Default Final Project for CS224N - Self-Attention

Jaak J. Uudmae
Stanford University
juudmae@stanford.edu

Abstract

Self-Attention is an additional attention layer for a reading comprehension system which is supposed to refine an already given attention layer with information from the whole passage. In this paper we add this layer to an already existing baseline model for SQuAD challenge and see how the model compares to the baseline model provided by the course staff. The results show self-attention model improving in all areas compared to the baseline model - becoming even better at whatever the baseline model was good at and improving on the weak spots.

1 Introduction

The main goal of the default project was to build a reading comprehension system that works well on the Stanford Question Answering Dataset (SQuAD)[1]. The SQuAD dataset provides a paragraph and a question about the paragraph, and the trained model is supposed to provide an answer to that question. Each question has multiple ground truths and the questions require different forms of logical reasoning to infer the answer.

The baseline model[2] provided by the course staff was already a fully functioning reading comprehension system, but it was basic in nature, allowing the author to easily improve on the system.

This paper focuses on adding a self-attention layer on top of the given baseline model for the SQuAD challenge. Self-Attention is supposed to refine a given attention layer with information from the whole passage, allowing for better answers.

In addition to just adding the self-attention layer to the model and seeing the improvement in the total F1/EM scores, this paper also compares the new model with the baseline model based on different types of questions asked.

2 Background and Related Work

Self-Attention used in this paper was based on a work-in-process model by the Natural Language Computing Group at Microsoft Research Asia called R-Net[3]. R-Net calls it a Self-Matching layer.

Since R-Net uses many other techniques in addition to the self-matching layer it would have been unfair to expect the model produced in this paper, which just adds the self-attention layer, to do as well as R-Net has done. Currently, there are many updated versions of the model on the SQuAD leaderboard[1]. The highest achieving one is by *r-net+(ensemble)* submitted on January 3, 2018 achieving an F1/EM scores of 88.493/82.650 which ties for a third place at the moment. The model with the self-attention that this paper was based on achieved scores of 80.6/72.3.

As we can see, the model keeps on improving. Unfortunately, the newest models do not have papers published to see if they have updated the self-attention layer.

[1]Extracted from SQuAD leaderboard http://stanford-qa.com on March. 20, 2018.

3 Approach

As mentioned before, the new model builds on top of the baseline model which can be deconstructed into three components: RNN encoder layer, an attention layer, and an output layer. The new model has an added self-attention layer and has modified the input to the output layer.

3.1 RNN Encoder Layer

This layer remains unchanged from the baseline model, thus we won't go into too much detail here. The layer encodes both the context and the question embeddings into hidden states. In order to do that, for each SQuAD example, the context and question embeddings are fed into a 1-layer bidirectional GRU, which produces a sequence of forward hidden states and backward hidden states. Then those states are concatenated into context hidden states ($\vec{c_i}$) and question hidden states ($\vec{q_j}$) to be used in the attention layer.

3.2 An Attention Layer

An attention layer in the new model is split up into two steps. First we do basic attention which is followed by the self-attention.

3.2.1 Basic Attention

In the basic attention, the context hidden states are going to attend to the question hidden states. For each hidden state $\vec{c_i}$ the attention distribution (α^i) is computed as follows:

$$\vec{e^i} = [\vec{c_i}^T \vec{q}_1, ..., \vec{c_i}^T \vec{q}_M]$$
$$\alpha^i = \text{softmax}(\vec{e^i})$$

After that we compute the attention output $\vec{a}_i$:

$$\vec{a}_i = \sum_{j=1}^{M} \alpha_j^i \vec{q}_j$$

Here M is the number of question word embeddings.

3.2.2 Self-Attention

With only basic attention, we get a representation that has very limited knowledge of the whole context. It only pinpoints important parts in the passage. This means that one answer candidate is often oblivious to important cues in the passage outside its surrounding window. Also, according to the R-Net[3] paper there exists some sort of lexical or syntactic divergence between the question and passage in the majority of SQuAD dataset[1]. A solution to that would be to include passage context to infer the answer. One solution to that would be self-attention. It dynamically collects evidence from the whole passage for words in passage and encodes the evidence relevant to the current passage word and its matching question information into the passage representation. Self-attention is added as follows.

With the $\vec{a}$ from basic attention, we calculate the self-attention output ($\vec{a_s}$):

$$\vec{e_j^i} = \vec{v}^T \tanh(W_1 \vec{a}_j + W_2 \vec{a}_i)$$
$$\alpha^i = \text{softmax}(\vec{e^i})$$
$$\vec{a_s}^i = \sum_{j=1}^{N} \alpha_j^i \vec{a}_j$$

As we can see, we need to define weight matrices W_1 and W_2, which will be be both (2*h, 2*h), where h is the size of hidden state. The weight vector $\vec{v}$ has the size of 2*h.

Before we feed this output to the output layer, we concatenate the self-attention output to the basic attention output and feed it to a bidirectional RNN to obtain a new set of hidden states $\{\vec{h}_1, ..., \vec{h}_N\}$:

$$\{\vec{h}_1, ..., \vec{h}_N\} = \text{biRNN}(\{[\vec{a_1}; \vec{a_s}_1], ..., [\vec{a_N}; \vec{a_s}_N]\})$$

Self-attention extracts evidence from the whole passage according to the current passage word and question information.

3.3 Output Layer

This layer also remains mostly unchanged besides the input to it, thus a lot of detail is left out. We calculate the blended final representation like this:

$$\vec{b_i'} = \text{ReLU}(W_{FC}\vec{h_i} + \vec{v}_{FC})$$

Here W_{FC} is a weight matrix with dimensions (h, 4*h). The $\vec{v}_{FC}$ is a bias vector with size h.
Next, a score (log) is assigned to each context location i by passing $\vec{b_i'}$ through a downprojecting linear layer with both start and end weight vectors and biases. After which we pass the two score vectors into a softmax function to get a probability distribution p^{start} and p^{end}.

3.4 Loss

The loss function is defined exactly like it is in the baseline model. The loss function is the sum of the cross-entropy loss for the start and end locations. If the gold start and end locations are i_{start} and i_{end}, then the loss for a single example is defined as:

$$\text{loss} = -\log p^{start}(i_{start}) - \log p^{end}(i_{end})$$

During training, the loss is average across the batch and minimized with the Adam optimizer

3.5 Prediction

The prediction is also defined exactly like in the baseline model. During test time, given a context and a question, we take the argmax over p^{start} and p^{end} to obtain the predicted span (l^{start}, l^{end}.

4 Experiments

4.1 Dataset

For both baseline and self-attention model, a GLoVE dataset with vectors in 100 dimensions was used. For training we used the official SQuAD training set train-v1.1.json. For develop test we used the official SQuAD develop set dev-v1.1.json. In addition we had access to tokenized train set data and develop set data with answers, context, questions, and spans. For evaluation both tiny-dev.json and the full dev-v1.1.json were used.

4.2 Experiment configuration

The following hyperparameters were used for both baseline and self-attention model:

Learning rate = 0.001
Max Gradient norm = 5.0
Dropout = 0.15
Batch Size (baseline) = 100, Batch Size (self-attention) = 50
Hidden Size (baseline) = 200, Hidden Size (self-attention) = 50
Context Length (baseline) = 600, Context Length (self-attention) = 300
Question Length = 30
Embedding Size = 100

The reason why self-attention used smaller batch size, hidden size, and context length, was to reduce the memory usage in the self-attention layer.

The model was trained using Adam optimizer. We tried using AdaDelta optimizer, but the model refused to train as the loss did not decrease. Thus, it was decided that Adam optimizer is good enough for this project.

The baseline was trained for 7h30min and the self-attention model was trained for 23h before they both stopped improving. The self-attention model also had to use extra memory (12GB) to be able to train.

4.3 Evaluation metric

To evaluate the results an F1 and EM metric was used. In addition to just comparing the total F1 and EM scores we separated the questions into seven different categories. Questions containing: ”what”, ”where”, ”when”, ”who”, ”which”, ”if”, ”how”. Note that some questions can contain multiple of those question tokens and there could have been some questions that were missed using this separation method, but all in all it captured most of the questions.

4.4 Results

As expected, the addition of the self-attention layer improved the scores. The baseline model in the develop leaderboard[2] for the class achieved scores of 43.723/34.56 for F1/EM. The self-attention model achieved 64.092/51.088 for F1/EM. That’s about 50 percent increase in both categories. The self-attention model was also used in the final submission in the test leaderboard[3], where it achieved 64.358/51.904 F1/EM scores.

[2] http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/dev.html
[3] http://cs224n-win18-leaderboard.westus.cloudapp.azure.com/test.html

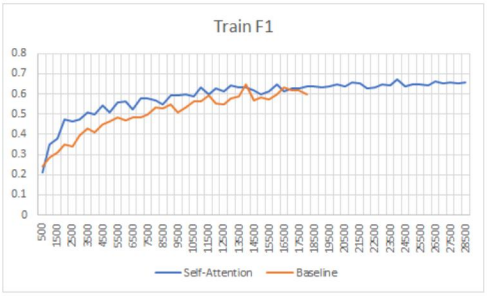

Figure 1: Training F1

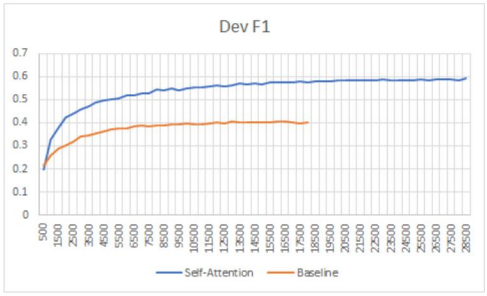

Figure 2: Development F1

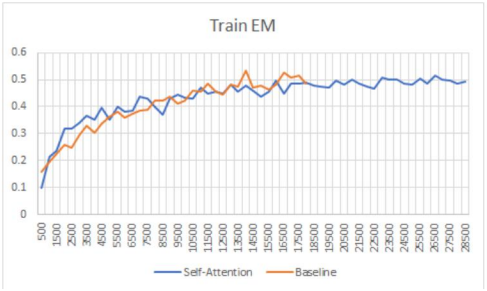

Figure 3: Training EM

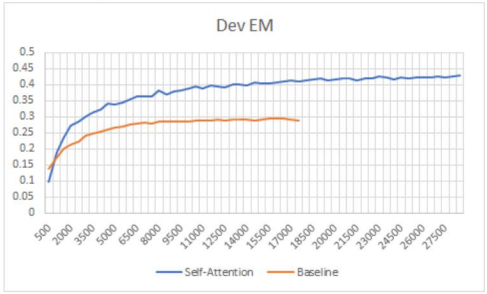

Figure 4: Development EM

As we can see from Figures 1 to Figure 4, the self-attention model generalizes to new data much better than the baseline model does. Even though the training F1/EM do not improve that much at all, the development set scores improved a lot. A similar effect happened in the training and development losses.

Next, we wanted to see how the F1/EM scores improved in the different question categories. The counts for each question category are represented in Table 1. We can see that Tiny-Dev dataset is more balaned than the full dev dataset, which is dominated by the question ”what”. Even so, the tiny-dev dataset is not perfectly balanced as questions like ”if”, ”where”, ”when”, ”which” seem to be under-represented. We can also see that the seven categories add up to more than total, meaning that there are some questions with multiple question tokens in them.

	Total	What	Where	When	Who	Which	If	How
Tiny-Dev	810	323	27	50	209	79	3	156
Dev-v1.1	10570	6056	505	862	1281	746	73	1241

Table 1: Counts of questions for each question category

As a point of comparison we can see from Figure 5 and Figure 6 that self-attention outperformed baseline for both the tiny and the regular develop dataset when all questions were included.

Figure 5: Comparison of F1 for all questions

Figure 6: Comparison of EM for all questions

Once we break the questions apart in Figure 7 to Figure 10, we see that there isn't a category where self-attention performs worse than the baseline. That means that the self-attention model strictly improves on the baseline model without any losses in accuracy.

Figure 7: Comparison of F1 on "what, where,when"

Figure 8: Comparison of EM on "what, where, when"

Figure 9: Comparison of F1 on "who, which, if, how"

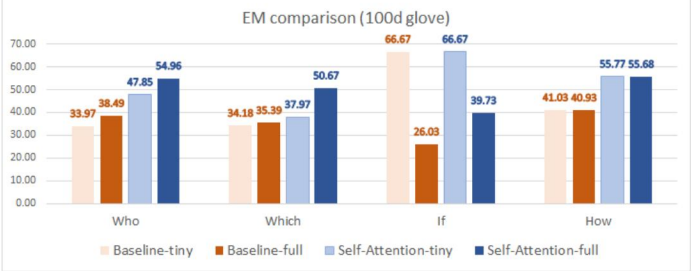

Figure 10: Comparison of EM on "who, which, if, how"

The self-attention model improves a lot on questions related to time (when), people (who), and objects (what) compared to the baseline model. The self-attention model also performs surprisingly well on questions that seem to require more complicated answers (how). Questions related to locations (where) and conditionals (if) don't do as well. From Table 1, we can also see that the latter two question types are the most under-represented in the the datasets.

It is interesting to see that categories that do really great in the baseline model like "when" and "how" do also increasingly better in the self-attention model. That shows that the attention gathered in the basic attention layer is reaffirmed in the self-attention model, making it even better at what it was at.

The big score boost comes probably from doing so well on questions related to "what", which is dominating the dev-v1.1.json dataset. Using the baseline model, the scores are considerably worse than other categories - 39.85 for F1 and 29.52 for EM in the full dataset. Meanwhile, the self-attention brings the performance on those types of questions up to 62.12 for F1 and 48.13 for EM in the full dataset. "What" is indeed a hard question to answer since it can capture many types of answers. We can see here how the information from the whole passage helps to give better answers to this type of question.

Even though training took longer for the self-attention to converge, the same results on the develop during training were achieved in about 40 percent of the time it took for the baseline to achieve them. The slow convergence time might also be related to smaller batch size. Computationally it makes sense that self-attention takes longer to train, since we are adding more code and computation to the model without taking anything out.

5 Conclusion

This project was a great way to learn how to improve on an existing neural network. It was a great learning experience to try to figure out how to implement a state of the art technique into an existing model and the problems that might arise while doing it - like memory issues and how to solve them. Trying to code the self-attention up also definitely improved my commenting skills as the dimensions were really important to keep track of.

In addition, this project shows how self-attention improves the baseline model. Making it even better at what the old attention was good at, and trying to improve on the areas that weren't as good. This confirms the whole idea of why self-attention was introduced into the R-Net. It seems that self-attention can be further improved by making the attention layer preceding it even better, thus allowing the self-attention to capture even better information from other words.

In the future, it would be interesting to see how the results differ using different hyperparameters (batch size, context length, etc.). Maybe the categories for question tokens could be further reduced (break "what" into different categories) to produce more balanced buckets for better comparison and further analysis.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang (2016) *SQuAD: 100, 000+ Questions for Machine Comprehension of Text*

[2] Abi See (2018) *Code for the Default Final Project (SQuAD) for CS224n, Winter 2018* `https://github.com/abisee/cs224n-win18-squad`

[3] Natural Language Computing Group (2017) *R-NET: Machine Reading Comprehension with Self-matching Networks*, Microsoft Research Asia .