Self-Attention QA System

Stanford CS224N Default Project
Track: Squad

Aryan Chaudhary and Kevin Yang
Department of Computer Science
Stanford University
achaud@stanford.edu, kevvyang@stanford.edu

Abstract

In this project, we built a neural network model that can answer questions for the Stanford Question Anwering Dataset (SQuAD) 2.0. We based our model off the 2017 R-NET implementation by incorporating the self-attentive features, including the gated attention based recurrent network. From the baseline BiDAF model, we included two key changes: character embeddings and R-NET self-attention. We expected these two changes to display improvements from the baseline (EM = 52.19, F1 = 55.69), and after including character embeddings, we saw a slight increase to EM = 54.51, F1 = 57.92. Our scores further improved after adding self-attention to EM = 56.15, F1 = 59.96.

1 Key Information to include

- Mentor: Lucia Zheng
- External Collaborators: None
- Sharing project: No

2 Introduction

Asking questions has always been an integral part of human behavior: we are a curious species that yearns to explore the intricate mysteries that the universe holds. Without this innate desire to seek out the unknown, humanity would never have progressed beyond the primary stages of civilization. Question-answering is a key component of our lifestyles that drives every interaction, which is why it is such an interesting topic to delve into. Now, with the power of artificial intelligence and natural language processing, we can imbue machines with the power to answer questions for us. This final project captures the spirit of inquisitiveness that is embedded in every living person's soul by allowing us to create a tool that finds answers despite all obstacles.

The goal of this project is to give a model the ability to pick out the important pieces of a passage and return the correct answer given a question. Obviously, there is a plethora of work in this area, even if we limit the scope to just the world-renowned SQuAD dataset. As students, we cannot hope to compete with the work of distinguished researchers who have dedicated their lives to natural language processing, but we can aspire to implement their ideas and possibly find methods to improve their baselines.

While it may be difficult to create such a model from scratch, we thankfully have a lot of resources to work with because so many people have taken part in this process already. Especially with the default project baseline, we are given a headstart into this realm of end-to-end neural network modeling. To

test our model, we used the provided SQuAD 2.0 dataset. This is the official dataset of SQuAD 2.0 and is the dataset used for all the default projects. Within the dataset, there are three sets: train, dev and test. The training dataset has 129,941 examples, the dev set is randomly selected and has 6078 examples and lastly the test set has 5915 examples.

3 Related Work

Our model is based off an existing model called R-NET, an end-to-end neural networks model for question answering given a context and a question. This implementation uses a gated attention-based recurrent network to determine the importance of information in a given passage regarding the question asked. Next, it uses self-attention to match the passage against itself, which is the piece that we chose to focus on and incorporate in our own model.

This model was tested on the SQuAD dataset and achieved the best results amongst all published results for it. In terms of specific metrics, R-NET achieved a high of 71.3 EM and 79.7 F1 for the dev set and 72.3 EM and 80.7 F1 for the test set. This appears to be the one of the best self-matching networks currently available so we want to try and build or model using it as a template.

There are also existing question answering models that we can use as baselines to measure our progress. The starter code for the default project implemented BiDAF (Bi-directional Attention Flow) to serve as a baseline for our improvements. The BiDAF implementation that we have averages around 52 EM and 55 F1, which indicates that it is slightly better than blindly guessing at the answer or only returning N/A.

4 Approach

4.1 Overview of Approach

Our ultimate goal was to incorporate both self-attention as well as character embeddings in order to improve the results from the baseline code.

We first approached character embeddings. The starter code had word level embeddings already implemented, and it wasn't too difficult to extrapolate from that implementation a rough picture of what character-level embeddings was going to look like. The implementation step was a little harder, as matching sizes and understanding pytorch functionalities such as max-pooling proved to be difficult at times.

In the end, we had a fully functioning character and word level embedding neural net, an improvement from the baseline, and our results of F1 and EM showed it.

To approach R-NET's self attention, we started by first extensively studying the R-NET self-attention equations, shown here:

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P)$$

$$a_i^t = \exp(s_i^t) / \Sigma_{j=1}^n \exp(s_j^t)$$

$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

After getting a firm grasp of each and every variable, we used the initial BiDAF Attention layer as our starter code and made changes until it achieved the same effect that R-NET's self attention did.

4.2 Character Embeddings

When implementing character embeddings, we started by understanding how word embeddings worked. The initial embeddings within our baseline model only took in x, the word vectors. The word vectors are then ran through a pretrained word_embeddings layer before applying a dropout layer, projecting onto a linearity function, and then running it through a highway encoder.

We knew there were going to be many similarities between the code of character embeddings and word embeddings, so we duplicated the code within embeddings and started there. Since the x input was the word vectors, and we can't use word vectors for the character embeddings, we needed to input character vectors as well, noted as y. Similarly, we also needed to include char_embeded, which thankfully is also a pretrained layer. However, the most difficult part is still up ahead.

The most difficult hurdle in character embeddings is that we need to implement character embeddings to have the same output size as word embeddings. This is because our forward layer can only include a singular tensor, so we must return a concatenation of both tensors together, meaning that they must have the same dimensions. Throughout this process, we got really familiar with torch.permute, as we had to permute the dimensions of char_emb a few times to match the final output dimensions.

Slightly different than word embeddings is the fact that in character embeddings, we must run it through a convolution layer and maxpool layer. Convolution was pretty spelled out for us, however maxpool had multiple possible implementations. The implemention we ended of choosing was using torch.amax. Torch.amax takes in an input tensor, and the dimension you want to find the max on. The reason why we choose amax instead of max is because max returns a list instead of a tensor when specifying a dimension. Amax however doesn't and still returns a tensor.

After running character embeddings through an amax, all we had to do was re permute the dimensions and then run it through a highway. Now within embeddings, we have two tensors of the same size, one for the character embeddings and oen for our word embeddings. Since they're the same size, we just need to concatenate the two, and return.

4.3 R-NET Self-Attention

After implementing character embeddings, we decided to take it to the next level by implementing R-NET self attention. Before we implement this step, however, we had to use a gated attention-based recurrent network to identify the important parts of a passage given a question that needs answering. This generates the 'v' value that we feed into our self attention layer which contains the sentence-pair representations. The full implementation of this layer is described in the R-NET documentation, but it is fundamentally very similar to the self-attention layer, only it uses a gate to focus on the relationship between the question and current word from the context.

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh(W_u^Q u_j^Q + W_u^P u_t^P + W_v^P v_{t-1}^P)$$

$$a_i^t = \exp(s_i^t) / \Sigma_{j=1}^m \exp(s_j^t)$$

$$c_t = \Sigma_{i=1}^m a_i^t u_i^Q$$

Evidently, this process is nearly identical to the R-NET self-attention equations except there is an

additional weight matrix in the tanh function to account for the context representations. Here, the u_t^P and u_t^Q matrices are context and question representations, respectively, that are used to generate the v_t^P self matching context representations.

Our self-attentive model is implemented using additive attention as described in lecture. The benefits of this approach are that it increases speed and computational simplicity while also increasing the model's accuracy. To go through the step by step of self-attention, we can reference the image of self-attention as well as the image of the gated attention based recurrent network.

In the first step, we are finding the similarity between two vectors and then multiplying it by a linearity, v^T. Our context representations v_t^P incorporate matching knowledge that was given by the context/question encodings. This first part was fairly straightforward as all we needed was three linearity layers, v^T, W_u^P, and W_u^Q. We then just find the sum of the two vectors, put it through a non-linearity function torch.tanh, and the multiply that by v^T. Te next part is obtaining a_i^t, which is just the softmax of the similarity between two matrices, which is just simply putting it through a softmax function.

5 Experiments

5.1 Data

This is the official dataset of SQuAD 2.0 and is the dataset used for all the default projects. Within the dataset, there are three sets: train, dev and test. The training dataset has 129,941 examples, the dev set is randomly selected and has 6078 examples and lastly the test set has 5915 examples.

There are two types of questions: answerable and unanswerable. Around half the total questions are those labeled unanswerable. The dataset contains tuples of (context, question, answers). In the case where there is no answer i.e. unanswerable question, the answer is empty. One restriction on the answer is that it's a subset of text from the context, meaning our task at hand is pretty much "highlighting" the right text from our dataset. Essentially, we are inputting (context, question, answer) tuples and expecting answers as output from the model.

5.2 Evaluation method

To evaluate performance, we had two metrics: EM metric, and the F1 metric.

The EM metric is a binary metric that measures if you got the right answer or not. This is a strict metric that doesn't even count synonmous answers as correct.

F1 is the harmonic mean of precision and recall. The precision is what percentage of the given answer is in the actual answer, and recall is what percent of the actual answer is in the given answer.

Finally, the mean of these two numbers is taken: we average both scores across the entire dataset to get the reported scores.

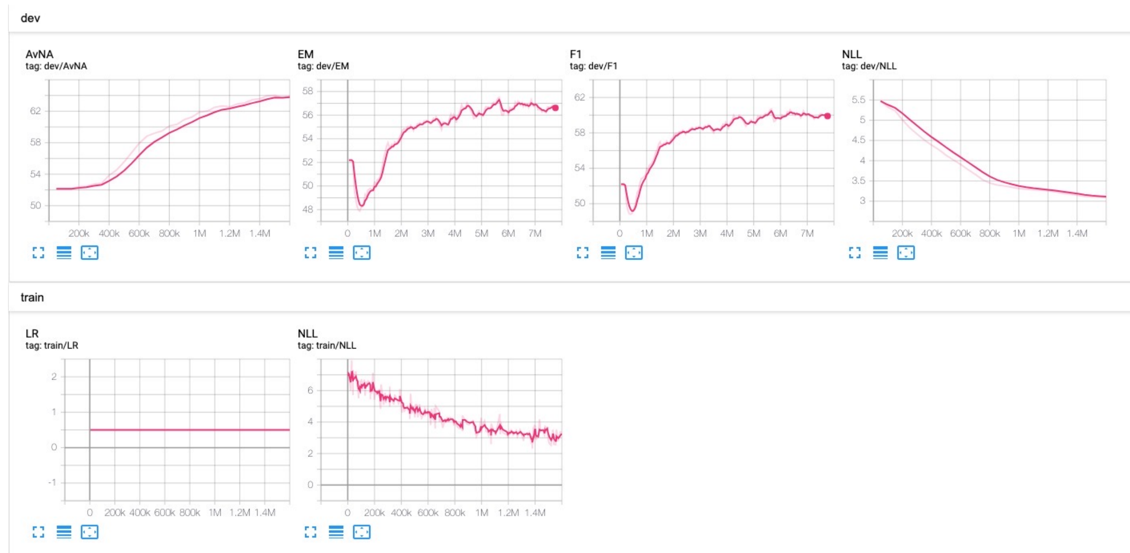
5.3 Experimental details

We first ran our code with the initial BiDAF model and classifier as provided in the starter code, then ran it again with additional character-level embeddings. We optimized the model using AdaDelta, setting the learning rate to 0.1, and epsilon to 1e-6. Following the R-NET implementation, our dropout layer has a dropout rate of 0.2, and our hidden vector length is 75 for all purposes (attention, linear layers, etc.).

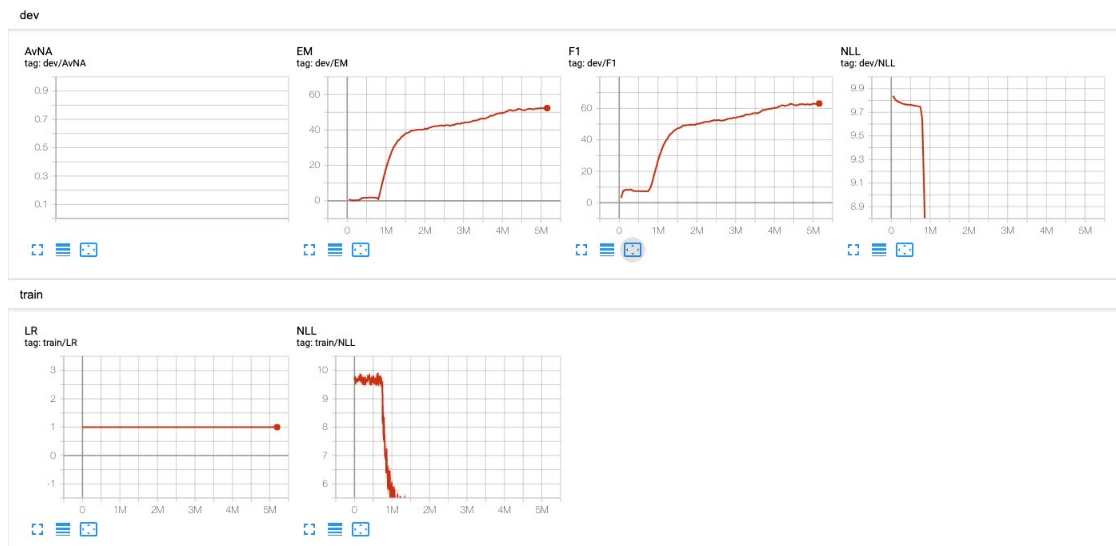
We then experimented with some different learning rates to see if model performance would improve. We tried out a learning rate of 0.5 and 1 as well, but found that these were relatively ineffective compared to our lower initial learning rate. This was an interesting difference from the R-NET

model, which proposed a learning rate of 1 and performed exceptionally well.

5.4 Results



Character Embedding + BiDAF Model graphs



Self-Attention + Character Embedding + BiDAF Model Graphs

We are on the PCE leaderboard. Our final F1 and EM scores were 56.15 and 59.96. This is a nice improvement from the baseline we got of 52.19 and 55.69.

We expected a little better than 56.15 and 59.96. Within R-NET, a huge portion of the implementation they had was their self-attention neural network, which we followed pretty closely and should have an exact implementation. With that being said, they achieved way higher scores of 72.3 and 80.6, so we were expecting at least to break 60s in EM and even break 63s in F1.

The reason this may be is because we never incorporated any sorts of self-matching or question-answer matching which was incorporated within R-NET. They also most likely had more resources than us as well resulting in better pretrained data.

6 Analysis

In order to learn more about when our model succeeds and fails, we can analyze the discrepancy between correct and incorrect outputs from the model. Observe the following two correct answers that the model gave:

- **Question:** Who decides who gets to address the members of Parliament to share their thoughts on issues of faith?
- **Context:** The first item of business on Wednesdays is usually Time for Reflection, at which a speaker addresses members for up to four minutes, sharing a perspective on issues of faith. This contrasts with the formal style of "Prayers", which is the first item of business in meetings of the House of Commons. Speakers are drawn from across Scotland and are chosen to represent the balance of religious beliefs according to the Scottish census. Invitations to address Parliament in this manner are determined by the Presiding Officer on the advice of the parliamentary bureau. Faith groups can make direct representations to the Presiding Officer to nominate speakers.
- **Answer:** Presiding Officer
- **Prediction:** Presiding Officer

- **Question:** Where was the Muslim Brotherhood founded?
- **Context:** Roughly contemporaneous with Maududi was the founding of the Muslim Brotherhood in Ismailiyah, Egypt in 1928 by Hassan al Banna. His was arguably the first, largest and most influential modern Islamic political/religious organization. Under the motto "the Qur'an is our constitution," it sought Islamic revival through preaching and also by providing basic community services including schools, mosques, and workshops. Like Maududi, Al Banna believed in the necessity of government rule based on Shariah law implemented gradually and by persuasion, and of eliminating all imperialist influence in the Muslim world.
- **Answer:** Ismailiyah, Egypt
- **Prediction:** Ismailiyah, Egypt

Here, we can see that when the question explicitly starts with a question word, the model is able to pull the correct answer out of the context. Through all the training samples, this phenomena was generalized to all question words like Who?, What?, When?, Where?, Why?, and How?, although each had relatively unique levels of accuracy. This indicates that the model learned how to correctly return a name, location, or a time period depending on the question word it was given. Sometimes, we would observe that the wrong name or time period would be returned, but these occurrences were few and far between.

Now, take notice of the next two incorrect outputs from the model:

- **Question:** Case complexities provide three likelihoods of what differing variable that remains the same size?
- **Context:** The best, worst and average case complexity refer to three different ways of measuring the time complexity (or any other complexity measure) of different inputs of the same size. Since some inputs of size n may be faster to solve than others, we define the following complexities:
- **Answer:** inputs
- **Prediction:** any other complexity measure

- **Question:** Besides Britain and North America, where else did Huguenot refugees settle?
- **Context:** The revocation forbade Protestant services, required education of children as Catholics, and prohibited emigration. It proved disastrous to the Huguenots and costly for France. It precipitated civil bloodshed, ruined commerce, and resulted in the illegal flight from the country of hundreds of thousands of Protestants, many of whom became intellectuals, doctors and business leaders in Britain as well as Holland, Prussia, and South Africa. Four thousand emigrated to the North American colonies, where they settled in New York and Virginia, especially. The English welcomed the French refugees, providing money from both government and private agencies to aid their relocation. Those Huguenots who stayed in France became Catholics and were called "new converts".
- **Answer:** Holland, Prussia, and South Africa
- **Prediction:** New York and Virginia

It is clear to see that the model was a bit confused by the wording of the question even though the fundamental question words are still present. We believe this to be the result of complex wording: since there is a phrase coming before the actual question itself, the model is less sure about which parts of the question to pay more attention too and therefore what the actual question it needs to answer is.

For the first example, the returned answer is a noun, as it should be, but the model did not know what exactly the question was attempting to highlight. Similarly for the second example, the model knew

that it needed to return a location, but it could not account for the fact that the question asked for a location "Besides Britain and North America". Since our model is being trained on the relationships between individual words in a sentence and their connection to the broader context (self-attention), it may be tough to unscramble complex sentence structures and return the accurate answer.

Let us now look at one final example where the model was accurate but struggled from lack of understanding:

- **Question:** In the determination of complexity classes, what are two examples of types of Turing machines?
- **Context:** Many types of Turing machines are used to define complexity classes, such as deterministic Turing machines, probabilistic Turing machines, non-deterministic Turing machines, quantum Turing machines, symmetric Turing machines and alternating Turing machines. They are all equally powerful in principle, but when resources (such as time or space) are bounded, some of these may be more powerful than others.
- **Answer:** probabilistic Turing machines, non-deterministic Turing machines
- **Prediction:** deterministic Turing machines

In this scenario, the question asked for two answers, but the model only returned one answer which was correct. There probably were not a lot of training examples that required multiple outputs for a singular question, so it would have been very difficult for it to figure out that it needed to return two answers in this case. Errors like this were less common but still existent, and we can see that a larger training data set could potentially resolve this issue by giving the model more exposure to more unique output requests.

Overall, our model learned how to recognize the differences between question words and return the corresponding answer type with some variations in accuracy. Complex structures tended to confuse the model because it did not understand where the question was exactly in the phrase it was given. Finally, the model struggled with returning multiple outputs when asked for it due to a lack of such examples in the training data.

7 Conclusion

Natural Language Processing is an intricate and concise system. The slightest error in tensor shapes or misalignment can result in an entire system to fail or produce drastic changes in output. Within implementing self attention and character embedding, we encountered and surpassed many hurdles.

Throughout our project, we can see even small improvements can result in significantly better performing test results such as seen from our improvements after implementing both character encoding and self attention on top of the baseline model that only incorporates BiDAF attention and word encoding.

Our model ended up being fairly successful, improving the baseline by more than 4% each. With more optimizations, we can expect this jump to skyrocket even more. The biggest limitation of our work has been the time constraint caused by 5 weeks within this class, and also our lack of NLP experience. Although this class has taught us a lot about NLP basics, I would say self implementation and thoroughly understanding how the initial BiDAF model worked is another level above what we've done prior in class. In the future we would love to see where the possibilities of creating multiple layers, maybe implementing the transformer-XL or coattention. Another possible realm of exploration is how minor changes - non artitectural changes can play a role in improving F1 and EM scores of our project.

References

[1] R-NET: Machine Reading Comprehension with Self-Matching Networks. Natural Language Computing Group, Microsoft Research Asia, May 2017, https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf.

[2] https://pytorch.org/docs/stable/generated/torch.amax.html

[3] Bansal, Aakash, et al. "A Neural Question Answering System for Basic Questions about Subroutines." 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021, https://doi.org/10.1109/saner50967.2021.00015.

[4] Robillard, Martin P., et al. "On-Demand Developer Documentation." 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, https://doi.org/10.1109/icsme.2017.17.