Implemention and Analysis of Character-level Embeddings, Self-attention, and R-NET on BiDAF for QA on SQuAD 2.0

Stanford CS224N {Default} Project

Rohin Manvi
Department of Computer Science
Stanford University
rohinm@stanford.edu

Avash Shrestha
Department of Computer Science
Stanford University
avash@stanford.edu

Abstract

Question Answering (QA) is an important task in the field of NLP. From chat bots to social media to medical text analysis, there are countless crucial applications of QA in the real world. In this paper, we aim to improve upon a given Bi-Directional Attention Flow (BiDAF) model for the task of QA on the SQuAD 2.0 dataset, by implementing various features such as character embeddings, self-attention, and another model's embeddings/encoding layers. We saw general improvement in our model's performance as we implemented features, with the exception of the other model's embeddings/encoding layers by themselves, which saw slightly worse performance. We analyzed our results and submitted our best model, which had tweaked hyperparameters, to the Gradescope Default Final Project - SQuAD track test leaderboard, where we got an EM score of 62.06 and an FM score of 65.14, an improvement on the provided baseline model's performance on the dev leaderboard. Finally, we qualitatively analyzed how our best model performed on different question types.

1 Key Information to include

- Mentor: Vincent Li
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Question answering (QA) is one of the most important tasks in the field of natural language processing (NLP). The overarching goal of QA is to make models that are able to extract meaning from texts in order to accurately answer questions on the model. This is a crucial task for a variety of applications, from chat bots to social media to medical texts.

A significant challenge in making these QA models is trying to figure out how to best assign meaning for the features in a text to a model. Take Figure 1 for instance, an example from the Stanford CS 224N Winter 2022 Default Final Project (IID SQuAD track) handout, henceforth referred to as the "project handout" [1]:

To properly answer this question, a model has to understand who/what Tesla and Gospic are and why Tesla returned to Gospic. This is made harder because the context paragraph includes information not related to the question at all, and so the model has to figure out the relevant information. Of

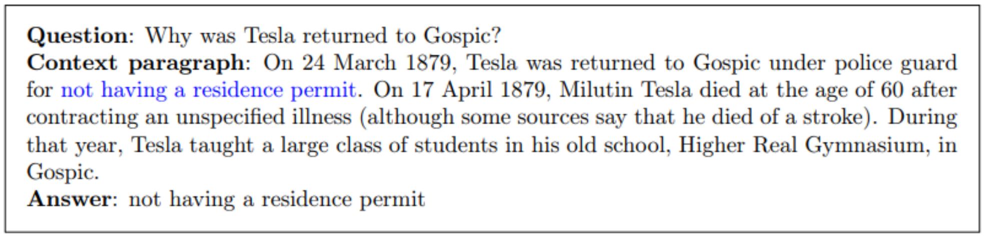
Question: Why was Tesla returned to Gospic?
Context paragraph: On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
Answer: not having a residence permit

Figure 1: In this example, the model has to ignore the irrelevant information to get the answer. [1]

course, for a human this problem seems trivial, since the exact reason was explicitly given. But, implementing this understanding in a model is not a trivial process.

In this paper, we will discuss extensions on a version of the popular Bi-Directional Attention Flow (BiDAF) model [2],which used a bi-drectional attention mechanism to better capture the important parts in a context text, given to us as part of the starter code for the Default project for CS 224N Winter 2022. At the time of the paper, the BiDAF model had achieved state-of-the-art results on the Stanford Question Answering Dataset 1.0, a dataset suitable for the task of QA (SQuAD)[3]. SQuAD 2.0 is an updated version of the original dataset that also includes adversial questions, meaning a model is also responsible for realizing when a question given to it is unanswerable [4].

We propose implementing character embeddings, self-attention, and a combination of embeddings and encoder layers on top of the starter model, so as to give our model more information about the relationships between the words/characters in a context example and have it pay more attention to the relevant parts of the input. We ran various combinations of the features we implemented on top of the starter model, and reported the results quantitatively and qualitatively. We also reported further on our best-performing model. We found that adding these features generally helped improve the EM/F1 scores of our model, likely because the information the model had access to was of higher quality.

3 Related Work

In order for QA models to be effective across a variety of contexts, and not be overfitted for their training dataset, it's crucial that the datasets they are trained on are high-quality and similar to real-world scenarios that a human might face. To fill this gap in the field, Rajpurakar et al. created the SQuAD dataset in 2016, which was a brand new Reading Comprehension (RC) dataset consisting of more than 100,000 questions, "almost two orders of magnitude larger than previous manually labeled RC datasets" [3]. These questions were created by crowdsourced workers on Wikipedia articles, where the correct answer is a direct subspan of the context passage given. This means a model has to understand more about the text, since there are a large number of possible answers, because it is free-response, instead of a set number of answer choices, as was given by previous RC datasets [3]. Later on, in 2018, SQuAD 1.0 was updated to SQuAD 2.0, and over 50,000 adversial unanswerable questions were added to it [4]. Previously, the unanswerable questions in SQuAD 1.0 were relatively easy to detect; this addition of crowdsourced adversial questions meant that models had become even better to perform well on SQuAD 2.0. Specifically, models now had to "Know what They Don't Know", in reference to the title of the SQuAD 2.0 paper [4].

A popular model that performed at state-of-the-art on SQuAD 1.0 was BiDAF [2]. The advancement of end-to-end trained systems in both text and image domains is partly due to the utilization of neural attention mechanisms, which allow the system to pay attention to particular parts of the context that are relevant to the query. This motivated the authors to further investigate attention and the limitations this mechanism had at the time. This paper attempted to improve the attention mechanisms by allowing the attention vector to be computed at every time step and by allowing previous representations in previous layers to flow into the subsequent modeling layer. This is opposed to generating a fixed-length vector of the attention, as done in previous work. This helps the attention mechanism not lose information, as compared to early summarization. The authors also used a memory-less attention that does not directly depend on attention at previous time steps. They hoped that this would better divide the work between the attention layer and the model layer, allowing each to focus on a more specific task. The attention is also then unaffected by premature attendances.

However, to perform well on SQuAD 2.0, a model needs to be more aware of its context. To attempt to fix this gap, attention was looked at again from a different view: self-attention. Made popular by the seminal "Attention Is All You Need" paper, by Vaswani et al., self-attention means the model looks at other words in its given context as it encodes/decodes a specific word, as can be seen by Figure 2.

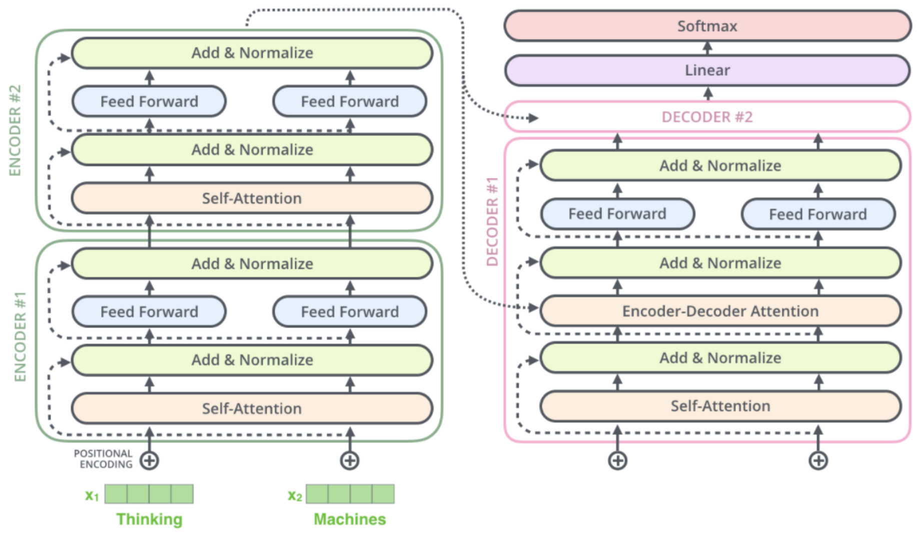

Figure 2: An example of a model using self-attention on its encoders/decoder. [5]

This technique changed the field of QA, with Transformer models, as mentioned in [6], becoming the new state-of-the-art. Nowadays, self-attention is seen as one of the most crucial elements of a QA model.

In this paper, we aim to implement on top of a given version of the BiDAF model. This is appropriate because the BiDAF model is a well-known model that performed at state-of-the-art at the time of its release. However, it is clear there are improvements to be made, since techniques such as self-attention have improved on BiDAF.

4 Approach

4.1 Baseline

Our baseline model was the starter model provided to us in the project handout, which was a version of the BiDAF model, notably without character embeddings implemented. Further information about the starter model can be found in [1] and information about the BiDAF model, in general, can be found in [2].

4.2 BiDAF Character Embeddings

To start, we implemented character embeddings as described in the BiDAF paper [2]. Generally, we ran initialized character embeddings provided to us in the starter model through a 2D Convolutional Neural Network and then max-pooled over the output, before concatenating them with the word embeddings provided for us in the starter model. The implementation of character embeddings was our original code, using PyTorch functions and written on top of the starter code. We found improvements on the performance of the model with character embeddings in comparison to the baseline, which will be covered in the Results section.

4.3 Self-Attention

Next, we implemented self-attention from our model, following the instructions of [7]. Specifically, we project down the output from the bi-directional attention layer and then feed it into a ReLU. We do this to compensate for the fact that the given bi-directional attention layer from the baseline did not feed the output through a ReLU, while the one in [7] did. From there, we follow the paper exactly, by then feeding the output through a bi-directional GRU, before applying the bi-directional attention layer again on the output, but this time on "between the passage and itself" [7]. Finally, we project the output from the self-attention and feed it through another ReLU layer, before summing it with the original output. The implementation of self-attention was our original code, using PyTorch functions and written on top of the starter code. This feature showed marked improvement in our models with self-attention implemented, as described further in the Results section.

4.4 R-NET Embeddings/Encoder

We also implemented a version of the R-NET Embeddings/Encoding, as described in depth in Section 3.1 of the paper [8]. We took initialized character embeddings, given to us in the starter code, and ran them through a single-layer bi-directional Gated Recurrent Unit (GRU), a type of RNN. We then took the final hidden states of the output and concatenated them with the word embeddings given to us in the starter model. Finally, we ran this through a 3-layer bi-directional GRU to produce encodings of the context and query, which were then fed into the bi-directional attention layer given to us in the starter model. The rest of the model was also the given starter code. The implementation of R-NET embeddings/encoder layers was our original code, using PyTorch functions and written on top of the starter code.

For this feature, we ignored the character embeddings we created and the encoder layer given to us, and wrote our own features before plugging in back to the baseline model for the attention, modeling and output layers. We had trouble with this feature, as it performed slightly worse than the given baseline model. To try to document this, we tested two versions of the model: one where the forward and backward character-level embeddings were concatenated and another where they were added together. This was done in an attempt to try to replicate the R-NET, which was ambiguous about how exactly the embeddings were created. We ended up finding that doing either didn't really help with the model performance, so there is most likely a different issue with our implementation. We propose that it's due to the mix-and-match aspect of this feature, since the majority of the model is from the BiDAF model, while only the embeddings/encoding layers are from the R-NET paper. There might be incongruities in the model because of this. However, as shown in the Results section, adding self-attention to this feature makes the model perform better than the baseline, so there is some hope with this feature.

5 Experiments

5.1 Data

To be able to run our experiments, we need an appropriate dataset to handle the task of question answering. In this paper, we are using a modified version of the official SQuAD 2.0 dataset, as further described in Section 3 of the project handout [1]. Throughout our experiments, we will keep the dataset the exact same, so as to make sure that our comparisons are on a common dataset.

5.2 Evaluation method

We will use 2 metrics to analyze the performance of our models: Exact Match (EM) and F1 Score.

- Exact Match: 1 if the predicted output is exactly the answer, 0 otherwise
- F1: $\frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$. More information on F1 can be found at https://en.wikipedia.org/wiki/F-score.

5.3 Experimental details

For the majority of our models, we kept the hyperparameters the same as the starter code, because of a lack of time in testing different hyperparameters for different models. Unless otherwise mentioned, the hyperparameters for our models include:

- Dropout Rate: 0.2
- Learning Rate: 0.5
- Hidden Size: 100
- Decay Rate: 0.999
- Batch Size: 64

More information about hyperparameters for our models can be found in the project handout [1]. We ran each model for 30 epochs, which took around 3.5 hours to 6 hours, on a mixture of Microsoft Azure and Google Colab.

5.4 Results

As seen in Table 1, we ran 11 distinct models, with various features added in combination.

As seen in Figure 3, there is a distinct increase in performance between our baseline, character embeddings only, and our best models. To be clear, a higher AvNA, EM, and F1 and a lower NLL means better performance.

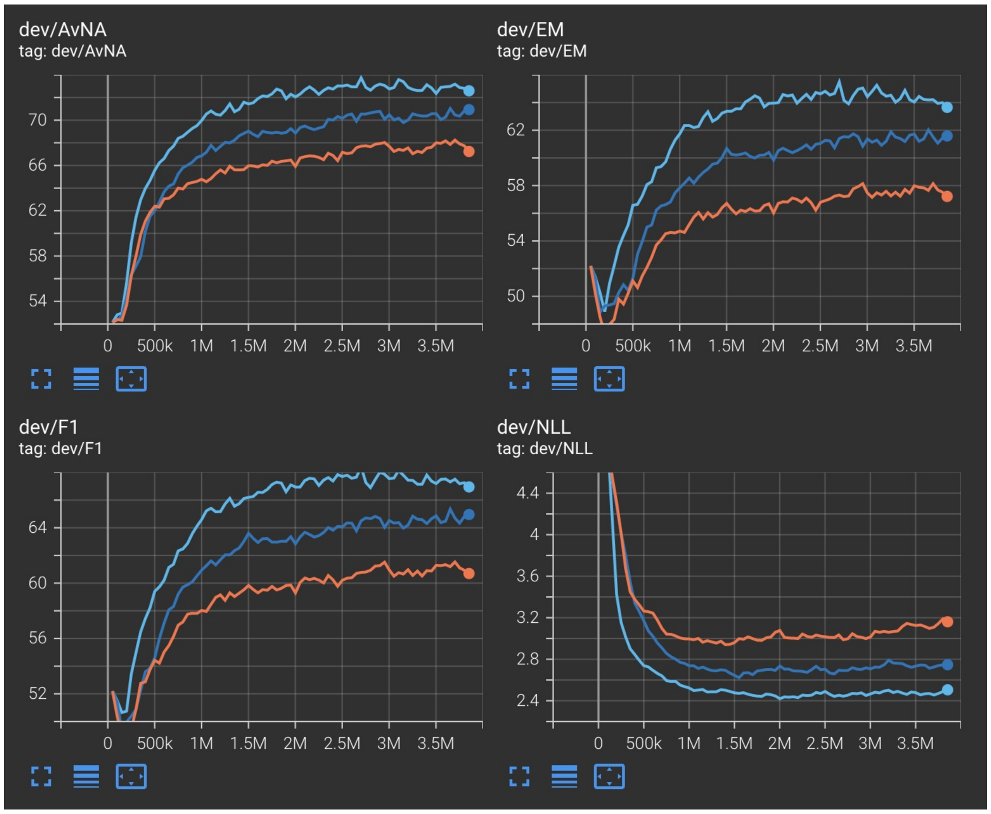

Figure 3: The light blue line is our best model, the blue line is our model with only character embeddings, and the orange line is the baseline model.

Table 1: Results

Model	Leaderboard	LR	Dropout	Hidden	EM	F1
BiDAF (baseline)	dev	0.5	0.2	100	58.58	61.90
BiDAF + Char Embed	dev	0.5	0.2	100	62.02	65.34
BiDAF + Char Embed	dev	0.6	0.15	100	62.81	66.21
BiDAF + R-NET (concat)	dev	0.5	0.2	100	57.50	60.90
BiDAF + R-NET (add)	dev	0.5	0.2	100	57.60	60.97
BiDAF + R-NET (concat) + Self-Att.	dev	0.5	0.2	100	61.91	64.89
BiDAF + R-NET (add) + Self-Att.	dev	0.5	0.2	100	63.20	66.32
BiDAF + Char Embed + Self-Att.	dev	0.5	0.2	100	63.91	67.26
BiDAF + Char Embed + Self-Att.	dev	0.6	0.15	100	63.23	66.16
BiDAF + Char Embed + Self-Att.	dev	0.5	0.2	125	65.48	68.40
BiDAF + Char Embed + Self-Att.	test	0.5	0.2	125	62.06	65.14

5.4.1 Character Embeddings

Implementing character embeddings on top of the baseline showed a marked improvement on the baseline, with an about 3.5 improvement in EM and F1 scores. When we tweaked hyperparameters, specifically dropout rate and learning rate, we got another increase in the score. This shows that there are promising results for character embeddings, and that the inclusion of this feature allows the model to perform better, likely because it has contains more information about the passage and query. It allows the model to better encode words that are out of its vocabulary. This leads to a better overall understanding of the passage.

5.4.2 Self-Attention

Implementing self-attention along with the previously implemented character embeddings also showed an increase: from 62.02 to 63.91 for EM and 65.34 to 67.26 for F1, in comparison to the model with character embeddings implemented. We tested a few different hyperparameter changes, and found that increasing the hidden size, while keeping all of the other hyperparameters the same, showed the best improvements. This was expected, because of how important self-attention is to the model, since it allows the model to focus on the most important aspects of the context.

5.4.3 R-NET Embeddings/Encoding

Our implementation of just the R-NET Embeddings/Encoding layers proved worse than the baseline, with worse scores by about 1 point for EM/F1 for both types. This was disappointing, and we attribute some of this to the R-NET paper being ambiguous on how to implement this layer as they did. We also think this worse performance could be because we are implementing a different embeddings/encoding as the BiDAF model originally did, which might cause a loss of information since different features for different models are being mixed and matched. However, when we added self-attention to the R-NET Embeddings/Encoding layers, we did see an improvement in performance as compared to the baseline, but still worse than the model with character embeddings and self attention. This shows there is some promise with the R-NET features, since it performed comparably to one of our best features once self-attention was added to it, even though it performed a lot worse in direct comparison to character embeddings by themselves. We didn't have enough time to test different hyperparameters for this feauture; however, we do note that when we followed the hyperparameters listed in the original paper [8], our model performed considerably worse. We think that trying to make the model with R-NET features implemented as similar to the baseline as possible helped its performance, so future hyperparameter testing would likely vary only slightly, so as to keep them similar to the baseline's as much as possible.

6 Analysis

As seen in Table 2, we collected information on the performance of our best model, BiDAF + Char Embed + Self-Att. with hidden size 125. As seen in Figure 4, "What" questions made up the majority of the dataset, with the rest being relatively small in comparison. We define Mean Answer Rate to be

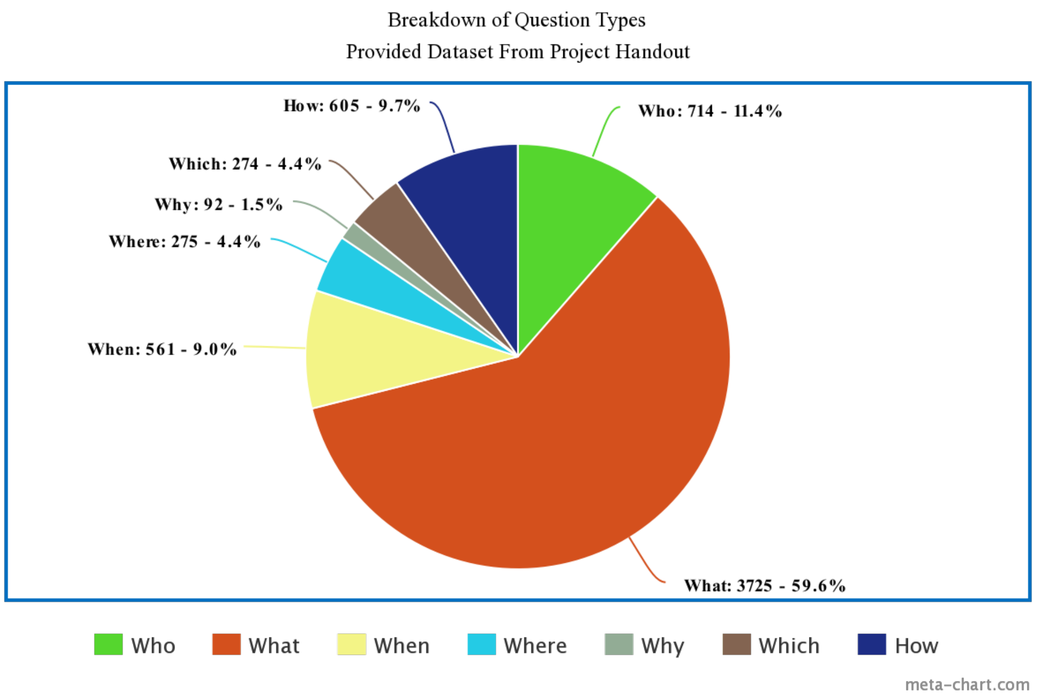

Figure 4: Breakdown of QUestion Types from provided dataset [1]

Table 2: Best Model by Question Type

Question Type	Who	What	When	Where	Why	Which	How
Size	714	3726	561	275	92	274	605
EM	65.28	65.26	68.10	65.44	59.09	68.69	62.64
F1	67.39	68.17	69.97	69.29	65.87	70.95	67.13
Mean Answer Length (characters)	17.6	21.6	11.6	18.9	45.3	17.2	18.5
Mean Prediction Length (characters)	17.8	20.3	11.3	17.8	48.9	16.9	16.1
Mean Answer Rate	0.44	0.45	0.52	0.48	0.50	0.53	0.45
Mean Prediction Rate	0.55	0.49	0.59	0.56	0.35	0.52	0.55

the percentage of answerable questions and Mean Prediction Rate to be the percentage of questions the model gave an answer for. We combed through our results, and found these trends in our model's performance, for the different question types:

- **Who:** "Who" questions may be difficult for our model because names do not carry as much subword information as other words. The answers must be proper nouns. Despite this, our model performed well on this question type.
- **What:** The largest percentage of questions are "What" questions, which constitutes over half of the questions in the evaluation set. "What" is a versatile question word and can be used to encompass many different types of questions. This is likely part of the reason why the EM and F1 scores are so close to the average.
- **When:** Our model performs well for "When" questions, likely because of its ability to predict well which numbers and words represent a point in time. The EM score is high due to ease in isolating the numbers/words that represent a point in time.
- **Where:** Our model performs well for "Where" questions, most likely due to the relative ease in choosing between a few words that it identifies as places. The EM score is not as high as F1, likely due to it being difficult to isolate the name of a place compared to say a date or time.

- **Why:** Our model performs worse for "Why" questions than on average, likely due to the questions being more difficult and nuanced compared to questions such as "When" and "Where". It may also be because there is a small number of “why” questions. This is supported by the fact that the average answer length is very large. It is around two to three times as long as the other type of answers.
- **Which:** Our model performs best for "Which" questions (EM and F1), most likely due to the relative ease in choosing between a few viable nouns or phrases. The EM score is close to F1 score due to ease in isolating the noun. It should also be noted that this is the only question type where the mean answer rate is greater than the mean prediction rate. This likely indicates that the model is almost “confident” in this category.
- **How:** Our model performs slightly worse for "How" questions, likely due to the questions being more difficult and nuanced compared to questions such as "When" and "Where". It’s similar to "Why" questions, where a reason has to be found instead of a simpler choice between phrases, like for "Which" questions.

In general, we see that the ratio between the mean prediction rate and the mean answer rate is an indicator of how “confident” the model is in a particular question category and how well it does. As this ratio increases, we believe the model is more "confident" in its answers, which can also be seen in its increase in EM/F1 performance as the "confidence" ratio increases. We also see that the model has a hard time with questions with greater average word length. This may be because understanding the semantic relationship between words that are far apart is difficult for the model.

We saw a large decrease in our best model’s performance on the dev set as compared to the test set. This large decrease in performance from the dev set to the test set likely indicates that the model is overfitting to the dev set and is not able to generalize well. This is probably due to our best model being one where we changed the hyperparameters from the given values, in an attempt to increase its performance on the dev set. Overall, though, we did improve on the baseline model’s dev set score with our test set submission.

7 Conclusion

To conclude, we implemented 3 distinct features on top of the given BiDAF model: character embeddings, self-attention, and R-NET’s embeddings/encoding layers. We found that as we implemented features to the given baseline, our model’s performance increased, with the exception of the R-NET embeddings/encoding layers by themselves. We found our best model to be a combination of the baseline, character embeddings, and self-attention, which we then ran hyperparameter tests on so we could increase its performance. Our best model did slightly worse on the test set, likely due to overfitting, but still performed well, and we showed an overall increase in performance as compared to the given baseline. We analyzed our best model, and reasoned through its performance on different question types on the provided dataset.

One major limitation of our project is that we did not have time to do more hyperparameter testing. Given more time and resources, we could have likely improved our performance even more with better hyperparameter tweaking. Finally, we believe that the R-NET embeddings/encoding layers have promise in them to perform well, since they performed better than the baseline when the self-attention was added to them; however, we were not able to understand exactly how the original authors of the R-NET paper, [8], exactly made their model. With more time, we could have figured it out, and likely performed better with those features than we did for this paper.

References

[1] CS224N Teaching Staff. Cs 224n default final project: Building a qa system (iid squad track). `http://web.stanford.edu/class/cs224n/project/CS224N_Final_Project_Milestone_Instructions.pdf`, 2022.

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical*

Methods in Natural Language Processing, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

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

[5] Jay Alammar. The illustrated transformer, Jun 2018.

[6] 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.

[7] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 845–855, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[8] Microsoft Research Asia Natural Language Computing Group. R-net: Machine reading comprehension self-matching networks. `https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf`, 2017.