Character CNN and Self-Attention for SQuAD

Jianqing Yang  
Department of Computer Science  
Stanford University  
Stanford, CA 94305  
yangjq@stanford.edu

Abstract

The Stanford Question Answering Dataset (SQuAD) and challenge is for researchers to develop reading comprehension and question answering models. The task is to correctly answer a question given a paragraph of context. This paper discusses the use of a character-level convolutional neural network and self-attention layer in tackling this challenge, as well as accompanying memory and performance optimizations.

1 Introduction

The SQuAD [1] challenge is to develop a model that is capable of answering a question given a paragraph of context text containing the answer. The dataset contains around 100K of crowdsourced question-answer pairs. The model is evaluated on its accuracy in selecting the correct span of text within the context paragraph for each question as either an exact match (EM) or partial match (F1) score.

The intended\(^1\) overall approach to this problem was to first experiment with a few independent model enhancements. These would then be combined and optimized through a hyperparameter search. To operate within available computing resources as well as to facilitate faster experimentation, this paper also discusses some performance optimizations for memory usage and processing time for training.

2 Background/Related Work

Character-level convolutional neural networks (CNN) are an increasingly popular mechanism for a variety of natural language processing (NLP) tasks [4]. Character-level embeddings are also commonly used in SQuAD solutions [2][3] although these are typically of secondary importance to boost the final task performance.

Attention methods are a heavily researched area for tackling SQuAD, with many different variants explored [2][3] and regarded a key component of most high-performing models. This is logical for the SQuAD challenge given that the key task is to find the span in the context which attends to the question. Given that the answer is a continuous span of text, it further makes sense to pick an attention method which naturally supports this such as [3].

3 Approach

First, a character-level CNN was implemented similar to [2], where characters are embedded into vectors and fed as 1D inputs in a moving window to a CNN. The CNN output is max-pooled to

\(^1\)Please see Additional Information document for more details.
Table 1: Batch training times with and without variable input length modification.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Average Batch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char CNN, Batch size 100</td>
<td>2.02s</td>
</tr>
<tr>
<td>+Variable input length</td>
<td>1.44s</td>
</tr>
</tbody>
</table>

produce a fixed size vector for each word, which is then concatenated with the pre-trained word embedding vector. This concatenation is then fed to the baseline bi-directional GRU encoder layer. The character embeddings are not pre-trained, but trained together with the model. The character vocabulary is assembled from all the words loaded from the GloVe embeddings. The input matrix of each batch of training data fed to the CNN was large due to the explosion of having a vector for each character instead of just each word. A memory optimization was implemented to subdivide each training batch processed by the CNN and recombine the results by concatenation.

It was observed that the default context and question length of 600 and 30 respectively was abundant in most cases, which added to training time unnecessarily. Variable context and input length was implemented to reduce this with the primary intention of speeding up iterative experiments for hyperparameter search later on. Being able to better handle contexts and questions longer than the default was a secondary benefit. This was done by setting the context and question lengths on a maximum-per-batch basis, as well as being the smallest possible product of the number of subdivisions configured for the CNN memory optimization. The latter is necessary for equal splitting to work as the last batch used before refilling may not have the same factors as a normal batch. For example, if the CNN subdivisions was set to 4 and the longest context in a particular batch was 102, the context length for the batch would be set as 104 which is the smallest product of 4 that is also larger than 102. All the other questions in the same batch would be padded to 104 words. This helped to reduce the training time required by around 29% as can be seen from Table 1; the table figures were calculated from averaging the processing times from batches 2-500 (omitting batch 1 time as it includes initialization overheads).

Next, a self-attention mechanism was added in reference to [3]. This is done by first adding a gate on the baseline attention layer output by multiplying the attention output with a trainable weight matrix and sigmoid activation. This is then fed into a self-attention layer. In [3], the self-attention layer is additive, i.e.:

\[ e_i = v^T \tanh(W_1 h_i + W_2 s) \]

However due to memory limitations, the implementation in this paper uses multiplicative attention, i.e.:

\[ e_i = s^T W h_i \]

The output of the self-attention layer is fed into another gate similar to the one used after the baseline attention layer, before being concatenated with the context hidden states as a blended representation and fed into a bi-directional GRU. The additional self-attention and RNN layers are configured to use the same dropout regularization rate as in the baseline GRU encoder and attention layers. Finally, this is passed into the baseline fully-connected layer and softmax function to generate the start and end answer locations. Figure 1 gives an overview of the full architecture.

4 Experiments

On their own, the character CNN and variable input length does not have any significant impact on the task performance. This is not unexpected as out-of-vocabulary (OOV) words have not been observed to be a major issue in the SQuAD dataset and the number of questions or answers exceeding the default fixed length is small. Due to the memory and speed improvements mentioned earlier, it was still possible to run training with a batch size of 100 at a comparable speed to the baseline.

After introducing self-attention, which significantly increased the number of trainable parameters, it was necessary to reduce the batch size to 50 to keep within memory limits. This yielded a significant
Figure 1: Model architecture.

Table 2: Dev dataset results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>F1 Score</th>
<th>EM Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>43.982</td>
<td>34.683</td>
</tr>
<tr>
<td>+Char CNN, +Variable input length</td>
<td>43.721</td>
<td>34.437</td>
</tr>
<tr>
<td>+Self-attention, -Batch size 50</td>
<td>69.024</td>
<td>58.666</td>
</tr>
<tr>
<td>+Hidden size 256</td>
<td>68.084</td>
<td>57.275</td>
</tr>
<tr>
<td>+GloVe 300D, -Hidden size 150</td>
<td>67.837</td>
<td>57.171</td>
</tr>
</tbody>
</table>

Improvement in task performance (Table 2). Examining the sample output of this model compared to the baseline, it is observed that self-attention does not suffer from incorrect answers where the end point is predicted behind the start point (Figure 2). Similarly, self-attention avoids selecting large spans of text as the answer, as seen in Figure 3; even though the self-attention answers in these examples are still wrong, they are arguably more coherent choices than in the baseline. Both these scenarios are consistent with the intuition discussed in [3] that self-attention expands upon the narrow context of basic context-question attention to find other relevant information.

From the tensorboard plots of this model (Figure 4), the dev loss does not increase greatly after the dev performance begins to plateau. This might suggest that the model is not overfitting that aggressively and there may be more headroom to increase the model expressiveness. This was thus the approach for the hyperparameter search. One experiment was to increase the hidden state size from 200 to 256, and another to use the largest GloVe embeddings of size 300 but reduce the hidden state size to 150. In both cases the parameters were empirically chosen to fit within available GPU memory limits. However, these were not successful in improving the task performance (Table 2).
Figure 2: Example results showing improvements in not predicting an end point behind the start point.

Table 3: Test dataset results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>F1 Score</th>
<th>EM Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Char CNN, +Variable input length, +Self-attention, -Batch size 50</td>
<td>69.637</td>
<td>59.761</td>
</tr>
</tbody>
</table>

Thus for the final test, the self-attention model with batch size 50, hidden size 200 and GloVe embedding size 100 was submitted, achieving the results shown in Table 3.

5 Conclusion

This project has been a useful exercise on many fronts. I learnt about the importance of strategizing the approach to deep learning problems. Under time and resource constraints it is helpful to identify key areas for improvement in order to prioritise work for the greatest performance gains. For instance, if I were to restart the challenge on my own, I would probably not have looked into character CNN so early. Strategizing also applies to knowing which methods work well with each other and how much additional work is required for integration. The variable input length required some effort to integrate with both already-implemented and subsequent model enhancements, but was likely still worth the time investment given the training time savings.

It was also a great learning experience to think about and experiment with all the different levers to pull (to borrow from bandit problem terminology) while considering the training graph plots, training time and memory limits. Especially regarding the latter two points, I have learned about the importance of building efficient models since simply using brute force was not an option.
For further work, it should be worth experimenting with pre-trained character embeddings for the character CNN. In [4], this gave marginally better results on a variety of NLP tasks over randomly initialized ones which were trained together with the task, even before tuning the pre-trained vectors for the specific task. This would remove the uncertainty of the character vector training being effective as a factor in the overall SQuAD performance. Another possible improvement is to replace other baseline components such as the final output layer with the Answer Pointer model [5].

References

