Answer Pointer Alternatives for Unanswerable Questions in SQuAD2.0

Evin Yang[*]
Department of Computer Science
Stanford University
Stanford, CA 94305
evin@stanford.edu

Abstract

Machine reading comprehension tasks where questions are answered by a segment of text in a passage are commonly approached by training a model to point to the start and end indices of the answer span within the passage. With the introduction of unanswerable questions in SQuAD2.0, the answer pointer method may be adapted to point to a zero-length span at the start of the input to indicate no answer. At the time of writing, fine-tuning BERT attached to an answer pointer "head" (final layer) achieves respectable results on SQuAD2.0, with pretrained weights for BERT readily available and the head easily interchangeable, making BERT an excellent candidate for running experiments. Using pretrained BERT, we investigate various alternative techniques in an attempt to beat the traditional answer pointer method: pre-classifying answerability, joint modeling with unanswerable questions filtered out during training, and "heat map" span prediction via token classification. While none of these methods significantly outperformed the answer pointer approach, error analysis reveals limitations and insights that may better motivate extensions and future work.

1 Introduction

Reading comprehension has long been a compelling problem in AI because improving benchmarks help push the frontier of natural language understanding, a subproblem of building intelligent machines. One such reading task, SQuAD, is formulated as a span prediction problem given a question and corresponding passage. In this paper, we probe the effectiveness of traditional answer pointer methods that train start/end index prediction via logits and softmax. Pointer methods were introduced with Pointer Net in 2015 [5] and applied to machine comprehension using match-LSTM architectures in 2016 [6]; answer pointer still remains the prevailing approach for tackling span prediction, even in SOTA models. Because of the lack of alternative span generation approaches in the field, we motivate and evaluate several hypotheses relating to the limitations of answer pointer and potential solutions, as described in later sections. In order to ensure these experiments are meaningful, we begin with a BiDAF baseline, then utilize and modify BERT to fit our needs.

2 Related work

2.1 BiDAF baseline

For a baseline, we use a Bidirectional Attention Flow (BiDAF) model without the character embedding layer included in the original [3]. In this BiDAF baseline, the embedding layer first converts words

[*]I use the collective "we" throughout this paper as a matter of habit; however, this was an individual effort.

Preprint. Work in progress.

(indices) into word embeddings of size D, which are projected to have dimensionality H by a learned matrix and then fed through a two-layer highway network. These hidden states of size H are now encoded by a bidirectional LSTM which concatenates forward and backwards hidden states for each time step, doubling their size to $2H$. In the attention layer, we compute a similarity matrix of scores for every pair of context and question hidden states and explicitly calculate both Context-to-Question and Question-to-Context Attention, taking row- and column-wise softmaxes followed by weighted sums, resulting in vectors of size $8H$. These are transformed by the modeling layer's two-layer bidirectional LSTM into vectors of size $2H$, which are consumed by another bidirectional LSTM in the output layer, projected, and run through softmaxes to yield p_{start} and p_{end}.

2.2 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a pre-trained deep language model that can be applied to a variety of NLP tasks, achieving state-of-the-art performance [1]. BERT is a Transformer encoder [4], a neural architecture that notably does not use convolution or recurrence, instead stacking blocks of attention and feed-forward layers. Within each Transformer encoder block, there is a multi-headed self-attention layer followed by a position-wise fully connected feed-forward neural network, with layer normalization in between and after these two sublayers. $\mathbf{BERT_{BASE}}$ has $L = 12$ Transformer blocks with a hidden size of $H = 768$, feed-forward size of $4H = 3072$, and $A = 12$ attention heads, totaling 110 million parameters. With $L = 24$ Transformer blocks, $H = 1024$ hidden size, $A = 16$ attention heads, and 340 million parameters, $\mathbf{BERT_{LARGE}}$ is a significantly bigger model and achieves markedly better benchmarks. However, due to limited compute resources, we will be using $\mathbf{BERT_{BASE}}$ to evaluate our hypotheses.

BERT's input schema uses three types of embeddings: token, segment, and positional. The input is tokenized and mapped to WordPiece embeddings and segment A/B embeddings are also added if necessary. To allow the Transformer to understand relative positioning, the input is also extended with positional embeddings, sine and cosine waves of different frequencies. A special [CLS] token prepends the input to allow for classification tasks, and a [SEP] token separates two sentences. As output, BERT produces a hidden state for each input token with hidden size H, which can be thought of as a deeply contextualized token embedding.

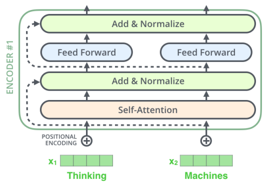


Figure 1: Transformer block

3 Approaches

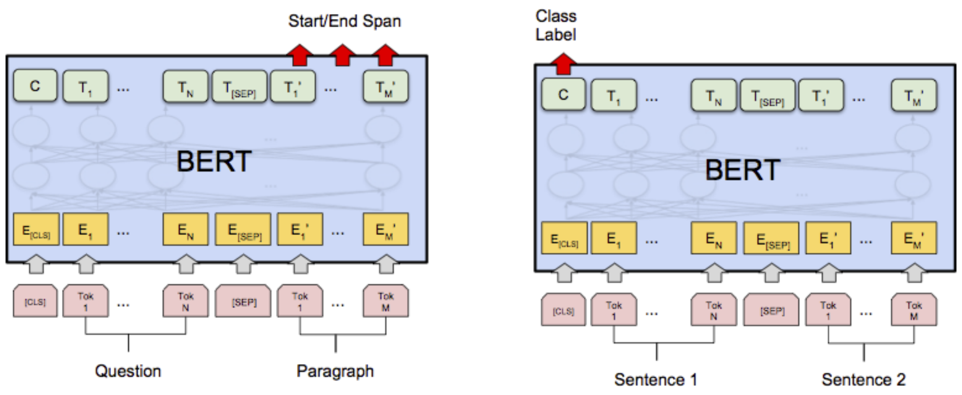
3.1 Fine-tuning BERT

Applying BERT to SQuAD is a relatively straightforward process. First, the question and passage are tokenized using WordPiece embeddings, then segment embeddings (segment A for the question, segment B for the passage) and positional embeddings are added. BERT generates hidden states of size H for each token, each of which is projected by linear layer of size $2 \times H$ (the answer pointer head) resulting in two sets of logits. Finally, we compute softmax for each of these two sets of logits, yielding p_{start} and p_{end} for each token.

We first attempted to fine-tune using the above approach as described by the paper authors, retraining all parameters of $\mathbf{BERT_{BASE-CASED}}$ on SQuAD, starting with HuggingFace's PyTorch implementation of BERT and pretrained weights. Next, we tried fine-tuning only the last linear layer instead by freezing all transformer blocks, leaving the final layer's $2 \times H$ weights and 2 biases as the only parameters to learn during training. Preliminary experiments revealed that the latter method—freezing BERT—was not viable, likely because adjusting a single linear layer was too shallow to adapt pretrained BERT to the SQuAD task.

3.2 Pre-classifying answerability

We hypothesized that SQuAD2.0 could be better approached with a joint conditional model instead of a single one. Specifically, given a question and its corresponding passage, a classifier could first determine whether or not the question was answerable based on the passage. If so, then a more



(a) Question answering head (b) Sentence pair classification head

Figure 2: Different heads, same BERT

specialized model could attempt to find the answer span for the question within the passage. The motivation here is that prior information—knowing that a question is answerable—could improve the quality of the answer span. Additionally, we believed that binary classification of the answerability of a SQuAD question is a reduced problem compared to finding the answer span, and therefore higher accuracy could be achieved.

Consequently, we swapped out the answer pointer head attached to BERT and replaced it with a two-class sequence classification layer. Specifically, for input tokens transformed to output hidden states by BERT, the hidden state corresponding to the special classification [CLS] token is projected by a linear layer of size $C \times H$ where $C = 2$ is the number of classes, resulting in two logits. Taking the softmax then yields the probability that a question is answerable or not given a passage.

3.3 Filtered training

In keeping with the motivation described in the previous section on pre-classification, we conjectured that the unanswerable questions in SQuAD2.0 might introduce noise into a model that seeks to predict answer spans. It follows that if a separate classifier could already determine with good accuracy whether or not a question was answerable, an answer span predictor should assume that the question is answerable. One way to encode such an assumption is to simply never introduce the answer span predictor to unanswerable questions, so we trained BERT with an answer pointer head on answerable questions only.

3.4 MultiBERT

Putting the previous parts together, we combined the pre-classifier with the answer span predictor trained on filtered data—two separate BERT models with different heads—to create a joint conditional model. Given a question and passage as input, the classifier first generates a binary signal for the input's answerability. If the question is deemed to be answerable, the BERT model fitted with an answer pointer head generates a span. Otherwise, the empty string is returned instead to indicate no answer.

We initially expected the classifier to have high accuracy given the perceived simplicity of the classification problem. However, when these expectations weren't met (as we'll discuss further in the results and analysis section), we replaced the classifier with a vanilla BERT model, overriding the vanilla model's span prediction with the filtered model's span while leaving the vanilla model's empty span prediction as is.

3.5 Heat map span prediction

We also formulated a novel strategy to generate spans for reading comprehension tasks in an effort to explore alternatives to the traditional answer pointer method. This approach, inspired by heat map graphics, involves classifying each token in the question-passage input as to the left of, within, or to the right of the answer. This requires attaching a three-class, token-level classification head to BERT which projects each token's output hidden state by a linear layer of size $C \times H$ where $C = 3$, yielding $N \times 3$ logits where N is the number of tokens in the input sequence.

Training examples were labeled accordingly, with each token classified based on its location with respect to the start/end indices of the answer span. For unanswerable questions, all tokens were labeled with the "left" class. In order to map the output logits to a span for evaluation, we take the log_softmax over each set of three logits, then use dynamic programming to check all N^2 start/end indices and choose the answer span that maximizes its global probability. Specifically, the global probability of a span is the sum of "left"-class log probabilities to the left of the start index, plus the sum of "within"-class log probabilities from the start index to the end index inclusive, plus the sum of "right"-class log probabilities to the right of the end index. Although this calculation takes $O(N^2)$ time, it only needs to be done at evaluation and it is relatively cheap compared to the cost of training BERT (couple minutes per dev evaluation versus few hours per training epoch).

The motivation behind seeking an alternative to answer pointer is that relying on start and end indices alone may be brittle; for a hard example, we could conceivably see a high start or end logit activation for a spurious token beyond the correct span, which might drag a span boundary to something unreasonable. With heat map span prediction, we hoped to see more robustness to such outliers and thus better performance. Because we have a left, within, and right classification for every token, even if a token outside the ground truth span has an incorrect classification, the predicted span may remain stable. Furthermore, associating each example with a single start and end index implies there is only one span in the passage that contains the answer. Even if our ultimate goal is to extract a single answer span, heat map span prediction may reinforce better learning when the answer appears in multiple locations in the passage.

4 Experiments

4.1 Data

We use the Stanford Question Answering Dataset v2.0 (SQuAD2.0) which consists of thousands of question-answer pairs on hundreds of passages [2]. Notably, unlike its predecessor SQuAD1.1, SQuAD2.0 also contains adversarially-written, unanswerable questions, forcing systems to distinguish whether or not a question is answerable by the given passage.

The data splits are as follows:

- **TRAIN** has 129,941 examples taken from the official SQuAD2.0 training set.
- **DEV** has 6078 examples randomly selected from the official SQuAD2.0 dev set. Some of these are then filtered out because they exceed max sequence length, so dev submission files actually have 5951 examples.
- **TEST** has 5915 examples—the other part (roughly half) of the official SQuAD2.0 dev set.

4.2 Evaluation method

The SQuAD leaderboard features two evaluation metrics:

- **Exact Match (EM)** is a binary indicator of whether or not the system response matches the ground truth answer exactly; partial matches earn a score of 0 here.
- **F1** is the mean of precision and recall, so if the ground truth answer is "New York City", a system response of "New York" would have 100% precision, 66.67% recall, and 80% F1.

Between these two metrics, EM is the stricter: mathematically, it must be less than or equal to F1. We seek to maximize both; however, F1 is somewhat better at capturing the usefulness of system responses, especially when the response differs only slightly from a ground truth answer.

4.3 Experimental details

We first trained BiDAF baseline for 30 epochs at a learning rate of 0.5, exponential moving average decay rate of 0.999, and random seed of 224. Each epoch took approximately 20 minutes on a single-GPU NV6 machine, totaling over 10 hours of training time. We then attempted to fine-tune $\mathbf{BERT_{BASE-CASED}}$ for 2 epochs with a batch size of 4, initial learning rate of 5e-5, and random seed of 42, which took around 11 hours on an NV6. Max sequence length and query length for truncation and padding were left at 384 and 64, respectively, and both attention and hidden dropout probabilities remained at 0.1. Because of GPU constraints on the NV6, the machine would run out of memory if the batch size was raised above 4. As we found out, having a sufficiently large batch size is absolutely crucial to training properly and makes a vast difference in resulting performance. We also tried fine-tuning $\mathbf{BERT_{BASE-CASED}}$ with all Transformer blocks frozen (leaving only the final linear layer to be trained) for 3 epochs on an NV6 with mostly identical hyperparameters, but with a batch size of 32. This increased batch size could be afforded because a gradient no longer needed to be propagated across all of BERT, saving compute resources. Each epoch took about 1.5 hours, totaling under 5 hours of training time.

To successfully fine-tune $\mathbf{BERT_{BASE-CASED}}$, we needed to upgrade to a two-GPU NV12 machine (roughly double the specs of the NV6), keeping all other hyperparameters the same except training batch size, which was bumped up to 12. For vanilla BERT (answer pointer head), we used the HuggingFace implementation of BertForQuestionAnswering. For the pre-classifier and heat map span prediction, we adapted BertForSequenceClassification and BertForTokenClassification. We trained all models after the BiDAF baseline for 2-3 epochs, (except heat map, which was capped at 1 epoch for reasons we'll discuss later), saving checkpoints and running dev evaluations after each epoch. On an NV12, each training epoch with batch size 12 took about 3.25 hours, and each filtered training epoch took 2 hours. A full epoch on a four-GPU NV24 (double the specs of the NV12) with batch size set at 24 took about 1.75 hours, but we ultimately avoided using the NV24 in favor of two NV12s for better parallelization.

4.4 Results

The BiDAF baseline achieved 60 F1, which is reasonable given the model architecture. Vanilla BERT (answer pointer) earned a respectable 74.6 F1, a significant increase from the baseline as we would expect. Surprisingly, the BERT classifier fell far short of solving the task of determining answerability, correctly tagging only 55.6% of unanswerable questions. Filtered BERT answer pointer did achieve 88.4 F1 on all answerable questions, but this is somewhat inflated since in practice, the filtered model won't be applied to ground truth answerable questions—only those that are deemed answerable by a preceding model. Indeed, when we pair the vanilla and filtered BERT models to create MultiBERT, the resulting F1 of 74.7 is not significantly better than the original BERT answer pointer. Finally, the BERT heat map predictor had a low EM of 47.8 and a mediocre F1 of 60.1. We explore potential reasons for failure in the following section.

Table 1: Dev set results

Method or experiment	Batch size	Epochs	Eval	EM	F1
BiDAF baseline	64	30	All	56.91	60.43
BERT answer pointer	12	2	All	71.74	74.60
BERT pre-classifier	12	2	NoAns	55.59	55.59
Filtered BERT answer pointer	12	2	HasAns	80.69	88.42
MultiBERT	12	2	All	71.77	74.71
BERT heat map	24	1	All	47.84	60.11
BERT answer pointer	4	2	All	0.10	10.24
Freeze BERT, fine-tune QA layer	32	3	All	52.16	52.17
No answer everything	-	-	All	52.19	52.19

It should be noted that all successful BERT experiments were run with batch size 12; at batch size 4, the resulting F1 of 10.2 after 2 epochs indicated that learning was not progressing at a reasonable pace. This is most likely because there was too much variance in the losses and gradients computed on small batches, preventing the model weights from being pushed in a productive direction. Furthermore, freezing BERT and fine-tuning just the last layer did not yield good outcomes even at a batch size of 32 after 3 epochs. The resulting F1 for frozen BERT was 52.2 because the vast majority of predictions were null, suggesting that fitting the pretrained BERT model to the SQuAD task requires a deeper adjustment than fine-tuning only a single shallow layer.

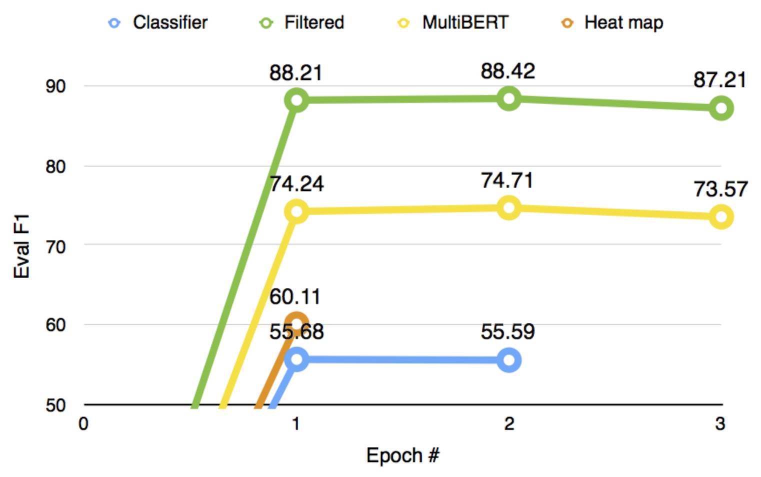


Figure 3: Dev eval F1 vs number of epochs trained

For all BERT experiments with batch size 12, dev performance appeared to peak after just two epochs. This is a likely a testament to the magnitude and consistency of SQuAD training and the power of the pretrained BERT model to adapt quickly to downstream NLP tasks.

Final MultiBERT eval submissions to the PCE SQuAD leaderboards were 71.769 DEV EM and 74.710 DEV F1, and 71.327 TEST EM and 74.711 TEST F1.

5 Analysis

The most surprising result was the failure of the BERT classifier to perform much better than base 50% binary classification. The initial hypothesis—that SQuAD formulated as an answerability classification problem would represent a reduction in difficulty—does not seem to hold up. Consider how the answer pointer head predicts an unanswerable question: start and end index logits are computed for every token, and if the start and end logits for the first index have the highest combined probability, then the predicted span is an empty one: no answer. Thus, there are explicit and intuitive comparisons in answer pointer across all output hidden states. With the classifier, we rely on BERT to instead encode the answerability classification within just a single vector of size $H = 768$ corresponding to the [CLS] token, as opposed to a vector for every token, up to 384 input tokens. It's possible that by "reducing" the problem to classification, we may have inadvertently introduced the very curse of dimensionality that attention mechanisms and non-recurrent Transformer architectures were designed to solve in the first place.

Even if classification is a bust, we can treat an original BERT answer pointer as a classifier and still execute a joint conditional model (MultiBERT) with filtered training. Recall that filtered training was backed by a suspicion that SQuAD2.0 unanswerable questions constitute noise that may confuse the BERT model, and keeping these unanswerable questions away from a "clean" model during training could yield better results on answerable questions. Although the filtered model did technically achieve higher HasAns EM/F1, this is because it treats all dev examples as answerable questions; after accounting for answerability classification error, the filtered model is effectively equal to the original BERT. This tells us that including unanswerable questions and empty spans in SQuAD does not distract much from answerable questions and non-null spans—it turns out that answerability classification and span prediction are similar, likely overlapping tasks. BERT is powerful enough and pretrained enough to do both, so filtering doesn't result in much of a gain, a notion supported by the fact that the BERT converges after two epochs.

Heat map span prediction training was cut short after one epoch because the generated spans were extremely sloppy, and there were straggling tokens where there should have been empty predictions (often punctuation marks). In hindsight, there's a fairly apparent reason why heat map didn't work: by classifying every token in the input, we introduced less important or even irrelevant targets to minimize loss on, and treated them as equally important to the answer token classifications. For example, consider a token on the very right edge of the input, far away from the answer span. If the [left, within, right] probabilities for that far right token are initially projected to be [0.2, 0.2, 0.6] and then improved to [0.1, 0.1, 0.8] (closer to the ground truth label probabilities of [0, 0, 1]), the decrease in loss would be the same as if we had improved the probabilities for a token within the answer span from [0.2, 0.6, 0.2] to [0.1, 0.8, 0.1] (closer to the ground truth label probabilities of [0, 1, 0]). The token far away from the ground truth span doesn't have much impact on the answer span itself, but tokens close to the answer span do, and this emphasis is not reflected in the loss function, resulting in very poor training.

6 Conclusion

We sought alternate methods besides answer pointer to address the SQuAD2.0 reading comprehension task with unanswerable questions, and our inability to significantly outperform answer pointer highlights its architectural strength. However, error analysis surfaced a couple insights for extending these methods or how to explore new approaches—both where to look and where not to look. Due to the nature of the task and the complexity of question answerability, binary classification is likely a dead end. However, filtered training and data segmentation strategies could prove useful for harder NLP tasks where noisy examples do actually "confuse" the model. Data/label augmentation, where additional meaningful labels are attached to examples and included in loss during training, could also help with performance. For heat map prediction, one possible solution is to localize and weight loss targets such that the closer a token is to the answer span, the higher priority given to minimizing its classification error. Reducing the three classes to two classes (inside/outside) could also support multiple-span prediction for difficult or lengthy passages where the answer appears multiple times (which is rare in the current version of SQuAD). More broadly, the biggest practical takeaway from running experiments with BERT is the importance of training batch size; valuable resources and time were spent to discover that small batch sizes cripple learning. This research provides a strong launching point to iterate faster in future work.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, art. arXiv:1810.04805, Oct 2018.

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

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *arXiv e-prints*, art. arXiv:1611.01603, Nov 2016.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, art. arXiv:1706.03762, Jun 2017.

[5] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. *arXiv e-prints*, art. arXiv:1506.03134, Jun 2015.

[6] Shuohang Wang and Jing Jiang. Machine Comprehension Using Match-LSTM and Answer Pointer. *arXiv e-prints*, art. arXiv:1608.07905, Aug 2016.