We would like to choose option 2 (ungraded). Thank you all course staff in making this quarter amazing and offering tremendous help in the current uneasy situation due to corona virus.

Machine Reading Comprehension on SQuAD 2.0

Stanford CS224N Default Project

Jiajia Luo, Yiyang Shen
Department of Computer Science
Stanford University
luojj19@stanford.edu, yiyangs@stanford.edu

Abstract

This project aims to produce a question answering system that works well on SQuAD 2.0, particularly, using an improved version of BiDAF model with Non Pre-trained Contextual Embeddings methods. We seek to increase both EM and F1 scores on the leader board as well as training our models to better understand and answer questions based on given contexts. We have explored character embedding, multi-head self attention, and syntax-guided self-attention and ensemble strategies to improve our model from baseline. So far with our best performing ensemble model, we have achieved an F1 score: 66.46 and EM score: 64.42 on dev set.

1 Introduction

In recent years, question answering as a machine reading comprehension (MRC) task has been the focus of many recent advancements in NLP research. One of the most popular datasets for measuring success at this task is the Stanford Question Answering Dataset (SQuAD). SQuAD consists of context paragraph, question, and answer triplets. A model that solves this task takes in the context paragraph and question as input, and predicts the answer to this question as a span of text from the context paragraph. SQuAD 2.0 also introduces unanswerable questions and motivates an additional metric, "Answer vs No Answer" (AvNA), which measures the models accuracy at classifying questions as answerable or unanswerable.

2 Related Work

The most successful models on the current SQuAD leaderboard now exclusively use pre-trained contextual embeddings (PCE). Specifically, PCE models use pre-trained word embeddings, for example bidirectional encoder representations for transformers (BERT) [1] and ALBERT [2], that depend on the context of the paragraph they appear in. We are inspired by innovative augmentations in PCE models by using syntax-guided self-attention network [3] and want to adapt and reproduce that feature onto Non-PCE models. The SG-Net paper [3] is unique in a way that it proposes a new perspective in developing attentive mechanisms. Adding syntactic constraints also agrees with our intuition that capturing meaningful syntactic information will help machine reading comprehension. This report will describe efforts to improve a baseline Bi-Directional Attention Flow (BiDAF) model using Non-PCE methods. We will reproduce the SG-Net implementation and verify its impact.

3 Approach

The work in this paper focuses on Non-PCE methods for question answering and builds off of a provided baseline BiDAF model that uses word-level embeddings. For the following discussion, let $H$ denotes the hidden size of our model.
3.1 Embedding layer

The embedding layer maps each word token into a vector representation using word-level and character-level features. Given some input word \( w_i \), the embedding layer performs embedding lookup and eventually generate word vectors with fixed dimensions. For word-level features, We lookup word embedding \( v_i \) from Glove 300-dimensional pretrained word embeddings. With a projection matrix \( W_{proj} \in \mathbb{R}^{\frac{M}{2} \times 300} \), it generates fixed-size word embedding representation \( x_{word} = W_{proj}v_i \in \mathbb{R}^{\frac{M}{2}} \) for each word.

Character-level embedding: We represent a word by its characters and generate fixed-size representation using 1D convolutional neural networks. Input character vectors are 64-dimensional learnable vectors with random initialization, and they are learned during training. Each word, padded to the maximum word length \( m_{word} \), is then represented by \( x_{emb} \in \mathbb{R}^{m_{word} \times 64} \). We then pass it to a 1-D convolutional neural network. We use \( \frac{M}{2} \) filters of various sizes, and use max-pooling with kernel of size 5 to reduce the output to a single dense fixed-size vector \( x_{char} = CNN(x_{emb}) \in \mathbb{R}^\frac{M}{2} \) for a given word.

For each word token, convolved character-level embedding \( x_{char} \) and word-level embedding \( x_{word} \) are concatenated to form vector \( x = [x_{char}, x_{word}] \in \mathbb{R}^H \). At end of embedding layer, we apply a two-layer Highway Network to refine the embedded representation.

3.2 Encoder layer

The encoder layer consists of 1 layer of bi-directional Long Short-Term Memory (LSTM) network. This layer incorporates temporal dependencies between different timesteps of the embedding layer’s output. The dimension doubles to \( 2H \) after this layer because it concatenates both forward and backward representation at a timestep. To enrich the encoder representation and encourage weight sharing, this layer of LSTM processes both context and question.

3.3 Attention Layer

The core part of our model is the attention layer implementation, which we will describe in detail here. The output of this layer gives representation of context that contains information from question. For the following discussion, let \( N \) be the length of context, and \( M \) be the length of question.

3.3.1 Bi-directional Attention Flow

This bidirectional attention flow layer enables the model to encode attention flow in both ways: from the context to the question and from the question to the context. After encoder layer, we obtain context hidden states \( \{c_1, \ldots, c_N\} \in \mathbb{R}^{N \times 2H} \) and also question hidden states \( \{q_1, \ldots, q_M\} \in \mathbb{R}^{M \times 2H} \). We compute similarity matrix \( S \in \mathbb{R}^{N \times M} \) which contains pairwise similarity value between each pair of context and question hidden states.

For each word \( i \) in context, we represent its context-to-question attention distribution as \( \alpha_{i,:} = softmax(S_{i,:}) \in \mathbb{R}^M \), which is a row-wise softmax of the similarity matrix. Then, we can compute the Context-to-Question (C2Q) attention output \( a_i = \sum_{j=1}^M \alpha_{i,j} q_j \in \mathbb{R}^{2H} \), which gives us information about attended question vectors.

Question-to-Context (Q2C) attention computes which context words are most relevant to the question. Similarly, we can compute question-to-context attention distribution as \( \beta_{,:} = softmax(S_{,:}) \in \mathbb{R}^N \), which is a column-wise softmax of the similarity matrix. Then, we compute \( N \times N \) matrix \( \beta' = \alpha \beta^T \in \mathbb{R}^{N \times N} \) to obtain the Q2C attention output \( b_i = \sum_{j=1}^N \beta'_{i,j} c_j \in \mathbb{R}^{2H} \). This computation intuitively means that we pay attention to a context word if it is relevant to some part of the question. Q2C attention gives us information about the most important context for answering the question.

At end of this layer, we have question-aware context representation \( g_i = [c_i; a_i; c_i \odot a_i; c_i \odot b_i] \in \mathbb{R}^{8H} \), where \( \odot \) denotes elementwise multiplication.

3.3.2 Multi-head Self-attention
Inspired by R-Net[4] which uses additive attention and Transformer[5] which uses dot product attention, we eventually decide to adopt the later approach to implement a simplified-version of multi-head scaled dot-product self-attention mainly because this mechanism is more computationally efficient in practice. We apply self-attention on context words to make each context hidden state attend to all other context hidden states, including itself. This enables information propagation between positions in the sentence.

**Single Attention Head:** We implement an $h = 8$ parallel attention layers or heads and for each head we want an output vector with size $d_{model} = \frac{d}{H}$. Consider a single attention head. Learnable weight matrices $W_Q \in \mathbb{R}^{2H \times d}$, $W_K \in \mathbb{R}^{2H \times d}$, $W_V \in \mathbb{R}^{2H \times d}$ are used to compute three vectors for a given input state $c_t \in \mathbb{R}^{2H}$: a query vector $q_t = W_Q^TC_t$, a key vector $k_t = W_K^TC_t$, and a value vector $v_t = W_V^TC_t$. Noted that query and key vectors have the same number of dimensions $d_q = d_k$, which by design is equal to $d_{model} = \frac{d}{H}$. The probability that word $i$ attends to word $j$ is proportional to $\exp(\frac{q_i \cdot k_j}{\sqrt{d_k}})$, where we can use a softmax function to calculate attention value. The values for all words that have been attended to are aggregated to form an average value $\bar{v}_t = \sum_j softmax(\frac{q_i \cdot k_j}{\sqrt{d_k}})v_j$, which is then projected back to desired dimension $d_{model}$ using another learnable matrix $W_O \in \mathbb{R}^{2H \times d_{model}}$. In summary, a single attention head is obtained:

$$ SingleHead(C) = [\text{softmax}(\frac{Q \cdot K}{\sqrt{d_k}})V]W_o \in \mathbb{R}^{N \times d_{model}} \quad (1) $$

where $Q = CW_Q$, $K = CW_K$, $V = CW_V$, and $C = [c_1, \ldots, c_N] \in \mathbb{R}^{N \times 2H}$ representing the context hidden states.

**Multi-head attention:** Multi-head attention allows the model to jointly attend to information from up to 8 different positions in the sentence at each attention sublayer. With a single attention head, averaging inhibits this. The output of this layer $H_{MH}$ concatenates vectors from each sublayer:

$$ H_{MH} = \text{MultiHead}(C) = [head_1; \ldots; head_8] \in \mathbb{R}^{N \times 2H} \quad (2) $$

### 3.3.3 Syntax-Guided Self-attention

As published in recent paper[3] for the first time, the innovative syntax-guided network (SG-Net) with BERT scored among top ten models on official SQuAD 2.0 leaderboard. The SG-Net paper[3] provides very little implementation details because they account that into preprocessing work for their published model. We dive deep into the paper and based on our own interpretation, we successfully generate our SG-Net adapted to our BiDAF model.

**Pretrained syntactic dependency parser:** We adopt the Glove embedding-based[1] pretrained syntactic dependency parser[2] which was made publicly available by Zhou and Zhao (2019)[6]. This high-performing syntactic parser achieves very high accuracy: 96.09% UAS and 94.68% LAS on the English dataset Penn Treebank (PTB)[7] test set. After a few days of debugging on the published code, we finally are able to recreate the dependency parser correctly. We feed Glove 100-dimensional pretrained word embedding into the parser and the parser can parse a sentence with at most 300 words.

**Sentence tokenizer:** We need to break context, which is a paragraph of words, into sentences in order to feed into the syntactic parser. One constraint we have is that our baseline model has already tokenize a context paragraph into a list of context word tokens. We have explored the method of breaking a context paragraph into sentences using sophisticated sentence tokenizer (like nltk.tokenize package), and then into word tokens, but it always leads into size mismatch as the sentence tokenizer uses different algorithms to detect the edge of a sentence and that results in slightly different inputs for word tokenizer. Eventually we use a simple method by detecting ‘.’, ‘!’, and ‘?’ to break a given list of context tokens into several sentences, which is sufficient for our purpose.

---

1. There are 3 different versions available and the SG-Net paper uses BERT-based parser, which is different from our choice due to Non-PCE constraint.
Syntactic dependency of interest (SDOI) mask: As published in the SG-Net paper [3], SDOI is a mask capturing the syntactic dependency information for a given sentence. SDOI mask is created by first using a syntactic dependency parser to produce the related nodes for each word in a sentence, and then generating a mask consists all its ancestor nodes and itself in the dependency parsing tree. Specifically, given input token for a sentence \([w_1, \ldots, w_n]\) where \(n\) denotes the sentence length, we first use syntactic parser to generate a dependency tree. Afterwards, we build up a SDOI mask \(M_s \in \mathbb{R}^{n \times n}\) where each row \(i \in \{1, \ldots, n\}\) denotes the dependency mask of all the words to the word \(w_i\). For word \(w_i\) and any word \(w_j\), \(M_s[i, j] = 1\) if \(w_j\) is an ancestor node of token \(w_i\); otherwise \(M_s[i, j] = 0\). As the example shown in Figure 1, the ancestors for token \(w_3\) : lower are \(w_5 : losses, w_2 : reflects\), and itself \(w_3\). Therefore, \(M_s[3, (2, 3, 5)] = 1\) and \(M_s[3, (0, 1, 4)] = 0\).

We piece the masks for difference sentences in the context together and pad empty spaces with 0 to produce a context SDOI mask \(M \in \mathbb{R}^{N \times N}\) for context of length \(N\).

Syntax-guided self-attention using SDOI: This implementation is the same to that of the multi-head self-attention, except the difference of using additional SDOI mask. We apply SDOI mask to each of the 8 parallel attention sublayers. Referring to equation (1) in a single attention head, to produce self-attention with syntactic constraint, we perform a dot product to score key-query pairs with the SDOI mask \(M\) to obtain attention weights \(A\):

\[
A = \text{softmax} \left( \frac{M \cdot (Q \cdot K)}{\sqrt{d_k}} \right) \in \mathbb{R}^{N \times N}
\]

Then carried on the rest of calculation in the same way to get the single head attention representation as \(\text{SingleSGHead}(C) = (AV)W_o \in \mathbb{R}^{N \times d_{model}}\), where the notations are the same as that in section 3.3.2. Then, we concatenate the results from 8 sublayers to obtain our syntax-guided self-attention representation \(H_{SG} \in \mathbb{R}^{N \times 2H}\).

Dual Context Aggregation for SG-Net: The final output of our SG-Net is a linear combination of a regular multi-head self-attention \(H_{MH}\) and a syntax-guided self-attention \(H_{SG}\):

\[
H_{SG} = \alpha H_{MH} + (1 - \alpha) H_{SG}
\]

where \(\alpha\) is a hyperparameter and \(0 \leq \alpha \leq 1\). We concatenate our SG-Net attention with the BiDAF attention which is then feeded into the output layer.

3.4 Output Layer

The output layer produces a vector of probabilities corresponding to each position in the context. It uses a bi-directional LSTM network to generate the starting position and the ending position representing the continuous span of words in context as our answer.

4 Experiments

4.1 Data

The dataset used in this project is SQuAD 2.0. This dataset combines the 100,000 questions in SQuAD1.1 [8] with over 50,000 unanswerable questions written adversarially by crowdworkers to
look similar to answerable ones. To do well on SQuAD 2.0, models need to first determine if there is an answer exists for a given question, and then provide an answer when possible. In this project, our training dataset (129,941 examples) is taken from official SQuAD 2.0 training set, and our dev (6078 examples) and test set (5915 examples) are each half of the official dev set.

4.2 Evaluation method

We are currently using the quantitative metrics proposed in the original SQuAD challenge: Exact Match Score and F1 Score. An overview of each metric is defined below:

- **Exact Match Score (EM)**: A binary measure on whether or not the model outputs exactly what is stated in ground truth.
- **F1 Score**: The harmonic mean of precision and recall: \( F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \)

Moreover, we can qualitatively analyze our results by inspecting the text output our model predicts from a given paragraph and a question about that paragraph.

4.3 Experimental details

We trained each model with 25-30 epochs (3-3.5M steps) for about 12 - 16 hours on Azure NV6 machine with batch size of 64, or 20-40 hours if we use smaller batch sizes. We used dropout rate of 0.2 for our character-embedding layer, encoder layer, BiDAF attention layer, and output layer. We used Adadelta optimizer with initial learning rate of 0.5. We attempted to tune the learning rate to a higher value (0.7) and smaller value (0.45) and we didn’t observe performance improvement in either direction. The hidden state size for all our models is 100. Unfortunately, our proposed finetuning on number of layers of LSTM networks in encoder layer ends up performing much worse than the initial setup of 1 layer. That means extra complexities added in encoder layer do not help the model better process the hidden states.

4.4 Results

We summarized our best performing models with different structures in the following table, where we reported results for our best single models and ensemble models:

<table>
<thead>
<tr>
<th>Model</th>
<th>On Dev Set</th>
<th>On Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EM</td>
<td>F1</td>
</tr>
<tr>
<td>Baseline BiDAF</td>
<td>57.08</td>
<td>60.30</td>
</tr>
<tr>
<td>Baseline + Char Embed</td>
<td>59.13</td>
<td>62.60</td>
</tr>
<tr>
<td>Baseline + Char Embed + Self-attn</td>
<td>61.03</td>
<td>63.99</td>
</tr>
<tr>
<td>Baseline + Char Embed + SG-Net</td>
<td>59.99</td>
<td>63.53</td>
</tr>
<tr>
<td>3-model Ensemble</td>
<td>64.42</td>
<td>66.46</td>
</tr>
<tr>
<td>4-model Ensemble</td>
<td>63.18</td>
<td>65.23</td>
</tr>
<tr>
<td>Non-PCE Test LeaderBoard submission</td>
<td>62.07</td>
<td>64.59</td>
</tr>
</tbody>
</table>

Table 1: Results on the SQuAD 2.0 dev and test set

**Character-level embedding** The extra character-level embedding implementation alone has boosted both EM and F1 performance by 3%. This is mainly because character-level embedding is much better at handling out-of-context words compared to word-level embedding only. One

---

3We choose the best multi-head model, the best SG-Net model, and the best character-embedding model to form the 3-model ensemble. We deploy a distributional ensemble strategy when combining different models.

4We choose the best multi-head model, the best 2 SG-Net model, and the best character-embedding model to form the 4-model ensemble. We deploy a distributional ensemble strategy when combining different models.
reason is that tokenized words in the training data are very noisy as many annotations in the context don’t have space separated themselves from the previous word and word tokenizer provided by the default implementation wrongly classify that concatenated group of words as one word. That results in many "unknown" words in model with word-embedding only. With the help of character-level embedding, we are better able to decipher these "unknown" words and thus facilitating the training performance.

**Multi-head self-attention** Additional multi-head self-attention alone has further boosted our model on top of the character-level embedding by another 2% on top of character-embedding BiDAF.

**SG-Net** We explored contribution of each potential component through different experiments.

<table>
<thead>
<tr>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char-embed BiDAF Baseline</td>
<td>59.13</td>
<td>62.60</td>
</tr>
<tr>
<td>+ Multi-head self-attention only</td>
<td>61.03</td>
<td>63.99</td>
</tr>
<tr>
<td>+ SG-attention only</td>
<td>59.13</td>
<td>62.62</td>
</tr>
<tr>
<td>+ Dual contextual attention</td>
<td>59.99</td>
<td>63.53</td>
</tr>
<tr>
<td>+ Concat dual attention</td>
<td>59.63</td>
<td>63.29</td>
</tr>
</tbody>
</table>

Table 2: Ablation study on potential components and aggregation methods on SQuAD 2.0 dev set

We observe that concatenating dual attention (multi-head self-attention and SG-attention) is performing slightly worse than using dual contextual attention method described in the paper [3], which agrees with what the paper has reported. Also, we see that with extra SG-attention only, there is almost no gain in performance comparing to our character-embedding BiDAF baseline. That is unexpected to us. We suggest that’s because BiDAF attention has already captured some of the syntactic information through training and hence rendering our extra syntactic self-attention less useful in the model. In addition, out baseline model is different from the baseline BERT model used in the paper. This difference in performance also warns us that naively transferring features from BERT model to non-BERT model might not work in the same way.

**Ensemble** We combine 3-4 different neural architecture outputs and these outputs may reflect different internal distributional representations, because we believe that the distributional ensemble strategy would outperform the majority vote and it indeed improves our Dev performance by more than 10% comparing to our best-performing single model.

## 5 Analysis

We first explored the EM and F1 curves on tensorboard and based on that, we concluded that our Multi-head self attention model is the best among baseline, Multi-head self attention and SG-Net. Here the orange curve represents baseline, the blue curve represents Multi-head self attention, and the gray curve represents SG-Net.

---

5 There are various forms of annotations in the context data, for example “[citation needed]”, “[note]”, “[web]”, and so on. Most of them begin with a square bracket and they don’t have space in front of them.
We then analyzed the composition of question types in training, dev and test data set. We categorized question types under the category of “what”, “when”, “which”, “how”, “why”, “who” and “where”. How we categorize each question is based on which keyword mentioned above exists in the question. As we can see from the following bar plots, the “what” type of questions is the most in our data set.

This motivates us to explore how our models perform how different types of question, and thus, we compare baseline, baseline + multi-head self attention and baseline + SG-Net. The plots are below:
As you can see, the scores achieved across all models quite spread out across various types of questions. This may explain why we couldn’t achieve much boost in performance since neither of the models could stand out from the “what” type of question which composes the most in the data set. In addition, although Multi-head self attention outperforms the other 2 models on dev evaluation, there are some types of questions where baseline and SG-Net performs better than Multi-head self attention. For example, for “which” type of questions, baseline outperforms both Multi-head self attention and SG-Net where SG-Net performs the worst. On the other hand, in “other” type of questions, SG-Net outperforms both baseline and Multi-head self attention where baseline performs the worst. We also plotted the actual scores Multi-head self attention achieves on different types of questions, and it seems “which” and “when” types of questions are Multi-head self attention most good at.

6 Conclusion

From this project, we find that both character embedding and self attention helps boost the performance of the model on top of BiDAF attention, and multi-head self attention allows the self attention to achieve in multiple dimensions. Although we successfully replicated SG-Net attention, we don’t find it improve our model performance. We learnt and achieved to implement character embedding, self attention and on top of that multi-head self attention, as well as SG-Net attention. However, we couldn’t figure out why SG-Net couldn’t boost our model performance, and it could lead to our future work.
References


