Megatron: Using Self-Attended Residual Bi-Directional Attention Flow (Res-BiDAF) to Improve Quality and Robustness of a BiDAF-based Question-Answering System

Matt Linker
Department of Computer Science
Stanford University
mslinker@stanford.edu

Zhilin Jiang
Department of Computer Science
Stanford University
zjiang23@stanford.edu

Zheng Nie
Department of Computer Science
Stanford University
zhengnie@stanford.edu

Abstract

Question Answering (QA) represents one of the fundamental problems in Natural Language Processing (NLP). In QA tasks, a model is given roughly a paragraph of text (known as context) and is then asked a question about the text, with successful models providing an appropriate answer, as predetermined by a human worker. A system that could compete with human performance on this task would represent an enormous leap forward in the field of artificial intelligence. One of the most well-known and popular datasets for benchmarking of this task is the Stanford Question Answering Dataset (SQuAD). Two key and related problems underlie existing approaches – the first being that existing systems will occasionally focus on irrelevant portions of the context passage, and the second being that not all questions have a valid answer given some context. We aim to improve upon a vanilla Bidirectional Attention Flow (BiDAF) based approach under both Exact Match (EM) and F1 scores on the SQuAD 2.0 dataset, by contributing a model that is focused on the questions of determining whether a relevant answer can be found in the context, and if so where it can be found. We present Megatron, which harnesses self-attended residual BiDAF to improve performance and reliability over the baseline model. Additionally, we present an index of negative results, particularly with term frequency-inverse document frequency weighting filters, found during our investigations in an effort to hopefully assist future researchers. Our findings show significant improvement over traditional BiDAF models and hopefully will prove of use in the goal of advancing the field of NLP.

1 Introduction

Question-answering (QA) has been an important task and popular research topic of natural language processing (NLP). There are various existing QA datasets available, among which the Stanford Question Answering Dataset (SQuAD) is an state-of-the-art reading comprehension dataset for training and evaluating QA systems [Rajpurkar et al., 2018].

In this project, we explore building and improving a system for the QA tasks defined by SQuAD 2.0, in order to contribute to this popular research topic as well as gaining practical experiences and better understanding of applying neural models to NLP tasks in general.

2 Related Work

2.1 Transformer

In preparing to work on our own model, we read and investigated the previous work of Vaswani et al. [2017] on Transformer. The idea here is essentially to build upon a standard Transformer-based approach by introducing recurrence into the self-attention network, reusing previous hidden states from previous segments to build up a “memory” over time. Doing so in theory should be substantially faster than “vanilla” Transformer, as we are not starting a new attention calculation “from scratch” each time. Over time, attention length is iteratively increased until no (or very little) incremental improvement is seen. More concretely, the approach is to re-parameterize to remove absolute position information from our data, replacing it with a trainable parameter, and seeing how that updates our attention distribution. Result-wise, Vaswani et al. [2017] found that this approach significantly lowered perplexity compared to previous Transformer-based models and works much faster comparatively, especially with longer attention parameters. This is particularly promising to us, as considering several sentences (or even paragraphs) in context may yield additional information not found in shorter contexts. For us, it could be useful to inspect whether this perplexity reduction aspect has an impact on model performance on our dataset (which is comprised of relatively-short passages).

2.2 TF-IDF

Another source of inspiration for us was that of Chen et al. [2017] on DrQA, a robust question-answering (QA) system building upon a vanilla RNN. In essence, DrQA seeks to mitigate the problem of determining the existence of, locating, and identifying relevant passages of text, which can then be fed into a more traditional RNN-based QA system. Such an idea was inspired by the immense breadth of Wikipedia – it would be far too computationally-intense to run an RNN over the whole of the Wikipedia corpus for a single query – some efficient means is needed to find the most relevant documents or paragraphs such that attention (figuratively, not neural-network attention) can be focused there by the actual RNN.

The implementation of this system is fairly straightforward. Essentially, it utilized an inverted index (an index that maps words or other similar features to documents/passages in which they are found) with TF-IDF weighting in order to identify and extract the most relevant contexts. TF-IDF weighting is a deterministic scoring system for relevance of a given search query – essentially, it computes a weighted similarity score between queries and documents, with higher weightings given to less common words found in both (e.g. amygdala), and lower weighting to more common words (e.g. the). From there, the authors compared their results to those of actual Wikipedia search, and found that they were able to obtain more relevant results in comparison – indicative of a reasonably-good localizing system for finding relevant documents.

From there, their system largely looks like that given in the baseline BiDAF model (described in further detail below), with the exception that the top 5 candidate documents/passages are fed in to the RNN, instead of just a single passage as in the baseline. The model then sees if a suitable answer is found, and if so, returns that answer. Our interest in [Chen et al., 2017]'s work, however, comes from a slightly different angle. In the SQuAD 2.0 dataset, we need not search a context space for an answer - instead, we only need to determine whether a given context has the answer to our question. Thus, instead of scoring, ranking, and retrieving some set of documents, we were inspired to use TF-IDF instead to help us identify and correctly answer "no-answer" results when appropriate.

3 Approach

3.1 Baseline Model and Observations

The foundation of our work is built upon a BiDirectional Attention Flow implementation in PyTorch [Seo et al., 2016]. From our initial evaluations of this model, we noted that two key challenges arose that hurt model performance.

First, in SQuAD 2.0 [Rajpurkar et al., 2016], approximately 33% of questions were adversarially generated by humans to appear similar to answerable questions for a given context, but in fact did not have an answer given that context. Under the BiDAF default model we begin with, rudimentary sanity checks are performed using existing techniques [Levy et al., 2017]. Specifically, an arbitrary out-of-vocabulary token v is prepended to the beginning of each sequence, and we calculate softmax over the full sequence, with some p_{start} and p_{end} representing the softmax probability that the answer starts or ends with a given word. If $p_{start}[0]p_{end}[0] > p_{start}[x_i]p_{end}[x_j]$ for all valid sequences of words in x, then the model predicts no answer. This allows the model to predict that there is no answer to the question in a given context, though since spurious low probabilities on start and end tokens could still be higher than those of entirely out-of-vocabulary words, under this model we only see no answer if the model is highly confident that the question cannot be answered from the context. Thus, one of our primary motivations is to increase reliability of the model by having it not answer invalid questions, while continuing to perform well on valid questions.

Second, in all versions of SQuAD, the baseline model at times returns irrelevant information. This problem is related to the first one, as this will also sometimes occur when the correct answer would be "N/A," however the problem extends to scenarios in which there is in fact a valid answer embedded in the context. In these scenarios, for one reason or another the BiDAF model attends to the wrong sub-passage of the context, and thus returns nonsensical or irrelevant answers instead of the correct answer. Our other guiding motivation, therefore, is to increase the performance of the model by improving the attention mechanism to better focus answers on the true answer segments.

3.2 Our Experiment Models

Throughout the project, we experimented with many different modifications on top of the baseline model, including:

- Transformer Model (Single & Double Transformer Models) **(Section 3.2.1)**
- BiDAF, with all LSTMs replaced by Transformers
- BiDAF, with bi-directional attention flow replaced by bi-linear attention
- Modified baseline, with embedding size increased from `hidden_size` to `word_emb_size`
- Cross-feeding transformer (Terminated due to high computation cost)
- Highway transformer
- Character-level Embedding **(Section 3.2.2)**
- Self-Attended Residual BiDAF (2 layers & 4 layers) **(Section 3.2.3)**
- Post-prediction filtering to mark no-answers using TF-IDF **(Section 3.2.4)**

In the sub-sections below, we highlight 3 different experiments that yielded significant improvements over the original baseline. Some of the experiments did not perform well and were terminated early due to observed mediocre performance.

3.2.1 Single Transformer Model

We experiment incorporating Transformer [Vaswani et al., 2017], the attention-based technique previously introduced, to improve our model performance. As our first attempt, we replaced phase word-embedding layer with Transformers.

3.2.2 Character-level Embedding

Inspired by Seo et al. [2016], we introduced subword (character-level) embedding to our model. For each word, we concatenated together word-embedding and Convolutional Neural Network (CNN)-

based char-embedding, and use the output as the final embedding being passed in to the rest of our model.

3.2.3 Self-Attended Residual BiDAF (Res-BiDAF)

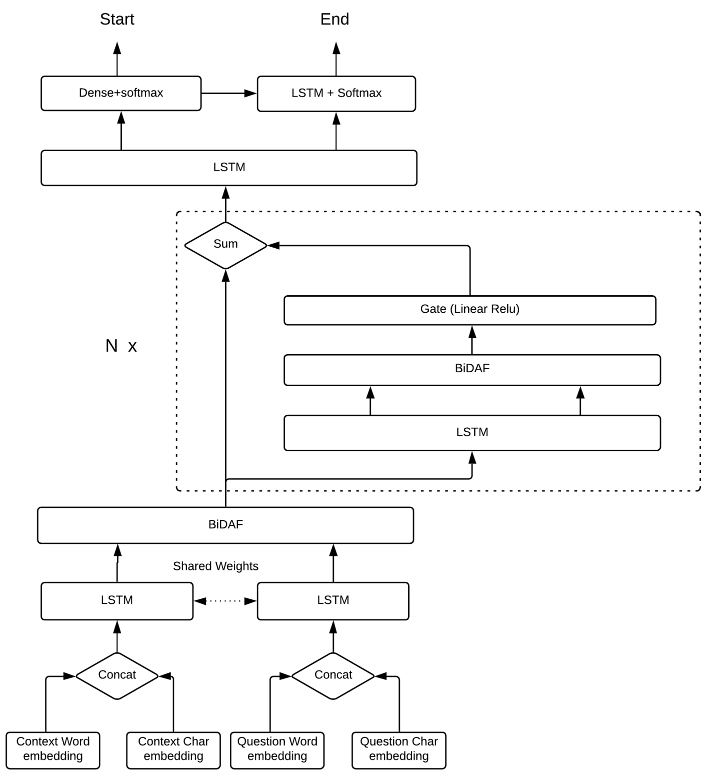

Figure 1: Architecture of our Residual BiDAF model.

In addition to previously described modifications to the baseline model, we propose a new architecture named "Self-Attended Residual BiDAF" (also known as Res-BiDAF). This architecture makes use of character-level embedding described in Section 3.2.2, and takes inspiration from ResNet [He et al., 2016] and Highway Network [Srivastava et al., 2015]. Figure 1 provides a good illustration of the model architecture. The context and the question are each taken in as word embedding and character embedding concatenated together. After passing through the shared-weight LSTMs, and combined through the initial BiDAF, the output is passed to a series of "Gated Residual Blocks", each consisting of an LSTM layer, a BiDAF, and a linear ReLU gate. The output is summed together with a pass-through connection, and the sum will become the input of the next residual block (if any). The part of the model after the last residual block output is identical with the baseline.

3.2.4 TF-IDF

TF-IDF is a numerical value to represent the relevance of a given document to a given word, by computing the relative rarity of a word in the corpus compared to whether it appears in the target document. Of course, this takes on even greater meaning if we search using full query strings instead of single words, as the frequency element will significantly influence our score just from that word alone. Mathematically, given some term t, we denote the number of times it occurs in document d as f. We then define the term-frequency score to be $tf(t, d) = 1 + log(f)$. The inverse-document score is then given by $idf(t) = log(1 + \frac{N}{n_t})$, where N is the total number of documents and n_t is the number of documents containing term t. Finally, we compute $tfidf(t, d) = tf(t, d)idf(t)$. Our implementation was heavily modified and improved from basic TFIDF code we wrote ourselves as part of prior coursework[1]. For our purposes, we have implemented a TF-IDF system that computes idf scores over the training dataset, then compute a length-normalized (average) TF-IDF score between a given context sentence and a provided "query" string, either representing the query itself or our candidate answer. To discount common words like "the," we compute this score over only values with a specific rarity, by default words appearing in less than half of contexts. From here, we tested various "filter" thresholds as hyperparameters, where the filter setting was the minimum average TF-IDF score we'd accept, ignoring common words, over either the query or the candidate response. The intention here was to remove those queries or answers that were asking non-specific and irrelevant questions, as we found that this was a common failure mode of the baseline model. TF-IDF is not intended to run as a standalone model - rather, it should be utilized as a pre- or post-processing filter on some existing QA system.

3.3 Implementation

In our experiment, we reused the baseline starter code. For character-level embedding implementation, we reused our solutions to the programming tasks in CS224N Assignment 5, whose core methods we implemented ourselves. We had a basic working TF-IDF implementation that we previously implemented for CS124, though significantly modified and improved it for usability and scalability purposes.

We implemented the remaining portion of our modifications on top of the starter code, including complete implementations of transformer (encoder and decoder) and other model variants, on our own.

Our code is made available at: `https://github.com/alongstar518/CS224N2019Final`.

4 Experiments

4.1 Data

At the current step, we train our models using the standard SQuAD 2.0 training set, and evaluate the models using the default project SQuAD 2.0 dev set, due to the fact that most models are still experimental and need further polishing. Once we lock on several good-performing model architectures, we will evaluate them using the test set.

4.2 Evaluation Methods

We use three metrics for evaluation: F1 score, Exact-Match score (EM score), and AvNA (Answer versus No-Answer) to evaluate the implemented model. To calculate F1 score, we compute the precision (P) and recall (R) of our predictions:

$$P = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \quad R = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

And the F1 score is defined as:

$$\text{F1} = 2\frac{PR}{P+R}$$

To calculate the Exact-Match score (EM score), we compute the percentage of exactly matched answers among all evaluated questions:

$$\text{EM} = \frac{\text{exactly matching answers}}{\text{total evaluated questions}} \times 100$$

4.3 Experiment Details

The baseline model took 2.3M steps (17h 1min on an Azure NV6 virtual machine) to reach the peak performance values.

The following 4 experiments yielded significant improvements over the baseline model:

[1]starter code at `https://cs124.stanford.edu`

4.3.1 Single Transformer Model

We trained the model with the following tuned hyperparameters:
Transformer $k = 64$, $q = 64$, $v = 64$; Number of transformer heads = 8; Number of layers = 6; Learning rate = 0.5 (flat rate); Dropout rate = 0.2; Batch size = 64.

The model took 1.25M steps (8h 1min on an Azure NV6 virtual machine) to reach the peak performance values.

4.3.2 Character-level Embedding

We trained the model with the following tuned hyperparameters:
Learning rate = 0.5 (flat rate); Dropout rate = 0.2; Batch size = 64; Character embedding size = 100; Character embedding CNN kernel size = 5.

The model took 1.4M steps (5h 48min on an Azure NV6 virtual machine) to reach the peak performance values.

4.3.3 Self-Attended Residual BiDAF, Single Gated Residual Block

We trained the model with the following tuned hyperparameters:
Learning rate = 0.5 (flat rate); Dropout rate = 0.2; Batch size = 64; Character embedding size = 100; Character embedding CNN kernel size = 5; Number of residual blocks = 1.

The model took 1.5M steps (9h 9min on an Azure NV6 virtual machine) to reach the peak performance values.

4.3.4 Self-Attended Residual BiDAF, 2 Gated Residual Blocks

We trained the model with the following tuned hyperparameters:
Learning rate = 0.5 (flat rate); Dropout rate = 0.2; Batch size = 64; Character embedding size = 100; Character embedding CNN kernel size = 5; Number of residual blocks = 2.

The model took 1.7M steps (13h 40min on an Azure NV6 virtual machine) to reach the peak performance values.

4.3.5 TF-IDF

We tried applying TF-IDF score-based filtering to the model predictions, manually removing predictions that are and outputting N/A (empty string) instead. We calculate the additive IDF score of the predictions (processed to be punctuation-stripped, case-insensitive), normalized to the sentence length, and we completely ignore common words (defined by having a word IDF score lower than 1.0). We filter our all predictions of additive IDF score lower than 2.0 using this metric.

Model	F1 Score	EM Score	AvNA Score
Baseline (Trained), no TF-IDF	61.13	57.97	67.84
Baseline (Trained), with TF-IDF	60.06	57.57	65.74
Char-Embedding, no TF-IDF	64.36	60.98	70.81
Char-Embedding, with TF-IDF	62.86	60.12	68.53
Res-BiDAF (2 blocks), no TF-IDF	67.69	64.48	73.40
Res-BiDAF (2 blocks), with TF-IDF	65.90	63.35	70.81

Table 1: Post-prediction TF-IDF filter experiment results

Table 1 shows the list of post-prediction TF-IDF filtering experiments. None of the experiment produced positive results, so we eventually decided to not pursue improving our models with this method. We believe a possible reason that this method did not work is that the trick questions in SQuAD 2.0 dataset actually include key words that are infrequent and important from the context, but are not applicable to the actual context due to more intricate meaning differences that TF-IDF cannot capture simply by counting word frequency.

4.4 Results

4.4.1 Training & Dev Set Evaluation Results

We trained each of our models over the SQuAD 2.0 training set, until the metrics start to decrease and the loss start to increase, incidating overfitting. At the end of each training, we record down the best evaluation performance on the dev set throughout the training process.

The evaluation results are as shown in Table 2.

Model	F1 Score	EM Score
Baseline (Given)	58.98	55.81
Single Transformer	59.70	57.18
Char-Embedding	64.36	60.98
Res-BiDAF (single block)	65.40	62.06
Res-BiDAF (2 blocks)	67.69	64.48

Table 2: Best Dev set evaluation performance of each model

It is apparent that each of our newer experiment obtains improved EM and F1 scores over the previous model, and the final Res-BiDAF model with 2 residual blocks reached a 8.71 increase in F1 score and 8.67 increase in EM score over the baseline evaluation results we are given.

The dev set evaluation score curves (monitored from TensorBoard) are shown in Figure 2. Note that the baseline we trained reached slightly higher EM and F1 scores than the baseline evaluation results we were provided with, due to extended training time.

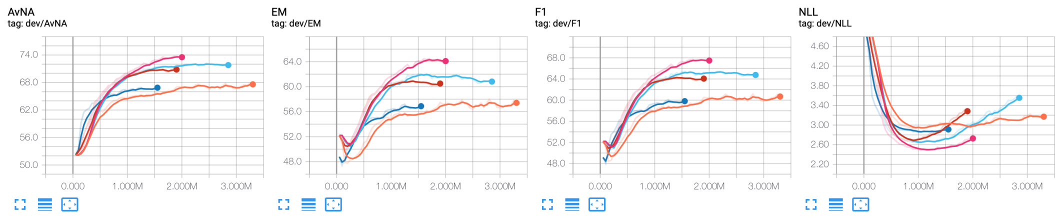

Figure 2: AvNA, EM, F1, and Loss plots for the Baseline (orange), Single Transformer (dark blue), Char-Embedding (red), single-block Res-BiDAF (baby blue), and 2-block Res-BiDAF (magenta).

4.5 Dev Non-PCE Leaderboard Results

We submitted our "Res-BiDAF (2 blocks)" model to the Dev non-PCE leaderboard, achieving an EM score of 64.443 and a F1 score of 67.449.

4.5.1 Test Non-PCE Leaderboard Results

We made a total of 2 submissions to the "TEST NON-PCE SQuAD Leaderboard":

- Res-BiDAF (single block), EM: 58.698, F1: 62.333
- Res-BiDAF (2 blocks), EM: 62.992, F1: 66.370

5 Analysis

In this section, we examine several question-answer examples to better understand performance differences among the models. Note that the dataset gets shuffled differently among models, and only a subset of questions were sampled and retained during evaluation, so we only compare among pairs of models.

5.1 Example: Predictions across models

- **Question:** Economy, Energy and Tourism is one of the what?

- **Context:** Subject Committees are established at the beginning of each parliamentary session, and again the members on each committee reflect the balance of parties across Parliament. Typically each committee corresponds with one (or more) of the departments (or ministries) of the Scottish Government. The current Subject Committees in the fourth Session are: Economy, Energy and Tourism; Education and Culture; Health and Sport; Justice; Local Government and Regeneration; Rural Affairs, Climate Change and Environment; Welfare Reform; and Infrastructure and Capital Investment.
- **Correct Answer:** current Subject Committees
- **Prediction (Baseline):** N/A
- **Prediction (Single Transformer):** The current Subject Committees
- **Prediction (Char-Embedding):** current Subject Committees
- **Prediction (Res-BiDAF, single block):** Subject Committees
- **Prediction (Res-BiDAF, 2 blocks):** current Subject Committees in the fourth Session

Here, the baseline failed to produce an answer to the question, while the Single Transformer model, the Char-Embedding model, and the single-block Res-BiDAF model all produced satisfactory answers to the question. What's the most fascinating is the prediction of our 2-blocks Res-BiDAF model - it is able to produce an answer that's even more accurate and detailed than the expected answer key!

5.2 Example: Proper noun, word-based vs. sub-word embedding

- **Question:** Against whom did the Camisards rise up to fight?
- **Context:** After this, Huguenots (with estimates ranging from 200,000 to 1,000,000) fled to surrounding Protestant countries: England, the Netherlands, Switzerland, Norway, Denmark, and Prussia — whose Calvinist Great Elector Frederick William welcomed them to help rebuild his war-ravaged and underpopulated country. Following this exodus, Huguenots remained in large numbers in only one region of France: the rugged Cévennes region in the south. In the early 18th century, a regional group known as the Camisards who were Huguenots rioted against the Catholic Church in the region, burning churches and killing clergy. It took French troops years to hunt down and destroy all the bands of Camisards, between 1702 and 1709.
- **Correct Answer:** the Catholic Church in the region
- **Prediction (Single Transformer):** Frederick William
- **Prediction (Char-Embedding):** Catholic Church

Here we see the Character-Embedding model making the correct prediction, while the Single Transformer model. We believe there are two possible reasons leading to this performance difference: first, character embedding is able to gain better performance especially on proper nouns due to their sub-word flexibility; second, character embedding allows the model to learn the embedding itself as a part of the training process, instead of taking fixed word embedding values as input like other models do.

6 Conclusion

In this project, we experimented with various modifications to the given BiDAF baseline model, such as Transformers and character-level embedding. We also proposed a new architecture named "Self-Attended Residual BiDAF" (Res-BiDAF), and successfully reached significant performance improvements, over any other model we experimented with. However, given the limitation of non-PCE models, our model cannot achieve the level of performance matching state-of-the-art PCE-based models. If given more time and computing resources, we would extend our project by trying our Res-BiDAF with a higher number of residual blocks, as well as its combination with various other non-PCE techniques. We also believe it will be an exciting research topic to apply our Res-BiDAF to PCE-based models for potential performance improvements.

References

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*, 2017.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.