Question Answering on SQuAD 2.0

Mayukh Majumdar
Stanford University
California, CA 94305.
maymaj@stanford.edu

Abstract

In this paper, we look at enhancing the results obtained by a baseline model for reading comprehension style question answering on the Stanford Question Answering Dataset (SQuAD 2.0). The baseline model is based on Bi-Directional Attention Flow (BiDAF). In order to improve on the baseline model provided, we first try to change the Bi-Directional LSTM(bi-LSTM) at the heart of the model with a Gated Recurrent Network(GRU) and look for performance improvements. To get further improvements, we attempt to study the effects of changing some hyper-parameters such as learning rate and the drop probability. We find that the GRU performs better than the bi-LSTM in all the metrics. The reduction in learning rate seems to only increase the time taken, but the increase in the drop probability has the effect of cratering the performance as well as disrupting the updates in a non-smooth manner.

1 Introduction

As our human interactions with machines increase on a daily basis, one of the most important issues to resolve has been the area of Machine Comprehension and Question Answering. With the progress in Natural Language Processing (NLP) 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 an architecture that handled variable length input sequences by using a recurrent, shared hidden state. Yet, even though they have been around for much longer, RNNs suffered from the problem of vanishing and exploding gradients. This problem was resolved by moving to LSTMs (as well as bidirectional LSTMs) which used bounded non-linearities to reduce the issue of vanishing and exploding gradients. Since then Gated Recurrent Networks have also been found which reduce the computation cost of LSTMs while keeping almost the same features. That is why this became the starting investigation point when we were looking for options to increase the performance of the baseline BiDAF based model provided.

2 Related Work

Availability of large datasets has been one of the pillars for the fast advancement and quick iteration in the fields of Machine Comprehension models. One of these datasets, 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 has been [3] 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. Other folks using the attention mechanism have been [6], which dynamically updates the attention weights given the query and context as well as the previous attention, as well as [7] who reverse the direction of the attention for SQuAD - attending on the query words as the context RNN progresses.

Given the aim of the exercise was to find ways to improve the performance of the baseline model, I started investigating other networks that could be used similar to the bidirectional LSTM at the heart of the baseline model. [8] and [9] provided me some basic background in the similarities as well as differences between LSTMs and GRUs, which provided me the hunch to use GRUs instead of the bi-LSTM. Reading through [10], [11], [12] gave some more ideas on how to investigate the hyper-parameters and focusing on learning rate and drop probability in particular.

3 Approach

The baseline model provided is built on Bi-Directional Attention Flow, as described in [3]. One can read the reference paper for details but the broad layers of the baseline model are :

Embedding Layer : which converts the input word indices into word embedding for both the question and the context.

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

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

Modelling Encoder Layer: This uses a 2-Layer LTSM to 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.

Since the goal of the experiments was to find changes which would provide a performance improvement, I focused on the Encoder layer which has the bi-directional LSTM. As indicated before, given that the GRU is a close replacement for LSTMs with a better computational profile, I decided to perform the experiments by replacing the bi-LSTM with a GRU. The replacement worked fairly well as even the arguments to the model are also similar between GRU and LSTMs.

After that, the aim was to investigate the how things would change due to the tweaks to hyper-parameters. As indicated in the above section, I 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.

4 Experiments

The baseline model and all the other models were run on the same VM with 6 GPUs(NV6) on the Microsoft Azure platform - so as to keep equality in terms of hardware infrastructure for comparison - and the results are discussed in the following section.

4.1 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.

4.2 Evaluation Method

There are three metrics used for evaluating the model accuracy, two of which are provided in [1]. They are :

Exact Match FM : This metric measures the percentage of predictions that exactly match one of the ground truth answers.

F1 score : This metric measures the average overlap between the predicted answer and the ground truth answers. The prediction and the ground truth are considered to be bags of tokens and the F1 computed on them. The maximum F1 over all the ground truths is taken for one question and then averages over all the questions.

Also, since there are at least 3 answers for each question in the set, the second answer is considered to be the human prediction and the others are considered ground truth answers.

AvNA : This stands for Answer vs No Answer and it measures the classification accuracy of the model when only considering its answer (any span predicted) vs no-answer predictions.

4.3 Experimental Details

There were 4 different experiments that were run.

1. First run was with the default configuration with the baseline model.

2. Second run was with the bi-LSTM network in the model with a GRU network. Since the GRU training ran much faster than the baseline model, we continued to tune the hyper-parameters based on the GRU settings.

3. Third run was to see the effect of reducing the learning rate from 0.2 to 0.5 on the GRU.

4. Fourth run was with the learning rate at its original value of 0.5 and then increasing the drop probability from 0.2 to 0.5 for the GRU network.

The following settings provide the default model configuration that was run for the baseline model :

Size of char embedding = 64

Size of the Glove vectors = 300

Max number of words in a training answer = 30

Max number of chars to keep in a word = 16

Max number of words to keep from a question = 50

Max number of words in a question = 100

Max number of words in a paragraph = 1000

Eval Steps = 50000

Learning Rate = 0.5

Epochs = 30

Drop Probability = 0.2

Maximum Gradient Norm for gradient clipping = 5.0

Seed = 224

Decay rate for exponential moving average = 0.999

After running the above configuration on a NV6 machine in the Microsoft Azure platform, I found the time to train runs overnight for about 11 hours and 35 minutes.

4.4 Results

The final results for all the runs after 30 epochs were as follows :

1. For the Baseline model run we got : Dev NLL: 03.35, F1: 59.17, EM: 55.67, AvNA: 66.48

2. For the model with the GRU replacing the bi-LSTM run, we got : Dev NLL: 03.01, F1: 60.93, EM: 57.49, AvNA: 67.79

3. For the model with the GRU and the learning rate reduced from 0.5 to 0.2. we got : Dev NLL: 03.44, F1: 58.41, EM: 54.90, AvNA: 66.11

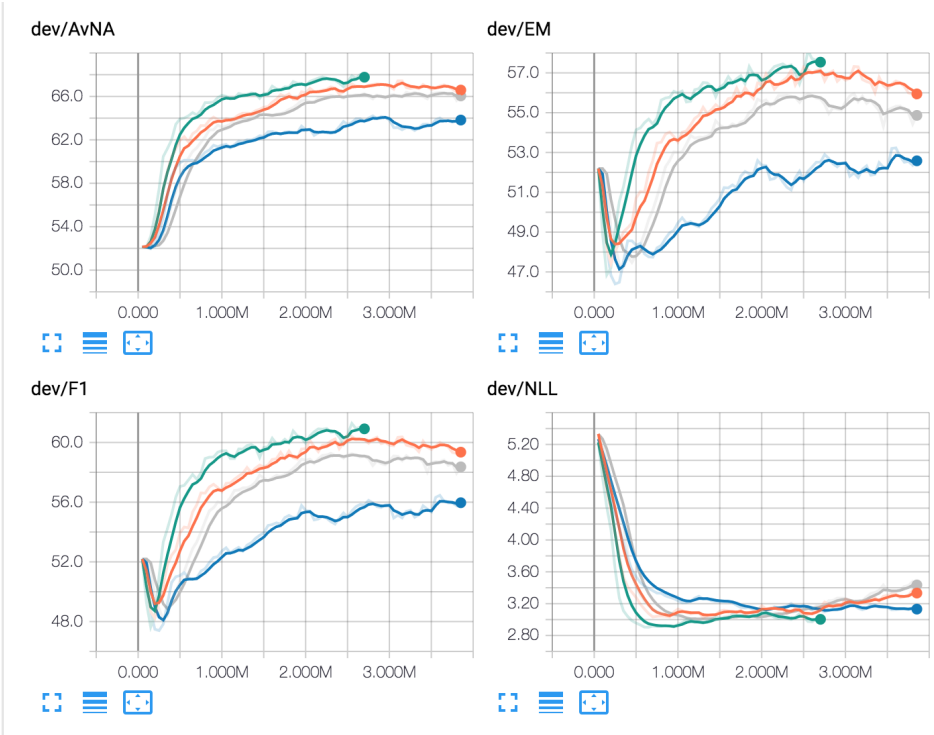
4. For the model with the GRU and the drop probability increased from 0.2 to 0.5, we got : Dev NLL: 03.12, F1: 56.06, EM: 52.66, AvNA: 63.97

I reported the top three runs to the *Test Leader Board* with the following results :

1. GRU run : EM = 58.563 F1 = 62.135

2. Baseline run : EM = 57.301 F1 = 60.371

3. GRU + LR run : EM = 55.300 F1 = 58.997



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. The best results on all the metrics are consistently obtained on the model with the GRU replacing the bi-LSTM (green lines). It has the lowest loss (NLL) at 2.90; the highest EM at 58.12; the highest F1 score at 61.275 and the highest AvNA score at 68.02. Moreover, since the GRU has lower computations than the bi-LSTM, we can clearly see that it ran to completion in almost a third-less number of iterations than the baseline model runs. This was reflected in the run time of the 30 epochs which took about 6 hours for GRU runs while it took about 10 - 11 hours for the baseline model runs. Another point to note is that it was only the GRU curves that seem to be monotonically increasing throughout the 30 epochs that were run, while in all other cases, we see some sort of a dip happening after reaching the maximum. All in all, in our experiments, the model with the GRU network was the clear winner in all evaluation metrics.

2. 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.

3. 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.

4. 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.

5. 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.

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.

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.

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.

I have to 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

The main findings of the experiments that we have run is that our initial hunch 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 points to the fact that when we are designing neural network 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. Without proper experimentation on the dataset, we are not able to declare for sure the values that will work the best for us. Previous experience can provide us some pointers in the values to be used but experimental results are what will confirm them.

Finally, we look at some of the work that can still be done and the future directions it can take.

6.1 Future Work

Research work implies that we are never done and there is always more to explore and learn. In the same vein, there are several avenues that we would have liked to explore in greater detail :

1. Introduce the character-level embedding which was part of the original BIDAF model[3]. The character-level embeddings allow us to look into the internal structure of the words and be better at handling words that are not part of the vocabulary.

2. We have looked into the use of self-attention from a few different angles. In the R-Net paper[4], they introduced the idea of Self-Matching Attention after having the Gated Matching Attention between the question and the passage (similar to the baseline model Context-to-Question Attention). One of the things to study would be to see how adding such a Self-Attention Layer would help.

3. In the Lin and Bengio paper[5], the model has been made even more simple. After the bidirectional LSTM, the model introduces a fully connected layer, which is then followed by a softmax layer. This fully connected layer would provide a set of weights that would find a correlation between all the tokens of the question as the those of the context, providing some more information for the answer to be extracted from the context. The self-attention mechanism allows extracting different aspects of the sentence into multiple vectors. These multiple vectors provide a way for the self-attention to work even without extra inputs being provided.

Acknowledgments

I would like to acknowledge the help that all the TAs have provided throughout this course, without whom I cannot see success at the end of the course. Also, special shout-out to Sahil for providing me valuable inputs at all times and especially during the project discussions. Finally, last but not the least, are innumerable thanks to Abi for providing me the ability to catch up on the assignments and the project after I had to miss 2+ weeks due to a family emergency in India. None of this would have been possible without all this support.

References

[1] Rajpurkar, Pranav, Zhang, Jian, Lopyrev, Konstantin and Liang, Percy. (2016). Squad: 100,000+ quetions for machine comprehension of text. In Proceedings of the Conference of Empirical Methods in Natural Language Processing. CORR, abs/1606.05250, 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] 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. arXiv:1703:03130

[5] 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

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

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

[8] https://jhui.github.io/2017/03/15/RNN-LSTM-GRU/

[9] https://medium.com/paper-club/grus-vs-lstms-e9d8e2484848

[10] http://ruder.io/optimizing-gradient-descent/

[11] https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

[12]https://www.kdnuggets.com/2017/11/estimating-optimal-learning-rate-deep-neural-network.html

[13]https://medium.com/udacity-pytorch-challengers/ideas-on-how-to-fine-tune-a-pre-trained-model-in-pytorch-184c47185a20