Bi-Directional Attention and Beyond: Double BiDAF with Residual Connections for Question Answering

Pedro M. Milani
Mechanical Engineering Department
Stanford University
Stanford, CA 94305
pmmilani@stanford.edu

Abstract

In the present work, I implement and test a deep learning system to perform a question answering task. This work is based on the popular SQuAD dataset [11], with over 100k examples. The model I used is based on the bi-directional attention flow proposed by Seo et al. [12], with some additions described in section 3. An ensemble of my final model obtained 75.20% F1 score and 65.36% EM score in the test set, putting me (at submission time) in the top 25 scores in the class (I assume there are over 100 teams). Section 4 presents some qualitative analysis of the current model, especially of character-level embedding.

1 Introduction

Deep learning techniques have revolutionized the field of Natural Language Processing (NLP) in the past few years. Previous state-of-the-art results quickly became obsolete across the board, from machine translation to dependency parsing. In the present work, I focus on applying deep learning to the task of question answering: given some context and a question, the model should be able to generate an answer. The dataset of choice for this project is the *Stanford Question Answering Dataset* (SQuAD) [11].

SQuAD was released in 2016 and is a popular dataset for question answering. It contains about 107k context-question-answer triples, randomly split into a training set (80%), a development set (10%), and a (secret) test set (10%). The contexts were obtained from sampling paragraphs in high-quality Wikipedia pages; the questions and answers were obtained by human annotators, through a crowd-sourcing effort [11]. The answers are a continuous sequence of words from the respective contexts, and each context has three answers (generated independently by three distinct human annotators). The official evaluation metrics compute the maximum accuracy over each of the 3 potential answers, with all lowercase text, and discard articles at either end. An example context, question, and answer triple from the development set is shown below:

Context: As of the 2010 United States Census, Southern California has a population of 22,680,010. Despite a reputation for high growth rates, Southern California's rate grew less than the state average of 10.0% in the 2000s as California's growth became concentrated in the Northern part of the state due to a stronger, tech-oriented economy in the Bay Area and an emerging Greater Sacramento region.
Question: Southern California had a population of 22,680,010 according to the census from which year?
Answer: 2010

For this task, the model should obtain the same answer as the one provided by the human annotator. The two evaluation metrics reported in the literature are the EM score (exact match), which is 1.0 for a correct answer and 0.0 for an incorrect one (however minor the mistake); and the F1 score, the

harmonic mean of precision and recall between the true and predicted answers. The average metrics in the development and test set are what I am trying to maximize.

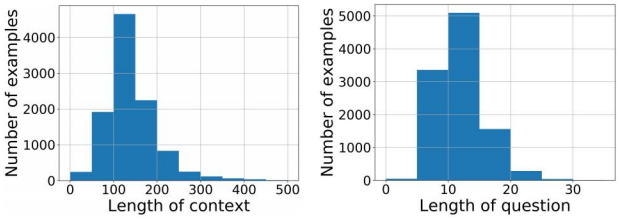

Figure 1: Histograms of context and question lengths

An initial analysis of the dataset shows that the contexts in the development (dev) set are on average 142 tokens long (each token is usually a words, but can also be punctuation, numbers, etc). Questions are on average 11 tokens long. Figure 1 shows the distribution of context/question lengths in the SQuAD dev set.

2 Related Work

SQuAD is publicly available and the authors maintain a leaderboard [1] to which researchers can submit their models to be evaluated on the test set. Rajpurkar et al. report a human accuracy of 91.22% F1 score and 82.04% EM on the test set [11]. The current best model, as of March 2018, is unpublished and authored by Joint Laboratory of HIT and iFLYTEK Research: it achieves 89.28% F1 score. Other high-performing, but more established models, include the Bi-Directional Attention Flow (BiDAF) from Seo et al. [12] (who report test F1 scores as high as 81.1%) and Wang et al.'s RNET [15], with F1 scores of up to 82.9%.

Of course, I cannot compete with such state-of-the-art numbers in the scope of a class project - I am working alone, and have only a few weeks to familiarize myself with the problem and design a model to solve the task. So, the work from students who took the class last year serves as a useful baseline of what I should be able to achieve. The top performing models in CS-224N in 2017 were that of Budianto [3], with an F1 score of 74.52% for a single model and 77.47% for an ensemble, and of Ke et al. [7], with an F1 score of 76.5% for an ensemble. Both of them based their implementations on the paper of Seo et al. [12], but differ slightly in their details.

In general, most models mentioned above are structured as follows: first, they use a Recurrent Neural Network (RNN) to process the embedded version of the context and question tokens separately; then, they combine them through some form of attention (producing a question-aware context embedding); and then use another RNN-based model to process the question-aware context embeddings. Finally, a softmax classifier is used to pick the index of the start and end token of the question in the context. From previous work, it seems that a simple model implemented correctly to do this would be able to obtain dev F1 score in the high-60's range [3]. From then on, each relevant modeling improvement built on top of this basic architecture seems to buy a few extra points in F1 score.

3 Approach

As a tried-and-tested approach for the scope of CS-224N, I decided to base my implementation on the generic architecture described in the last section, and focus particularly on implementing a model similar to BiDAF, presented by Seo et al. [12]. There are two new features that I add, when compared to Seo et al's original implementation. One is a **double BiDAF** instead of a single BiDAF in the attention stage; the second is **residual connections** whenever two layers are stacked on top of each other. Those will be explained in detail, together with the general architecture, in the following subsection.

[1] https://rajpurkar.github.io/SQuAD-explorer/

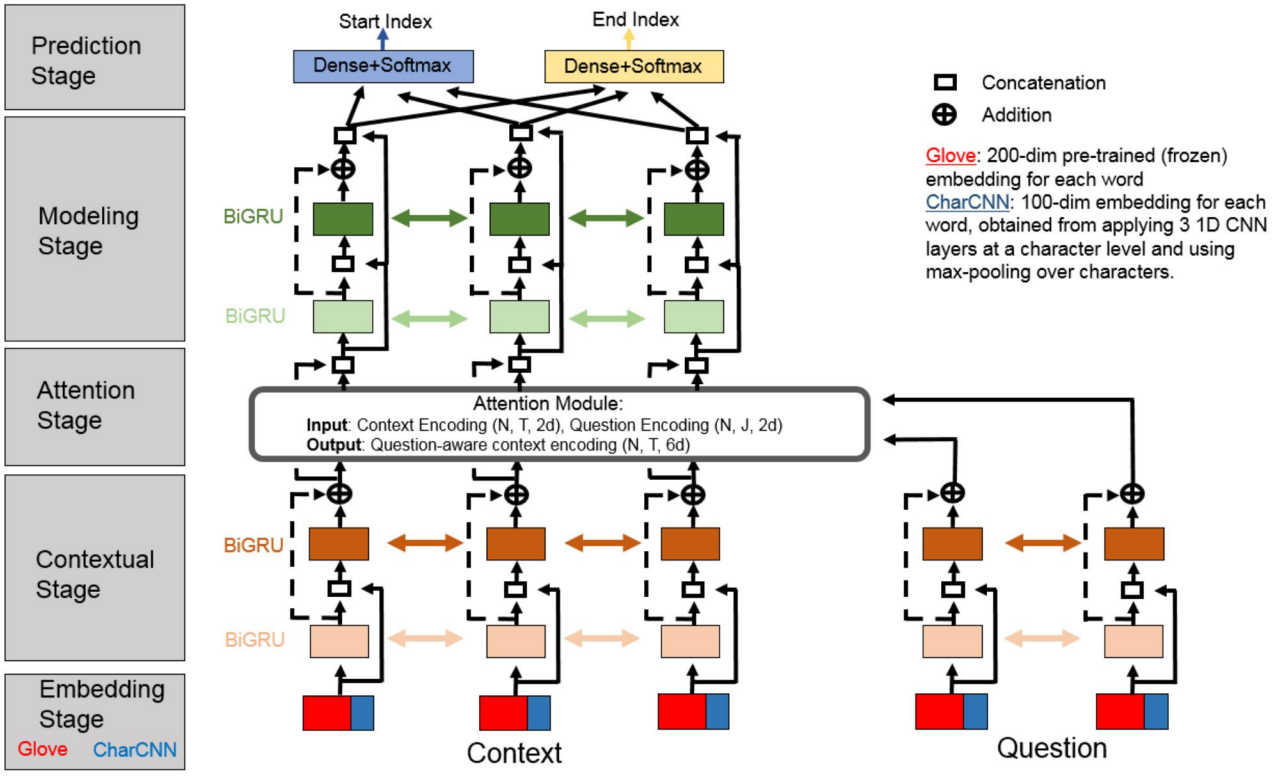

Figure 2: My final model architecture. Note that residual connections are denoted by dashed lines. The attention module is a placeholder for any attention desired. In the schematic, N is the batch size, T is the context length (3 in the diagram), J is the question length (2 in the diagram), and d is the hidden state size of the bi-directional GRUs.

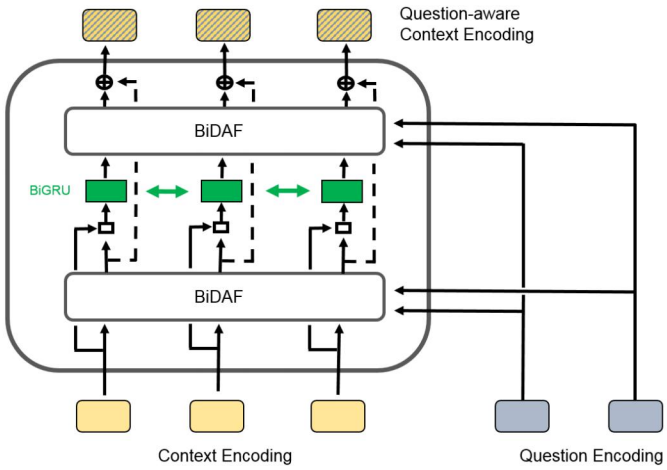

Figure 3: My final attention architecture. This shows how I stacked two BiDAF modules on top of each other. Residual connections are again showed by dashed lines. The bi-directional GRU has the same hidden state size d.

3.1 Final Model

My final architecture is represented schematically in Fig. 2, which contains a general attention placeholder, and Fig.3 which contains the double BiDAF attention module I implemented. The design is modular, and the context and questions go through the following sequential stages:

Embedding Stage

In this stage, the objective is to take each token from the context and the question and convert it to a vector of size D_{emb}. In the final architecture, we use pre-trained GloVe 200D embeddings [10] (which are kept fixed during training), concatenated to the output of a 1D convolutional neural network (CNN) applied on the character level [8].

For the character-level CNN, we start with an 50-dimensional embedding for each character (initialized at random and trained), then apply 3 consecutive 1D CNN layers of filter-width 3. I chose this design (rather than Seo's single CNN layer of width 5) because there is some evidence in the computer vision community, where CNNs are widely employed, that it is better to stack multiple small filters than use fewer, bigger filters [14]. In all three CNN layers, a ReLU non-linearity is used, and there are 100 filters. Then, the maximum activation over the characters is used at the end to convert it into a 100-dimensional embedding for each word. Concatenated with GloVe 200D vectors, this makes $D_{emb} = 300$.

Contextual Stage

This stage receives a sequence of D_{emb} dimensional vectors for each token in the context and question, and processes them with two layers of a bi-directional RNN (the gated recurrent unit - GRU - is employed [4]). For computational speed, it is desirable to assume all contexts and all questions have the same length, so the computations can be paralellized more easily. Using the histograms of Fig.1, I decided to set the context length at $T = 400$ and the question length at $J = 30$. Smaller context/questions are padded and the computations are masked; longer sequences (less than 1% of the total) are discarded at training time and truncated at test time. Note that the weights are shared between the context and the question - I found that this speeds up training and in general produces slightly better results. The GRU have hidden length of dimension $d = 100$, and the forward/backward hidden states are concatenated as the output.

Note that the contextual stage employs residual connections to connect the two layers of the bi-directional GRU (denoted by dashed lines in Fig. 2). Residual connections were introduced in the computer vision context [5]. Their main idea is to add the activations of layers that are used sequentially to allow the gradients to backpropagate more easily through deep architectures (since gradients flow in an unmodified way through additive gates). The intuition behind it is that the second layer is not trying to predict the correct activations from scratch - it is trying to predict the difference (or *residual*) between the activations of the first layer and the correct activations. I decided to add those connections because as I stacked more layers, particularly the double BiDAF, I noticed a reduction in the overall gradient magnitude during training. Thus, I conjecture that residual connections would help training this architecture.

Attention Stage

This stage is responsible for transforming context/question encodings (respectively of dimension $T \times 2d$ and $J \times 2d$) into a single context encoding, but that contains information from the question. So, the output of this layer is a sequence of length T. In the BiDAF paper, the attention encoding has dimension $8d$ for each word: $6d$ from the attention manipulations, that are subsequently concatenated with the $2d$ context encoding from the contextual stage. For the double BiDAF implementation, the output dimension is exactly the same. Fig. 3 shows how the double BiDAF is implemented. The rationale for it is that, as deep learning research has often shown, more layers usually improve accuracy metrics (especially with residual connections). Besides, I conjecture that one of BiDAF's short-coming is that the question encoding has little impact in the answer prediction (it only enters the flow once, through the BiDAF module). My double-BiDAF implementation would double the number of layers which take input from the question encoding. The reader is referred to Seo et al. [12] for the exact implementation of the BiDAF module.

Modeling Stage

Here, the input is a sequence of length T of question-aware context embeddings (dimension $8d$). They are manipulated through a two-layer bi-directional GRU of hidden dimension $d = 100$, and then concatenated to the original input. So, the output is of dimension $T \times 10d$. Note that the general architecture here is very similar to the contextual layer.

Prediction Stage

The prediction stage is simple: each of the T encodings of dimension $10d$ passes through a fully connected layer followed by a ReLU non-linearity that reduces them to dimension $D_{pred} = 150$. Then, two independent linear layers followed by a softmax layer predict the probability that each particular element of the sequence is the start/end token of the answer. At training time, a cross-entropy loss is used between the predicted probability and the ground truth start/end locations (for simplicity, I only used the first human answer in SQuAD to train).

4 Results

The architecture described in the previous section was implemented in Python using Tensorflow [2], using the infrastructure given to us as the project starter code [1]. To regularize the model, I used dropout [13] in every layer, with a drop probability of 0.2 and L2 regularization with strength 0.001. The batch size was 48 for the final model, and 64 for smaller models (i.e., which did not contain all modules of the final model). I used the Adam optimizer, with learning rate 0.0008 for batch size of 48, and 0.0015 for batch size of 64, and trained the models for about 20 epochs. The hyper-parameters were tuned by running a few epochs on a subsampled training/dev set. Even though this approach assists one in getting a rough idea for best hyper-parameters, it is not ideal (since running on a bigger dataset might require different hyper-parameters). I am confident that I am using reasonable hyper-parameters, but there might be better combinations. If I had much more time, I would choose hyper-parameters by running around 5 epochs on the full dataset across a wide range of hyper-parameter combinations.

4.1 Accuracy Metrics

Table 1 shows the relevant accuracy metrics in the dev and test sets of 10 selected models. The first 5 are from previous work and serve as comparisons to my own results, in the last 5 rows. Note that the best F1 score I obtained on the test set was 75.20%.

In Table 1, final model corresponds to the architecture described in section 3. The ensemble is obtained by loading the weights of 6 different versions of the model, saved at different snapshots of the same training run, and then averaging the predicted probabilities for start/end tokens. Single BiDAF corresponds to the architecture of Fig. 2 but with BiDAF used as the attention module (instead of the double BiDAF). Single BiDAF, no Char CNN is a yet simpler model, where no character level embedding is used (i.e., only a GloVe 200D vector is used to embed each token in the embedding stage). Finally, the publicly released Baseline code [1] (whose results I added in the last row) represent GloVe 100D embedding, a single bi-GRU layer in the contextual stage, simple dot product attention, and no modeling stage.

From the results in the table, it is clear that my models achieved comparable performance to last year's best. My single model beats Budianto's, but my ensemble is worse than his. In the literature, model ensembles usually obtain 2-3 $\%$ higher F1 score than the component single models, but mine only improved results by less than 1%. Maybe averaging probabilities is not the best way of obtaining the ensemble predictions in this problem - summing logits or a voting strategy (the latter was used by Budianto [3]) might be better alternatives. My single BiDAF model fell short of the original implementation by about 4% in F1 score. Some of the differences that might account for that include the fact the Seo et al. used LSTM cells [6] instead of GRU cells, and had a more elaborate prediction state where the end token prediction depends on the start token.

Comparing my different models, it seems that the double BiDAF attention managed to improve performance in the dev set by about 1% in F1 and EM. Using character level embedding also proved useful: it improves performance by about 2% in the single BiDAF implementation (and the same

Table 1: Comparison of accuracy metrics in SQuAD. The first five rows correspond to previous work. The last five are from the current work. The bold line shows my best results.

Model	Dev Set		Test Set	
	F1	**EM**	**F1**	**EM**
Human Performance [11]	–	–	91.22	82.30
Original BiDAF - Ensemble [12]	80.7	72.6	81.1	73.3
Original BiDAF - Single Model [12]	77.3	67.7	77.3	68.0
Budianto - Ensemble [3]	77.71	69.03	77.47	68.49
Budianto - Single Model [3]	74.04	63.92	74.52	64.26
My Final Model - Ensemble	**75.22**	**64.65**	**75.20**	**65.36**
My Final Model	74.41	63.82	–	–
Single BiDAF	73.62	62.96	73.72	63.68
Single BiDAF, no Char CNN	71.49	60.54	–	–
Baseline Code	43.14	34.15	–	–

level of improvement is obtained when adding the character-level embedding to the raw baseline model). Finally, all my models represent a significant improvement over the baseline results. The main responsible for this is the addition of the modeling layer: even a single bi-directional RNN layer between the attention and prediction stages increased the F1 metric by over 20%.

4.2 Character Embedding Analysis

In the current subsection, I attempt to analyze the character-level embedding used in the embedding stage. I chose to represent by an embedded vector every character that appears more than 0.01% of the times in the training set - that amounted to 57 characters, including all letters of the alphabet, all digits, punctuation, and some foreign characters. If an out-of-vocabulary character appears (i.e., one that is not part of the 57 most common), I assign it an embedding corresponding to `<unk>` (which is also trainable). Also, when mapping tokens to characters, each token is assumed to have the same length - longer tokens are truncated, shorter ones are padded. I chose a maximum token length of 16, which is longer than 95% of all tokens in the context training set. For padding, another special character embedding is used, `<pad>`. Overall, the embedding matrix is of shape 59×50, since each character is embedded as a 50D vector. When applying 1D convolutions, it is important to mask the words appropriately (so activations in locations where we originally had a `<pad>` character are not considered). The way I did it is to add a very large negative number to the final activations relative to padded locations right before taking the maximum over characters - that way, those will never be part of the final $100D$ character-level embedding for that token. From early experiments, the character-level embedding does not produce any gains in F1 score when compared to just GloVe embedding if masking is not handled appropriately.

Fig. 4 shows a visualization of the trained character embeddings in the single BiDAF model. The dimensionality reduction technique called t-SNE (t-Distributed Stochastic Neighbor Embedding [9]) is used to reduce the dimension from 50D to 2D, so that they can be visualized in a plot. Note that to obtain a meaningful 2D encoding, it is necessary to use low perplexity when running the t-SNE algorithm [9], since there are few overall points (only 59, one for each character). Fig. 4 was produced with a perplexity of 2. It shows some interesting patterns, and intuitively helps to explain why character-level token embedding helps in obtaining better accuracy in the current task.

Note that all the 10 digits are clustered together in the center-right of the plot, so the algorithm learned that they all have similar meanings. This is probably very helpful to deal with numbers. For example, if the context contains the number 224 but that particular number is not part of the tokens contained in the pre-trained GloVe embeddings (which is probably true of most numbers), the character-level embedding will correctly identify it as being similar to some other number (e.g., 229), but dissimilar to a rare word not in the GloVe vocabulary (for example, my middle name "Montebello") . A purely GloVe-based encoding would fail to recognize this, since all three tokens (224, 229, Montebello) would be encoded as `<unk>`. Many punctuation characters (such as `( ) : ; \`) are also close-by in the 50D space, which again shows that the trained model learned to recognize that these characters have similar meaning. One interesting observation is that the letter `d` is isolated from the other characters in the bottom-left of the plot; it is unclear whether this is a

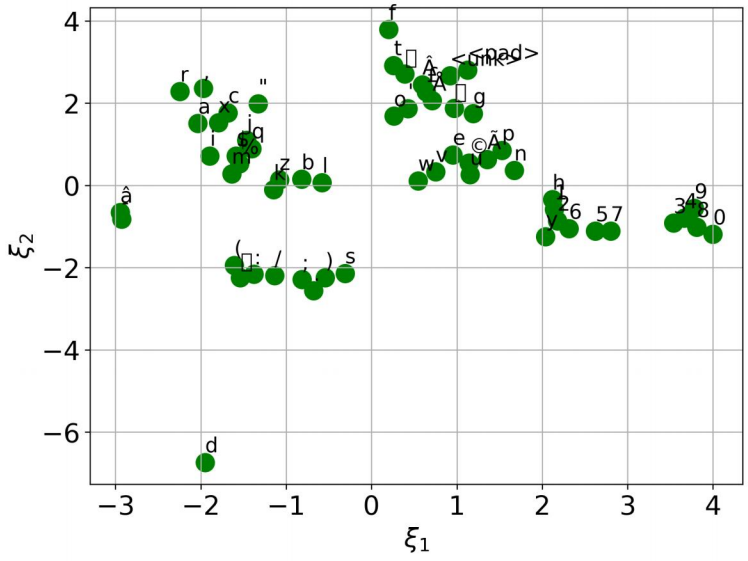

Figure 4: Trained character embeddings after applying t-SNE to reduce them from 50D to 2D. Axis are the two t-SNE coordinates.

true, unintuitive pattern in the English language or whether it is an artifact of poor training or poor t-SNE encoding.

Another interesting experiment I performed was to verify the intuitive but vague claim that several authors make (e.g. [12, 15]) that character-level embeddings "have been shown to be helpful to deal with out-of-vocab (OOV) tokens" [15]. I used the context-question pairs in the development set[2] and split it into two sets: those where at least one unique token appears, and those where no unique tokens appear (a unique token is defined as one that is outside of the 400k vocabulary of the GloVe embeddings [10]). 63% of the examples fell in the first group (at least one unique token), and the remaining 37% are in the other group. Then, I used the single BiDAF model with and without character-level embedding in both sets, and compared their F1 and EM scores. The results are shown in Table 2. As we can see from the last row (showing the improvement in score when character-level embedding is employed), adding character embeddings improves predictions across the board. However, as suggested in literature, this improvement is indeed more significant in the set of context-question pairs with unique tokens. This supports the intuition that character-level embedding helps the model interpret out-of-vocabulary words more appropriately.

Table 2: Comparison of accuracy metrics in two subsets of the pre-processed dev set: one with at least one unique token, and one without unique tokens.

Model	Unique Tokens		No Unique Tokens	
	F1	**EM**	**F1**	**EM**
Single BiDAF, no Char CNN	65.37	49.76	67.82	52.53
Single BiDAF	67.84	52.33	69.40	54.67
Improvement	**2.47**	**2.57**	**1.58**	**2.14**

4.3 Error Analysis

Finally, in this subsection I briefly analyze the error modes of my final model. Like previous authors have reported, most errors made by my final model are due to imprecise boundaries in the answer: the model predicts an answer that either misses or adds one or two tokens at the beginning/end of

[2]This is not the full, official dev set; it is the one we used at training time, with just one of the human answers. That is why the accuracy metrics are a little worse than the ones from Table 1

the correct answer (obtaining 0 EM and a non-zero, smaller than one F1 score vfor that example). This is also the main mode of human disagreement ([11]). For sake of brevity, I will not present any such examples here.

Instead, I will highlight one error mode that, whereas not frequent in the dataset, shows how hard it is to obtain human-like accuracies in SQuAD: questions that require previous world knowledge to answer. These are questions which, just by reading the context, it would be impossible to get right. An example is shown below:

Context: 20th Century Fox, Lionsgate, Paramount Pictures, Universal Studios and Walt Disney Studios paid for movie trailers to be aired during the Super Bowl. Fox paid for Deadpool, X-Men: Apocalypse, Independence Day: Resurgence and Eddie the Eagle, Lionsgate paid for Gods of Egypt, Paramount paid for Teenage Mutant Ninja Turtles: Out of the Shadows and 10 Cloverfield Lane, Universal paid for The Secret Life of Pets and the debut trailer for Jason Bourne and Disney paid for Captain America: Civil War, The Jungle Book and Alice Through the Looking Glass. [citation needed]
Question: What famous July Fourth holiday movie did Fox pay to advertise a sequel of during the Super Bowl?
True Answer: Independence Day
Predicted Answer: Walt Disney Studios

From the context, it is clear that Fox paid to advertise four movies. But the question asks for the name of a prequel of one of the four advertised movies, and the prequel is a "July Fourth holiday movie". Humans will be able to identify Independence Day if they know what the movie is about. A model that only has access to this context would never be able to do that. My model gives a completely incorrect answer because it failed to identify how key tokens in the question (like "prequel" and "July Fourth") relate to words in the context.

5 Conclusion and Future Work

This work was performed in the scope of the final project in CS-224N at Stanford University. I tackled the SQuAD dataset, and implemented a model that obtained up to 75.20% F1 score in the test set. I based my implementation on the BiDAF model of Seo et al. [12], with a few simplifications and some additions (notably, double BiDAF attention and residual connections). In sections 3 and 4, I attempted to describe my model and hyper-parameters in detail so my results can be replicated (possibly by future students in the class). Section 4 shows my full range of accuracy metrics, and also interesting analysis on the character-based encoding that I used. In particular, I provided evidence for the claim that character-level embedding helps the model to deal with out-of-vocabulary words.

Future work should probably focus on further model improvements, particularly in the prediction stage. It would also be interesting to experiment with LSTM cells instead of GRU cells. On the analysis side, I would be interested on an in-depth study of the benefits of residual connections in such networks; I conjecture that they are beneficial for the reasons I previously explained, but unfortunately I did not have enough time to conduct those experiments. Finally, at a higher level and in the longer term, it would be interesting to study what mechanisms one could introduce to make sure a SQuAD model also has access to some previous world knowledge besides just the context when answering a question. As the example in section 4.3 shows, this might be a crucial ingredient to eliminate the gap between human and machine performance on this task.

Acknowledgments

I would like to acknowledge Microsoft for graciously providing (virtually unlimited) computing time on a GPU-equipped machine through Microsoft Azure. Without it, I would not have been able to train the models I described in this paper. I would also like to acknowledge the CS-224N staff, who worked tirelessly to support the students on this project. One example out of many: when I was running out of computing credits, I posted on Piazza that I needed some more, and within 5 minutes the credits were available in my account. Thanks a lot!

References

[1] CS-224N Default Final Project: Question Answering. *CS-224N Winter 17-18*, 2018.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[3] F. Budianto. Reading Comprehension on the SQuAD dataset. *CS-224N Final Project Report*, 2017.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[6] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] J. Ke, Y. Wang, and F. Xia. Question Answering System with Bi-Directional Attention Flow. *CS-224N Final Project Report*, 2017.

[8] Y. Kim. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882*, 2014.

[9] L. v. d. Maaten and G. Hinton. Visualizing Data Using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[10] J. Pennington, R. Socher, and C. 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.

[11] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ Questions for Machine Comprehension of Text. *arXiv preprint arXiv:1606.05250*, 2016.

[12] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. *CoRR*, abs/1512.00567, 2015.

[15] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou. Gated Self-Matching Networks for Reading Comprehension and Question Answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198, 2017.