension 2, one corresponding to logits of the predicted start index and one corresponding to logits of the predicted end index. We take the cross entropy loss of both the start and end logits against their corresponding labels, and we use Adam optimization to learn on the loss.

3.2 Pointer Net

The first extension we made to the network architecture is to replace the single linear layer from vanilla BERT with Pointer Net, a RNN that has the ability to generate dynamically-sized outputs. It was created as an improvement to Sequence-to-Sequence, which has an output dimensionality fixed by the problem [9].

For each timestep, Pointer Net outputs a softmax distribution over the length of the input. We specifically draw two outputs from Pointer Net, one corresponding to logits of the predicted start index and one corresponding to logits of the predicted end index. If the predicted start index is beyond the predicted end index, our final prediction is that this question does not have an answer. The following equation details how the logits of the ith timestep are calculated. W_1, W_2, and v are all learnable parameters, while e is the output from the encoder and d is the output from the decoder.

$$p(C_i) = \text{softmax}(v^T \tanh(W_1 e + W_2 d))$$

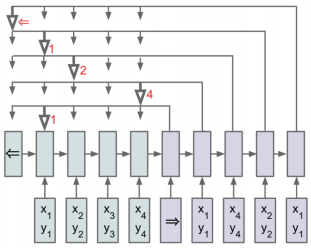

Figure 1: Pointer-Net [3]. Blue indicates a RNN used to encode the input sequence, and purple indicates the generating network that at each step generates a softmax vector over inputs that corresponds to the probability of each word being the next potential output token.

3.3 Dynamic Pointing Decoder

The next extension we implemented is a Dynamic Decoder, or DD, which is a part of the Dynamic Coattention Network [4]. Because of the single-pass nature of most question answering models, they cannot recover from incorrect but probable answers associated with local maxima. To solve this problem, the dynamic pointing decoder iterates over potential answer spans in order to recover from local maxima and find the true answer span.

The DD is similar to a state machine, where the state maintained by an LSTM. With each iteration, the decoder updates the state by processing the BERT embeddings in conjunction with the current estimates of the start and end positions to create a new hidden state. Using a multilayer highway maxout network, the decoder uses this hidden state to create new estimates of the start and end states.

Let the variables h_i, s_i, and e_i denote the hidden state, predicted start index, and predicted end index, respectively, for iteration i. Our Decoder LSTM update is as follows:

$$h_i = \text{LSTM}_{\text{dec}}(h_{i-1}, [u_{s_{i-1}}; u_{e_{i-1}}])$$

where $u_{s_{i-1}}$ and $u_{e_{i-1}}$ are the representations corresponding to previous start and end estimates in the BERT encodings, $U \in \mathbb{R}^{b \times m \times 2l}$ where b, m, and l correspond to the batch size, context length, and hidden size respectively.

For each LSTM iteration, we use two Highway Maxout Networks to compute start and end scores α_t and β_t for each token in the context passage. As a result, our predicted start and end span is defined by token indices s_i and e_i as follows: $s_i = \text{argmax}_t(\alpha_1, ..., \alpha_m), e_i = \text{argmax}_t(\beta_1, ..., \beta_m)$.

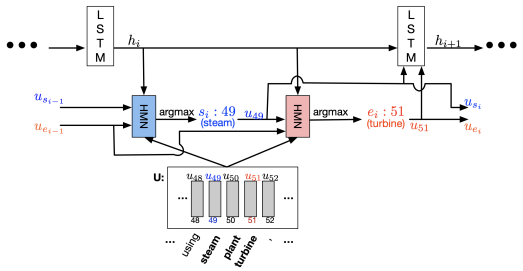

Figure 2: The Dynamic Pointing Decoder, from Xiong, et. al [4]. Blue denotes the variables and functions used to estimate the start position, red for the end position.

The SQuAD 2.0 dataset is partly challenging because a variety of question and context types makes it more difficult to predict the correct answer span. The Highway Maxout Network, or HMN, is particularly useful because it can pool across multiple model variations to determine the best answer span. In our model, we use two HMNs, where one computes s_i and the other e_i. Each HMN has the same architecture, but does not share parameters.

The full structure of the Highway Maxout Network is illustrated in the Appendix (Section 7.1).

As with the Pointer-Net implementation, we use our pre-trained BERT encodings as input to our DD. It minimizes Cross Entropy Softmax Loss of the start and end points compared with the correct span, or no answer if applicable.

3.4 Answer Chunk Ranker

The last extension we implemented is a modified version of the Answer Chunk Ranker, or ACR, which represents the final two layers of the Dynamic Chunk Reader (DCR) architecture [5]. The first of those two layers is a Chunk Representation layer, which dynamically extracts sequential chunks of the passage as potential answers. The original DCR paper proposes two possible chunk generation algorithms. The first runs an external part-of-speech (POS) tagger on each answer in the training set to generate a "POS pattern trie" of POS sequences. At test time, all sequences of tokens in the passage that match a path down the POS pattern trie are extracted as the set of possible answer candidates. The second approach in the paper uses all possible subsequences up to a user-defined maximum length, *N*, as the set of answer candidates. We choose to implement the second approach due to the large computational overhead of iteratively running a POS tagger on the extracted subsequences.

Given an answer candidate from index m to index n of the passage, our ACR architecture represents the answer span as: $[B_m^{\text{answer}}; B_n^{\text{answer}}]$, where $[x; y]$ represents vector concatenation, and B_x^{answer} is the hidden state at token index x of BERT's contextual representation of the answer. Likewise, we represent the question as: $[B_0^{\text{question}}; B_{\text{len(question)}}^{\text{question}}]$, where we concatenate the BERT contextual representations of the first and last word tokens in the question. The DCR authors experimented with different pooling methods of the hidden states within the span, but found that concatenation of the first and last states was the optimal function to represent the span's context [5]. Furthermore, while the original architecture uses a bidirectional RNN/GRU encoder to generate hidden states and concatenates the first state of the forward RNN with the last state of the backward RNN, BERT's original paper states that its deeply-bidirectional hidden states are more powerful than the "shallow concatenation of [the hidden states of] a left-to-right and right-to-left model" [2].

The second of the two layers in our ACR implementation is the Ranker layer, in which the representations of each extracted answer span is compared against the question representation, using cosine similarity as follows:

$$S(c_i^{m,n}) = \text{cosine\_sim}([B_0^{\text{question}}; B_{\text{len(question)}}^{\text{question}}], [B_m^{\text{answer}}; B_n^{\text{answer}}])$$

where $c_i^{m,n}$ is the answer span from m to n. At test time, after exhaustively computing the similarity scores, the span with the highest cosine similarity with the question is returned as the predicted answer [5].

The original DCR paper uses the softmax of the vector of all answer span similarity scores in its cross-entropy loss, but in order to adapt this ranking approach to the fixed start and end logits cross-entropy loss calculation in BERT, we propose a novel greedy logit-generation algorithm. We iterate over the passage, and for the word at index x, we compute the cosine similarity scores between the question representation and all the answer representations in the set: $[[x; x], [x; x+1], ..., [x; x+N]]$, where $[a; b]$ is the concatenation of BERT contextual representations at indices a and b. We take the span, say $(x, x+j)$, with the maximum cosine similarity and set the x^{th} index of the zero-initialized

$start\_logits$ to the cosine similarity score, and the $x + j^{th}$ index of the similarly zero-initialized end_logits to the same score. If there is a potential collision in end_logits (i.e. the value is already set by a prior span), then we replace the existing value if it is less than the current span's similarity. We therefore preserve the globally maximum similarity score in both logits. We furthermore implement a thresholding mechanism to adapt the span extracting and ranking mechanism to the no-answer constraint in SQuAD 2.0. The maximum value of the logits is extracted after all answer spans are considered, and if it is less than one standard deviation above the mean of either the start or end logits, then the model negates all logits to force a null answer.

3.5 Original Work and References

3.5.1 Pointer Net

We made use of the HuggingFace's BERT PyTorch code repository, which has existing logic for training and evaluating our baseline model on custom datasets [8]. We drew the majority of our Answer Pointer logic from an implementation of Match-LSTM followed by Pointer-Net by Github user Laddie123 [3]. Our original work in developing BERT-Pointer includes updating the hidden size of Pointer-Net to be compatible with that of BERT, generating an attention mask, and extracting an answer (or no answer) from the output logits [10].

3.5.2 Dynamic Pointing Decoder

Like our implementation of BERT with Pointer Net, we used Atul Kumar's implementation of the Dynamic Coattention Network on Github in conjunction with HuggingFace's BERT repository [8] [11]. Our original work in developing DD-BERT included modifying tensors from BERT's outputs to be compatible with the Dynamic Decoder and Highway Maxout Networks, generating an attention mask, and extracting an answer (or no answer) from DD-BERT's output logits [10].

3.5.3 Answer Chunk Ranker

We implemented a modified version of Answer Chunk Ranker in PyTorch, following the equations and framework in the original DCR paper [5]. Our original work also included modifying HuggingFace's BERT implementation to pass in the indices of the question-answer separating tokens to ACR [8]. Finally, we altered the logit-generation step of ACR to work with BERT's loss function by implementing a novel greedy algorithm and no-answer thresholding mechanism as described in Section 3.4 [10].

4 Experiments

4.1 Dataset

We are using v2.0 of the Stanford Question Answering Dataset (SQuAD 2.0), a question-answering dataset containing around 150,000 tuples of a question, passage (from Wikipedia articles), and answer [12]. This recently-released update to the previous version of SQuAD adds a layer of complexity to the reading comprehension task, wherein a subset (roughly one third) of the question/passage pairs have no answer. The remaining question/passage pairs have an answer that is a continuous span of text from the corresponding passage. We are using a slightly modified train, dev, and test split (129,941, 6,078 and 5,915 examples, respectively) of the official SQuAD dataset, where the official dev set is split into a new dev and test set, and the new test set is augmented with hand-labeled tuples [6]. An example of a training example tuple is below in Figure 1.

Question: What population descended from the Vlachs?
Answer: Moravian Wallachia
Context: "... Romance-speaking Vlachs who migrated into the region ...
The population of *Moravian Wallachia* also descend of this population."

Figure 3: SQuAD training example (italics added in context) [12].

Given a question and context pair, our task is to either predict a start and end index pair corresponding to the contiguous answer span in the passage, or predict that the question isn't answerable.

Model Name	Test F1 Score	Test EM Score
Baseline BiDAF (from leaderboard)	59.920	56.298
DD-BERT	**75.035**	**71.784**

Table 1: Test F1 and EM results for the trained models

4.2 Evaluation Method

The two evaluation metrics we use are provided by the SQuAD dataset. The first metric is *Exact Match*, which is equal, in the answerable case, to 1 if the predicted answer matches directly with at least one of the three possible human-annotated answers in the dev and test sets, and 0 otherwise. The second metric is the *F1 Score*, defined as: $2 * \frac{p*r}{p+r}$, where p represents precision (percentage of predicted tokens that appear in the ground truth) and r represents recall (percentage of ground truth tokens that are present in the prediction). Given that there are three possible answers for each question/context pair, we compute the *F1 Score* with each human-annotated answer and keep the highest score. In the no-answer case, both *Exact Match* and *F1 Score* are equal to 1 if the model prediction and the ground-truth are no-answer, and 0 otherwise [12] [6].

4.3 Experimental Details

Using a GPU and CUDA-enabled Azure Virtual Machine, we first trained the BiDAF baseline using the default parameters, with a maximum of 30 epochs, batch size of 64, and an initial learning rate of 0.5. We then fine-tuned the pre-trained, uncased BERT implementation from the HuggingFace repository with the suggested learning rate of $3 * 10^{-5}$, 2 training epochs, and a batch size of 96. We also used half-precision floating point training. The BERT-Base-Uncased pre-trained model consists of 110 million parameters in a 12-layer architecture [13].

For the BERT-related models, we tuned the batch size parameter by increasing it until the VM returned a CUDA out-of-memory error. For the Answer Pointer-BERT hybrid model and the DD-BERT model, we used full-precision training, trained for two epochs, and used a learning rate of $3 * 10^{-5}$. However, we used a batch size of 42 for Answer Pointer-BERT, and a batch size of 36 for DD-BERT. For ACR-BERT, we used half-precision training, trained for one epoch, used a learning rate of $3 * 10^{-5}$, and trained with a batch size of 56.

4.4 Results

In Table 1, we see that our DD-BERT architecture outperforms the baseline BiDAF model by a significant margin on the test set, in terms of both F1 and EM. As shown below in Table 2, two of the three models that we trained and evaluated, including the finetuned BERT and DD-BERT hybrid architectures, outperformed the baseline BiDAF model by a significant margin on the dev set. Answer Pointer-BERT underperformed the baseline on the dev set, and upon further examination, the model predicted no answer for all dev set examples. ACR-BERT also significantly underperformed the baseline on the dev set. While ACR-BERT's metrics on the questions that have answers ($EM = 20.447$, $F1 = 39.454$) were low, the architecture failed on questions with no answers ($EM = F1 = 3.409$).

While our test set metrics for DD-BERT are still lower than the state-of-the-art implementations on the official SQuAD 2.0 leaderboard, they are in line with what we expected, as we have not conducted hyperparameter tuning. It does make sense that DD-BERT would outperform vanilla finetuned BERT, as vanilla BERT uses a single dense layer to predict the answer span, while our extension instead uses an LSTm-based prediction architectures that can better capture long-range dependencies in the answer passage. The dev set results for ACR and Answer Pointer-BERT were significantly lower than expected, likely because of the complexities in adapting mechanisms that work well on SQuAD1.1 (i.e. does not contain "no-answer" questions) to SQuAD 2.0. This implies that further research and experimentation is needed to deal with this novel class of questions.

We are in the Pre-Trained Contextual Embeddings (**PCE**) division.

Model Name	Dev F1 Score	Dev EM Score
Baseline BiDAF	58.75	55.42
Fine-tuned BERT (vanilla)	72.898	69.826
PointerNet-BERT	52.122	52.122
ACR-BERT	20.666	11.566
DD-BERT	**75.884**	**72.771**

Table 2: Dev F1 and EM results for the trained models

5 Analysis

5.1 Pointer Net

Pointer Net underperformed the class-provided baseline and predicted "no answer" on every single question in the dev set. We initially thought that the model quickly learned to always predict a null answer to minimize the NLL loss function, and we therefore continued to train for another two epochs [6]. However, the dev set results remained nearly unchanged for that model – the model continued to predict no answer for almost all questions. Upon further analysis, we discovered that all predicted start and end index logits had values around -4 or -5, which were negative to a large enough extent that the HuggingFace SQuAD algorithm deemed all predictions as not confident enough.

One possible reason why our model underperformed is that Pointer-Net does not generalize well to SQuAD 2.0's answer domain that includes "no answer" as an option. Previous work done on Pointer-Net has only seen it applied up to SQuAD 1.1, so even though the absolute values of Pointer-Net logits need to be uniformly shifted, their values relative to each other (most importantly the highest-valued logit) may still be correct.

If the reason why our model underperformed is due to human error, it may be our loss function that led to these incorrectly low logit predictions. We might have accidentally flipped a sign in our cross entropy loss function, which means we're attempting to minimize the logit values at the label index rather than maximize. However, this particular scenario is unlikely since we use the same loss function on PointerNet-BERT that we do with DD-BERT, which is our highest performing model.

5.2 Dynamic Pointing Decoder

The DD was our most successful experiment, improving from vanilla BERT by a few points on both F1 and EM. To analyze the results, consider the following passage and respective question examples. We examine DD-BERT's output compared to that of vanilla pre-trained BERT. Note that this test set example was contained in the original SQuAD 2.0 dev set [12]. We manually extracted the corresponding ground truth answers below after training and evaluation on the *course-provided* train/dev sets for the sake of more descriptive analysis [14].

Context: A particularly simple example of a probabilistic test is the Fermat primality test, which relies on the fact (Fermat's little theorem) that np≡n (mod p) for any n if p is a prime number... More powerful extensions of the Fermat primality test, such as the Baillie-PSW, Miller-Rabin, and Solovay-Strassen tests, are guaranteed to fail at least some of the time when applied to a composite number.

Question: What does the Carmichael primality test depend on?
Ground Truth Answers: No Answer
DD-BERT Prediction: No Answer
Vanilla BERT Prediction: the fact (Fermat's little theorem) that np≡n (mod p) for any n if p is a prime number

Above, DD-BERT correctly predicted "No Answer" while vanilla BERT predicted an incorrect span. Notice that the correct answer is "No Answer" because a "Carmichael primality test" does not exist; only a "Fermat primality test" does. The vanilla BERT model likely hit a local maximum similar to the previous example, making an assumption that Carmichael is truly a primality test.

This is where the advantages of DD-BERT manifest, as the contextual representations for "Carmichael primality test" extracted from BERT are iteratively applied to the context, updating the start and

end token predictions each time. Because each token is individually but sequentially applied to the Hidden Maxout Networks, the lack of the token sequence "Carmichael primality" in the context warrented the question irrelevant, yielding a correct prediction that the proposed question has no answer.

Question: What is the name of one impressive continuation of the Fermat primality test? *Ground Truth Answers*: Baillie-PSW *DD-BERT Prediction*: Baillie-PSW, Miller-Rabin, and Solovay-Strassen tests *Vanilla BERT Prediction*: Baillie-PSW, Miller-Rabin, and Solovay-Strassen tests

Above illustrates why the SQuAD task is so difficult. While the context lists many continuations of the Fermat primality test, the question asks for just one. Both DD-BERT and vanilla BERT incorrectly output all continuations. This might motivate better architectures for question encoding, as the contextual attention for the word "one" was not enough to produce the correct span here.

Upon further analysis of test output from DD-BERT, we see cases where the model should improve from more training time and hyperparameter tuning. For example, DD-BERT missed a closed parenthesis in a span prediction, resulting in a wrong answer. While DD-BERT does improve from vanilla BERT by more closely examining the question's direct relation to tokens in the passage, it still misses the more challenging task of understanding the exact meaning of the question and therefore what to look for or avoid in the passage.

5.3 Answer Chunk Ranker

As our worst-performing model, the ACR-BERT model largely failed on all types of questions. Looking into the test set outputs, the ACR model performs exceptionally poorly on no-answer questions. For example, the ACR model predicts the same answer span as Vanilla BERT for the above no-answer question: *"What does the Carmichael primality test depend on?"* If BERT reaches a local maximum span as described in Section 5.2, the cosine similarity of the respective BERT contextual representations will likely reflect that local maximum, as ACR simply runs iterative cosine similarity comparisons of the start/end BERT representations of the question and answer spans.

For other examples where ACR-BERT predicts an answer for a no-answer question, contributing to our low no-answer metrics, we may be able to attribute those predictions to a poor choice of the no-answer thresholding function. Our architecture generates a no-answer when there is low variance among cosine similarity predictions. However, it might be necessary to also incorporate a fine-grain measure of skew, as the system generating all high similarity scores, for example, may not represent answer uncertainty. Finally, it may be worthwhile to train the model for significantly more epochs, as we were limited by the slower training speed (around 7.50 seconds per iteration at its fastest).

However, an upside that we noticed about the ACR architecture is that it is relatively light on GPU memory utilization in comparison to DD and Answer Pointer-BERT, allowing us to train with larger batch sizes.

6 Conclusion

Through this work, we've explored variations on how to interpret text and question encodings from BERT. We've evaluated the performance of BERT with a single dense layer, Pointer-Net, dynamic decoding, and dynamic chunking on SQuAD 2.0. Of those four models, the Dynamic Decoder performed the best with a test set F1 score of 75.884 (17.134 above baseline) while ACR performed the worst with a dev set F1 score of 20.666 (38.084 below baseline).

We've learned that the SQuAD 2.0 "no answer" extension to previous SQuAD datasets is actually a significant barrier that must be overcome by prediction models, as seen by our Pointer-Net and ACR implementations that have historically performed well on SQuAD 1.0/1.1 but see a massive drop in performance on SQuAD 2.0. The Dynamic Pointing Decoder, however, handles this extended domain well.

Given more time, we hope to better understand how Pointer-Net and ACR could be improved to handle SQuAD 2.0, whether it's by general hyperparameter tuning or specifically adjusting the no-answer prediction generation. We will also attempt to train the network longer, which should yield better performance as logit values systematically shift upwards.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. *arXiv e-prints*, page arXiv:1806.03822, Jun 2018.

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

[3] Laddie132. Match-lstm. `https://github.com/laddie132/Match-LSTM`.

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

[5] Yang Yu, Wei Zhang, Kazi Saidul Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end reading comprehension with dynamic answer chunk ranking. *CoRR*, abs/1610.09996, 2016.

[6] CS224n Course Staff. Cs 224n default final project: Question answering on squad 2.0, Feb 2019.

[7] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2017.

[8] HuggingFace. Pytorch pretrained bert: The big extending repository of pretrained transformers. `https://github.com/huggingface/pytorch-pretrained-BERT`.

[9] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

[10] Personal dev repo: Pytorch pretrained bert: The big extending repository of pretrained transformers (private). `https://github.com/dzhxo/cs224n_bert`.

[11] Atul Kumar. co-attention. `https://github.com/atulkum/co-attention`.

[12] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[13] Google-Research. google-research/bert, Feb 2019.

[14] Squad 2.0 the stanford question answering dataset. `https://rajpurkar.github.io/SQuAD-explorer/explore/1.1/dev/Prime_number.html`.

7 Appendix

7.1 Highway Maxout Network for Dynamic Decoder

We describe the structure of the HMN below. First, we use Tanh to create a non-linear projection of the current state: $r = \tanh(W^{(D)}[h_i; u_{s_{i-1}}, u_{e_{i-1}}])$ where $r \in \mathbb{R}^{p \times l}$ and $W^{(D)} \in \mathbb{R}^{l \times 51}$.

Our first Maxout layer is: $m_t^{(1)} = \max(W^{(1)}[u_t; r] + b^{(1)})$ with $W^{(1)} \in \mathbb{R}^{p \times 1 \times 3l}$ for pooling size p, and $b^{(1)} \in \mathbb{R}^{p \times l}$. The max operation computes the maximum value over the first dimension of our tensor. Similarly, the second Maxout layer is: $m_t^{(2)} = \max(W^{(2)} m_t^{(1)} + b^{(2)})$ with $W^{(2)} \in \mathbb{R}^{p \times l \times l}$, and $b^{(2)} \in \mathbb{R}^{p \times l}$. Altogether, these two layers are fed into our third Maxout layer as follows:

$$\text{HMN}(u_t, h_i, u_{s_{i-1}}, u_{e_{i-1}}) = \max(W^{(3)}[m_t^{(1)}; m_t^{(2)} + b^{(3)}])$$

where $W^{(D)} \in \mathbb{R}^{p \times 1 \times 2l}$, $b^{(3)} \in \mathbb{R}^{p}$, and $m_t^{(1)}$ and $m_t^{(1)}$ are the outputs of the first and second Maxout layers.

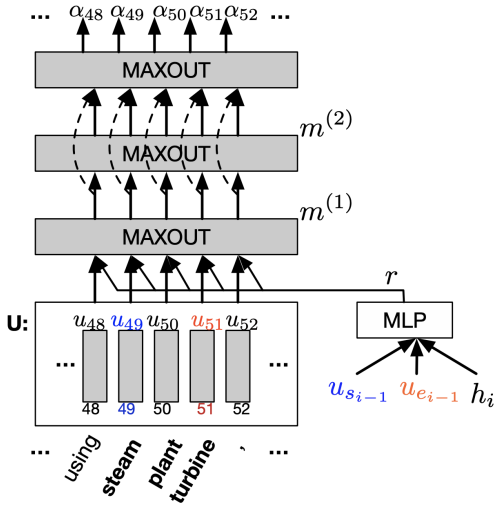

Figure 4: The Highway Maxout Network, from Xiong, et. al [4] with three maxout layers. The dotted lines represent the highway connections between the first and last maxout layers.