A Transformer Augmentation of BiDAF

Steven Zheng
Department of Computer Science
Stanford University
Stanford, CA 94305
`szzheng@cs.stanford.edu`

Abstract

Question-Answering (QA) describes a topic in natural language processing in which the goal is to build a machine model that best answers questions posed in natural human language. Effective methods in QA often involve recurrent networks, convolution, or self-attention networks. In this paper, we construct a model using all three methods. We achieve an F1 score of 64.246 and an EM score of 60.558 on the SQuAD 2.0 dataset.

1 Introduction

Question-Answering (QA) is a problem in which machine models attempt to provide answers to human-posed questions. Two of the most important motivations for studying question-answering are its relations to machine understanding and to information retrieval, which we can see in the following characterization of the QA problem: given a context passage and a query, the goal is to *select* a span of the context that answers the query (as opposed to actively *generating* answers) or conclude that such a span does not exist. Success in such a system requires a machine's ability to comprehend text and filter out important / target details at a level that is basic to humans, and the mastery of which is fundamental to deeper levels of natural language processing as a whole. This specific type of QA comes from the SQuAD 2.0 dataset [1], which contains both span-answerable and non-answerable question-context pairs.

Recurrent models (e.g. BiDAF [2]) are widely considered as leaders in machine comprehension and interpretation of human language, owing to a natural fit towards sequential inputs. More recently, non-recurrent transformer-based models have been shown to be as successful in some applications in QA. These non-recurrent models employ mechanisms such as convolution, self-attention, or both (e.g. QANet [3]), to adequately capture relational dependencies in sequential input. Based on the success of these existing approaches, we attempt a hybrid architecture for QA that leverages all of these mechanisms including recurrent networks. We attempt to show the addition and diversification of relational information will lead to stronger model representation, comprehension, and interpretation levels. Towards that end, we start with a BiDAF-based model and attempt to add elements of QANet, while maintaining core elements such as bidirectional LSTM layers and bidirectional attention flow. Therefore, the hybrid model can be considered an augmentation to the BiDAF model, rather than a pure modification of it.

2 Approach

2.1 Baseline (Provided)

The provided baseline as part of the default final project is a BiDAF-like model that conspicuously lacks the use of character-level embeddings. This basic architecture has 5 central components: (1) an input embedding layer, (2) an embedding encoder layer, (3) a context-query attention layer, (4) a model encoder layer, and (5) an output layer. This is a fairly standard high-level template that we will follow in our proposed architecture. For details regarding the baseline architecture, refer to the default final project handout[1]. Additionally, the started code containing the basic BiDAF implementation is provided along with the handout[2].

2.2 Related Works

While the architecture we will propose uses influences from multiple papers and multiple influences within a paper, we will focus this section on just the changes to the baseline.

Character-level embeddings
Main article: Bidirectional Attention Flow for Machine Comprehension [2]

Building off the baseline model and architecture, the first modification to attempt would be to augment the input embedding for each word with a character-level component. This is a standard approach in many well-performing model, and one key difference between the BiDAF-like baseline and the actual BiDAF model. Character-level vectors provide additional information to word representation, and can be particularly helpful for words that lie outside the given vocabulary.

Non-recurrent encoders
Main article: QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension [3]

The current baseline uses solely recurrent networks to encode relational dependencies in the embedding layer and attention layer outputs. An alternative and successful approach, used in QANet, is to use a combination of convolution (local dependencies) and self-attention (global dependencies) layers to achieve the desired encoded features. This approach is an enhancement of the transformer architecture [4], each of which show the value of convolution or self-attention, respectively. For our purposes, it would be interesting to use a combination all these types, since they all capture inter-word relations in different ways. This change would take place in the embedding encoder and model encoder layers. Note, the original QANet approach was designed to be a replacement for the recurrent architecture, though we intentionally keep the original recurrent networks for experimentation.

2.3 Model

The model stars with a context sequence of words $C = \{c_1, ..., c_n\}$ and query sequence of words $Q = \{q_1, ..., q_m\}$. It returns either a continuous sub-sequence of words (span) $S = \{c_i, ..., c_{i+j}\}$ that best answers the query or the null span S_0 denoting a "no-answer".

Input Embedding Layer
The input embedding layer builds upon the infrastructure of the baseline embedding layer by including character-level information in a word's final representation. For each word x in the context and query, let $x_w \in \mathbb{R}^{n_1}$ denote its word-level embedding and let $x_c \in \mathbb{R}^{n_2}$ denote its character-level embedding. For our purposes, we use pre-trained vectors of size n_1. x_c is a fixed-length vector of size n_2, generated through the 1D-convolution and max-pooling of its constituent character vectors, which are also of size n_2 and are trainable. We aggregate these base representations, $x_{wc} = [x_w; x_c]$, and pass the result through a two-layer highway network [5], before finally linearly projecting the output of that to h-dimensional space.

$$x_{wc} = [x_{i_w}; x_c] \in \mathbb{R}^{n_1+n_2}$$
$$x_{hwy} = \text{Highway}(x_{wc}) \in \mathbb{R}^{n_1+n_2}$$
$$x_{emb} = [W_1(x_{hwy}); ...; W_h(x_{hwy})] \in \mathbb{R}^h$$

[1]http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf
[2]https://github.com/chrischute/squad

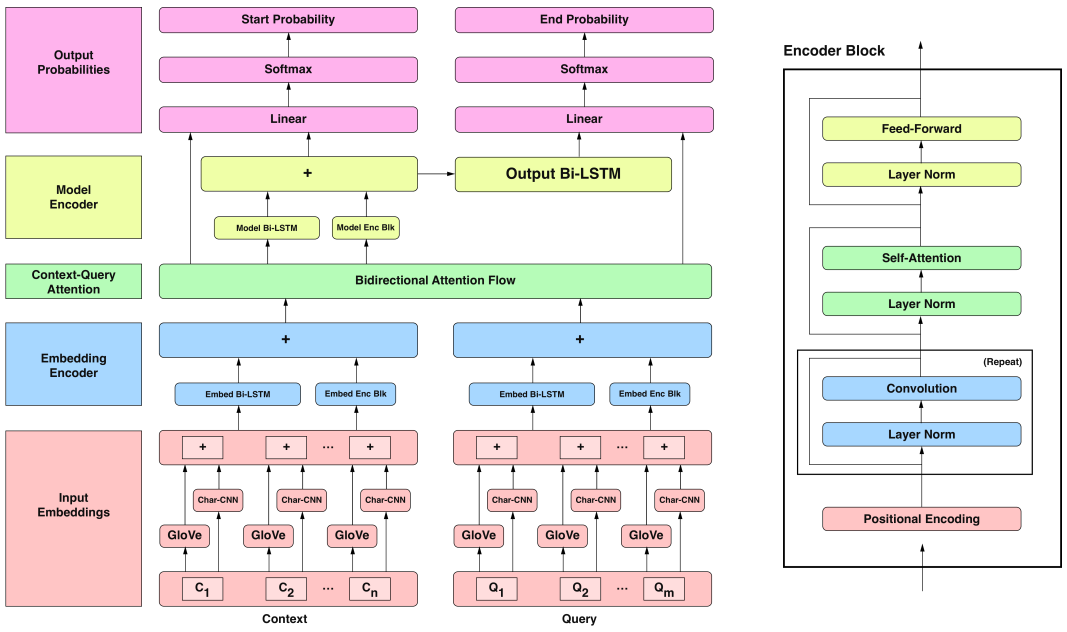

Figure 1: The augmented BiDAF architecture. Note the addition of encoder blocks at the embedding and model encoder layers.

Note: The original baseline applies a projection to the base embedding before applying a highway network on top of that. Here, we apply the highway network first and then project the result to the hidden size.

Embedding Encoder Layer
This layer takes influence from both the recurrent BiDAF architecture and the transformer-based QANet architecture. Specifically, the input to this layer, the embedded context or query X, is simultaneously but separately passed to a 1-layer bidirectional LSTM network [6] and a 1-block QANet-encoder. The recurrent network generates an encoding of each position of the context and query, of dimensionality $2h$ because of the bidirectionality. The QANet-encoder generates an encoding of dimensionality h using the following architecture. Firstly, the QANet-encoder consists of a sequential stack of encoder blocks. Each encoder block applies a positional encoding to its own input and passes the result through a series of depthwise-separable convolution layers [7], then a multihead self-attention layer, and finally a feed-forward layer. A layer normalization is applied right before each of those layers, and output of a layer is summed with the input to that layer to form the input to the next layer. The resulting encodings are concatenated to form a final encoding in $\mathbb{R}^{3h}$

$$
\begin{aligned}
E_{i_{rnn}} &= \text{Bi-LSTM}(X)_i \in \mathbb{R}^{2h} \\
E_{i_{blk}} &= \text{Enc-Block}(X_i) \in \mathbb{R}^{h} \\
E_i &= [E_{i_{rnn}}; E_{i_{blk}}] \in \mathbb{R}^{3h}
\end{aligned}
$$

Note: The self-attention and feed-forward section of the QANet-encoder forms the transformer outlined in the original transformer paper [4]. The implementation of the transformer that is used is provided by Samuel Lynn-Evans[3]

Context-Query Attention Layer
The attention layer sits at the juncture of the query encoding Q and the context encoding C and is responsible for producing a query-sensitive representation of the context. Here, we use bidirectional attention flow [2], which computes both a context-to-query score and a query-to-context score for each position in the context and is based on a given similarity function. Let $S \in \mathbb{R}^{n \times m}$ represent the similarity matrix between the context encodings ($\mathbb{R}^n$) and query encodings ($\mathbb{R}^m$) from the previous

[3]https://github.com/SamLynnEvans/Transformer

layer. Let $\bar{S}$ be the row-softmax form of S and let $\bar{\bar{S}}$ be the column-softmax form of S. Furthermore, let $c_i \in \mathbb{R}^{3h}$ be the encoded context embedding at position i and $q_j \in \mathbb{R}^{3h}$ be the encoded query embedding at position j. The context-to-query attention (a_i) for c_i is computed as the sum, from $j = 1, ..., m$, of q_j multiplied by $\bar{S}_{ij}$. The query-to-context attention (b_i) for c_i is computed as the sum, from $j = 1, ..., n$, of c_j multiplied by $(\bar{S}\bar{\bar{S}}^T)_{ij}$. The output of this layer at a given position (g_i) becomes the concatenation $[c_i; a_i; c_i \odot a_i; c_i \odot b_i] \in \mathbb{R}^{12h}$, where $\odot$ represents elementwise multiplication. The output of this layer is immediately mapped to a vector of $\mathbb{R}^h$ size.

$$S_{ij} = \alpha(c_i, q_j) \in \mathbb{R}$$

$$a_i = \sum_{j=1}^{m} \bar{S}_{ij} q_j \in \mathbb{R}^{3h}$$

$$b_i = \sum_{j=1}^{n} (\bar{S}\bar{\bar{S}}^T)_{ij} c_j \in \mathbb{R}^{3h}$$

$$g_i = [c_i; a_i; a_i \cdot a_i; c_i \cdot b_i] \in \mathbb{R}^{12h}$$

Note: This structure of this layer only differs from the default baseline by the inclusion of the final projection layer to dimensionality $d = h$

Model Encoder Layer
The model encoder shares a similar approach and structure to the embedding encoder layer, being a concatenation of both an RNN network and a QANet-encoder layer. Here, the RNN-network is a 2-layer LSTM and the QANet-Encoder stack consists of 2 [convolutions / self-attention / feed-forward] blocks (instead of 1 for both). The input to this layer are the attention outputs G. The output of this layer is once again of dimensionality $d = 3h$.

$$M_{i_{rnn}} = \text{Bi-LSTM}(G)_i \in \mathbb{R}^{2h}$$

$$M_{i_{blk}} = \text{Enc-Block}(G_i) \in \mathbb{R}^{h}$$

$$M_i = [M_{i_{rnn}}; M_{i_{blk}}] \in \mathbb{R}^{3h}$$

Output Layer
The output layer computes the probabilities of the starting and ending positions of the answer span, at each position in the context. The computational flow of this layer takes as input the attention layer output (G) and model encoder layer output ($M1$), runs $M1$ through a bidirectional LSTM to produce $M2$, and computes a linear combination of $[G; M1]$ and a linear combination of $[G; M2]$ at each position. These two outputs at each position are the scores relating to the probability of the answer starting and ending there. The actual probabilities can be acquired through a soft-max. The corresponding span is the one that maximizes the product of its start and end probabilities among spans where the end position is at least the start position.

$$p_1 = \text{softmax}(W_1[G; M_1])$$

$$p_2 = \text{softmax}(W_2[G; M_2])$$

During training, we minimize the negative sum of log-likelihoods at the actual start and end index, across each batch. The null span is handled by prepending an OOV-token to the beginning of each context, and denoting it as the span beginning and ending at 0.

$$l(\theta) = -[\log(p_{1_{y_1}}) + \log(p_{2_{y_2}})]$$

Note: This layer differs from the original baseline output layer only to accommodate the size of the input which is now $3h$ instead of $2h$.

Table 1: Results

Model	EM	F1
Baseline	57.016	60.436
BiDAF (Baseline + char-embed)	59.721	62.822
BiDAF + Embedding Enc Blk	60.343	63.460
BiDAF + Embedding and Model Enc Blk	60.813	64.043
BiDAF + Emb. and Mod. Enc Blk (Test)	60.558	64.246

3 Experiments

3.1 Data

The experiment data is the SQuAD 2.0 dataset [1], which is similar to the original SQuAD [8] data used in the QANet paper except that the newer version contains examples with "no-answer" as an acceptable response. The dataset is aptly split into 129941 training examples, 6078 dev examples, and 5915 test examples as outlined in the project handout.

3.2 Evaluation Method

The two evaluation metrics that are used are F1 and EM scores. EM (Exact Match) measures the ability of a model to produce exact answers. F1 is a more lenient measure of the similarity between the predicted and target phrases.

3.3 Experiment Details

Setup Details
We invoke a context limit of 400 words, a question limit of 50 words, and answer limit of 30 words in the following experiments. Texts are truncated and padded accordingly during training. For the input embeddings, we use pretrained GloVe vectors ($\mathbb{R}^{300}$) [9] and randomly-initialized character vectors of dimension $d = 64$. Furthermore, the character limit, used in computing a word's character-level embedding component, is set to 16. For the baseline and baseline-char experiments, we use a hidden size of 128. For the transformer-augmented experiments, we use a hidden size of 96 to account for the augmented dimensionality between the embedding encoder and attention layers and between the model encoder and output layers. The kernel size is set to 7 for all convolution layers. The number of convolution layers is 4 in embedding QANet-encoder and 2 in the model QANet-encoder.

Training Details
The dropout rate between layers is set to 0.1 (0.05 in the CharCNN). For the fully-augmented model, we also employ a stochastic depth dropout between convolution layers in the transformer block [10]. We use the Adadelta optimizer [11], an initial learning rate of 0.5, and a batch size of 64. The number of epochs is 30. Because of large differences in training time, each experiment is run on a different machine. The non-transformer based models are trained on an NV6 machine (20-30 minutes per epoch) and the transformer-augmented models are trained on an NV12 machine (30-40 minutes per epoch). All models are implemented using PyTorch [12].

3.4 Results

In terms of dev set performance, there is a +3.797 in EM performance and +3.607 in F1 performance from the baseline to the final model. We also see incremental improvements between each level of augmentation in the architectures. Although these improvements are non-trivial, we initially expected a larger performance increase from a hybrid RNN/Encoder-Block approach that seeks to combine the best of each individual approach. One thing to note is that compared to the model outlined in the QANet paper, we used fewer convolution layers in each block, and significantly fewer blocks in the model encoder (2 instead of 7).

4 Analysis

4.1 Non-answerable Questions

Upon inspection of the output, it appears that most of the mistakes that the model makes are due to the non-answerable queries introduced in SQuAD 2.0. For comparative purposes, the same model achieves an EM score of 70.54 and F1 score of 79.21 on the dev set without non-answerable queries, a stark contrast from the model's performance on the full dev set (EM: 60.813, F1: 64.043). To account for this discrepancy, we will on the issue of "consistent" answers in our analysis. Consistent answers make sense on some syntactical or semantical levels. For example, a consistent answer to a "Who?" query would be the name of a person. Consistent answers are "partially correct" in the sense that they can get some components of a query correct, but not necessarily all. Models that break down contexts and queries into salient features can lead to heavily component-based systems that check for consistency. For a model that knows that a correct answer exists (SQuAD 1.1 [8]), this consistency often leads to correctness. For a model that does not (SQuAD 2.0), achieving good consistency is no longer a guarantee of correctness and the problem becomes much harder.

EXAMPLE 1
Question: What is the oldest work of Norman art?
Context: By far the most famous work of Norman art is the Bayeux Tapestry, which is not a tapestry but a work of embroidery. It was commissioned by Odo, the Bishop of Bayeux and first Earl of Kent, employing natives from Kent who were learned in the Nordic traditions imported in the previous half century by the Danish Vikings.
Answer: N/A
Prediction: Bayeux Tapestry

This is one of many examples that illustrate the above phenomenon. Here, the query is looking for an answer that is first and foremost a "work of Norman art". The model produces an answer that is indeed an example of Norman art, so it has already achieved a great deal of consistency. The issue is that the modifier "oldest" in the query completely nullifies the prediction, and the model is ignoring it or not attending to it sufficiently. Somehow, the model needs a mechanism to encourage that all parts of the query get proper attention in the context.

EXAMPLE 2
Question: Of Poland's inhabitants in 1901, what percentage was Catholic?
Context: Throughout its existence, Warsaw has been a multi-cultural city. According to the 1901 census, out of 711,988 inhabitants 56.2% were Catholics, 35.7% Jews, 5% Greek orthodox Christians and 2.8% Protestants. Eight years later, in 1909, there were 281,754 Jews (36.9%), 18,189 Protestants (2.4%) and 2,818 Mariavites (0.4%). This led to construction of hundreds of places of religious worship in all parts of the town. Most of them were destroyed in the aftermath of the Warsaw Uprising of 1944. After the war, the new communist authorities of Poland discouraged church construction and only a small number were rebuilt.
Answer: N/A
Prediction: 56.%

Here is another example of the model producing a consistent answer (percentage of inhabitants that were Catholic) that is nearly correct, but overlooks a key aspect between the answer and the query (Warsaw $\neq$ Poland). The model needs to make better use of the fact that "Poland" does not appear in the particular section describing the answer span.

4.2 Answerable Questions

Although non-answerable questions make up the bulk of the mistakes in the model, we would also like to address some of the errors in the answerable set. Some of these errors give an insight into the deficiencies in the question-answering system in general.

EXAMPLE 3
Question: A forced trade agreement between two countries would be an example of what?

Context: The definition of imperialism has not been finalized for centuries and was confusedly seen to represent the policies of major powers, or simply, general-purpose aggressiveness. Further on, some writers[who?] used the term imperialism, in slightly more discriminating fashion, to mean all kinds of domination or control by a group of people over another. To clear out this confusion about the definition of imperialism one could speak of "formal" and "informal" imperialism, the first meaning physical control or "full-fledged colonial rule" while the second implied less direct rule though still containing perceivable kinds of dominance. Informal rule is generally less costly than taking over territories formally. This is because, with informal rule, the control is spread more subtly through technological superiority, enforcing land officials into large debts that cannot be repaid, ownership of private industries thus expanding the controlled area, or having countries agree to uneven trade agreements forcefully.
Answer: "informal" imperialism
Prediction: N/A

In this example, the model struggles with deeper levels of inference. The correct answer lies on the understanding that a “forced trade agreement” is an example of “less direct rule though still containing perceivable kinds of dominance”, but this can be hard to conclude from the structure or wording of the sentence. There may be some similarities more deeply embedded in the context and query that could be picked up in the context-query attention layer, which would require a more extensive layering at the embedding encoder stage or higher dimensional word vectors.

5 Conclusion

Our work with QANet encoder blocks appears to be a moderately successful improvement over the baseline. The improvement amounts to a +3.607 in the F1 metric and +3.797 in the EM metric, on the dev set, using comparable training parameters, and consistent performance on the test set. Additionally, each stage of model augmentation had a positive effect compared to the previous stage: adding character-level embeddings, adding encoder blocks in the embedding encoder layer, and finally adding encoder blocks in the model encoder layer. The primary limitation, which the additional of encoder blocks does not directly address, is that of non-answerable questions and challenge they pose to the validity of consistent answers.

References

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

[2] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[5] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[6] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.

[7] Jianbo Guo, Yuxi Li, Weiyao Lin, Yurong Chen, and Jianguo Li. Network decoupling: From regular to depthwise separable convolutions. *CoRR*, abs/1808.05517, 2018.

[8] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[11] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.