No RNN SQuAD

Amanda Jacquez, Chase Davis, Vikas Munukutla
Department of Computer Science
Stanford University
Stanford, CA
chased@stanford.edu, vikasm2@stanford.edu, ajacquez@stanford.edu

Abstract

In this paper, we present a deep learning approach to answering questions in SQuAD, the Stanford Question Answering Dataset. We train and test on SQuAD 2.0, where the model either gives an answer if one exists or predicts N/A if an answer does not exist. Given that current state-of-the-art models fall short of human performance, our goal is to understand the challenges that come with implementing a system to answer these questions, analyze the benefits and shortcomings of such systems, and implement our own system that relies on successful techniques. Our aim was to improve a baseline BiDAF model by first incorporating small yet significant changes to the original model, such as adding character embeddings to the model. Then, we sought to change the structure of the model to a QANet model, which relies on an intricate structure based on encoder blocks and advanced attention techniques to provide answers as output to the input questions. Our team experimented with a variety of techniques, including analyzing a character-embedding system versus a QANet system, tweaking hyperparameters such as the hidden size, batch size, and the dropout rate, and experimenting with various optimizers. We found that our implementation of QANet produced mediocre results, even though we checked the dimensions and followed the paradigm as closely as we could. Instead, we ended up experimenting with a character-embedding system, modifying it strategically to analyze which factors would result in higher EM/F1 scores. Our system achieved dev scores of F1: 62.006 and EM: 65.472 and test scores of F1: 60.101 and EM: 63.716. Ultimately, the Adam optimizer with a variable learning rate outperformed the Adadelfa optimizer in the short run but fell short by the end, larger hidden size and batch size were better, and a dropout probability of 0.1 seemed to be most optimal.

1 Introduction

As automation continues to become a more significant component of modern life, interest has grown in implementing question-answering systems and implementing machine reading comprehension. Through the implementation of an automated question-answering system, users will be able to retrieve information at astonishingly faster rates, and avoid wasted time involved with sifting through extremely large passages to find a single answer to a question. The impact of automation can especially be useful in a real life context. Machine reading comprehension can be integrated in a variety of situations that include customer support, and dialog and customer relationship management. It has potential to extend to seemingly unrelated fields, such as the field of medicine. An efficient question-answering system would enable a doctor to find extremely important documents in potentially life-threatening situations at a remarkably fast rate. Question-answering is a versatile task that could assist anyone from a student sifting through Wikipedia articles, to a doctor panicking in the Emergency Room.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

There has been a large amount of work done so far to improve the accuracy of machine reading comprehension and produce the correct answers to their corresponding questions. The overall purpose of the project was to propose a new Q&A architecture called QANet, which does not require recurrent networks, but rather exclusively uses convolutions and self-attentions as the building blocks of encoders that encodes the query and context. This enables training speed to increase by 13 times and inference to increase 9 times, compared to its RNN counterparts [3]. This dramatic increase in speed enables us to train the data on a much larger data set. While recurrent networks provide statistically significant accuracy in answer predictions, they inevitably result in extremely slow training, which makes it extremely difficult to use the model in extremely large data sets.

The key idea of our approach was to solve the question-answering task through a model that achieves more reliable results than the baseline model given. Furthermore, we wanted to make the given model more efficient by replacing slower RNNs with faster CNNs while also improving on accuracy. In addition, we wanted to experiment with hyperparameters to how they affect accuracy and reduce overfitting. Through our implementation, we received the test scores: EM 60.101 and F1 63.716, respectively.

2 Related Work

A lot of work has been done to produce a question-answering model with high accuracy. As a baseline, we were given a BiDAF model that used recurrent neural networks. In order to gain more information on this baseline model, we used Bidirectional Attention Flow for Machine Comprehension to clarify our understanding of the model, which takes in a word embedding, computes temporal interactions with words through an LSTM, computes context-query attention on the result, runs the attention through bi-directional modeling layer, and finally outputs start and end probabilities for the answer [1].

In our work to update this model to a more advanced framework, our primary source of inspiration was QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. This paper is related to the paper on BiDAF because it aims to remove a bottleneck from the model - at the time, most Q&A models were primarily based on RNNs with attention. However, RNN-based models are often slow for training and inference due to the sequential nature of RNNs, which is why the authors of the QANet paper aimed to use only convolutional layers and attention techniques, effectively removing the bottlenecks that come with using an RNN-based approach [3]. We also used Attention is All You Need in order to understand how to incorporate multi-head attention into our QANet - a crucial aspect of the model that allows the model to jointly attend to information from different representation subspaces at different positions [7]. With a single attention head, averaging inhibits this [7]. Futhermore, we also used Layer Normalization in order to understand the layernorm and feed-forward layers within the QANet's encoder block. The encoder block seemed to be the most crucial aspect of the QANet since it is used on the embeddings as well as in repetition on the result from the Context-Query Attention.

As the QANet is a relatively new model, and the acronym, Question Answering, limits the model to question-answering tasks, there do not exist papers that apply our methods to different tasks. However, there does exist literature that attacks the question-answering dilemma through different means. A common method to predict answers to questions has been done through two components: (1) the Document Retriever module for finding relevant articles and (2) a machine comprehension model, Document Reader, for extracting answers from a single document or a small collection of documents [4]. Another common method involves the usage of neural network models. These systems include GRU and LSTM units that allow RNNs to handle longer texts required for question-answering [5]. Using several deep learning models is a very common method, and although this method works, it is not efficient if trained on an extremely large dataset. It is for this reason that we implement our model with a QANet, which also improves the accuracy of predictions.

3 Approach

Our first step was to implement character embeddings and transition to a more advanced model that uses the original word embeddings along with the word embeddings produced by sending the character embeddings through a convolutional neural network. This is integral to our later

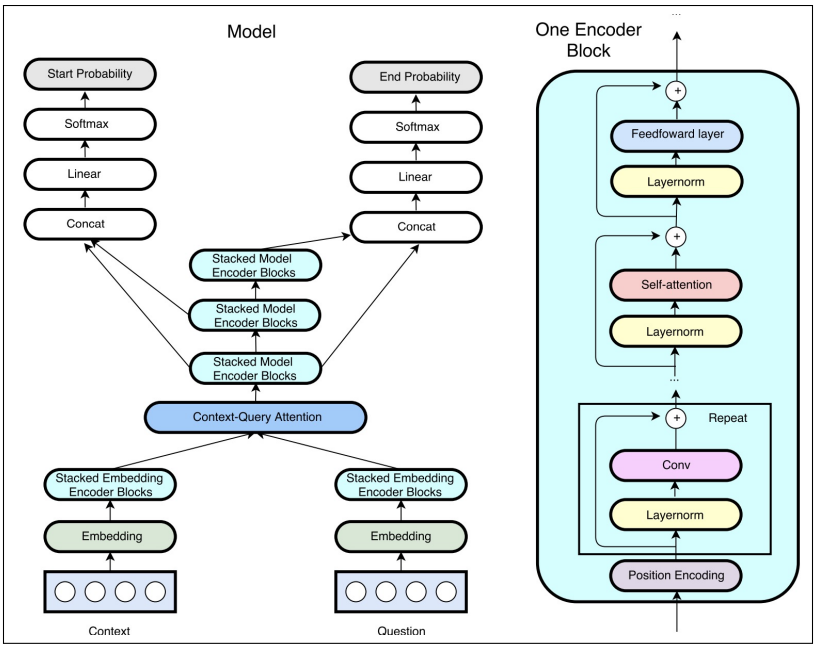

Figure 1: Full QANet model

implementation of the QANet, which is completely reliant on convolutional neural networks.

Referring to **Figure 1**, we have updated the current code to include the updated embedding layers and the stacked encoder blocks with Contex-Query Attention. Our next step is to update the attention layer to a Dynamic Coattention Network for a higher level of optimization.

Our baseline model was the one provided to us as a starting point for the default project. On a high level, the baseline model uses highway layers on word embeddings before passing them into an RNN. Then it uses a basic bidirectional attention flow model to compute attention scores, ultimately computing the most likely answer to SQuAD reading comprehension questions. The model achieved a dev EM score of 56.88 and an F1 score of 60.46. A more detailed description of the baseline BiDaF model is presented in the default project handout [2].

We added to the baseline by modifying some of the files running the baseline model. Specifically, we modified train.py to retrieve tensors from a pre-stored character embeddings file, and passed character indices into the BiDaF model, which originially only took in word indices. We updated the BiDaF model by changing the Embeddings class in layers.py, which now also initializes an embedding layer for characters. In addition, we modified the forward function of the Embeddings class to run a CNN on the character embeddings, combining the individual character vectors into a word vector of the same dimension as the GLoVe vectors, 300. Finally, we concatenate this result to the original word embedding to produce an overall embedding which has dimension 600. Our approach is different from the QANet paper here because the QANet paper uses character embedding dimension 200, so their overall embedding is of dimension 500. However, the difference in embedding dimension would not produce a large enough discrepancy with the original model.

Next, we implemented the encoder blocks. The first thing that happens in the encoder block is that the input is passed into a positional encoder that relays information about relative or absolute positions of tokens in the input. The encoder block consists of a number of convolutional layers, in our case 4, where the output of one layer is fed as the input of the next. The dimension of the input and output are both hidden size - in our case, our hidden size was 96 so our input and output were of dimension 96 throughout the convolutional layers. After the convolutional layers, the input is passed into a self-attention layer that uses multi-head attention. Finally, the input is then passed into a feedforward layer. At each step the input is added to the output - for an input x and a given operation f, the output is f(layernorm(x)) + x, where the layernorm function in-

dicates that the input mini-batch undergoes layer normalization before proceeding to the next layer [3].

We used https://towardsdatascience.com in order to borrow implementations of general attention, multi-head attention, and the positional encoder [8]. The positional encoder is used to inject some information about the relative or absolute position of the tokens in the sequence [7]. Furthermore, we used the feed forward layer from https://www.deeplearningwizard.com [9]. We decided that we wanted to use the neural network version of the feed forward layer instead of the single-layer version - the neural network maps the input to a hidden layer of smaller dimension, and then maps to an output layer of the same dimension as the input. The feed forward net is a non-linear transformation since it uses a non-linear layer.

Every other aspect of the QANet was coded by us. We implemented the Embedding Encoder class, using default values dmodel = 96, number of heads = 4, convolution number = 4, kernel size = 7, and dropout probability = 0.1. Using this Embedding Encoder class, we also implemented a Model Encoder class that takes in the output of the context-query attention and passes it into 3 blocks of 4 encoder stacks each with convolution number 2, sharing weights between the 3 blocks like the paper mentions. The original paper actually uses 7 encoder stacks in each block, but we used 4 to make the model run faster. Finally, we pass this into the QANet output layer, which takes in M1, M2, and M3, the results from each model encoder block. The output layer concatenates M1 and M2 together, and after running the result through a linear layer and softmax function, produces the probability distribution for start probabilities. The output layer also concatenates M2 and M3 together, and after running the result through a linear layer and softmax function, produces the probability distribution for end probabilities. In other words, these probabilities are represented by $p^1 = softmax(W_1[M_0; M_1]), p^2 = softmax(W_2[M_0; M_1])$. These are then used to compute loss, which is given by the formula L(θ) = $\frac{-1}{N} \sum_{i}^{N}$ [log($p^1_{y^1_i}$) + log($p^2_{y^2_i}$)].

4 Experiments

We used the Stanford Question Answering Data set, or SQuAD, in our model. SQuAD, the Stanford Question Answering Data set, is a data set that is comprised of questions posed on an extremely large set of different Wikipedia articles. This data set contains questions and their corresponding answers that are a segment of text from the relevant passage. There exist over 500 articles with over 100,000 corresponding question-answer pairs. The first few pages of the default project handout [2] contain an in-depth description of the data set we use in training and testing our model.

The main evaluation metrics we used were the ones provided, the EM and F1 scores. The EM score, otherwise known as the exact match score, is a strict metric that requires predictions to be exactly the same as the actual output. The F1 score is a less strict metric - it is the harmonic mean of the precision and recall. In other words, the closeness of the prediction to the actual answer determines the F1 score, rather than an exact, word-to-word match. When a question has no answer, both the F1 and EM score are 1 if the model predicts no-answer, and 0 otherwise. We evaluate F1 and EM scores with equal importance, looking for increases in both metrics as we improve our model.

We primarily ran our experiments by training on the full training data set, using TensorBoard to carefully track our progress for each implementation. We noticed that the baseline implementation took around 8 hours to train, while our first implementation with character embeddings took more than twice as long, clocking in at around 17 hours. The QANet model was projected to take even longer than that, clocking in at close to an hour per epoch even with a lower number of model blocks. The more we added to the baseline, the longer the process took for the model to run on the virtual machine.

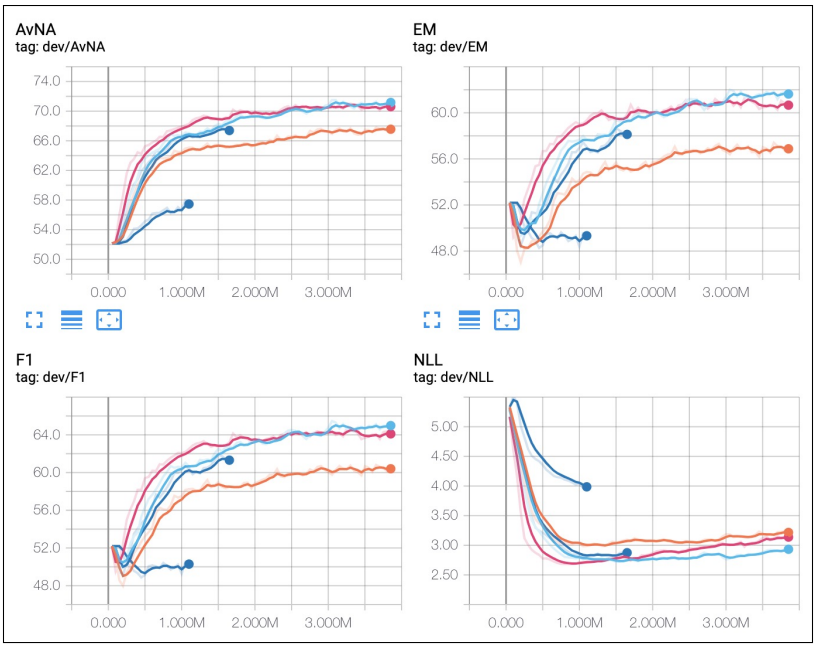

Figure 2: Results after implementing QANet

Our experimental process can be subdivided into two processes. We will first discuss the process we undertook in the milestone phase of the project. We will then discuss the process we undertook in the final phase of the project.

After running our enhanced model for 17 hours, we tested the model on the dev set. After the code finished running, we received an EM score of 62.01 and an F1 score of 65.47. Initially, our implementation of the character embeddings was incorrect because we incorrectly concatenated two tensors together along the wrong axis, which caused the model to perform worse than the original model. However, after fixing this line, our model was much better than the original model, which surprised us because we didn't expect to see such a drastic improvement.

We will now discuss the final implementation of our model. **Figure 2**, above, shows the baseline model versus the enhanced character-embedding model, where orange represents the baseline, light blue represents the enhanced character embedding model with the ADADELTA optimizer, red represents the enhanced character embedding model with the ADAM optimizer, and the upper dark blue represents our implementation of QANet. As can be seen, error rates went down at a faster rate in our character-embedding models as well as QANet than in the baseline, and their accuracies went up faster than in the baseline. Furthermore, the initial dip in accuracy at the beginning of training was not as pronounced in the enhanced model as it was in the baseline.

The figure also includes a number of other experiments that we performed hoping to enhance our model. As shown by **Figure 2**, the QANet implementation unfortunately did not perform as well as the enhanced character embedding implementation or the baseline, so we decided to terminate the operation early. We also tried using only the first half of the model, stopping after the context-query attention layer in the QANet model and then running the remainder of the BiDAF model. This proved unsuccessful, as shown by the bottom dark blue line in **Figure 2**. We performed a number of operations to improve performance, one of which was to change the number of heads in the multi-headed attention layer as shown by the upper dark blue line. However, this did not result in any improvement of the QANet. We also experimented with our model by changing the optimizer of the enhanced character embedding model from an ADADELTA optimizer to an ADAM optimizer, as shown by the red line in **Figure 2**. This originally provided promising results; however, it ultimately ended up under-performing the character embedding with ADADELTA optimizer. Ultimately, we decided to use our character embedding implementation as our best model, which was a non-PCE test submission that resulted in scores of F1: 60.101 and EM: 63.716. An interesting aspect to note

about the character-embedding model versus the baseline model is that the baseline model ends in a flat slope while the character-embedding model is still increasing by the end; this implies that if the model kept running for more iterations, then it may have improved. The character-embedding model has scope to further improve through additional layers on top, or by tweaking hyperparameters; it has more potential than the BiDAF model. Throughout the course of the project, we ended up not implementing a Dynamic Co-attention Network, or DCN, as the paper suggested, as the paper implies that a DCN only resulted in a slight increase in performance. To be conscientious of the time allotted for the project, we decided as a team that we would not implement the DCN, but rather spend more time optimizing the character embeddings and implementing the QANet.

Our quantitative results for QANet were not as good as we expected, whereas we were surprised by how large of a difference in accuracy we were able to achieve by simply appending the character embedding to the original word embedding. Our analysis details some potential reasons as to why our QANet did not perform as expected.

5 Analysis

Ultimately, we found that our implementation of the baseline enhanced with character embeddings performed the best of all models that we tested. We tested this model with two different optimizers: the ADAM optimizer and the ADADELTA optimizer. For the ADAM optimizer, we experimented with $\beta_1 = 0.8$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, a learning warm-up rate scheme with an inverse exponential increase from 0.0 to 0.001 in the first 1000 steps, and then a constant learning rate for the remainder of training, and L2 weight decay on all the trainable variables, with parameter $= 3 \times 10^{-7}$ [3]. We drew inspiration for these values from the QANet paper. We found that using ADAM optimizer performed in better results initially and seemed to learn faster than ADADELTA, and by a significant amount as well, which is why we kept it running as it was training. This is most likely a result of the fact that ADAM uses an added momentum variable while ADADELTA does not. This causes ADAM to learn faster than ADADELTA; however, this may have up negatively effected ADAM in the long run as it began to under-perform ADADELTA around 2.5 million iterations as it may have begun to over-fit the data.

When implementing the QANet, we found that other teams on piazza noticed increased performance when setting the number of hidden layers in their model to 96 and using 4 heads instead of 7 heads for the multi-headed attention layer, thus we decided to use these parameters for our model. However, our QANet still did not perform as expected. We believe that the discrepancies in our model were in the implementation of the convolutional layers as well as the multi-headed attention layer. We did not use depth-wise convolutional layers despite the paper using depth-wise convolutional layers, and thus we ran into memory errors when training on a batch size larger than 10. We also believe that the multiplications in the multi-headed attention layer could be taking up a large amount of memory and thus resulted in memory deficiencies when the batch size is larger than 10.

Our error analysis shows that the character embedding model outperforms the baseline model in terms of subject-verb agreement. We note this as we see an increase in performance when the answer to a question is the subject of a sentence, which is indicated by a verb. For example, the answer to the question *What German ruler incited Huguenot immigration* is Frederick William. This is indicated by the sentence *Frederick William, Elector of Brandenburg, invited Huguenots to settle in his realms, and a number of their descendants rose to positions of prominence in Prussia.* The character embedding correctly associates "invited Huguenots" with the answer, while the baseline model indicates that the answer is not in the passage. This is most likely because the word embedding in the baseline model is looking for a synonym to the word incite, whereas the character embedding finds a similar spelling to the word incite and invite. This prevents the character embedding model's performance from being inhibited by not being able to find the exact word used in the question, like the baseline model most likely is.

Some other issues with the baseline model is in predicting N/A. If the question is exactly the same as some phrase in the context except for one word, which is completely different, then the predicted answer should be N/A, but the baseline model looks at the overall temporal interactions between words and sometimes misses this minute difference. The incorrect word in the question

could be wrong for a variety of reasons - it could be misspelled, the word could be referring to a different subject (for example, the subject of the question for one of the examples was a garden, but the prediction answer's subject was a university library), or the differentiating word could be an antonym to the word in the context. In all cases, character-level convolutions have a higher chance of capturing these minute differences because of the character-level analysis that was previously lacking, thus possibly increasing the F1 and EM scores overall. However, both models are still somewhat lacking in non-N/A predictions, which happen to be incorrect at a similar rate. The reason some non-N/A predictions are incorrect is because the question is too different from the sub-phrases within the context; for example, the model is not able to understand that if a question asks if something is "too brief", then this equivalent to a sub-phrase in the context that states that something is "insufficiently long".

6 Conclusion

In our project, we found that question answering is a possible, but challenging task. Overall, our project was relatively accurate in predicting the answers to their corresponding questions. Though this task inevitably came with its challenges, we learned how to create a model, wherein we were uncertain on the predictions it would make. This would more closely align to the projects we would face in the future, whether we approach artificial intelligence tasks in research or industry in the future of our careers. Throughout the process, we learned the framework required to implement a QANet. In the research paper we based our project on, there was a diagram that clearly outlined the framework that defined a QANet in its implementation.

We were successful in implementing the character embeddings, which allowed our model to preform significantly better than the baseline model given. However, we did not implement a QANet that made our predictions more accurate than those of the refined character embeddings model. For future work, we would continue to work on the QANet such that it would outperform the predictions of the character embeddings. We would also implement a DCN as specified by the paper to further polish the predictions by the model, since DCN attention can provide a little benefit over simply applying context-to-query attention [3]. This would be done by further analyzing the layers that we implemented in the QANet and test whether we need to add more or modify the current layers we have in our current implementation. Other than the QANet, further work could be devoted to tuning hyperparameters that could, in turn, increase the performance of our model.

7 References

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

[2] CS 224N Default Final Project: Question Answering on SQuAD 2.0. 28 Feb. 2019, web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf

[3] Yu, Adams Wei, et al. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. 23 Apr. 2018, arxiv.org/pdf/1804.09541.pdf.

[4] Chen, Danqi. Reading Wikipedia to Open-Domain Questions . Reading Wikipedia to Open-Domain Questions. http://aclweb.org/anthology/P17-1171.

[5] Stroh, Eylon, and Priyank Mathur. Question Answering Using Deep Learning . Question Answering Using Deep Learning. https://cs224d.stanford.edu/reports/StrohMathur.pdf.

[6] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. CoRR, abs/1607.06450, 2016. arxiv.org/abs/1607.06450.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017a. arxiv.org/abs/1706.03762.

[8] Ng, Ritchie. "Feedforward Neural Network with PyTorch¶." Deep Learning Wizard, www.deeplearningwizard.com.

[9] Lynn-Evans, Samuel. "How to Code The Transformer in Pytorch." Towards Data Science, Towards Data Science, 27 Sept. 2018, towardsdatascience.com.