QA System Using Feature Engineering and Self-Attention (IID SQuAD track)

Stanford CS224N Default Project

Alex Oseguera, Jay Saleh
Department of Computer Science
Stanford University
`aoseg@stanford.edu, jsaleh@stanford.edu`

Abstract

Machine reading comprehension is an exceedingly important task in NLP and is a desired feature in many of the latest consumer and research projects. Therefore, using this task as motivation, we set out to build a reading comprehension model that performed well on the SQuAD 2.0 question answering dataset. To do this, we built upon the existing BiDAF [1] machine comprehension model given to us through the CS224n staff. Our contributions to this model are a character embedding layer on top of the existing word embedding layer, a self attention layer, and added features to the character and word embeddings which include Part of Speech tags (POS), named entity recognition (NER) tags, and dependency tags. As a result of implementing these layers we found that character embedding with additional input features performed the best with an F1 dev score of 64.38 and an EM dev score 61.29. On the test set we achieved F1 and EM scores 62.17 and 59.04 respectively.

1 Introduction

From the search engine's we use everyday to the chatbots we interact with on websites to the personal assistants in our rooms, question answering systems have become a staple of modern NLP technologies. Not only is the question answering task useful in a variety of different applications, it is an important tool for evaluating how well machines can "understand" text. Therefore, working on improving and fine-tuning these systems has been a major focus of NLP research and has led to many state of the art systems in the space. As part of our venture into the space, we built upon one of the most popular models in the space, Bi-Directional Attention Flow (BiDAF), which is a hierarchical multi-stage architecture for modeling the representation of context paragraphs at different levels of granularity [1]. Although there are many datasets that have been created for the exact task of question answering, we used the Stanford Question and Answering Dataset (SQuAD), particularly, version 2.0 which includes questions that cannot be answered through the context provided.

Looking at the BiDAF model we were provided by CS 224n, we noted that we could improve upon the model using new question answering methods that have been published since the release of the BiDAF model. For starters, we noticed that the BiDAF model given to us did not have the character embeddings discussed in the BiDAF paper [1], therefore, we looked towards adding the functionality to see how scores improved. From there, we took inspiration from the seminal "Attention is all you need paper" [2] and noted that self attention could be a useful layer to add onto our model. In doing research for this model involving self attention, we found that the RNET self attention layer for question answering [3] had derived excellent results in the question answering domain and thus would be a good fit for our self-attention purposes. Additionally, we looked into the embedding layer and noted that adding sentence and word structure information to our embeddings could help the model learn, and thus decided to research into feature engineering. In this research, we found that the feature engineering found in the DrQA [4] paper could be helpful in our own model.

As a result, our model incorporated a character embedding layer, a self matching layer as specified in the RNET [3] paper, as well as feature engineering that included Part of Speech Tags (POS) for words, Named Entity Recognition (NER), and dependency tags. These contributions gave us better results over the baseline BiDAF model provided and resulted in the highest F1 and EM scores of 64.38 and 61.29 on the dev set and 62.17 and 59.04 on the test set respectively.

2 Related Work

2.1 Baseline

Our baseline is inspired from the BiDAF model [1]. BiDAF is a hierarchical multi-stage architecture for modeling the representation of context paragraphs at different levels of granularity. This model, as described in the paper, uses word-level and contextual embeddings as well as a bi-directional attention flow to obtain query aware representations. This paper's major contribution was the creation of a novel attention mechanism detailed within the paper which is hypothesized to force the attention layer to focus on learning the attention between the query words and the context words, and allows the modeling layer of the model to focus on learning the interaction within the query-aware context representation (the output of the attention layer). Prior to Transformer models and other pretrained models BiDAF achieved state of the art results and is thus a great starting point for improvements.

2.2 Self Attention

In our BiDAF baseline, the attention performed within the model is between context words and query words which then generates query aware feature vectors which are modeled later on [1]. However, recurrent models such as these can only memorize a limited amount of passage context and thus may lose out on answer clues that are found in other parts of the passage. As a result, the RNET [3] paper proposes a self matching attention layer which aggregates evidence from the whole passage to infer the answer. It does so by taking the output of context query attention, similar to that of the baseline model, and performs self attention on these outputs to aggregate information relevant to each passage word from all other words in the passage. In this way, we add an extra layer of information to our previous attention layer which can be useful in training.

2.3 Feature Engineering

In natural language processing feature engineering involves generating input features for a model based on a certain word or words in a corpus of text. In "Reading Wikipedia to Answer Open-Domain Questions" [4] Chen et al. utilizes a novel document reader to parse through Wikipedia articles and generate context paragraphs in which the answers to given questions can be found. Each word in a context paragraph is processed to generate features.

In Chen's approach, the first feature involves having each word in the paragraph embedded using the 300-dimensional Glove word embeddings trained from 840B Web crawl data (Pennington et al., 2014). The second feature involves having three binary values for each word. The first binary value indicates whether the exact match of the word is found in the question, the next binary value indicates whether the lemma form of the word is found in the question, the last binary value indicates whether the lowercase form of the word is found in the question. The third input feature involves storing additional token features. Some token features include part of speech, named entity recognition and normalized term frequency. Lastly, the final feature involves generating an aligned question embedding where an attention score ($a_{(i,j)}$) captures the similarity between paragraph token (p_i) and question token (q_j).

As a result of using these features, Chen saw excellent results on question answering datasets and thus is an avenue that is worth considering for our approach.

3 Approach

3.1 Baseline

Our baseline is the Bidirectional Attention Flow Model (BiDAF) with word level embeddings as described in the default project handout provided by the CS224n teaching team [1].

3.2 Character Embedding

As described in the introduction, the BiDAF model given to us by the CS 224n teaching team lacked the character embeddings that are found in the original BiDAF paper. Due to the fact that words at training and test time can be out of the vocabulary and that character composition can provide more information about a given word, we found that it would be useful to use character embeddings for our model.

To do this, we took the pretrained character vectors provided by the teaching team and used those vectors, as well as the prepared character indices provided by the teaching team, to pass to our embedding layer. In our embedding layer we followed the character embedding conventions described by Kim in *Convolutional Neural Networks for sentence classification* [5]. More precisely, we embedded the character indices using an embedding layer. We then performed dropout on this embedding. After this process, we passed the embedding through a 1 dimensional convolution (although in code we use torch.Conv2d due to the 4-dimensional shaping of the character embeddings) and then performed max-pooling after the convolution. Finally, we took this result and concatenated the character vectors onto our word vectors to generate the new input for the model. This code was written originally by the team.

3.3 Self Matching Layer (Self attention)

Our self attention layer is built off of the description and math described in the RNET paper. After we go through the embedding layer we pass the input to the BiDAF Attention layer [1]. The BiDAF attention layer performs Context-to-Question attention on the input similar to that of the Context-to-Question attention layer detailed in the RNET paper. For the sake of time as well as development resources, we decided to use the provided BiDAF attention layer instead of the attention layer described in the RNET paper as both layers produce similar attention outputs with some implementation details different between the two.

Once we have the output from the Context-to-Question layer we pass the question-aware passage representation $\{v_t^P\}_{t=1}^n$ to the self attention layer. This self attention layer works by matching the question-aware passage representation against itself and thus can gain awareness of the passage itself which can help in providing question answering context. The layer thus produces the passage representation using the following process [3]:

$$h_t^P = \mathrm{BiRNN}\left(h_{t-1}^P, \left[v_t^P, c_t\right]\right)$$

where ct = $att(v^P, v_t^P)$ is an attention-pooling vector of the whole passage (v^P):

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh\left(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P\right)$$
$$a_i^t = \exp\left(s_i^t\right) / \Sigma_{j=1}^n \exp\left(s_j^t\right)$$
$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

We then concatenate v_t^P with c_t and run this through an additional gate to control the input to the modeling layer as so:

$$g_t = \mathrm{sigmoid}\left(W_g \left[v_t^P, c_t\right]\right)$$
$$\left[v_t^P, c_t\right]^* = g_t \odot \left[v_t^P, c_t\right]$$

$\left[v_t^P, c_t\right]^*$ is then passed to the modeling layer provided by the CS224n teaching team. The output is then conditioned off of the BiDAF attention layer and the modeling layer described above. The implementation of this self-attention layer is done originally by the team.

3.4 Feature Engineering

For additional input features we decided to incorporate part of speech tags, named entity recognition tags, and dependency tags. There are two main parts involved with generating these features: running the document through spaCy and encoding the string tokens into numbers.

To run the document through spaCy, we first generated a new spaCy document tokenizer by loading the "en_core_web_sm" language model from spaCy. This language model contains the tokenizer, dependency parse, and named entity recognizer for the English language. In the process_file function of setup.py, we run each context paragraph and question through the new spaCy document tokenizer. From this document, we are able to loop through each token and construct arrays containing the text, part of speech, named entity tag, and dependency tag for each token. Each of these arrays are stored in the example which is returned by the function.

Next, we encode the features into numbers in the build_features function. For each unique feature, we assign an integer value representing the type of each feature. We do this by keeping a dictionary that contains a key that corresponds to a part of speech, dependency tag, or named entity tag and the value of each key maps to a unique integer. If the tag already exists in the dictionary, we return that value else we create a new value and add that tag to the dictionary. After constructing these arrays they get stored to the output file as specified by the build_features function. These features then get concatenated with the word indices and character indices in the forward function of the model. This code was written originally by the team.

4 Experiments

4.1 Data

We are using the SQuAD dataset. The SQuAD dataset is a set of paragraphs from Wikipedia. The questions and answers in the dataset are crowdsourced using Amazon Mechanical Turk with around 150k questions in total. Roughly half of these questions cannot be answered using the paragraph that is provided. Preprocessing is done through the setup.py file provided which will download GloVe 300-dimensional word vectors and prepocesses the SQuAD dataset for efficient data loading. Additionally, the data will be split according to the project handout to create class leaderboard rankings.

4.2 Evaluation Method

For evaluation methods we will be using F1 and EM scores as specified in the default handout project description.

The F1 score is the weighted average of precision and recall when it comes to the the predicted outputs. Precision is calculated as the ratio between the correctly predicted positive observations and the total predicted positive observations. This is calculated using the following equation, $\frac{\text{True Positive}}{\text{True Positive + False Positive}}$. Recall is the ratio between correctly predicted positive observations to all the possible observations. This is calculated using the following equation, $\frac{\text{True Positive}}{\text{True Positive}+\text{False Negative}}$. Using both of these equations we can calculate the F1 score using the following equation: $\frac{2\cdot(\text{Precision}\cdot\text{Recall})}{(\text{Precision}+\text{Recall})}$.

The EM score is defined as the percentage of questions that the model generated a predicted answer that exactly matched the actual answer. Every character of the predicted answer must match the actual answer for a certain question answer pair to be correct.

4.3 Experimental Details

In running experiments we decided to follow the following format. After running the baseline model we wanted to see what improvements we would get from solely adding character embeddings to the model. After getting the results from this model, we decided to add the self-attention layer and the feature engineering separately on top of the character embedding model to see how both perform individually on top of the baseline with the character embeddings. Finally, we added all of the different features we implemented together to see how they interacted.

Although hyperparameter tuning is important and can lead to better results for all of the models stated above, we chose not pursue this path. The reasoning being that training time for most of the models we implemented was about 12 hours with self attention running from about 16 to 24 hours dependent on the tests we were trying to run. Once we upgraded to larger computers the training times would take about 6 to 8 hours but computation costs went up so we decided to stick to model testing and not hyperparameters.

As a result, the learning rate we used was 0.5, the dropout probability was 0.2, and the hidden size was 100. For the convolution layer for the char embeddings we used a 1D filter of size 5 and for the self attention encoding layer we used 2 layers (although 3 were specified in the RNET paper).

All of the models were trained on the dev set provided by the teaching team and then tested separately on both the final leader board test set and dev set.

4.4 Results

Once we trained on the dev set and tested all of the trained models on the dev set, we took the best scoring model and tested it on the leader board test set. The following are the results from the experiments:

Table 1: Results

	Dev EM	Dev F1	Test EM	Test F1
Baseline (BiDAF)	58.360	61.478	–	–
Baseline+Char-Embeddings	59.973	63.362	–	–
Baseline+Char-Embeddings+Self-Attention	60.612	63.876	–	–
Baseline+Char-Embeddings+Features	61.284	64.379	59.036	62.171
Baseline+All	60.007	63.557	–	–

As can be seen in the table, both the self attention layer and the added features produced better results over simply adding character embeddings to the BiDAF model. In our research, we found that adding features performed the best as opposed to the self attention model as well as the final aggregate model. However, in running the feature model on the test set we ended up seeing worse results than with our dev set. This will be explored in our analysis.

5 Analysis

Figure 1 demonstrates the breakdown of questions within the evaluation set. As is seen in the graph, the largest percentage of questions are "what" questions which constitute over half of the questions in the evaluation set. The second largest group of questions comes in the "who" questions which constitutes 10%.

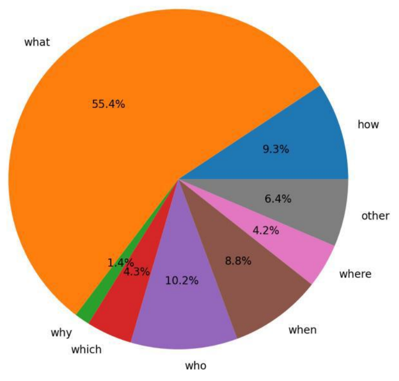


Figure 1: Breakdown of questions in the dev evaluation set

5.1 General Analysis of Models

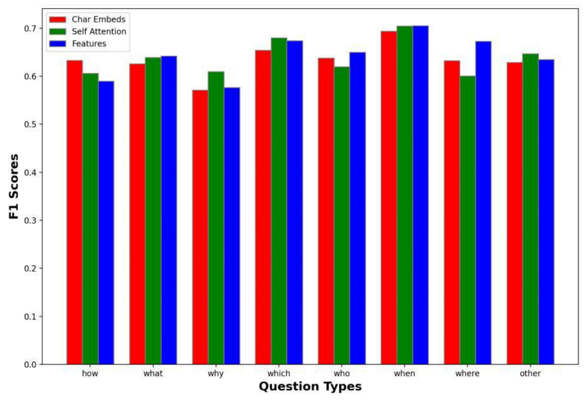


Figure 2: How each model variant performed on different question types (F1)

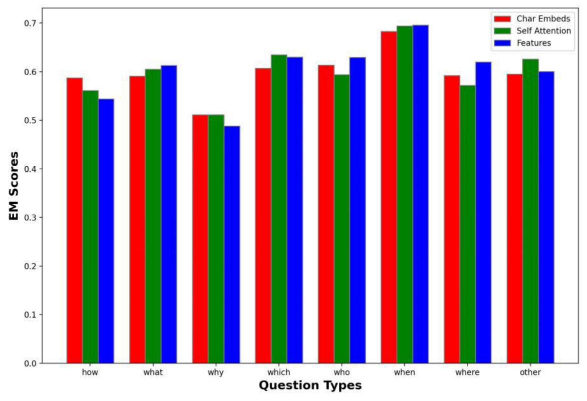
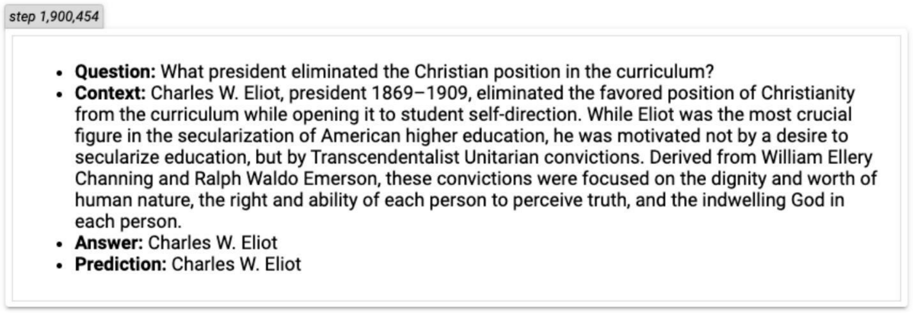


Figure 3: How each model variant performed on different question types (EM)

The figures above represent how each model performed on different question types. As noted in Figure 1, "what" questions constituted the majority of the questions in the data and we observed that the features model did do the best of all the other models on those types of questions which follows it being the best model trained. Something interesting that was noted was the performance on the "who" questions and the "where" questions by the features model. Here we substantial gains in the features model over the other models which leads us to hypothesize that the Named Entity Recognition Feature did help the model understand questions involving names and places. As a result, in questions dominated by answers requiring names as answers, we saw substantial gains over other models. We also see that the self attention model performs well on the "why" and "other" questions which leads us to hypothesize that running self attention over the context helps the model learn about the context and cues in the passage outside its surrounding window which therefore helps it solve more complex "why" and "other" questions.

5.2 Best Model Analysis



step 1,900,454

- **Question:** What president eliminated the Christian position in the curriculum?
- **Context:** Charles W. Eliot, president 1869–1909, eliminated the favored position of Christianity from the curriculum while opening it to student self-direction. While Eliot was the most crucial figure in the secularization of American higher education, he was motivated not by a desire to secularize education, but by Transcendentalist Unitarian convictions. Derived from William Ellery Channing and Ralph Waldo Emerson, these convictions were focused on the dignity and worth of human nature, the right and ability of each person to perceive truth, and the indwelling God in each person.
- **Answer:** Charles W. Eliot
- **Prediction:** Charles W. Eliot

Figure 3: An example with question, context, answer and prediction

Our Baseline+Char-Embedding+Features performed best out of all of our model that we trained having a Test F1 score of 59.036 and a Test EM score of 62.171. While this was our best performing model there are some persistent problems that occur in the test examples.

The first issue that is repeated by the model is the tendency to underweight adjectives or descriptive properties within the question. A good example of this is in the question,"What can be viewed as a **limited** *collection of instances together with a solution for every instance*?". The first sentence of the context paragraph states, "A computational problem can be viewed as an **infinite** *collection of instances together with a solution for every instance*." In this example, the italicized portions of the text are identical. Given that this part of the context is identical as the question, the weight of the adjective **limited** and **infinite** which are opposites, are not valued as much as the rest of the sentence. Due to this the subject of the sentence "A computational problem" is returned as the correct answer when in fact no correct answer is found in this passage. Another example of this is in the question,"What is the **oldest** *work of Norman art*?". The first sentence of the context question states,"By far the most **famous** *work of Norman art* is the Bayeux Tapestry, which is not a tapestry but a work of embroidery." Similar to the previous example, not enough attention is placed on the adjectives, resulting in the model returning a noun that does not answer the question correctly.

The next issue that is repeated by the model not being able to connect synonyms. One example of this is in the question,"Which directive mentioned was **created** in 1994?" The context passage states, " The UK subsequently **adopted** the main legislation previously agreed under the Agreement on Social Policy, the 1994 Works Council Directive, which required workforce consultation in businesses, and the 1996 Parental Leave Directive." In this example, the model failed to connect the verbs **created** and **adopted** which can be viewed as synonyms given the context. Another example of this occurs in the question, "What can the exhaust steam not fully do when the exhaust event is **insufficiently long**?" The context passage states, "... if the exhaust event is **too brief**, the totality of the exhaust steam cannot evacuate the cylinder." Again, in this example, the model failed to recognize that the adjectives **insufficiently long** and **too brief** are very similar.

The final issue that is repeated by the model is not being able to properly determine the noun that is connected to a pronoun. An example of this is in the question,"How did **peace** start?". The passage states,"**It** began with a dispute over control of the confluence of the Allegheny and Monongahela rivers." The model fails to recognize that **It** referrers to **war** in the previous sentence. Since it fails to recognize this, the model returns "a dispute over control of the confluence of the Allegheny and Monongahela rivers" as the incorrect answer. Another example of this issue is found in the question,"What percentage of electrical power in the United States is made by **generators**?" In this question the subject is **generators**. The answer returned by model is, "Today most electric power is provided by steam turbines. In the United States 90% of the electric power is produced in this way using a variety of **heat sources**." **Heat sources** is the subject that generates 90% of the power for the United States yet the model returns 90% as the answer for how much power **generators** create. The model has an issue where it has difficult using the context of the passage to determine which pronouns refer to which nouns, results in incorrect answer.

6 Conclusion

In this project we tackled several modifications on top of provided BiDAF model in order to boost question answering performance on the SQuAD 2.0 model. Specifically, our model added character embeddings, input features, and self attention to the model. After extensive research, it was found that the input features were able to improve upon the baseline model, with the best overall scores being an F1 Score of 64.38 and EM Score of 61.28 on the dev set. When running on the test set, this model ended up scoring a F1 score of 62.17 and and EM score of 59.04. Due to time constraints we were not able to experiment with model hyperparameters nor implement the Context-Query attention layer of the original RNET paper discussed. These explorations could lead to better results and could be experimented on for future work.

References

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

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762v5. 2017

[3] Group, Natural Language Computing. R-NET: Machine Reading Comprehension with Self-matching Networks. https://www.microsoft.com/en-us/research/publication/mcr/. 2017

[4] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[5] Yoon Kim. Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882v2. 2018.