Building QANet from Scratch

Stanford CS224N Default Project

Nathaniel Chien
Department of Computer Science
Stanford University
nchien2@stanford.edu

Abstract

Question answering models have been the standard in NLP for demonstrating reading comprehension, and recent developments in model architecture have greatly increased performance. One of the most important innovations has been the application of the transformer architecture to question answering through the QANet architecture. In this paper, I describe my own implementation of QANet based on the original model. My model was able to improve performance on the Stanford question answering dataset when compared to a standard BiDAF model, despite issues with overfitting that decreased test performance. Through analysis of my model results, I have identified potential limitations of QANet, and found that it trains more slowly than recurrent models when limited by memory and computational power.

1 Key Information to include

- Mentor: Elaine Sui
- External Collaborators: N/A
- Sharing project: No

2 Introduction

The problem of question answering has been considered since the 1960s, both as a potential tool for everyday use and as a measure for evaluating machine learning models' accuracy. Intuitively, a model that has reading comprehension good enough to answer questions about a passage is able to demonstrate a strong understanding of language. Besides their roles as metrics, question answering models are also commonly utilized in modern search engines, and there is ongoing research about the potential of using them as knowledge bases [1].

A majority of question answering models are developed using the Stanford question answering dataset (SQuAD), which is the most popular dataset for testing reading comprehension. Performance on SQuAD has increased drastically over the past years as architecture has developed, with the use of attention and recurrence in particular leading to significant improvements in accuracy and speed. Two models of note that use SQuAD are the Bidirectional Attention Flow (BiDAF) [2] and QANet [3] models. The BiDAF model was state of the art until recently, and uses a combination of self-attention and recurrent neural networks to encode passages. QANet improved upon BiDAF by applying the transformer architecture, significantly speeding up training and improving accuracy. Newer models have continued to build upon the concepts from BiDAF and QANet.

Current state of the art models have 'solved' SQuAD, and are able to achieve performance that is comparable with human understanding. Despite this, there are still limitations for many models. In order to challenge models and force them to greater understanding of language, researchers have developed new adversarial data with more difficult questions. One such example is the triviaQA dataset [4], which contains manually curated questions that require greater understanding. Another

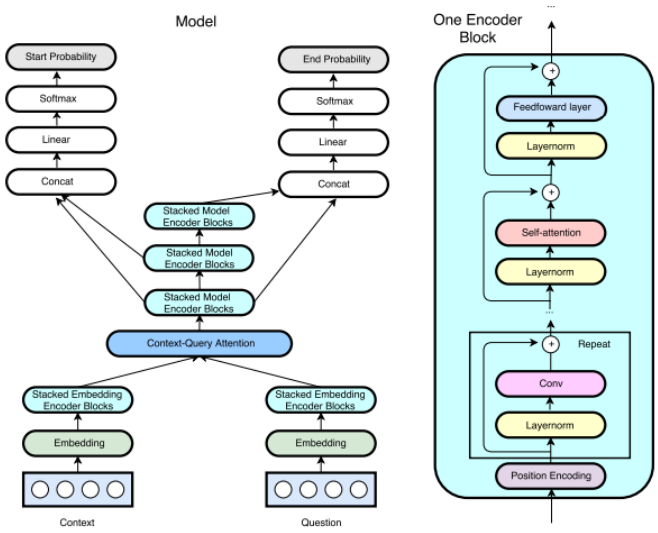

Figure 1: Model architecture from QANet paper. (Will replace with my own graphic once my implementation is finalized)

example is SQuAD v2.0, the dataset used in this paper, which also contains some questions with no answers. While the innovations of QANet have lead to great improvements in question answering, there are still many improvemnets yet to be made.

3 Related Work

For this project, my goal was to implement my own version of the QANet model from scratch, as defined in (Yu et al., 2018). Specifically, QANet is a transformer model that has been adapted to work on text data, utilizing convolutional layers instead of the LSTMs used in the Bidirectional Attention Flow model that preceded it. These LSTMs apply a network with shared weights over the entirety of the text in order to encode local structure. However, they are slow to train since data must be fed into the layer sequentially. Convolutional layers don't have this limitation and are parallelizable, which allows QANet to have much faster performance both for training and evaluation while being competitive with BiDAF. The authors of the original QANet paper also leveraged this greater training speed, using data augmentation to generate more questions and further increasing performance. Since the development of QANet, there have been several other models that have significantly improved upon it. The most well-known current model, BERT, uses the same basic transformer structure while also applying bidirectional training and pre-training on objectives with easily generated data. While QANet is no longer the state of the art, its innovations have paved the way for current research and models.

4 Approach

Starting with the BiDAF baseline, I modified the architecture in order to imitate the original QANet model, with some minor changes.

4.1 Embeddings

The first part of my implementation was to add character-level embeddings. As described in (Seo et al., 2018) [2], these embeddings can more accurately encode meaning behind subparts of words, such as specific prefixes and morphemes. The provided baseline already provides character embedding initializations which I processed based on the methods described in (Kim, 2014) [5]. The initializations are used to create character-level representations for each word, which are then convolved and maxpooled to obtain a vector of fixed length for each word. These character-level embeddings

are then concatenated with the word-level embeddings, and passed into a 2-layer highway network before being used by future layers.

4.2 Encoder

The next part of the model, which is the key innovation of QANet, is the encoder block. These blocks use parallelizable convolution layers instead of recurrence. Each block is composed of a a positional encoding, a repeated layer of convolutions, a self-attention layer, and a feedforward network.

4.2.1 Positional Encoding

The positional encoding, as described in (Vaswani et al., 2017)[6], makes up for the limitations of removing recurrence by allowing the model to utilize the sequence's order. For each position index pos, and for each dimension of the embedding i up to half of its total size, the positional encoding is defined as follows:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/emb\_dim})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/emb\_dim}).$$

This sinusoidal function allows the model to encode word position into the embeddings. This is important since the QAnet model removes the recurrent networks from BiDAF, which implicitly take position into account. Intuitively, the sinusoidal function is particularly good for representing position since PE_{pos+k} can be represented as a linear function of PE_{pos} for any k. After the positional encodings are calculated, they are then added to the word and character-level embeddings, before being passed further into the encoder block.

4.2.2 Depthwise separable convolutions

The next layer is a stacked depthwise separable convolution (Chollet, 2016) [7]. A depthwise separable convolution is a convolution that has been split into two operations: a depth-wise and point-wise convolution. The depth-wise convolution transforms the matrix while maintaining its depth, and the point-wise convolution uses 1x1 kernels that iterate through every point and expand the output depth to be equal to the number of kernels. In practice, this decomposition is much more computationally efficient than a typical convolution, and its reduced expressivity has relatively low impact in complex models. The usage of convolutions instead of recurrent networks is the key innovation of QANet. Convolution are able to perform the same function of capturing local structure, while being parallelizable and allowing for faster model training. By itself, QANet's performance is on par with BiDAF's. But when you take advantage of its increased training speed and use data augmentation to generate more questions, it is able to achieve a higher performance.

4.2.3 Self-attention

Next, the output of the convolutional layers is passed into a self-attention layer. As described in the original R-net paper[1], self-attention allows representations of questions and passages to attend to important context clues outside of its local window. Specifically, self attention is found by calculating similarity score between vectors within a query/context, taking the softmax to generate a distribution, and then using this distribution to take the weighted sum and generate an attention output. Specifically, the R-net paper defines it as

$$s_j^t = \mathrm{v}^\top \tanh(W_v^P v_j^P + W_v^{\bar{P}} v_t^P)$$
$$a_i^t = Softmax(s_i^t)$$
$$c_t = \sum_{i=1}^{n} a_i^t v_i^P$$

Where t represents the index of the word for which attention is being calculated, and i represents the index of the word that t is attending to. The final output, c_t, is the attention output for word t. I

[1] https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf

utilized the built-in pytorch MultiheadAttention class, which acts as a self-attention layer when the same data is passed in as the key, value, and query. Finally, this output is passed to a final feedforward layer that generates an output with the .

4.2.4 Residual Connections and Dropout

Following the architecture as described in the paper, I also implemented residual connections between every sub-layer in the encoder block. These connections directly pass data between layers without transforming it, which helps avoid distorting important information as it is passed through the block. The original QANet model also included layer dropout, in which layers in the encoder have a certain probability to be skipped completely. Due to time limitations, I did not implement this feature. However, I did add ordinary, node-wise dropout between layers.

4.3 Context-Query Attention

I then pass the output of the encoder to a Context-Query attention layer which finds the most relevant words in the passage based on the question. I simply used the same layer defined in the baseline BiDAF model, which first calculates a similarity matrix $\mathbf{S}$ as follows

$$\mathbf{S}_ij = \mathbf{w}_{sim}^{\top}[\mathbf{c_i}; \mathbf{q_j}; \mathbf{c_i} \circ \mathbf{q_j}] \in \mathbb{R}$$

Softmax is then applied to this similarity matrix to calculate a distribution $\overline{\mathbf{S}}$, which is applied to the question encodings to calculate attention outputs.

$$\overline{\mathbf{S}}_{i,:} = \text{softmax}(\mathbf{S_{i,:}})$$

$$\mathbf{a_i} = \sum_{j=1}^{M} \overline{\mathbf{S}}_{i,j}\mathbf{q}_j$$

4.4 Model Encoders

The output of the context-query attention layer is passed to three layers of stacked encoder blocks with the same architecture as described above, except with only two stacked convolutions per encoder. In order to train the model with the limited memory on my virtual machine, I also decreased to number of stacked blocks per layer to 5, rather than the 7 in the original paper. These three layers also share weights, using the same 5 encoder blocks.

4.4.1 Output

The output is calculated as described in the original paper, by concatenating the intermediate outputs of the model encoder layers, then passing them into a linear layer and softmax. This gives us two probability distributions over the positions in the context, one for the span start position and another for the end position.

5 Experiments

5.1 Data

I tested my model on the Stanford question answering dataset (SQuAD), which is composed of manually annotated datapoints. Each point includes a passage selected from English Wikipedia, a crowd-sourced question, and an answer which is defined as a short span in the passage. The goal of any model using SQuAD is to predict span that gives the correct answer given the passage and question. I am also using the updated SQuAD v2.0 dataset [8], which also includes crowd-sourced questions that are designed to be unanswerable, based on the same articles as the original SQuAD dataset. In total, there are approximately 150k question-passage pairs, of which around one-third are unanswerable. The inclusion of these unanswerable questions makes the task of predicting answer spans significantly more difficult, and potentially forces models to express a better understanding of text and relationships.

5.2 Evaluation method

I use the same two evaluation metrics used in the original QANet paper, which are F1 and exact match (EM) accuracy. F1 accuracy is a measure of the proportion of overlap between the predicted and true answers, while EM accuracy is 1 if the predicted answer exactly matches the true answer, and 0 otherwise. For the purposes of this project, I also compared my results directly with a baseline BiDAF model that was provided to students. This model achieved a dev set F1 score of 61.431, and EM score of 58.074.

5.3 Experimental details

Before implementing QANet, I familiarized myself with the baseline code by modifying the provided BiDAF to include character embeddings. I ran this model for 30 epochs with the same hyperparameters as the baseline: a batch size of 64, a learning rate of 0.5, a dropout rate of 0.2, and a model dimension of 100.

After implementing QANet, I ran my model for 30 epochs with the same hyperparameters described in the original paper. I used a batch size of 32, a learning rate of 0.001 with an exponentially increasing wake-up period, a dropout rate of 0.1, and a model hidden dimension of 128.

5.4 Results

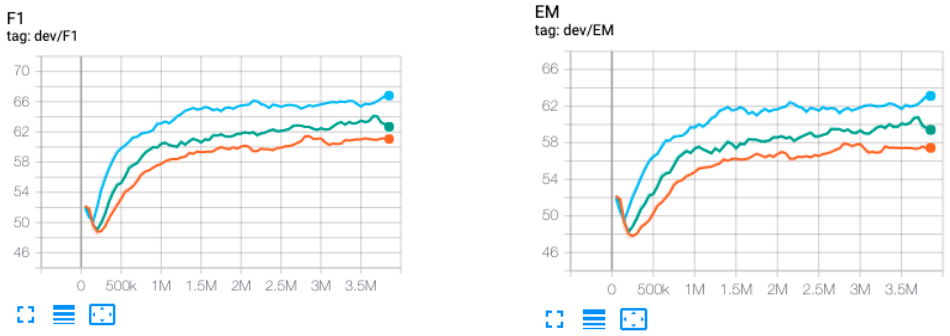

Figure 2: F1 and EM scores for the baseline (orange), character embedding (green), and QANet (blue) models

After training my models, I observed increased performance for both the character embedding and QANet model on the dev dataset when compared to the baseline scores of 57.049 for EM and 60.686 for F1. My character embedding model achieved an EM score of 60.881 and an F1 score of 64.244, and my QANet model achieved an EM score of 63.199 and an F1 score of 66.895. However, upon submitting my results to the test leaderboard, I found that my scores for QANet were significantly lower on the test dataset than the dev dataset, with an EM score of 58.428 and an F1 score of 62.343. On the test dataset, my QAnet model achieved only minor increases in accuracy over the baseline.

I first believed that this discrepancy in accuracies was due to overfitting on the dev dataset, and so I retrained the model using a higher dropout, and stopping at an earlier epoch when the training loss had started to plateau. However, this actually led to a further decrease in accuracy and scores below the baseline. Given these strange results, I am unsure whether or not this is an issue with overfitting, or some other bug with my model implementation. Unfortunately, due to the limited number of submissions to the test leaderboard I was unable to resolve this issue. If given more time I would continue to experiment with regularization methods and learning rate schedulers to try to resolve this issue.

6 Analysis

My QANet architecture is able to obtain better performance on the dev set than the baseline, but it still fails at many of the same tasks that the BiDAF model does. For example, consider the following examples:

Question:Orientalism refers to how the South developed a what of the North?
Context:Orientalism, as theorized by Edward Said, refers to how the West developed an imaginative geography of the East. This imaginative geography...
Answer: N/A
Prediction: imaginative geography of the East

In this example, the model makes a prediction when the question asks about 'the South', despite the word 'South' not appearing anywhere in the text. It instead answers about 'the East', seeming to interpret the two directions as the same entity. This could be indicative that the generated word embeddings are not specific enough. Because 'South' and 'East' are both directions often used in similar contexts, their embeddings are similar enough that the model cannot distinguish them. This is a demonstration of the limitations of our embeddings.

Question:What country was under the control of Norman barons?
Context:Subsequent to the Conquest, however, the Marches came completely under the dominance of William's most trusted Norman barons, including Bernard de Neufmarché, Roger of Montgomery in Shropshire and Hugh Lupus in Cheshire. These Normans began a long period of slow conquest during which almost all of Wales was at some point subject to Norman interference...
Answer:Wales
Prediction:Cheshire

The reasons for the model's incorrect prediction are more ambiguous in this example, but I believe it demonstrates one specific failing of my model. This answer demonstrates a lack of understanding of the world. A human reading this passage would immediately discount Cheshire as a potential answer, since Cheshire is not a country. However, our model has very little capacity to learn this general information. The inability to answer questions such as this is one of the major motivations for ongoing research in integrating knowledge into language models.

Another interesting observation was that my QANet model trained significantly slower than the baseline, with about one-third of the iterations per second. This result was surprising because one of the main draws of QANet is it's fast training time. However, through discussion among students I've learned that this may be due to the limited memory and processing power of the machines that we were working on. QANet is faster only because it is parallelizable, and without the computational power to take advantage of this parallelizability, it runs more slowly than BiDAF.

7 Conclusion

Building my own version of QANet proved to be much more difficult than I anticipated, but through my implementation I became much more familiar with transformer architecture, the motivation behind it, and the small details that one must account for when constructing and training a model. There are countless potential improvements to be made, from implementing details that I left out such as layer-level dropout and data augmentation, to experimenting with different regularization techniques and tweaking the model architecture. Despite bugs or problem with overfitting that have led my model to have poor results on the test dataset, I successfully implemented the key components of QANet, including character and positional embeddings and the encoder block.

References

[1] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: enhanced language representation with informative entities. *CoRR*, abs/1905.07129, 2019.

[2] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.

[4] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *CoRR*, abs/1705.03551, 2017.

[5] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[6] 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, 2017.

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[8] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.