# SQuAD: Integrating PCE and Non-PCE approaches

Oghenetegiri Sido
Department of Computer Science
Stanford University
osido@stanford.edu

March 20, 2019

**Abstract**

Question Answering (QA) is an increasingly important NLP problem with the proliferation of chatbots and virtual assistants. In October 2018, **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**) was released and achieved state-of-the-art results on a variety of NLP tasks, including QA, Token Classification, and Next Sentence Prediction. We seek to extend BERT with other performant QA architectures for SQuADv2.0. We utilize traditional pre-trained word vectors and **P**re-trained **C**ontextual **E**mbeddings (**PCE**). Specifically, we experiment with a variety of non-PCE approaches, including **Bid**irectional **A**ttention **F**low (**BiDAF**) [3], **D**ynamic **C**oattention **N**etwork (**DCN**) [4], and Answer-Pointer. We find that using BERT embeddings in conjuction with non-PCE approaches is more performant for training than relying solely on PCE or non-PCE techniques. Our best model, BERT-CASED achieves an F1 of 75.3 and EM of 71.9 on the dev set and an F1 of 77.1 and EM of 73.6 on the test set.

## 1 Introduction:

Machine Comprehension - the ability for a system to understand text at a deep level - is a well-researched yet challenging task. As applied to the **S**tanford **Q**uestion **A**nswering **D**ataset (**SQuAD**)[1], we are tasked with building a system that is not only adept at finding the answer to a question in the a context paragraph but also determining whether the question can even be answered. Traditionally, previous state-of-the-art QA systems relied on pre-trained word vectors and a variety of high-performing networks, including **Bid**irectional **A**ttention **F**low (**BiDAF**) [3] and **D**ynamic **C**oattention **N**etwork (**DCN**) [4]. These systems all performed extremely well on SQuAD1.1 but far worse on SQuAD2.0, which contains over 50,000 adversarial questions that could be answered. Then, in October 2018, BERT [2] was released and achieved state-of-the-art results for several different NLP tasks, including QA. This is due to the fact that, among other additions, BERT uses **P**re-trained **C**ontextual **E**mbeddings (**PCE**), where embeddings for every token are learned specifically within the context of their surrounding words. This encodes much more locale-specific information than traditional pre-trained vectors that are more general and not tied to a particular context.
Our paper is focused on exploring how high-performing non-PCE techniques can be applied to BERT and vice-versa for increased performance in both paradigms, PCE and non-PCE.

1. We apply non-fine-tuned BERT embeddings to standard BiDAF and DCN implementations, using the hadamard product of BERT and GLoVE before feeding these embeddings to the networks. We find that using BERT embeddings in conjuction with GLoVE speeds up training time and convergence but results in similar performance on the dev and test sets.

2. We then seek to apply a time-tested non-PCE approach, Answer-Pointer, to the BERT implementation, which currently uses a simple linear layer. We ran Answer-Pointer experiments using both an RNN and GRU in place of BERT's linear output layer. Additionally, we combined the outputs of our Answer Pointer layer and their linear layer. We find that our Answer Pointer implementation performs far worse than their simple linear layer. We will analyze why this is the case below.

# 2 Related Work:

- Rajpurkar et. al [1], creator of the SQuAD dataset, published a paper describing the dataset's characteristics and the results of a simple logistic regression baseline, which achieved and F1 and EM score of 51.0.

- Early summarization - the consequence of a lacking a comprehensive summary of the context in relation to the question - has hampered the performance of earlier systems that only employed Context-to-Question (C2Q) attention [3]. Accordingly, Seo et. al [3] and his associates introduced **B**i-**D**irectional **A**ttention **F**low (BiDAF), which not only utilizes combined Context-to-Question (C2Q) attention but also Question-to-Context (Q2C) attention with stellar results. BiDAF, alongside its attention layer, implements character embedding, word embedding, encoder, modeling, and output layers. Our BiDAF and BERT-BiDAF experiments rely upon this existing implementation, with the exception of the character embedding layer.

- Xiong's et. al [4] **D**ynamic **C**oattention **N**etworks (DCN) differentiates itself from BiDAF with the addition of learnable hidden states that are concatenated to the context and question hidden states. They faciliate the correction from local maxima during training. Additionally, DCN employs a biLSTM encoder - LSTM decoder duo to estimate the start and end indices of the span. Our model implements the DCN attention layer in place of BiDAF attention in the larger BiDAF model.

- Wang's et. al [5] Answer Pointer addresses the issue of independent start and end index prediction. Answer Pointer feeds modeling layer outputs to the RNN, whose outputs are then used to compute the start logits via attention. Next, it feeds the *final hidden state* from the prior RNN to the same RNN with the inital modeling layer outputs, repeating the process to derive the end logits. This encodes a dependence between the start and end indices. We integrate Answer Pointer into BERT, using both an RNN and a GRU.

- Finally, with respect to the landmark BERT paper [2], it is designed to pre-train bidirectional representations by conditioning on both left and right context in all layers. Additionally, these representations can be fine-tuned with additional output layers that can be used to build state-of-the-art models for a variety of NLP tasks like QA. Unlike other models, BERT can outperform humans on SQuADv1.1. Its model architecture includes a multi-layer bidirectional transformer encoder, transformer blocks,

hidden states, self-attention heads, and a feed-forward filter. The BERT transformer performs birectional self-attention. We modify the output layer of BERT with Answer Pointer and experiment with its non fine-tuned embeddings in BiDAF and DCN.

# 3 Approach:

## 3.1 BiDAF Baseline

BiDAF relies on the following layers:

1. *Embedding Layer:*
   After looking up a pre-trained GLoVE for a particular token, $h_i$, we run it through two highway networks.

2. *Encoder Layer:*
   We use a bi-LSTM to encode temporality into the embedding sequence, honing in on importance and meaning based on the position of the word in the context or question. This produces hidden states that are twice the dimension of the embeddings due to concatenating the forward and backward states at a particular timestep.

3. *Attention Layer:*
   Hidden states are fed to this attention layer that performs both context-to-query and query-to-context attention. To facilitate this, it uses a similarity matrix, $S$, where $S_{ij} = w_{sim}^T[\ c_i; q_j; c_i \odot q_j]$ . This encodes the strength between a particular context token ($c_i$) and question token ($q_j$).

   (a) Context-to-Question (C2Q)
   Our context-to-question attention is derived by taking the softmax over every row as $\hat{S}$. Our attention outputs $a_i$ are a weighted sum of the question states, where the weights are $\hat{S}_{ij}$.

   (b) Question-to-Context (Q2C)
   Our question-to-context attention is computed by taking the softmax over every column as $\tilde{S}$. We then get $S' = \hat{S}\tilde{S}$. Our attention outputs $b_i$ are the weighted sum of the context states, where the weights are $S'_{ij}$.

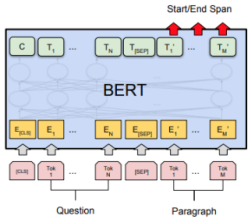
   Finally, we concatenate $c_i$, $b_i$, and $a_i$ and their hadamard products as our final output.

4. *Modeling Layer:*
   Uses a bi-LSTM to encode temporality into the attention outputs.

5. *Output Layer:*
   Takes the output of the modeling layer ($M$) and the attention outputs ($G$), applying a bi-LSTM to produce $M'$. Using trainable weight matrices and a softmax layer, $M$ is transformed into start logits and $M'$ is transformed into end logits.

## 3.2 BERT-BiDAF Model

Utilizing the HuggingFace [1] implementation of BERT, we extract the embeddings for every token in the context and question.




Since BiDAF and other non-PCE methods are proven to work effectively with pre-trained word vectors, we use an original approach to retain the GLoVE embedding information for a given token while adding the hadamard product of that GLoVE embedding and its corresponding BERT embedding. This retains the information from the GLoVE embedding but adds useful BERT information, which is derived from the particular context in which the token was taken. Note that we project the BERT Embedding (*bert*) down to match the dimensions of corresponding GLoVE embedding.

$$bertProj = Wbert \in \mathbb{R}^{300}$$
$$combinedEmbedding = glove + glove \odot bertProj \in \mathbb{R}^{300}$$

We put these *combinedEmbedding*(s), through our BiDAF [3] network.

## 3.3 Dynamic Coattention Network Model

We implement the attention layer from the Dynamic Coattention Network paper [4]. Unlike BiDAF, it includes secondary-level attention output. Taking our hidden states for both question and context, $c_1, ..., c_N \in \mathbb{R}^l$ and $q_1, ..., q_M \in \mathbb{R}^l$, we project and use tanh to get our new question hidden states: $q'_j = tanh(Wq_j + b) \in \mathbb{R}^l$. Next, we add trainable sentinel vectors to both context and question matrices:

$$c = \{c\ c_0\} \in \mathbb{R}^{(N+1)xl}$$
$$q' = \{q'\ q_0\} \in \mathbb{R}^{(M+1)xl}$$

Then, akin to the similarity matrix in BiDAF, we build an affinity matrix, $L$, where $L_{ij} = c_i^T q'_j \in \mathbb{R}$. Afterwards, we perform C2Q attention by taking the row-wise softmax over L to yield $\alpha$. We then compute the attention output $a_i \in \mathbb{R}^l$ by taking weighted sums of the new question states, where the weight is $\alpha_j^i$. Then, we perform Q2C attention by taking the column-wise softmax over L yielding $\beta \in \mathbb{R}^{Mx(N+1)}$. We derive our attention output $b_j \in \mathbb{R}^l$ by taking the weighted sum of the context hidden states, where the weight is $\beta_i^j \in \mathbb{R}^l$. Then, we compute our second level outputs, $s_i \in \mathbb{R}^l$ as the weighted sum of the Q2C

[1] `https://github.com/huggingface/pytorch-pretrained-BERT`

attention outputs where the weight is $a_j^i$. Finally, we use a bi-LSTM to encode our stacked outputs:

$$\{u_1, \ldots, u_N\} = biLSTM(\{\{s_1; a_1\}, ..., \{s_N; a_N\}\}) \in \mathbb{R}^{Nx4l}$$

### 3.4 BERT-Answer-Pointer

As mentioned previously, BERT's current implementation uses a linear layer to compute the start and end logits *independently* from the sequence outputs which come after BERT's encoder and pooler layers. Our Answer Pointer implementation enforces dependence between the start and end logits by conditioning our end prediction on our start prediction. To facilitate this, we use an RNN (also a GRU in a separate experiment) to run our sequence outputs through producing a probability distribution for each of the start indices and a final hidden state. We compute attention on the distribution where the hidden state attends to the initial sequence outputs, producing our start logits. To compute our end logits, we repeat the same process, but also use the final hidden state from the first pass of the RNN as an additional input alongside the initial sequence outputs to the RNN. We again, compute our logits in the same way, but now, the end logits have a dependence on the start logits via the passed through hidden state.

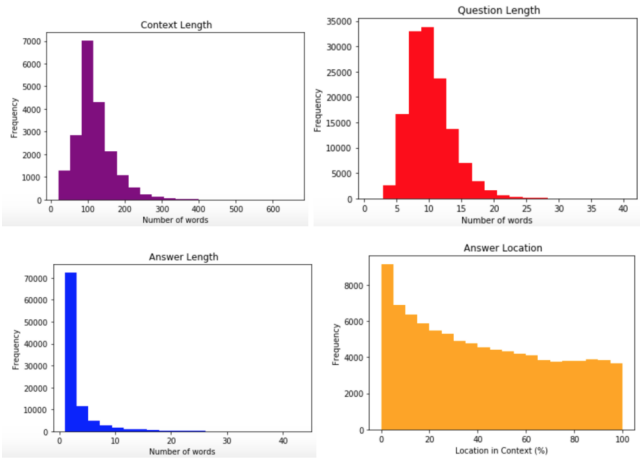
#### 3.4.1 BERT-Linear-Answer-Pointer

Here, we use a GRU instead of an RNN, due to its increased flexibility with its update gate. We add the logits outputted by the Answer Pointer layer with the logits created with the linear layer.

## 4 Experiments

### 4.1 Data

SQuAD [1] is comprised of paragraphs, questions, and their corresponding answers from Wikipedia. Our data split has 129,941 training examples, 6078 dev examples, and 5915 test examples.



#### 4.1.1 Data Exploration

The histogram for context length, reveals that the vast majority of context paragraphs are fewer than 300 words in length. Additionally, we see from our question length histogram that the majority of questions are less than 20 words in length. The starter BiDAF implementation allows for contexts of up to 400 words in training and 1000 words in evaluation. For question length, it allows up to 50 words in training and 100 words in evaluation. Shrinking this max based on the histogram will likely drop a few examples but keep most, drastically reducing computational cost. However, our BERT implementation has a max sequence length of 384 for both the question and context lengths combined, which is appropriate given that our exploration revealed that an appropriate max would be around 320 for both combined. Then, an examination of answer location reveals that while answer location appears uniformly distributed, many answers are in beginning of the context with noticeable drop-off as we near the end of the context.

### 4.2 Evaluation Method

Quantitatively, we use *EM*, the expectation-maximization metric based on matching the number of correct words for a particular answer as well as *F1*, which is the harmonic mean of precision and recall. Qualitatively, we conduct error-analysis of 25 randomly selected examples, comparing our model's predictions to the true answers. This helps in identifying persistent shortcomings and strengths of the models.

### 4.3 Experimental Details

#### 4.3.1 BERT embeddings

For BERT-BiDAF and BERT-DCN, to integrate the non fine-tuned BERT embeddings into our implementation, we instantiate a *bert-based-cased* model to generate embeddings for each example. We write these embeddings to disk and read them when needed in place of the embedding layer in the standard implementation.

#### 4.3.2 Hyperparameter Search

More hyperparameter searching will be conducted in future work, but here, we start with most of the default values:

- Starter Implementation:
  Settled on the defaults for *learning rate*: 0.5; dropout: .2; *max paragraph length*: 400; *max question length*: 50; *batch size*: 16, we reduced it from 64 to 16 due to memory constraints.

- BERT Implementation

  Settled on the defaults for *learning rate*: 5e-5 (we experimented with faster learning rates but dealt with too much divergence that affected increased loss in different periods of training); *max sequence length*: 384 (as seen from the histograms, we could have lowered to 320; however, this likely would only lower computational cost, not improve performance); *dropout*: .1 (lower than what we used in the standard implementation); *batch size*: 6 (due to memory constraints since each embedding is 768 dimensions); *gradient accumulation steps*: 4 (produces noisy error but is better than stochastic in a appropriate gradient update).

#### 4.3.3 Training Time

- Starter Implementation:
  We train over 30 epochs. Our BERT-BASIC model only has two linear layers and requires approximately 50 minutes per epoch; BERT-BiDAF uses a linear projection, and the complex BiDAF architecture takes approximately 1 hour; BERT-DCN performs complex attention and has an additional biLSTM encoder, taking approximately 1.3 hours per epoch. DCN takes approximately 30 minutes per epoch, which is similar to standalone BiDAF.

- BERT Implementation:
  Here, we train over 3 epochs. For BERT-cased and BERT-uncased, each epoch takes approximately 4 hours; however, BERT-Answer-Pointer and BERT-Linear-Answer-Pointer both rely upon either a RNN our GRU and require 5.5 hours per epoch.

### 4.4 Results And Analysis

We submitted our best model, BERT-CASED, to the test leaderboard: **EM: 73.609 ; F1: 77.065**. To understand the relative strengths and weaknesses of our models, we've selected 25 random examples and analyzed their responses.

Table 1: Dev Set Perfomance

| Model | EM | F1 |
|---|---|---|
| **BERT-Cased** | **71.9** | **75.3** |
| BERT-Uncased | 71.6 | 74.7 |
| BERT-BiDaF | 56.4 | 59.4 |
| BERT-DCN | 52.75 | 56.14 |
| BERT-Answer-Pointer-RNN | 43.4 | 49.7 |
| BERT-Linear-Answer-Pointer-GRU | 67.7 | 71.1 |
| DCN | 54.1 | 56.8 |
| BiDAF Baseline | 55 | 58 |

Table 2: Random Examples (25)

| Question Type | Frequency |
|---|---|
| What? | *14* |
| Which? | 1 |
| Where? | 4 |
| How? | 2 |
| Who? | 3 |
| Whose? | 1 |
| Unanswerable | 13 |

| Model | Accuracy (%)* | AvNA (%) |
|---|---|---|
| *BERT-Cased* | *74.0* | *53.8* |
| BERT-Uncased | 66.0 | 38.5 |
| BERT-BiDaF | 68.0 | 38.5 |
| BERT-DCN | 52.0 | 38.5 |
| BERT-Answer-Pointer-RNN | 54.0 | 61.5 |
| BERT-Linear-Answer-Pointer-GRU | 62.0 | 30.7 |
| DCN | 48.0 | 7.7 |

ID: 00d60faa383c8a6beffbc2bff
**CONTEXT**: Several commemorative events take place every year. Gatherings of thousands of people on the banks of the Vistula on Midsummer's Night for a festival called Wianki (Polish for Wreaths) have become a tradition and a yearly event in the programme of cultural events in Warsaw. The festival traces its roots to a peaceful pagan ritual where maidens would float their wreaths of herbs on the water to predict when they would be married, and to whom. By the 19th century this tradition had become a festive event, and it continues today. The city council organize concerts and other events. Each Midsummer's Eve, apart from the official floating of wreaths, jumping over fires, looking for the fern flower, there are musical performances, dignitaries' speeches, fairs and fireworks by the river bank.

**QUESTION**: What will maidens be able to predict by floating their programmes down the Vistula

**TRUE ANSWER**:

**is_impossible**: True

### 4.4.1 Discoveries:

Each discovery is motivated by its performance on the randomly selected examples and experimental results.

- *Specific Example above*:

  In this example about maidens, all models – with the exception of BERT-DCN and BERT-BiDAF – fail to determine that this question is unanswerable and answer something about "marriage". The key here is in "will" inside the question. The models are not understanding past versus future here, demonstrating a lack of deep understanding.

- *BERT-CASED* vs *BERT-Answer-Pointer*

  The only difference between BERT-CASED and BERT-Answer-Pointer is the insertion of the Answer-Pointer layer in place of the linear layer to generate the logits. Answer-Pointer has proven to be highly successful in non-PCE implementations due to its ability to encode dependence between start and end logits, so one would expect better results from Answer-Pointer. However, BERT-Cased's simple linear layer performs much better. We attribute the difference in performance to a couple of factors: a linear layer trains much faster than an RNN and clearly picks out important information needed for logits. Additionally, RNN's have been proven to be plagued by early-summarization, as they don't have the additional gate memory that LSTMs and GRUs have. This suspicion was confirmed by the randomly selected examples, where BERT-Answer-Pointer suffered from generating logits that resulted in a shorter parts of a complete answer approximately 20% of the time.

- *BERT-CASED* vs *BERT-UNCASED*

  We find that case matters in QA, as BEST-CASED surpasses its uncased counterpart in both EM and F1. Our error analysis finds that BERT-UNCASED suffers from predicting that a question is unanswerable when it, in fact, does have an answer. Therefore, we hypothesize that converting everything to lower case lowers the effectiveness of the attention and output layers.

- BERT-BiDAF/DCN vs BiDAF/DCN

  By examining our training metrics, we see that the hadamard of BERT and GLoVE aids in allowing the models to converge faster to their maximum EM and F1. In most cases, the end performance is the same as the

non-BERT models. This is due to the fact that the BERT embeddings we are using are not fine-tuned. They are frozen, but they add value in contextual information due to BERT's underlying FastText approach [2]. This valuable information is learned later on in training by the non-BERT models, which is why BERT helps in converging faster.

- BERT-CASED vs BERT-Linear-Answer-Pointer (GRU)

  While both of these implementations are based on the *bert-base-cased* pretrained models, BERT-Linear-Answer-Pointer uses both the linear layer as well as Answer-Pointer, adding the logits from the different sources. We find that it performs worse. We had hpped that a linear layer and Answer-Pointer could learn different components of the logits but that did not occur. This time, our Answer-Pointer used a GRU instead of an RNN, to capture more long-range dependencies which in turn could produce better probability distributions; however, this was not the case. Our error analysis revealed that BERT-Linear-Answer-Pointer suffers from attempting to answer unanswerable questions. This makes sense because we add the logits without any normalization. Accordingly, the probabilities for every index is magnified, so the system attempts to answer more than appropriate.

# 5 Conclusion:

- Non fine-tuned BERT embeddings can help speed up training in non-PCE implementations.
- BERT-CASED is the most performant model; however, it's main issue is attempting to answer unanswerable questions. We need to investigate on more advanced mechanisms to determine answerability to improve performance.
- True understanding of text is still a significant challenge, as evidenced by our error analysis.

# References

[1] Rajpurkar, P. & Zhang, J. (2016) Squad: 100,000+ Questions for Machine Comprehension of Text, *abs/1606.05250*

[2] Devlin, J. & Chang, M.W. (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *arXiv:1810.04805*

[3] Seo, M. & Kembhavi, A.F. (2016) Bidirectional Attention Flow for Machine Comprehension, *arXiv:1611.01603*

[4] Xiong, C. & Zhong, V. (2016) Dynamic Coattention Networks for Question Answering., *arXiv:1611.01604*

[4] Wang, S. & Jiang, J. (2016) Machine Comprehension Using Match-LSTM and Answer Pointer, *arXiv:1608.07905*