Question Answering on SQuAD 2.0

Revanth Bandaru[1] **Manasa Rangineni**[2] **Sowmya Davuluri**[3]

Data Science Department

University of New Haven

Abstract

The key job of Reading Comprehension is to answer the question using the information in the provided context. So, here we have implemented the standard Stanford Question Answering Dataset (SQuAD 2.0) version 2.0. So here we have taken the baseline model Bi-Directional Attention Flow (BiDAF). To improve on the baseline model provided, we need to alter the Bi-Directional LSTM(bi-LSTM) at the heart of the model with a Gated Recurrent Network(GRU) and look for any performance improvements. For achieving better results, we tried changing the hyperparameters like learning rate and drop probability. It came out that performance wise GRU shows better results than bi-LSTM.

1 Introduction

In 2016, Rajpurkar et al. built a dataset designed for the task of Extractive Question Answering: the Stanford Question Answering Dataset (SQuAD 1.1). Later on the updated version of SQuAD 1.1 came into picture which includes unanswerable questions, which make the problem more challenging. So in our project we are using this updated version SQuAD 2.0. by acknowledging that our model must not only rely on answering questions correctly when possible, it should also be able to answer when no answer is provided in the passage(context). So, this project idea came into picture because with the progress in Natural Language Processing and Deep Learning it has now become even more possible to achieve promising results on a variety of tasks in the text and image domains.

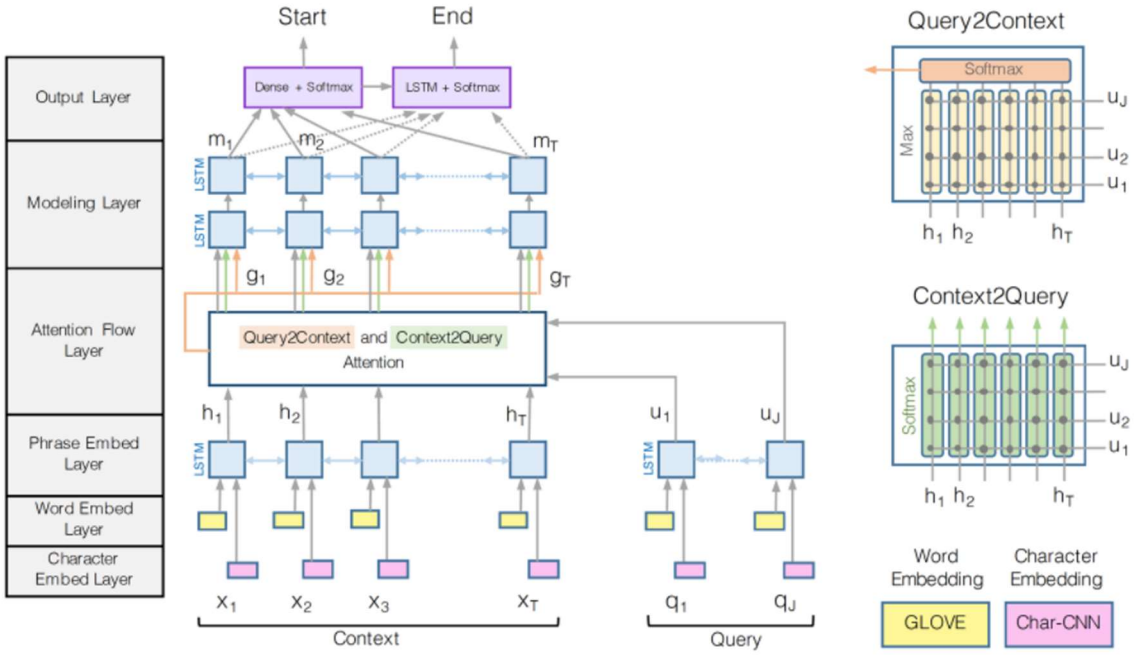
Recurrent Neural Networks (RNN)s were capable of handling variable length input sequences by using a recurrent, shared hidden state. But RNNs suffered from the problem of vanishing and exploding gradients. LSTMs came into picture providing solution for this problem, by using bounded non-linearities to reduce the issue of vanishing and exploding gradients. Since then GRUs have also been found which helps in reducing the computation cost of LSTMs while keeping almost the same features.

As mentioned above, the Stanford Question Answering DataSet (SQuAD) is considered the gold standard in machine comprehension experiments. This dataset was enhanced by adding 50000 unanswerable questions in addition to the exclusively answerable questions available before This new dataset is called SQuAD 2.0. To do well on this dataset, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering.

To provide Question Answering results from this dataset, there have been several models proposed. One of them is which uses a Bi-Directional Attention Flow (BiDAF) model. the BiDAF includes character-level, word-level and contextual embeddings and uses the attention flow in both directions to get a query-aware context representation.

2 Approach

First let's look into the architecture of Bi-Directional Attention Flow (BiDAF) network. The baseline model provided is built on Bi-Directional Attention Flow, as described.



Architecture of BiDAF

We can see that this network has multiple layers, connected sequentially, Let's go through each layer:

Embedding Layer : After the pre-training process, the transformer outputs the full sequence of hidden-states corresponding to the last attention block. The hidden states are the concatenated vector representation of the the context and questions T1, T2 · · · TN ∈ Rh, which converts the input word indices into word embedding for both the question and the context.

Encoder Layer: It Uses a bi-Directional LSTM to cover the temporal dependencies between timestepsof the embedding layers outputs.

Attention Layer: Attention layer is the core layer that makes sure that the attention flows both ways - from Context-to-Question and Question-to-Context.

Modelling Encoder Layer: Modeling layer consist of two layers of bi-directional LSTMs with hidden size h to scan the vector space representations after the highway layer. It is designed to better capture the relation between context and question, so this take in the temporal information between the context representations given the background of the questions. This happens as this appears after the attention layer were the cross-conditioning between questions and context has already taken place.

Output Layer: This provides a vector of probabilities for each position in the context. This is achieved by using a softmax function.

The loss function is the cross-entropy loss for the start and end locations. Additionally, the losses are average across the batch and the Adadelta optimizer is used to minimize the loss.

As the aim was to investigate the how things would change due to the tweaks to hyper- parameters. As indicated in the above section, we focused on changing two of the hyper-parameters :learning rate and drop probability. The learning rate only affected the Adadelta optimizer, but the drop probability change was more widespread as every layer of the model used the drop probability.

3 Experiments

This paper uses the SQuAD 2.0 dataset Contrary to SQuAD 1.1, this version also includes unanswerable questions. An example input is a pair (context, question) and an output is the corresponding answer or 'No Answer' if there is none. Both the context and the question can be represented as strings of variable lengths. The answer, if there is one, can be represented as a substring of the context or equivalently as a pair of start and end word indices of the context.

3.1 Data

The SQuAD dataset contains context, question, answer triplets. Each context is a paragraph, passage or document - which is excerpted from Wikipedia. The question is the question to be answered based on the given context. The answer is a set of textual words derived from the context that provide an answer to the question posed on the context. It is possible for the model to not have an answer - making a no-answer prediction.

The SQuAD 2.0 dataset used with the baseline model has three splits : train, dev and test. The training dataset is used to train the model. The dev dataset is used to fine tune the model parameters. The test dataset is used to evaluate how well the model got trained.

4 Evaluation Method

There are three metrics used for evaluating the model accuracy, two of which are provided. They are

• Exact Match is a binary measure (0 or 1) checking if the model outputs match one of the ground truth answer exactly

• F1 score is the harmonic mean of precision and recall for the answer words.

Exact Match EM : EM is a binary measurement of whether the percentage of output from a system exactly matches the ground truth answer(the proportion of questions that are answered in exact same words as the ground truth)

F1 score : F1 score is a harmonic mean of precision an recall. For each question, precision is calculated as the number of correctly predicted words divided by the total words in the predicted answer. Recall is the number of correctly predicted words divided by the number of words in the ground truth answer. The F1 score is averaged among questions. Also, since there are at least 3 answers for each question in the set, the second answer is consideredto be the human prediction and the others are considered ground truth answers.

4.1 Experimental Details

All the models were trained for 30 epochs. The training hyperparameters are unchanged from the baseline: Adadelta optimizer with the same learning rate and exponentially weighted moving average of the model parameters during evaluation.

The training ran much faster than the baseline model, we continued to tune the hyper-parameters based on the GRU settings.

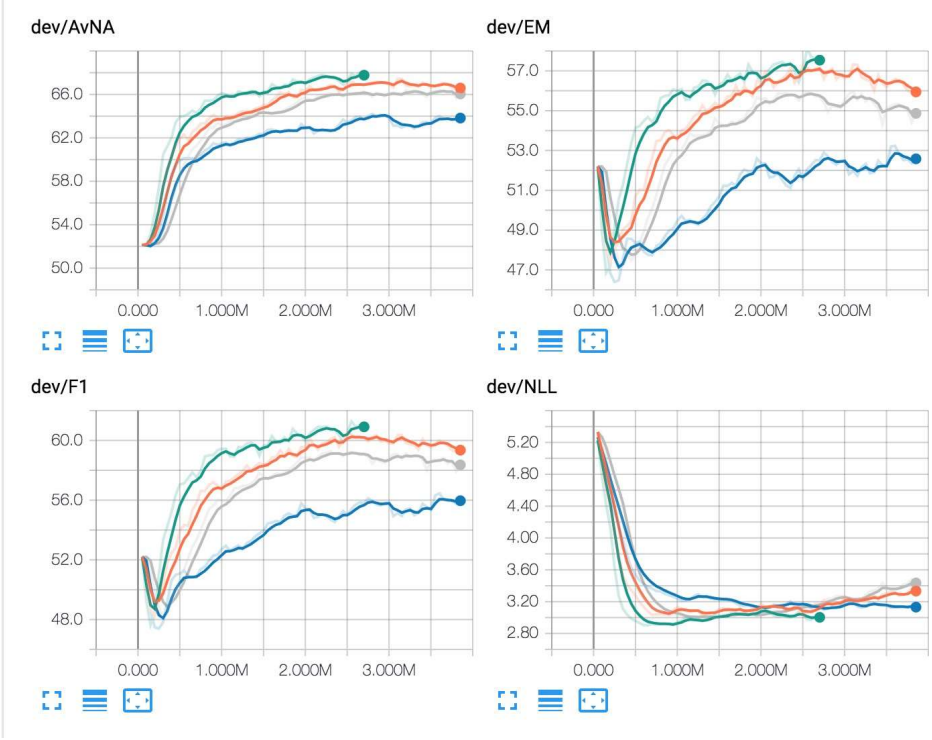
1. Third run was to see the effect of reducing the learning rate on the GRU.

2. Fourth run was with the learning rate at its original value and then increasing the drop probability for the GRU network.

]

4.2 Results

The final results for all the runs after 30 epochs, The training hyperparameters are unchanged from the baseline: Adadelta optimizer with the same learning rate and exponentially weighted moving average of the model parameters during evaluation



The above graphics contains the overall snapshot of the Tensor board that compared all the four runs described above. The legend for the graphs are as :

1. Baseline Model with bi-LSTM : Orange Lines

2. Model with GRU replacing bi-LSTM : Green Lines

3. GRU model with Learning rate: 0.5 -> 0.2 : Gray Lines

4. GRU model with Drop Probability: 0.2 -> 0.5 : Blue Lines

We can add some further comments based on the graphs seen above :

1. Given that the GRU model was running in such a shorter run time, I decided to pursue all the remaining runs for hyper-parameter changes using the GRU model itself. This allowed me to perform more experiments in the short amount of time available to me. Thus, the runs for the learning rate changes and the drop probability changes were run with the GRU model in place.

2. Even though the GRU curves were better than the baseline model curves in all the graphs, the difference was not very much. This was showcased by the fact that as soon as we change any of the hyper-parameter in my experiments (either by changing the learning rate or the drop probability), the baseline model did better than them in all cases as well.

3. In the case of the learning rate reduction, the results are much worse as compared to the only GRU case, but only slightly worse than the baseline model. In all these cases, we see that the reduction in the learning rate does cause the EM and F1 updates to slow down as well. This is reflected in the larger number of iterations required in this case.

4. In the case of the drop probability, we do see that increasing the drop probability just simply craters the performance of the model in all curves as well. We also see the curve not following a smooth pattern indicating the disruption caused by increasing the drop probability too high.

5 Analysis

There are three main analysis statements we can make from the results seen above.

1. The first is that the GRU with good hyper-parameter values was able to perform better than the baseline model having the bi-LSTM. This was a hunch I could follow as the GRU is known to have less computation requirements than the bi-LSTM.

Looking into the details of the main differences between the LSTMs and GRUs, I found that both GRUs and LSTMs have the same goal of tracking long term dependencies while mitigating the vanishing and/or exploding gradient problems so prevalent in Recurrent Neural Networks. The LSTM does this using the input, forget and output gates. Input gate regulates how much of the new cell state to keep. Forget gate regulates how much of the existing memory to forget. Output gate regulates how much of the cell state to be exposed to the next layers. Both LSTMs and GRUs have the ability to keep memory/state from previous activations allowing them to remember features for a long time and allowing back-propagation to happen through multiple bounded non-linearities, which reduces the issue of vanishing gradients.

On the other hand, the GRU operates using a reset gate and the update gate. Reset gate sits between previous activation and the next candidate activation to forget previous state. Update gate decides how much of the candidate activation to use in updating cell state.

That is why LSTMs and GRUs perform similarly but the reduced computation requirement of the GRUs helped them reduce the iteration time and provide better performance in this case.

2. The next one was the fact that the learning rate did not seem to effect the result by much. This is true of the Adadelta optimizer, which is derived from the Adagard optimizer, whose major feature is that the learning rate does not need to be specified. These optimizers have been very successfully been used to train GLOVE word embeddings as well. The Adadelta optimizer is used more as it

reduces the tendency in Adagard optimizers to lose the learning rate as the squared gradients in their denominators keep increasing.

We should note though that even though the expectation was not to see much change due to changing the learning rate hyper-parameter, the curves of my experiment do show the slow down caused by reducing the value of the learning rate.

3. Finally, the reason the run with the drop probability increased seemed to show that not only the performance cratered but also the curves were extremely shaky. This can be explained by the fact that the higher drop probability caused higher number of updates to be dropped arbitrarily. Some of these updates were important to the learning and so the learning trends kept getting affected by these drops. These drops also were intrusive enough that the EM and F1 values were jumping all over the place rather than following a smoother curve.

6 Conclusions

Our results shows that our initial estimation was correct and the GRU network not only provided us a more efficient and faster model but provided results that indicated that it was the better network for this model working on this dataset. This adds to the fact that when we are designing neural network models and NLP language models we should definitely try all the different kinds of networks and experiment and see which might be the best fit for our case. The baseline model, though published in literature, was overcome with another model with a slight modification.

We also found that performing a hyper-parameter search of the best possible values for the parameters is extremely important, As we got better results after tuning the model with various sets of hyperparameters. This work can actually be furtherly investigated and modified.

Future Work

We can say that this project can be altered by taking a different baseline model like well known **BERT language model,** even we can try with some pre-trained models. We can also try using different hyperparameters like momentum and weight decay. Also we can try other regularization techniques like L1 and L2.
For future development, we hope to incorporate more additional features to embedding output, such as name entities, POS tag and the question-context matching features, to further boost its performance.

Acknowledgments

We would like to thank our Professor Vahid Behzadan for giving us a chance to explore into the field of Natural Language Processing.

References

[1] Zhang Rajpurkar and et al Lopyrev. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250, 2016, 2016.

[2] Rajpurkar, Pranav, Jia, Robin and Jiang, Percy. (2018) Know what you don't know.: Unanswerable questions for squad. arXiv:1806:03822, 2018.

[3] Seo, Minjoon, Kembhavi, Aniruddha, Farhadi, Ali and Hajishirzi, Hannanah. (2016) Bidirectional attention flow for machine comprehension. arXiv: 1611.01603, 2016.

[4] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[5] Lin, Zhouhan, Feng, Minwei, Santos, Cicero Noguiera dos, Yu, Mo, Xiang, Bing, Zhou, Bowen and Bengio, Yoshua (2017). A Structured Self-Attention Sentence Embedding. in International Conference on Learning Representations (ICLR), 2017.

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

[7] Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. (2015) Neural machine translation by jointly learning to align and translate. ICLR, 2015.

[8] Wang, Shuohang and Jiang, Jing (2016). Machine comprehension using match-lstm and answer pointer. arXiv:1608.07905, 2016.

[9] Bolei Zhou, Alex Andonian, Aude Oliva, and Antonio Torralba. Temporal relational reasoning in videos. In Proceedings of the European Conference on Computer Vision (ECCV), pages 803–818, 2018.

GITHUB Repository Link:

https://github.com/revanth99/Question-Answering-on-SQuAD-2.0