Squadobots and Decepticonvs

Stanford CS224N Default (IID SQuAD) Project
CA Mentor: Kathy Yu

Amine Elhafsi
Department of Aeronautics and Astronautics
Stanford University
amine@stanford.edu

Patrick Washington
Department of Aeronautics and Astronautics
Stanford University
phw@stanford.edu

Abstract

We consider the problem of building a model capable of performing Question Answering (QA) on the Stanford Question Answering Dataset (SQuAD) 2.0. The QA problem refers to the problem of producing a response to some query given some context information (e.g., a document database, the web, etc.) and is relevant to many modern day services ranging from digital assistants like Siri and Alexa to Google search. In this work, we build upon the BiDAF architecture and investigate the effect of introducing Transformer blocks and convolution layers on the model's QA performance. Our best model yields respective F1 and EM scores of 64.39 and 61.55 on the development data set and 63.936 and 60.372 on the test data set. We show that pretraining the model was responsible for a respective increase of 0.58 and 0.89 in these scores evaluated on the development set.

1 Introduction

Natural Language Processing (NLP) technologies drive many modern day technologies and conveniences. These technologies drive services ranging from intelligent digital assistants like Siri and Alexa to Google search. A key desideratum of the aforementioned services is their ability to correctly respond to a users queries such as general trivia questions, song lyrics retrieval, or general web search.

In this work, we consider the related problem of Question Answering (QA). The QA task refers to the problem of producing a response to some query given some context information (e.g., a document database, the web, etc.) and has spawned a research area that sits under the umbrella of Natural Language Processing (NLP) and information retrieval. In particular, this project explores deep learning solutions for the question answering task put forth by the Stanford Question Answering Dataset (SQuAD) 2.0 [1].

We consider the BiDAF model [2], the state-of-the-art model on the SQuAD leaderboard in 2017, as a baseline architecture and investigate the effects of various modifications and augmentations on its QA performance as measured by the Exact Match (EM) and F1 metrics. The BiDAF model relies on Long Short-Term Memory (LSTM) networks for much of its sequence modeling, which are recognized as seeing difficulty in modeling long range dependencies [3]. This issue is particularly relevant to the QA task since the context length in the SQuAD 2.0 dataset can reach lengths of hundreds of words. To this end, we attempt to overcome these limitations by improving the BiDAF sequence modeling capabilities through the introduction of Transformer [4] and convolution layers. We additionally explore the benefit of pretraining on the SQuAD 2.0 context data.

2 Related Work

Numerous models have been proposed to solve not only the SQuAD QA challenge, but the QA and reading comprehension problems related to NLP more broadly. Classical approaches to reading comprehension tasks have typically relied on statistical methods, simple classifiers using manually defined features, and rule-based methods. For example, [5] considers the problem of machine answering of multiple-choice questions pertaining to fictional stories. This work considered two baseline systems: one uses a sliding window to match a bag of words and the other computes distances between words in the question and context documents. In [6], the context document is mapped to a learned structured representation and the question is transformed into a formal query, which is then matched to the context to produce an answer. Taking a statistical approach, [7] proposes a baseline QA solution using a conventional classifier for each candidate answer based on a set of handcrafted feature templates (e.g., frequency of entity in context, sentence co-occurrence between entity and question words, etc.).

With respect to the modern deep learning era, earlier approaches to performing reading comprehension ([2], [7], [8], [9]) generally relied on Recurrent Neural Networks (RNNs) for sequence modeling usually in conjunction with some attention element. [10] proposed using neural networks as a long-term memory knowledge base from which context information may be drawn.

However, most recent approaches to QA incorporate Transformer architectures given their improved sequence modeling capabilities over RNNs. For example, QANet [11] largely resembles the BiDAF model [2] in its use of a core attention module, with the notable displacement of LSTM layers by Transformer-like blocks to improve global modeling of text. QANet also introduces convolutions within the Transformer blocks to improve the local comprehension. There has also been a movement towards the use of large pretrained Transformer models such as BERT [12] and GPT-3 [13], which has proven to be successful at priming NLP models for a variety of tasks given the vast quantity of unsupervised natural language data available on the web.

In this work, we begin with the BiDAF architecture and attempt to adapt it by adopting elements that have shown success in recent QA systems, such as the incorporation of convolution layers for local context handling, Transformer architectures for better global context modeling and model pretraining.

3 Approach

3.1 Problem Formulation

Given a question of N words, $\{q_1, \ldots, q_N\}$, and a context paragraph of M words, $\{c_1, \ldots, c_M\}$, the objective of the QA task is to identify a span of context words $\{c_{\text{start}}, \ldots, c_{\text{end}}\}$ that correctly answers the question. In certain cases, the question may not be answerable with the provided context. In these situations, the QA system must indicate this fact by returning the span $\{c_0, c_0\}$ with c_0 representing a special token for this specific purpose.

3.2 Model Structure and Variations

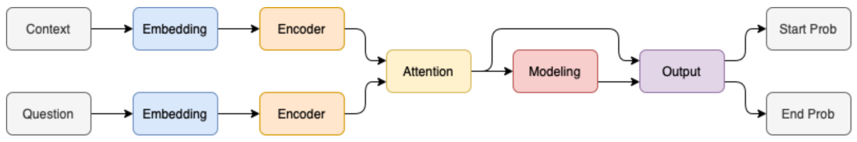

Figure 1: This is the structure of the BiDAF model that we based our models on.

Our model follows the overall structure of the BiDAF implementation and consists of 5 main components as depicted in Fig. 1. For simplicity, we maintain largely the same layer nomenclature used in [2], however we note that the specifics of each layer were varied as we investigated the effects of different modifications and augmentations.

3.2.1 Embed Layer

The Embed Layer maps each input word token into a word embedding. Word embeddings were simply taken as the 300-dimensional pretrained GloVe embeddings [14]. We opted to hold these embeddings fixed during training across all of our model variants given that we could not guarantee that the word distribution of the SQuAD 2.0 dataset matched that of the dataset with which the GloVe embeddings were originally learned.

Our implementation can optionally produce a character embedding for each word. In this case, words considered as arrays of 16 characters with shorter words padded and longer words truncated. Individual character representations were randomly initialized as 64-dimensional vectors and learned during training. To produce a fixed-size character embedding, the individual character representations were input to a 1D convolution layer with input channel equal to the individual character representation size. Convolutions were performed over windows of 5 characters with 200 output channels. Max-pooling-over-time is then perform to yield a 200-dimensional character embedding. When character embeddings are used, the Embed Layer concatenates the word and character embeddings to produce a 500-dimensional embedding.

The resulting embeddings are then passed through a two-layer Highway Network [15] to refine the embeddings into 200-dimensional features.

3.2.2 Contextual Embed Layer (Encoder Layer)

The Contextual Embed Layer serves to further process the embeddings and incorporate contextual information from the input sequence (i.e., context or question). We considered multiple instantiations of this layer.

Bidirectional RNN

As our baseline we retained the single-layer bi-directional LSTM used in the BiDAF model. Thus, the 200-dimensional inputs at each position in the sequence would result in a 400-dimensional output, corresponding to the concatenation of the hidden states produced by the LSTMs proceeding in either direction.

Convolution-Transformer

As an alternative architecture, we implemented this layer taking significant inspiration from [11]. Our motivation was to combine local context modeling through the use of convolution layers, while still maintaining the capability to handle global context modeling through the use of multiheaded self-attention. This architecture is constructed by stacking one-dimensional convolutions, a self-attention module as defined by [4] and a two-layer feedforward network. A LayerNorm [16] was applied before each of the set of convolutions, the self-attention module and feedforward network, with residual connections over each of these LayerNorm-sublayer blocks. Each convolution layer was designed with a kernel size of 5 and 200 output channels, with max-pooling-over-time performed to return to a 200-dimensional embedding following each convolution. We used four attention heads in the self-attention module and ReLU activations in the feedforward network. A final linear layer was used to map the output to the appropriate dimensionality (i.e., $\mathbb{R}^{400}$).

Deep Convolution-Transformer

It was found that this Convolution-Transformer resulted in nearly the same performance as the Bidirectional RNN implementation. As such, we attempted stacking the Convolution-Transformer layer three times to observe whether the increased depth would increase performance, however the benefit was marginal.

It was suspected that a much higher hidden dimension would be required to truly leverage the strengths of this architecture. However, we quickly found that larger dimensions would be beyond the computational and memory capabilities of our local machines or educational cloud computing solutions. Instead, we tried to combine the benefits of deep architectures and large hidden sizes by applying weight sharing ideas as in [17]. We took a single Convolution-Transformer and doubled the size of the hidden dimension, but applied it three times consecutively to create artificial depth in the Contextual Embed Layer. Unfortunately, we still found memory constraints to be limiting as this attempted solution did not lead to any noticeable benefits.

3.2.3 Attention Flow Layer

The Attention Flow Layer serves to refine the context embeddings with information from the question embeddings and is the first layer in the model where both context and question embeddings are jointly processed. For this we retain the Query2Context and Context2Query attention mechanisms proposed by [2] and implemented in the starter code. We found this element critical to the functionality of the QA system, with any other attempts (e.g., multiheaded self-attention, multiplicative attention, etc.) yielding significantly degraded performance and thus focus on possible improvements elsewhere. Each 200-dimensional input embedding passing through the Attention Flow Layer results in an 800-dimensional output.

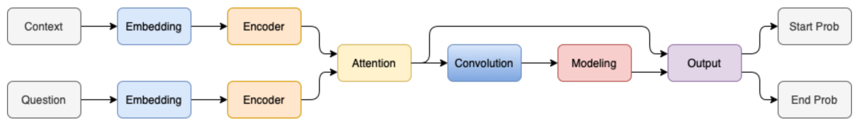

Figure 2: Our best model used a 1D convolution between the Attention Flow and Modeling Layers.

3.2.4 Convolution Layer

We added several convolution layers at various places within the model. The one that was most successful was after the Attention Flow Layer, shown in Figure 2. It is a 1D convolution with a kernel size of 5 and a ReLU activation. The idea of this convolution is to identify sequences of words that merit high attention scores in an attempt to find the answer. We also tried kernel sizes of 3 and 7, finding that 3 was approximately equivalent but 7 performed much worse. Note that the convolved attention is passed to the Modeling Layer but is not passed directly to the Output Layer.

3.2.5 Modeling Layer

The Modeling Layer processes the attention embeddings and serves as the final processing step before predictions are made. Similar to the Contextual Embed Layer we considered as our baseline a two-layer bidirectional RNN that would map each input embedding to a 200-dimensional hidden state vector. We also considered a Convolution-Transformer for this layer in the same way that we did for the Contextual Embed Layer, though we used 800 output channels in the convolution layers and a hidden dimension of 800 through the self-attention and feedforward portions of this architecture. We note that due to the larger dimensionality, we could not easily increase the depth within this layer given the aforementioned memory constraints. Attempts were made to reduce the dimensionality, though this severely degraded performance.

3.2.6 Output Layer

The default Output Layer applies a linear layer to the outputs of the Attention Flow and Modeling Layers to produce a vector containing the probability of each word in the context being the start of the answer. The output of the Modeling Layer is passed through an RNN and then another linear layer is applied to it and the Attention output to produce a similar vector for the end of the answer. The vectors contained a special index to indicate no answer. We ultimately stayed with that Output Layer but considered several modifications.

In particular, one idea came from basic probability. We realized that the start and end probabilities are not independent distributions. Among other factors, the end position cannot come before the start position. The provided discretization function handles this by excluding invalid combinations but we still wanted to have some form of a true joint or conditional distribution. Unfortunately, outputting such a distribution is extremely memory intensive and the method was intractable.

3.2.7 Training Loss

We used the negative log-likelihood training loss on the start and end probabilities in our final model but considered several augmentations. The main augmentation effort was an attempt to create a soft EM approximation using the start and end probabilities. It used the output probabilities and the provided correct start/end to estimate the probability of each word being in the answer. The loss added a penalty for assigning a lower probability to context positions that were in the answer and for assigning a higher probability for those not in the answer. This briefly showed some promise but was ultimately unsuccessful. We believe that estimating the joint distribution from the start and end distributions is simply not accurate enough to be useful. Had we been able to predict the full joint distribution, this loss would probably have performed well.

3.3 Other Attempted Models

We have also implemented a purely Transformer-based model from the ground up. The motivation for doing so was to gain familiarity in constructing models from scratch as well as the intricacies of working with transformers. This model was designed to take fixed, pretrained GloVe [14] word embeddings as input and produce start and end word distributions over context positions. This model was built following the simplest possible approach and was designed with an encoder-decoder structure. The encoder jointly encoded the context and question embeddings, with a separation token introduced between them. We implemented the encoder by stacking four vanilla Transformer encoder layers as described by [4] with sinusoidal positional encodings. The decoder was implemented as two separate context-to-query multiplicative attention modules, one corresponding start index and the other to end index of the answer. In particular, each attention module was responsible for computing a matrix of attention scores between the encoding for each context word i, and question word j. We sum the attention scores for each context word (i.e., over all question words) and pass them through a linear layer to obtain a vector of logits and pass them through a softmax to predict the start or end index. We show the result of this model in Table 1 (denoted “Vanilla Transformer + Multiplicativee Attention Decoder"), however we state here that it did not perform competitively with the baseline. We suspect that additional tuning and the introduction of additional structure may be required to render this model performant, which were deferred in favor of our aforementioned investigations as those proved more fruitful.

3.4 Pretraining

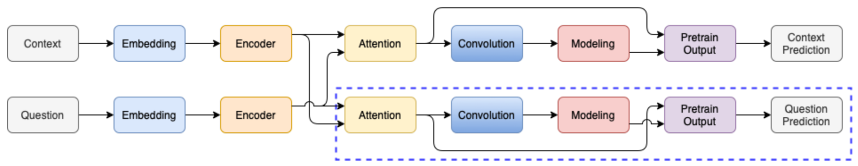

Figure 3: This is the model structure during pretraining. The output layer was changed to reconstruct the non-corrupted input. The components inside the blue rectangle swap the context and question in order to reconstruct the question, though that section was not used in the best model.

We implemented a pretraining approach that attemped to reconstruct corrupted contexts and questions. Before being fed into the model, each context had a sequence of 1 to 5 words set to random word/character indices. Note that the word and character representations had the same positions corrupted but the corrupted words and characters were generated separately. We did this instead of adding special corruption tokens with the idea that the model should be able to figure out which words are contextually out of place and fill in reasonable words. The output layer for pretraining used an adaptive softmax [18] to predict the probability of each word being at each position in the context. We used an adaptive softmax because it can efficiently handle the large number of words that need to be assigned probabilities. All layers except for the output layer were shared between the pretraining and task-specific training phases.

We also tried a similar pretraining strategy that corrupted the questions in the same way as the contexts. This required a modified forward pass, shown in the blue box of Figure 3, that swapped the context and question in a parallel forward pass. However, we found that excluding question corruption performed slightly better. Other pretraining efforts included predicting word vectors and highlighting segments of the text, but those efforts were unsuccessful in improving the final result.

4 Experiments

4.1 Data

We use a modified form of the SQuAD 2.0 dataset [1] provided by the Stanford CS224N teaching staff, which we note differs from the official SQuAD 2.0 dataset [1]. The dataset consists of paragraphs from Wikipedia, with the questions and answers obtained via crowdsourcing through Amazon Mechanical Turk. In particular, the data we will work with has a train, dev and test split. The train split consists of 129,941 examples, the dev split consists of 5,951 examples and the test split consists of 5,915 examples. An example consists of a question, context paragraph, and answer triple. A question and context paragraph are represented as lists of words while an answer consists of (potentially multiple viable) start and end word indices of the correct response. We represent words using pretrained 300-dimensional GloVe [14] embeddings which are held fixed throughout training.

4.2 Evaluation method

To evaluate our models, we are reporting the F1 and EM performance metrics on the SQuAD 2.0 development dataset. Our most promising models were uploaded to the test leaderboards to evaluate their performance on the test dataset. We primarily compare the performance of our models against the Stanford CS224N teaching staff provided implementation of the BiDAF model. This implementation largely resembles the BiDAF model as presented in [2], with the notable exclusion of character embeddings in the input.

4.3 Experimental details

4.3.1 Training

Training was performed for 30 epochs over the training set of 129,941 examples with a batch size of 64. Models were trained locally on two machines respectively equipped with an 11 GB RTX 2080 Ti and an 8 GB RTX 3070, each possessing at least 32 GB of RAM. Training times were roughly equal across both machines with a complete training run averaging 3.5 hours with low spread.

We found that training past 30 epochs did not yield improvements. Several experiments ran for longer but the performance on the development set always plateaued before 30 epochs had passed, so only the most promising models were allowed to continue for the sake of not wasting resources.

Pretraining slightly improved training speed once we switched to task-specific training. However, the overall training time was essentially unchanged once the time spent pretraining was considered. This is demonstrated in Figure 4, where the solid red line represents the pretrained model's progress on task-specific training and the dotted red line is shifted right by the number of pretraining iterations completed prior. Larger versions of the Figure 4 plots are in the Appendix.

4.3.2 Tuning

We kept the learning rate at the default 0.5 for most experiments. We did try changing the learning rate as training went on in a few experiments but results were mixed and insignificant. The best model we found used a learning rate of 0.1 for the final 50,000 iterations to get a small improvement, but continuing after that, along with several other similar attempts, led to worse performance.

4.4 Results

We achieved an F1 score of 63.936 and and EM score of 60.372 on the test dataset. Our development set results from several model variations are shown in Table 1. We were able to beat the baseline with several of our model variations.

Figure 4 shows the progress of AvNA, EM, and F1 on the development set during training for four selected models. In all of our relatively successful experiments, we noticed fast improvements for approximately 10 epochs followed by slower progress and then a plateau. This is the expected behavior. However, we found that many of our model variations ended with similar performance, which appears to indicate that a fundamental model change would be required to boost it higher. One particular surprise was that combining the Transformer Contextual Embed Layer and the Post-Attention Convolution Layer hurt performance, despite each giving improvements individually.

5 Analysis

5.1 General Observations

Using character embedding was the single largest improvement we saw when experimenting with various changes. This enabled the model to better reason about numbers, symbols, or words that were poorly represented in the training data. This is further discussed in Section 5.2 with examples.

Contrary to expectations, replacing LSTM layers with Transformers appeared to show equivalent or at times slightly degraded QA performance. We hypothesize that Transformers require much larger hidden sizes and deeper architectures to see their full benefit (e.g., $\text{BERT}_{\text{LARGE}}$ [12] uses 24 layers with a hidden dimension of 1024 and 16 attention heads). Unfortunately, our local machines

Model	**AvNA**	**F1**	**EM**
Expected Baseline from Handout	65	58	55
Baseline	68.46	61.39	57.81
Vanilla Transformer + Multiplicative Attention Decoder	52.76	51.25	51.36
Baseline + Character	70.02	63.63	60.04
Baseline + Character + Transformer Contextual Embed Layer	69.82	63.77	60.36
Baseline + Character + Transformer Modeling Layer	67.42	60.37	57.12
Baseline + Post-Attention Convolution	67.01	60.13	56.78
Baseline + Character + Post-Attention Convolution	71.1	64.39	60.66
Baseline + Character + Transformer Contextual Embed + P-A Conv	69.84	63.75	60.14
Baseline + Character + P-A Convolution + Pretrain	**71.48**	**64.97**	**61.55**

Table 1: Selected results on the development dataset. The model labels show which combination of components were used in each experiment. Our best model is bolded.

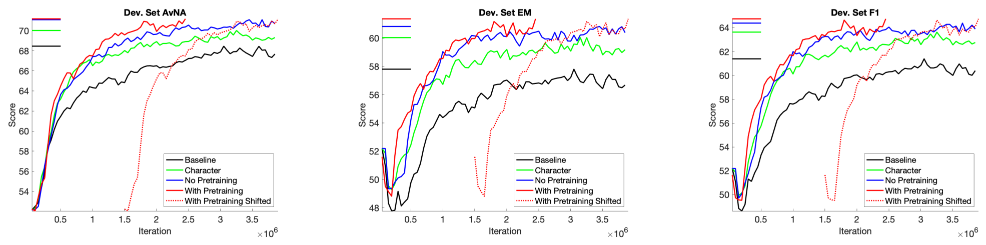

Figure 4: AvNA, EM, and F1 scores on the dev set during training. The line on the left edge of each plot shows the maximum achieved value for the corresponding curve. “Character” refers to the baseline+character embedding model. “No pretraining” refers to our best model, the last line in Table 1, without pretraining. “With pretraining” refers to that model after pretraining.

and educational cloud computation solutions memory constraints were much too limiting for a Transformer model of appreciable size.

We added convolutions at several places in the model and found that their effect was highly variable depending on the location. If a convolution was placed before the Attention Flow Layer, performance dropped. This was likely due to the convolution smearing information across groups of words at the expense of per-word information. When this happens too early, the rest of the model can lose the ability to look at individual words. However, a convolution between the Attention Flow and Modeling Layers proved helpful, specifically with a kernel size of 3 or 5. This convolution found groups of words that had been identified as important. A kernel size of 3 or 5 working better than 7 is also understandable because many answers are 5 words or fewer, so 7 is looking at too large of a window in most cases.

One interesting behavior was that having a direct line from the Attention Flow Layer to the Output Layer, skipping the convolution, was helpful. If the convolved attention was sent to the output, the performance got much worse. This can be explained with the same reasoning as the earlier convolutions, in that losing per-word information is harmful. Modeling word windows helps but since the output is concerned with individual words, it cannot lose that information.

5.2 Examples of Successes and Failures

Several informative results are shown in Table 2. These demonstrate the importance of character embedding and show some interesting behaviors of our model.

The first two examples clearly show the benefit of character embedding. It is unlikely that “CD40” or “90°” were well represented in the training data. However, similar chemical or angle terms were likely there enough to figure out that these words relate to those ideas, respectively. The second example also demonstrates long-range context understanding, with an 11 word gap between relevant pieces of the context. The third example shows a case where character embedding was not enough. There are many symbols in a row where the letters are acronyms and the other symbols are obscure, which meant that the model was unable to understand the context and find the correct answer.

Question	Relevant Part of Context	Answer
What is a ligand on the cell surface that is upregulated after helper T cell activation?	... helper T cell activation causes an upregulation of molecules expressed on the T cell's surface, such as CD40 ligand...	CD40 ligand
At what angle were the groups of pistons set in relation to one another in a 4 cylinder compound?	...producing a 4-cylinder compound, ...[11 words]... the groups being set at $90°$ to each other...	90
What is an expression that can be used to illustrate the suspected inequality of complexity classes?	Many known complexity classes are suspected to be unequal, but this has not been proved. For instance $P \subseteq NP \subseteq PP \subseteq PSPACE$,...	$P \subseteq NP \subseteq PP \subseteq PSPACE$
What year did a toxic waste spill from an American ship prompt the Commission to look into legislation against waste?	In 2006, a toxic waste spill off the coast of Côte d'Ivoire, from a European ship, prompted the Commission to look into legislation against toxic waste. ...	N/A
What sort of motion did Newcomen's steam engine continuously produce?	... James Watt patented a steam engine that produced continuous rotary motion. ...	N/A
What King and former Huguenot looked out for the welfare of the group?	... following the death of Henry IV, a Huguenot before converting to Catholicism, who had protected Protestants through the Edict of Nantes. His successor Louis XIII, under the regency...	Henry IV

Table 2: Informative examples seen in development set while training. The model's prediction is highlighted in the context. For the example with no highlighted answer, the model predicted N/A.

The fourth and fifth examples demonstrate partial understanding. In answering the "what year" question, the model recognizes that it needs to identify a number. Indeed, it identifies the year corresponding to the toxic waste spill. In the "what sort of motion" question, the model correctly identifies "rotary motion" as relevant to the motion of a steam engine. However, in both cases, the model did not identify the correct ownership. In the former, the ship was European but the question asked about an American ship. In the latter, Newcomen had a steam engine but it was Watt's that produced continuous rotary motion. We were able to verify that misunderstanding ownership was the issue in the engine example because the development set had the same question except for swapping out Newcomen for Watt. In that case, the model gave the same answer, which was correct. This issue of missing modifiers, such as those indicating possession, showed up in many of the incorrect answers that we found.

The sixth example, asking about King Henry IV, is an impressive success by the model. In many question/context pairs, the question words are almost exactly in the context and the answer is adjacent to those words. In this pair, we see several features that provide evidence of a true understanding of the language. First, the question identifies Henry IV as a "former Huguenot" from "a Huguenot before converting" which indicates an understanding that "converting" is a state change and connects "former" to "before." Second, the question says "looked out for the welfare of the group" but the context says "protected the Protestants," meaning that the model identified "the Protestants" as the relevant group and recognized that "looked out for the welfare of" and "protected" are the same in this context. Third, the context never directly calls Henry IV a King, yet the model identifies him as one anyway. One clue to that is the "IV" in his name. The other is given at the end of the identified section of the context, which states that he had a successor with a regent relationship, like a King might have. Altogether, answering this question correctly demonstrated the model's ability to understand language rather than simply find words from the question in the context.

6 Conclusion

In this work, we analyzed the effect of different modifications to the BiDAF architecture. We found the Attention Flow Layer to be critical to the functionality of the model with the absence of this component preventing meaningful training. To improve upon the provided BiDAF implementation, we found that subword modeling provides a significant performance boost, and the benefit was shown for questions pertaining to specialized terminology, numerical entities, or obscure words. Contrary to expectations, replacing LSTM layers with Transformers appeared to show equivalent or at times slightly degraded QA performance. We hypothesize that this is due to the limited size of Transformer models we were able to train on our available hardware. However, we did find the convolution layers introduced after the Attention Flow Layer helped aggregate local context information, though introducing convolutions before it appears to smear information where per-word information seems important. Finally, we also found that pretraining yielded a minor performance improvement, though we suspect a larger benefit should be seen with a larger unlabeled data set.

As future work, we believe that one possible avenue of improvement would be to condition the answer end index probability on each start index probability since this would give probabilities for each possible window instead of treating the start and end points as independent entities. In particular, we believe that this would help with incorrect answer vs. no answer decisions that come from keeping the probabilities separate.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

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

[3] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do rnn and lstm have long memory? In *International Conference on Machine Learning*, pages 11365–11375. PMLR, 2020.

[4] 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*, 30, 2017.

[5] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 193–203, 2013.

[6] Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D Manning. Modeling biological processes for reading comprehension. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1499–1510, 2014.

[7] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*, pages 2358–2367. Association for Computational Linguistics (ACL), 2016.

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

[9] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. 2016.

[10] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

[11] 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. *arXiv preprint arXiv:1804.09541*, 2018.

[12] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

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

[16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[17] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[18] Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, et al. Efficient softmax approximation for gpus. In *International conference on machine learning*, pages 1302–1310. PMLR, 2017.

A Appendix

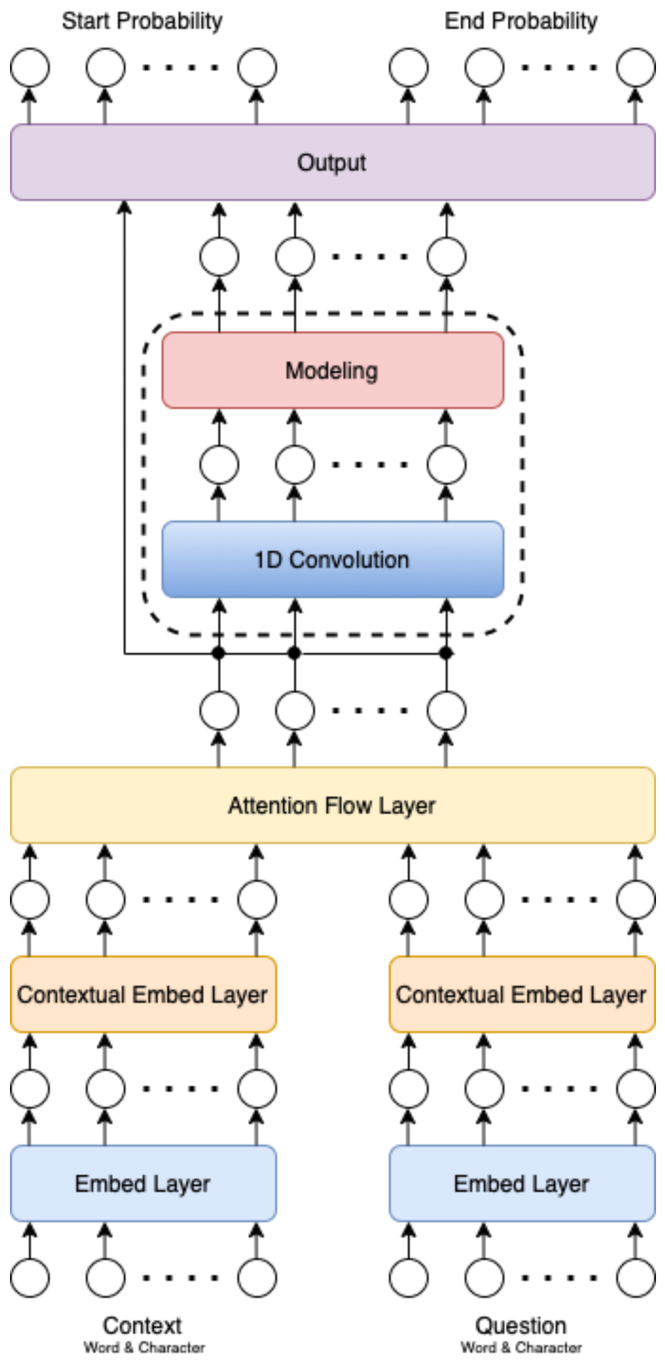

Figure 5: A larger, more detailed diagram of our model.

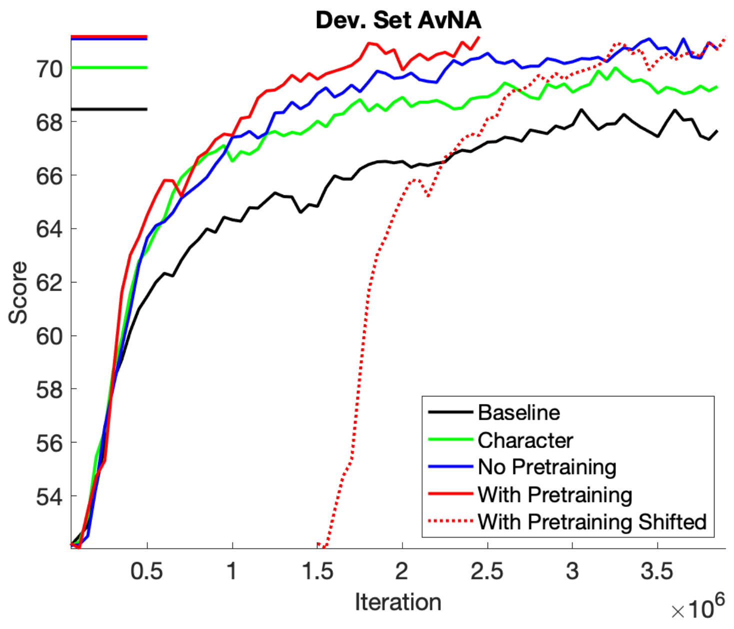

Figure 6: Larger version of AvNA from 4.

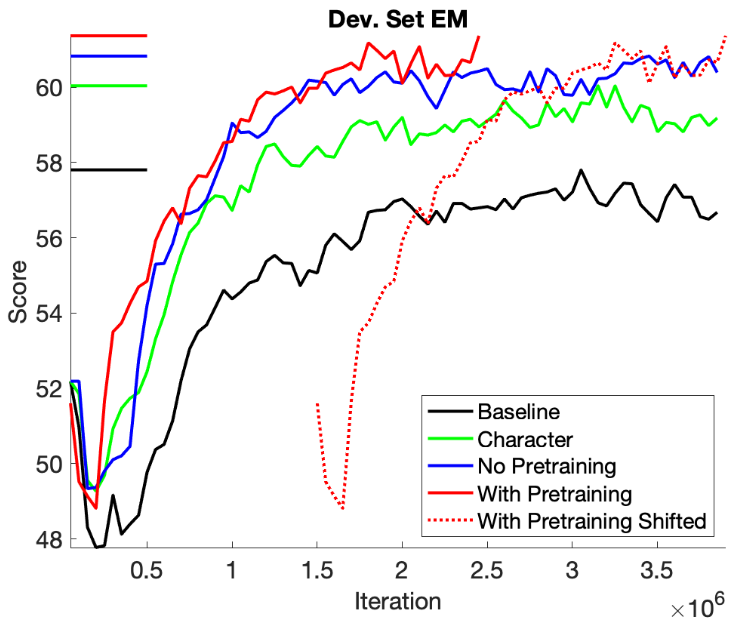

Figure 7: Larger version of EM from 4.

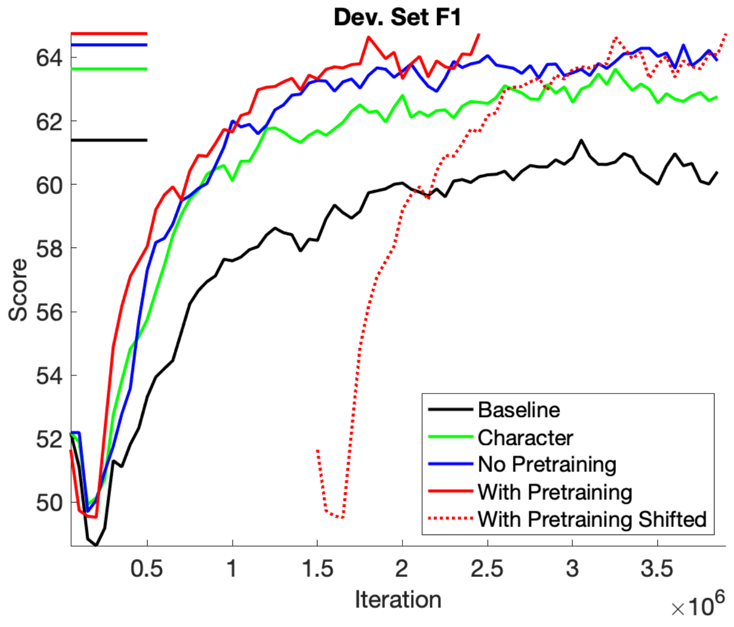

Figure 8: Larger version of F1 from 4.