Contextual Question Answering using variations of BiDAF and QANet

Stanford CS224N Custom Project

Achillefs Martinis
Department of Computer Science
Stanford University
martinis@stanford.edu

Abstract

Our goal with this project is to better take into account the compositional properties of language, as measured by the performance on the task of closed-domain question answering in the Stanford Question Answering Dataset (SQuAD) 2.0. In this vein, we implemented and studied two high-performing question answering models, which may be able to achieve this goal: Bidirectional Attention Flow (BiDAF), and QANet. Firstly, we extended the BiDAF implementation by Chris Chute [1], by adding character embeddings to handle unknown words and by adding self-attention to capture dependencies within the input text. Secondly, we implemented QANet from scratch, in order to better capture dependencies within the input text. We experimented with variations of layers and attention heads. Thirdly, we evaluated how our best-performing models perform on a variety of question types and answer types. We assumed our baseline to be Chris Chute's implementation of BiDAF, which has an F1 score of 62.45 and an Exact Match (EM) score of 59.17. Our best performing model, based on QANet, achieved an F1 score of 65.61 and an EM score of 62.31.

1 Key Information to include

- Mentor: Gaurab Banerjee
- External Collaborators (if you have any): Yash Dalmia and Julian Chu
- Sharing project: No

2 Introduction

Question answering systems attempt to provide an answer to a question based on a document. In open-domain question answering task, the system is expected to provide answers to questions based on a large unstructured document. In closed-domain question answering, which we focus on, the system is provided a passage and a question as inputs and it has to answer the question correctly based on the passage. Closed-domain question answering is interesting from a research point of view because it evaluates how well systems can capture the compositional property of language, which refers to the ability to capture the meanings of smaller parts of a passage and how they are related to each other to create meaning in a passage. From a practical point of view, closed-domain question answering has applications for search engines, voice assistants, and FAQ bots [2].

SQuAD 2.0 is an example of a closed-domain question answering dataset. It is based on Wikipedia paragraphs on diverse topics. A system receives as inputs a paragraph (i.e. context, passage) and a question (i.e. query). If the question is not answerable based on the paragraph, the system should predict that the passage cannot answer the question, while if the question is answerable, the system should predict the start and end indices of a span in the passage that answers the question (as opposed

to generating an answer). SQuAD 2.0 includes diverse question types, answer topics, and syntactic properties. In addition, it contains questions, which require different types of reasoning, which means that the answer in the passage may be a paraphrased or rearranged version of what the question is asking.

In the past, some of the highest performing question answering models employed recurrent units, such as models based on LSTMs (long-short-term-memory), or GRUs (gated recurrent units). This meant that these previous models had significant time constraints imposed by the serial processing of input in recurrent computations. Currently, many of the highest performing question answering models are pre-trained models (e.g ALBERT [3]).

Our goal is to capture the compositional property of language through a system. In pursuit of this goal, we make three contributions over the course of this project. First, we extended upon a Bidirectional Attention Flow model (BiDAF) implementation by Chris Chute [1], by adding a self-attention layer to capture longer-range relations within the passage. Second, we implemented from scratch another model, QANet[4], which draws inspiration from transformers. Third, we made modifications to our QANet layers and evaluated performance. Although we could use a pre-trained model to achieve superior performance on SQuAD 2.0, we wanted to explore how changing the architecture influences the ability to understand language. Hence, we focused on the two aforementioned architectures.

3 Related Work

In the 2010's, some of the highest performing models employed recurrent computations, and one of those models was BiDAF. BiDAF, as proposed by Seo et al [5], was the highest performing model for SQuAD 2.0 in 2016. Seo et al's key insight was in realizing that bidirectional attention between question and passage could improve models' understanding and ability to answer questions. Unfortunately, however, BiDAF made use of recurrent computation in the form of LSTMs to encode the context and query, something that led to significant time constraints imposed by the serial processing endemic to recurrent computation. In addition, although BiDAF performs attention between the passage and the question, it does not perform self-attention on the passage and, hence, may fail to capture long-range dependencies within the passage or within the question (example in E Appendix).

QANet, by Yu et al[4], is a newer model that dispenses the recurrent computations of BiDAF. QANet relies heavily on attention mechanisms, both for encoding query and context, as well as allowing the context and query to attend to each other. QANet is inspired in large by Vaswani et al "Attention is All You Need" [6]. QANet employs a couple of different ideas from this paper and from transformer, including their positional encoding and their scaled multiheaded attention formulation.

4 Approach

Our overarching approach to the SQuAD question answering task is to create a model that better understands queries (i.e. questions) and context (i.e. passage) by capturing structure of its language inputs. We made modifications to Chris Chute's implementation of BiDAF[1], and we implemented QANet from scratch.

4.1 Extending BiDAF

Below we describe BiDAF's layers, including its Attention layer, which is an element in the model's success, since it allows BiDAF to compute attention both from the context to the question and from the question to the context.

4.1.1 Embedding layer (Word+Character-level embeddings)

Provided with input word indices $w_1, ..., w_k \in \mathbb{N}$ and character indices $c_1, ..., c_n \in \mathbb{N}$, the layer looks up GloVe embeddings for each index, resulting in word embeddings $we_1, ..., we_k \in \mathbb{R}^{300}$ and character embeddings $ce_1, ..., ce_n \in \mathbb{R}^{200}$. This lookup is performed for both the passage and the question. Chris Chute's implementation only looked up word embeddings, but we added character embeddings in order to condition on characters in case a word is unknown. We cap each word to

16 character embeddings to have consistent tensors across words. We maxpool the 16 character-level embeddings per word to obtain one "summarised" character embedding per word. We then concatenate the "summarised" character-level embedding of each word with its word-level embedding to obtain embeddings of $e_1, ..., e_k \in \mathbb{R}^{d=500}$, one for each of the words.

Each embedding is projected to have dimension $H = 128$ for computational efficiency reasons. We achieve this with a linear layer which multiplies $W \in \mathbb{R}^{d \times H}$ with each $e \in \mathbb{R}^d$ embedding.

Lastly, we apply a Highway Network to each embedded representation, as per [7], in order to help prevent the vanishing gradient problem. We use a two layer high-way network, which outputs embedding for each word to give $h_1, ..., h_k \in \mathbb{R}^H$. Note that H is the size of each output embedding, while h_i refers to a particular output embedding. Further details about the layers of the Highway Network are provided by Chris Chute [1].

4.1.2 Encoder Layer

The Encoder layer uses the output embeddings, h_i, from the Embedding layer as input in a bidirectional LSTM. A bidirectional LSTM (depicted in A Appendix) helps in order to obtain dependencies from both past and future contexts of each word. The output at each forward timestep of the LSTM is $h'_{i,fwd} \in \mathbb{R}^H$ and the output at each backward timestep is $h'_{i,rev} \in \mathbb{R}^H$. The output at each timestep, i, of the entire bidirectional LSTM is the concatenation of the forward and backward hidden states of the LSTM, giving us $h'_i = [h'_{i,fwd}; h'_{i,rev}] \in \mathbb{R}^{2 \times H}$.

4.1.3 Attention Layer

The Attention Layer is the innovative layer of the BiDAF. It takes as input the passage's output states and the question's output states from the Encoder Layer. Suppose that the Encoder Layer output is $p_1, ..., p_i, ..., p_N \in \mathbb{R}^{2 \times H}$ for the passage and $q_1, ..., q_j, ..., q_M \in \mathbb{R}^{2 \times H}$ for the question. In the Attention Layer, we find a similarity matrix $S \in \mathbb{R}^{N \times M}$ with similarity score $S_{ij} \in \mathbb{R}$ for each pair (p_i, q_j) [1].

BiDAF's unique strength is that attention flows both from the passage to the question and from the question to the passage. In the matrix S, each row represents a passage word, i, and each column represents a question word, j. To perform Passage-to-Question attention, we take the row-wise softmax of S and we use it to take a weighted sum across question hidden states, q_j. We obtain $a_i \in \mathbb{R}^{2 \times H}$, for each passage word, i.

To perform Question-to-Passage attention, we perform similar computations. We obtain embeddings, $b_i \in \mathbb{R}^{2 \times H}$, for each passage word, i.

Last, we get output g_i for each passage word, i, where $g_i = [p_i; a_i; p_i \circ a_i; p_i \circ b_i] \in \mathbb{R}^{8 \times H}$ and where $\circ$ represents the element-wise multiplication of two vectors[8].

4.1.4 Self-attention Layer

The original BiDAF model by Seo et al [5] did not have a Self-attention Layer. However, we were inspired to add it after reading Microsoft's paper on its R-NET model. Microsoft's R-NET model had the insight to perform self-attention (which they refer to as self-matching) among all the embeddings after the Passage-to-Context attention[9]. Taking inspiration from this paper, we implemented a self-attention layer, whose inputs are the output embeddings, $g_i \in \mathbb{R}^{8 \times H}$, from the Attention Layer. There is one embedding g_i per passage word, and it can be interpreted as a question-aware representation. Self-attention on the g_i's makes them attend to each other, hopefully capturing global dependencies within the question-aware representation of the passage. We performed Multi-Headed Attention (with 8 heads), since having Single-Headed has shown to have reduced descriptive power, and the output of this Self-attention layer consists of embeddings $s_1, ..., s_N \in \mathbb{R}^{8 \times H}$.

4.1.5 Modeling Layer and Output Layer

In accordance with the original BiDAF, we use the outputs of the previous layer (the Attention Layer or the Self-attention Layer if it has been applied) as input timesteps for another bidirectional LSTM on the passage. The input to the LSTM consists of embeddings of size $\mathbb{R}^{8 \times H}$ for each word. This layer could help capture dependencies between words in the question-aware passage representation.

In contrast to the Encoding Layer's bidirectional LSTM, which has one layer, the Modeling Layer's bidirectional LSTM has two layers. The output of the Modeling Layer consists of hidden states $m_1, ..., m_N \in \mathbb{R}^{2\times H}$, one embedding for each passage word.

In the Output Layer, we need to find the probabilities that each passage word, i, is the start of the answer or the end of the answer. In other words, we want to find $p_{start}(i)$ and $p_{end}(i)$ for each word. The Output Layer takes as input, either the Attention Layer outputs, $g_1, ..., g_N \in \mathbb{R}^{8\times H}$, or the Self-Attention layer outputs, $s_1, ..., s_N \in \mathbb{R}^{8\times H}$, if Self-Attention was applied (let's use $v_1, ..., v_N \in \mathbb{R}^{8\times H}$ as a variable to equivalently refer to either g_i's or s_i's, depending on what is the input). In addition, the Output Layer takes as input the Modeling Layer's output, $m_1, ..., m_N \in \mathbb{R}^{2\times H}$. We apply a bidirectional LSTM using $m_1, ..., m_N$ as inputs for each timestep of the LSTM, and we get hidden states $m'_1, ..., m'_N \in \mathbb{R}^{2\times H}$ as output. Let's say that $G \in \mathbb{R}^{8H\times N}$ is the matrix with $v_1, ..., v_N \in \mathbb{R}^{8\times H}$ as columns, $M \in \mathbb{R}^{2H\times N}$ is the matrix with $m_1, ..., m_N \in \mathbb{R}^{2\times H}$ as columns and $M' \in \mathbb{R}^{2H\times N}$ is the matrix with $m'_1, ..., m'_N \in \mathbb{R}^{2\times H}$ as columns. We find p_{start} and p_{end} as follows:

$$p_{start} = softmax(W_{start}[G; M])p_{end} = softmax(W_{end}[G; M'])$$

where $W_{start} \in \mathbb{R}^{1\times 10H}$ and $W_{end} \in \mathbb{R}^{1\times 10H}$ are projection matrices, $[G; M]$ is the concatenation of matrix $G\mathbb{R}^{8H\times N}$ with matrix $M \in \mathbb{R}^{2H\times N}$ along the rows, and $[G; M']$ is the concatenation of G with M'.

4.2 Implementing QANet

We've also reimplemented QANet mostly from scratch, without reference to other existing implementations of QANet. At high level, QANet contains an embedding layer, an encoding layer, a bidirectional attention layer, a modeling layer, and a layer for outputting the final predictions. The details of each layer follow:

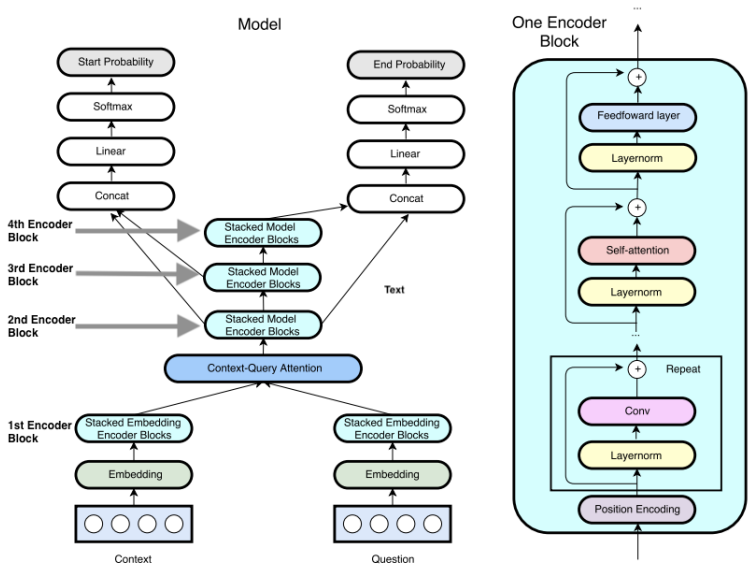

Figure 1: Model architecture of the QANet model from the Yu et al paper [4]

4.2.1 Embedding layer:

The Embedding Layer in the QANet is mostly similar to the BiDAF embedding layer. This layer takes in a sequence of word indices and character indices for both the context and the question. For each word index and character index in the context and the question, the embedding layer looks up its pre-trained GloVe word embeddings and the pre-trained character embeddings. We cap or pad all the words to 16 character embeddings per word and we pass the 16 character embeddings per word through a pooling layer, which "summarises" the 16 character embeddings to 1 embedding. We

concatenate the word embedding for each word with its "summarized" character embedding getting a new word embedding $e_i \in \mathbb{R}^{d=500}$ per context word and per question word. We apply dropout layer on the context word and character word embeddings to prevent the model from overfitting these parameters. The layer then passes each of these embeddings through a linear layer. We apply a Highway Network layer (that we reused from Chris Chute's implementation[1]), which empirically helps avoid the vanishing gradient problem in back-propagation. Lastly, we pass the embeddings through a convolution layer, which reduces each embedding to have dimension of size $H = 128$. Hence, the output embeddings per word are $e_i \in \mathbb{R}^{H=128}$.

4.2.2 Encoder Block

The Encoder Block is the key element distinguishing Yu et al's QANet. With inspiration from the "Attention is All You Need" paper [10], this block computes positional encodings for each word to capture sentence order and performs self-attention between words to capture long-range dependencies within the passage and within the question. The Encoder Block is employed in two places in the QANet model pipeline, once to encode the question and answer embeddings, and, second, to encode the bidirectional-attention-enhanced passage.

We first add positional encodings to our inputs, which are the output embeddings per word from the Embedding Layer, $e_i \in \mathbb{R}^H$. We reused the positional encodings from the Annotated Transformer, an implementation of the "Attention is All You Need" paper [10] by Harvard University. Following the positional encoding is a stack of convolutional layers. The convolutions help to learn local dependencies between position-aware word embeddings [4]. We take the output embeddings from the convolutional layer, and we apply a self-attention layer in order to learn global word relationships [4]. Lastly, a Feedforward layer is applied. Before layers, we employ layer norms and residual connections, which empirically improve training by avoiding vanishing and exploding gradient issues on deep neural networks [4].

Between the two applications of this encoding layer, there are some key differences. Notably, when encoding the question and context embeddings (1st encoder in Figure 2), our encoder stack uses a single Encoder Block, and, within this block, we use 4 convolutional layers and kernel size 7, as per Yu et al's paper [4]. The Multihead self-attention layer employs 4 heads in the 1st Encoder Block. In contrast, when applying the Encoder Blocks to the bidirectional-attention-enhanced context, we stack three Encoder Blocks (2nd, 3rd, and 4th encoders in Figure 2), each one operating upon the input of the last. These three blocks each contain 2 convolution layers, but instead use a kernel size of 5. This layer uses 8 headed self attention instead of 4.

4.2.3 Bidirectional Attention (i.e. Context-Query Attention layer in Figure 2)

Next is the Bidirectional attention layer. Our inputs are the output embeddings per word from the 1st Encoder Block. Suppose that the Encoder Block output embeddings are $p_1, ..., p_i, ..., p_N \in \mathbb{R}^{H=128}$ for the passage and $q_1, ..., q_j, ..., q_M \in \mathbb{R}^H$ for the question. These are self-attended passage word and question word representations, which may capture long-range dependencies within the passage and within the question. We apply to them BiDAF's highly successful Attention Layer, where attention flows from context words, p_i, to question words, q_j, and vice versa. This layer within QANet was borrowed from Chris Chute's implementation of BiDAF[1]. The output of this layer is a question-aware embedding, $g_i \in \mathbb{R}^{4 \times H}$, for each passage word, where $g_i = [p_i; a_i; p_i \circ a_i; p_i \circ b_i]$, is created just like was described in section 4.1.3.

4.2.4 3 Encoder Blocks

The inputs to each of the three Encoder Blocks are the passage word representations, g_i, from the Bidirectional Attention layer. The three Encoder Blocks share weights with each other in training. Unlike the original Yu et. al QANet paper[4], where each of the 3 blocks is repeated 7 times (for a total of 7*3=21 blocks), we only go through each of the three blocks once, for computational efficiency. The output of each block consists of representations b_i, for $i = 1, ..., N$ passage words. By passing the output of a block as input to another block we find three outputs.

4.2.5 Output Pointers

Finally, we take the three outputs from the encoder layers and pipe them into the output layer. The first two outputs are concantenated, projected, and ultimately softmaxed to determine a start pointer. Likewise, the first and last are concatenated, projected, and ultimately softmaxed. The first operation results in a probability distribution over likely start indexes, while the second operation results in a probability distribution over end indexes.

4.3 Loss function

As proposed in the Seo et al paper [5] and the Yu et al paper [4], our loss function is $L = \frac{-1}{N}\sum_i^N[\log(p^{start}_{y_i^{start}}) + \log(p^{end}_{y_i^{end}})]$, for both the BiDAF model and the QANet model, where y_i^{start} and y_i^{end} are the true start and end indexes of the answer span.

5 Experiments

5.1 Data

We are training on a modified version of the official SQuAD 2.0 dataset. We split the official dev set and only use part of it as our dev set while we use the rest of the official dev set as our test set (this is a split from previous offering of CS224N and can allow us to compare our results against students from previous years). There are, all in all, 129,941 examples in our train set, 6,078 examples in our dev set and 5,915 examples in our test set. These examples are in the form of triples <context, question, answer>. We will additionally make use of GloVe precomputed word vectors and precomputed character vectors. [11].

5.2 Evaluation method

We use Exact Match (EM) scores and F1 scores scores to evaluate our implementations of QANet and BiDAF versus the BiDAF baseline. Each EM score per test example is a binary variable telling us whether our prediction (i.e. start and end indices) exactly match the ground truth indices. F1 is a less strict measure (and is usually higher). It represents the harmonic mean of precision and recall.

For example, suppose the ground true answer is "Not John" while our model predicts "John". The EM score would be 0 for this example since the ground truth diverges from the prediction. F1, however, would be 66.7%. The precision is 100% since our answer is a subset of the ground truth answer, but recall would be 50% since our answer only has one of the two ground truth words. Hence F1 score would be $\frac{2\times50\times100}{100+50} = 66.7\%$.

5.3 Experimental details

We used Google Colab default GPU's, which are typically NVIDIA T4 Tensor Core GPU's. Colab provided 12GB of RAM per session and allowed for 12 hours of continuous training after which the session would time out. We did not have access to Google Cloud resources, which could have afforded us higher performance GPU's, due to technical difficulties with AWS credits. Below are some details for the models we trained.

We used dropout probability of 0.1 and the Adadelta optimizer with learning rate of 0.5. The total training time was about 40 minutes for 1.4 million training examples for BiDAF baseline and BiDAF with character embeddings. For the remaining models, the training times were about 4 hours for QANet models with 2-4 attention heads, 15 hours for the QANet model with 8 attention heads, and 17 hours for the BiDAF model with self-attention.

5.4 Results

We experimented with variations of BiDAF and QANet. For BiDAF, we have three models: 1) the Baseline model, which is from Chris Chute's implementation[1], 2) BiDAF with character-level embeddings, and 3) BiDAF with character-level embeddings and self-attention. For QANet, we experimented with different numbers of attention heads for the self-attention and different repetitions

of the 3 Encoder Blocks (as we noted earlier, the Yu et al paper has 7 repetitions of each of the three Encoder Blocks). We were not able to repeat the approach of Yu et al. [4](7 Encoder Blocks and 8 attention heads), which might have yielded optimal results, due to Google Colab's memory restraints and GPU's. However, I think that if we were able to replicate 7 Encoder Blocks repetitions, we could have increased performance of our QANet model further.

		F1	EM
BiDAF	BiDAF Baseline	62.45	59.17
	BiDAF w/ character embeddings	65.24	62.21
	BiDAF w/ character embeddings and self-attention	59.62	56.23
QANet	**QANet 1 encoder block 8 attention heads**	**65.61**	**62.31**
	QANet 2 encoder blocks 2 attention heads	59.3	56.76
	QANet 2 encoder blocks 4 attention heads	58.02	55.28
	QANet 4 encoder blocks 2 attention heads	59.05	56.28

Figure 2: Scores of different model variations

Our QANet model with 1 Encoder Block and 8 attention heads outperformed our baseline model in terms of F1 and EM scores. This is as expected and is evidence in favor of the benefits of self-attention and positional encodings, which are the key elements of QANet. I hypothesize that both self-attention and positional encodings endemic to QANet help capture global dependencies within the passage and question, something which the LSTMs of BiDAF might fail to do to the same extent. Expectedly, the QANet models with fewer attention heads for the self-attention performed worse than the QANet model with 8 attention heads. They also performed worse than the baseline, which I think is because using fewer attention heads results in less descriptive power in the learned self-attention embeddings.

Surprisingly, our BiDAF model with self-attention performed worse than the BiDAF baseline. I had expected that the self-attention layer in BiDAF would enhance BiDAF's performance by helping it capture long-range dependencies between the question-aware passage embeddings. The worse performance of the BiDAF self-attention model could be attributed to the fact that the model was not trained for sufficient epochs to converge.

6 Analysis

6.0.1 Model performance by answer type

We wanted to explore how our models performed in different answer categories (e.g. answers about People, Dates, Percentages etc.). We used the SpaCy entity recognition module [12] in order to classify the answers of each dev set example, and we evaluated our three highest performing models on these answer groups.

As seen in Figure 3, the three models tend to perform well in cases where the answer is a Percentage, a Date, or a Monetary value. One explanation of this is training data bias, which leads the models to become better at recognizing these types of answers, due to more training examples (this could be the case for Dates, which have 10,146 examples in the train set). Another possible explanation behind the high performance on these answer categories, could be that they have fixed formats (e.g. dates are separated by /, Percentages and Monetary values are post-pended by % or $), which might make it easier for the model to identify them in the question and match to the answer in the passage. For example, the models identify when a question asks "What percentage ..." and respond with a %, which they find in the passage. There are even specific cases, where this pattern matching occurs finding a wrong number (Example in E Appendix).

Despite their successes, all three models underperform on Ordinal answers (e.g. "fist", "second"), and I hypothesize that this could be because these answers require temporal reasoning, which our models are lackluster at despite self-attention mechanisms. I found multiple examples where the models exhibit their lack of temporal reasoning and one of these is in C Appendix. If the problem is a lack of temporal reasoning training examples, one way to improve the models is to train them on datasets with temporal reasoning, such as TempEval3 [13]. If the problem is an inability by the model architecture to capture temporal relations, we would have to change our architecture. One idea is to finetune the pre-trained word embeddings in the Embedding Layer of our model, in order to capture temporal information for temporal indicators (e.g. before, after, during, etc.) [14]

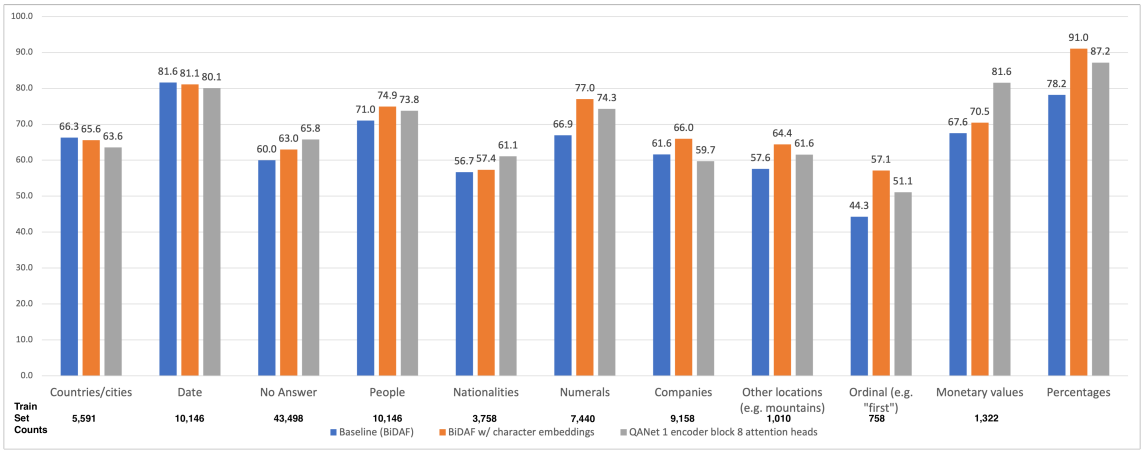

Figure 3: F1 by answer category of top 3 performing models

6.0.2 Model performance by question type

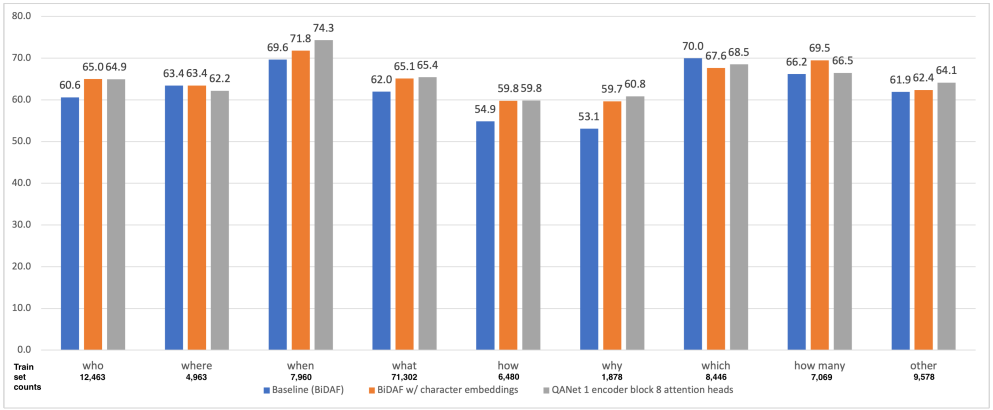

Figure 4: F1 by question category of top 3 performing models

As we can see from Figure 4, the 3 top performing models underperform in questions, which start with "how" or "why". We went through specific examples of such questions and realized that many of the questions require causal inference in order to answer. Our hypothesis is that our models may be unable to understand the causal arrows of logic despite self-attention. One example of a question requiring causal inference is in D Appendix. One idea to improve upon the cause-and-effect reasoning is to finetune the pre-trained word embeddings to capture information about causal indicators (e.g. because, in response). Another idea, inspired by Hudson et al's Neural State Machine [15] is to represent both the context as a language graph (where words are states and relations between them are edges) and represent the query question as a series of manipulations to that graph. This is a large change to our current architectures but would be interesting to explore.

7 Conclusion

In conclusion, we have extended Chris Chute's BiDAF implementation with character emmbeddings, allowing it to perform better than the baseline BiDAF. We also implemented QANet from scratch and achieved relatively good F1 and EM scores on SQuAD 2.0. We showed that lowering the number of attention heads in QANet significantly lowers performance. In the future, we would want to experiment with increasing the number of encoder blocks in QANet, while keeping 8 attention heads. We are also interested in ways of improving the model's temporal reasoning and cause-and-effect reasoning, by finetuning the pre-trained word embeddings and by using graphs to represent the passage.

References

[1] Chris Chute. Starter code for stanford cs224n default final project on squad 2.0, 2020.

[2] Use cases for question answering - azure cognitive services.

[3] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

[4] 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, 2018.

[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016.

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

[7] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks, 2015.

[8] Cs224n: Natural language processing with deep learning.

[9] Natural Language Computing Group. R-NET: Machine Reading Comprehension with Self-matching Networks, may 2017.

[10] Alexander Rush. The annotated transformer, 2018.

[11] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD, 2018.

[12] Linguistic features · spacy usage documentation.

[13] Siddharth Vashishtha, Adam Poliak, Yash Kumar Lal, Benjamin Van Durme, and Aaron Steven White. Temporal reasoning in natural language inference. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4070–4078, Online, November 2020. Association for Computational Linguistics.

[14] Rujun Han, Xiang Ren, and Nanyun Peng. Deer: A data efficient language model for event temporal reasoning, Dec 2020.

[15] Drew Hudson and Chris Manning. Learning by abstraction: The neural state machine. In *NeurIPS*, 2019.

[16] Enes Zvornicanin. Differences between bidirectional and unidirectional lstm, Nov 2022.

A Appendix

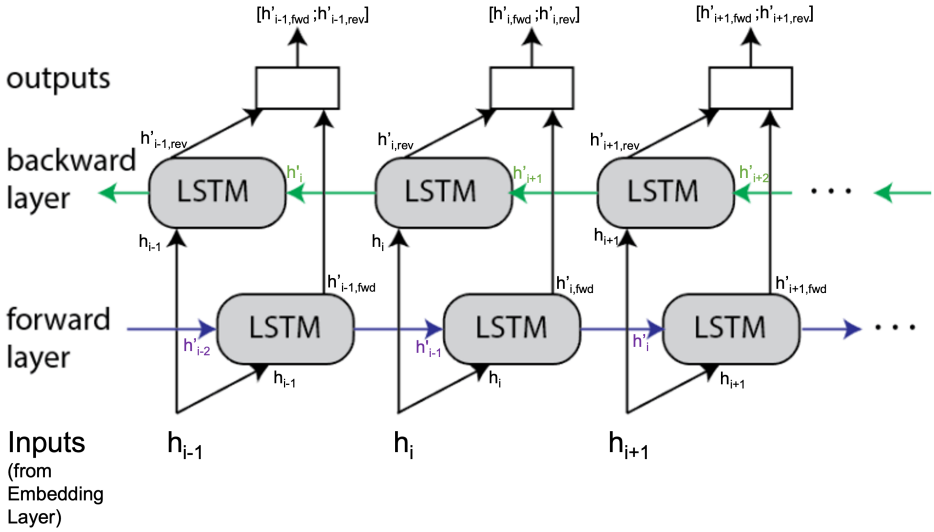

Figure 5: Bidirectional LSTM [16]

B Appendix

	Countries/cities	Date	No Answer	People	Nationalities	Numerals	Companies	Other locations (e.g. mountain	Ordinal (e.g. "first"	Monetary values	Percentages
Count in dev set	216	339	3168	295	166	185	352	43	18	17	26
Baseline (BiDAF)	66.32	81.63	60.01	71.01	56.67	66.95	61.60	57.57	44.28	67.56	78.21
BiDAF /w character embeddings	65.57	81.14	62.97	74.94	57.36	77.02	65.97	64.44	57.12	70.48	91.03
QANet 1 encoder block 8 attention heads	63.59	80.08	65.78	73.79	61.09	74.30	59.72	61.59	51.10	81.59	87.18

Figure 6: F1 by answer category of top 3 performing models

	who	where	when	what	how	why	which	how many	other
Count in dev set	604	245	436	3358	317	89	264	271	494
Baseline (BiDAF)	60.6	63.4	69.6	62.0	54.9	53.1	70.0	66.2	61.9
BiDAF w/ character embeddings	65.0	63.4	71.8	65.1	59.8	59.7	67.6	69.5	62.4
QANet 1 encoder block 8 attention heads	64.9	62.2	74.3	65.4	59.8	60.8	68.5	66.5	64.1

Figure 7: F1 by question category of top 3 performing models

C Appendix

uuid: f4aaa69a30d2b14934096002f

Question: What architecture type came before Norman in England?

Context: In England, the period of Norman architecture immediately succeeds that of the Anglo-Saxon and precedes the Early Gothic. In southern Italy, the Normans incorporated elements of Islamic, Lombard, and Byzantine building techniques into their own, initiating a unique style known as Norman-Arab architecture within the Kingdom of Sicily.

Ground truth answer: Anglo-saxon, **BiDAF baseline answer:** Early Gothic, **BiDAF w/ character embedding answer:** Early Gothic, **QANet 1 encoder block 8 attention heads answer:** No answer.

Analysis: Answering this question requires temporal reasoning in order to understand the temporal relation among Norman, Anglo-Saxon, and Early Gothic styles and identify which style came before the Norman. None of the models predict the answer correctly, indicating their weakness in the area.

D Appendix

uuid: 0a7b56f8eca762a677c16c67b

Question: Why did OPEC raise the price of oil to $5.11?

Context: In response to American aid to Israel, on October 16, 1973, OPEC raised the posted price of oil by 70%, to $5.11 a barrel. The following day, oil ministers agreed to the embargo, a cut in production by five percent from September's output and to continue to cut production in five percent monthly increments until their economic and political objectives were met. On October 19, Nixon requested Congress to appropriate $2.2 billion in emergency aid to Israel, including $1.5 billion in outright grants. George Lenczowski notes, "Military supplies did not exhaust Nixon's eagerness to prevent Israel's collapse...This [$2.2 billion] decision triggered a collective OPEC response." Libya immediately announced it would embargo oil shipments to the United States. Saudi Arabia and the other Arab oil-producing states joined the embargo on October 20, 1973. At their Kuwait meeting, OAPEC proclaimed the embargo that curbed exports to various countries and blocked all oil deliveries to the US as a "principal hostile country".

Ground truth answer: In response to American aid to Israel, **BiDAF baseline answer:** response to American aid to Israel, **BiDAF w/ character embedding answer:** a barrel, **QANet 1 encoder block 8 attention heads answer:** a barrel.

Analysis: The BiDAF baseline model surprisingly obtains an answer which is close to the ground truth. However, the other two models falsely predict. This question requires cause-and-effect reasoning, in order to identify that after American aid to Israel, the OPEC raised the price in response. The BiDAF and QANet models mispredict, indicating that they fail to understand the logical relationship inherent in the sentence.

E Appendix

uuid: f4af15a3ae19eb5b7dc223b3a

Question: What percentage of Warsaw's population was Protestant in 1901?

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.

Ground truth answer: 2.8%, **BiDAF baseline answer:** 2.4%, **BiDAF w/ character embedding answer:** 2.4%, **QANet 1 encoder block 8 attention heads answer:** 2.8%.

Analysis: The BiDAF models both predict the wrong answer, 2.4% (the 1909 figure), while the QANet model correctly predicts 2.8% (the 1901 figure, which is being asked for). My hypothesis as to why the BiDAF models predict 2.4% incorrectly is that that they see "What percentage..." and relate it to "Protestant" in their local window, but they fail to capture the long-range relation between the words "percentage" and "1901" within the question. I think QANet makes this long-range relation because it applies the Encoding Block on the question, hence capturing long-range dependencies through the self-attention layer.