Question Answering by QANet and Transformer-XL

Stanford CS224N Default Project

Chi Wang
Department of Computer Science
Stanford University
chiwang@stanford.edu

March 17, 2021

Abstract

Question answering is a classic and interesting NLP task, although the pre-trained contextual embedding models (like BERT) have dominated the leaderboards, in order to gain a deep understanding of transformers, I chose to re-implement QANet[1] architecture and integrate it with Transformer-XL[2] attention calculation method in this project. My hope is by introducing the recurrent structure into the attention computation, the combined model (QANet-XL) could learn better since it can take an unlimited length context all at once and it should be able to look further when finding answers. Despite my experiments didn't show a clear performance improvement with Transformer-XL, but the DEV NLL comparison suggests that QANet-XL might outperform QANet with proper tuning and longer training time.

1 Key Information to include

- Mentor:
- External Collaborators (if you have any):
- Sharing project:

2 Introduction

Questions Answering systems, usually a computer program, can pull answers from an unstructured collection of natural language documents. An effective and accurate question answering approach is badly needed since it could save people

tremendous amount of time on searching answers through large volume of texts.

With the rapidly development in NLP field, the recent BERT-based models managed to even surpass human performance. Despite the pre-trained contextual embedding models (like BERT) have dominated the leaderboards in question-answering space, I choose to focus on Non-PCE transformer approaches: QANet and Transformer-XL in this project.

Why bother trying QANet if it was out-performed by pre-trained models? My goal was to gain a deep understanding of transformer model by re-implement the classic transformer-based QANet model and extend it with the ideas of Transformer-XL. I hope this experiment could show that by introducing a recursive structure to allow transformer to draw connections between farther apart sections of text, it would improve the performance of a QANet model. In the end, however, which I will describe in more detail later, the combined model didn't gain any significant improvement compare to pure QANet model. Also. after comparing with fine-tuning pre-trained DistilBERT model, we could see clearly why people are in favor pre-trained models.

3 Related Work

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.

SQuAD 1.1, the previous version of the SQuAD dataset, contains 100,000+ question-answer pairs on 500+ articles. SQuAD2.0 combines the 100,000 questions in SQuAD1.1 with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on SQuAD2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering.

The original QANet[1] ("QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension") came from a research by Yu et al. (2018) at Google. They combined local convolution with global self-attention for reading comprehension. It was once the leader on score board on SQuAD 1.0 with F1 84.6 in 2018.

Transformer-XL[2] ("Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context") addresses the limitation imposed by fixed length context in language models. According to the authors, it can enable capturing longer term dependency, resolve the problem of context fragmentation, and achieve better performance on both short and long sequences. The complete set of solutions consist of two techniques: a segment level recurrence mechanism and a

new positional encoding scheme.

More recent work with pre-trained contextual embeddings, such as BERT have further pushed the NLP research to a new level, by using pre-trained embeddings, can offer huge improvements in performance, allowing new state-of-the-art benchmarks to be reached across Question Answering and other NLP tasks.

4 Approach

4.1 Baseline

The baseline model approach is BiDAF[3], thanks for the default project, it's already provided in the starter code. Please see the BiDAFF details in project handout.

4.2 QANet Implementation

The overall QANet architecture is illustrated in Figure 1, my QANet implementation is started from the default project, for reference and debugging code errors, I considered following two git repositories: andy840314/QANet-pytorch-[4] and heliumsea/QANet-pytorch[5]. Dimensions and hyper parameters are chosen directly from the QANet[1] paper. QANet layers are discussed in below 5 subsections.

4.2.1 Input Embedding Layer

I reused the input embedding layer of the baseline code (BiDAF) embeding layer. Input embedding is built by concatenating word embeddings (300 dimension GloVe) and character embeddings (64 dimension).

4.2.2 Embedding Encoder Layer

Seeing the big blue box in Figure 1, each encoder block is built by a positional encoder, depthwise separable convolutions layer, self-attention layer, and feed-forward layer. The self attention adopt the multi-head attention mechanism defined in (Vaswaniet al., 2017a), for each position in the input, called the query, computes a weighted sum of all positions, or keys, in the input based on the similarity between the query and key as measured by the dot product. The number of heads is 8 throughout all the layers.
The same Encoder Blocks are used throughout the model; only the number of convolutional layers for each block is different.

4.2.3 Context-Query Attention Layer

Adopted directly from the BiDAF Attention, see more details in the default project handout("Attention Layer - layers.BiDAFAttention" section)

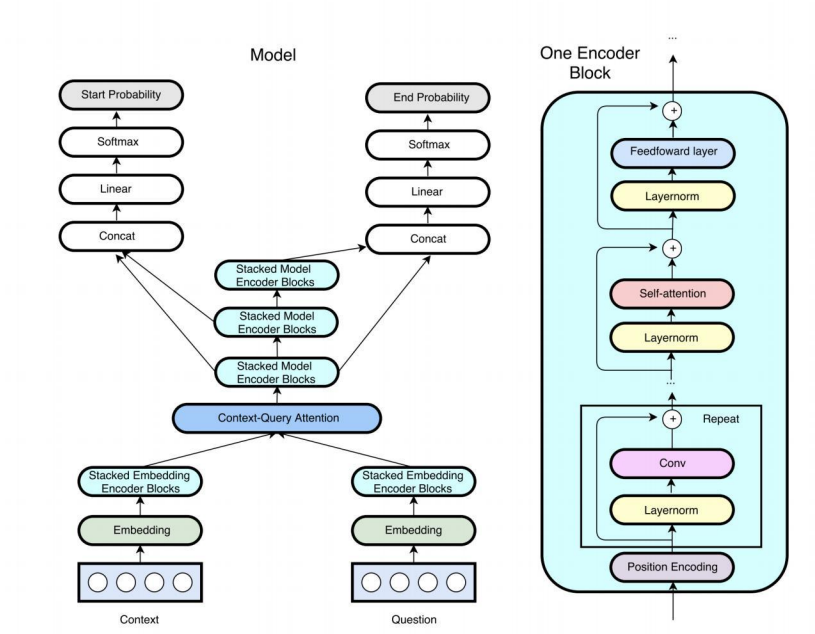

Figure 1: QANet overall architecture

4.2.4 Model Encoder Layer

Similar to the Embedding Encoder with the same parameters, except convolution layers number and block number are set to 2 and 7 respectively.

4.2.5 Output Layer

Adopted from the BiDAF Output, The output of this layer is probabilities of the starting and ending position.

4.3 TransformerXL Implementation

The discussion of TransformerXL will be focused on below two methods I applied to QANet, other parts of TransformerXL architecture are less relevant for this project, thus it will not be discussed here.

4.3.1 Segment Level Recurrence with State Reuse

To address the limitations of using a fixed-length context, the idea is to reuse the hidden states obtained in previous segments as an extended context when processing the next new segment. Instead of computing the hidden states from scratch, the cached and reused hidden states serve as memory for the current state. Since information can be propagated through the recurrent connection

The $n-th$ layer hidden state for segment $S_(t+1)$ can be expressed as the following three equations:

$$\tilde{h}_{T+1}^{n-1} = [SG(h_T^{n-1}) \circ h_{T+1}^{n-1}], \text{(extend context)}$$
$$q_{T+1}^n, k_{T+1}^n, v_{T+1}^n = h_{T+1}^{n-1} W_q^\top, \circ h_{T+1}^{n-1} W_k^\top, \circ h_{T+1}^{n-1} W_v^\top \text{(query, key, value)}$$
$$h_{T+1}^n = \text{Transformer-Layer}(q_{T+1}^n, k_{T+1}^n, v_{T+1}^n)\text{(self-attention + feed-forward)}$$

4.3.2 Relative Positional Encodings

Since TransformerXL processes segments recurrently and reuse previous hidden states, it changes to encodes the relative positional information in the hidden states and replaces absolute embeddings U_j with relative embeddings R_{ij} , where R is a sinusoid encoding matrix. The original absolute attention equation ($A_{i,j}^{abs}$) and the new relative attention equation ($A_{i,j}^{rel}$) are shown below:

$$A_{i,j}^{abs} = q_i^\top k_j = E_{x_i}^\top W_q^\top W_k E_{x_j} + E_{x_i}^\top W_q^\top W_k U_j + U_i^\top W_q^\top W_k E_{x_j} + U_i^\top W_q^\top W_k U_j$$
$$A_{i,j}^{rel} = E_{x_i}^\top W_q^\top W_{k,E} E_{x_j} + E_{x_i}^\top W_q^\top W_{k,R} R_{i-j} + u^\top W_{k,E} E_{x_j} + v^\top W_{k,R} R_{i-j}$$

4.3.3 QA-XL, Integration with QANet

All the TransformerXL changes are made inside the QANet encoder block, the major changes includes:

- Define memory cache for MultiHead Attention layer in the encoder block.
- Extending the context by concantenating queries with memory(mems) and replace the QANet self Attention layer.
- Add new variables to represent relative positions.
- Implement the position wised feedforward layer to replace the QANet feedforward layer.

PS: Although above summary seems quite straightforward, but the actual implementation is quite challenging, I constantly got dimension mismatch exception on the relative position variables and the memory cache. At the end, I corrected my implementation by padding those variables, I got inspired by looking into a git repo inSam/QA-XL[6].

5 Experiments

5.1 Data

The dataset (SQUAD 2.0) for this project came from the default project code, and was set up by setup.py script. It has three splits: train, dev and test.

- train (129,941 examples): All taken from the official SQuAD 2.0 training set.
- dev (6078 examples): Roughly half of the official dev set, randomly selected.
- test (5921 examples): The remaining examples from the official dev set, plus hand-labeled examples.

5.2 Evaluation method

Results are evaluated based on the Exact Match (EM) score and F1 score.

5.3 Experimental details

All the experimentation are done on my local machine, Ubuntu 20.04, NVIDIA Quadro RTX 5000 Graphic Card.

- **Baseline - BiDAF**: default configurations from the starter code, AdaDelta optimizer, learning rate set to 0.5, hidden dimension set to 100, and 64 batch size set. The total training time is close to 8 hours.
- **QANet+Char Embedding** : beside the default BiDAF configurations, I chose ADAM optimizer, with learning rate set to 0.001, β_1 set to 0.8, hidden dimension set to 96 and batch size set to 8. I use a warm-up learning rate that exponentially increases from 0 for the first 2000 iterations, this is for compensating the smaller batch size. Training time is close to 50 hours.
- **QANet+TransformerXL**: I chose same parameters from QANet, the training time is around 48 hours.
- **DistilBert**: I fine-tuned a distilBert model by using huggingface[7] code to compare its performance with QANet and QA-XL model. I chose batch size as 12, max sequence length as 384, doc stride as 128, learning rate as 3e-5 and 6 training epochs. The training time is around 2 hours.

5.4 Results

Here are the F1 and EM results for each experiment, please refer to figure 2,3,4,5 for QANet and QA-XL's training metrics.

See from table 1 below, QANet result outperforms the baseline BiDAF model, which is expected, but QA-XL result is below expectation, I think there are a few reasons:

- Lots SQUAD context is not long text, so transformerXL's advantage is not showing up.

Models	F1	EM	Epochs
Baseline BiDAF	52.19	52.19	30
QANet, Dev Set	68.198	64.325	30
QANet, Test Set	64.862	61.386	30
QA-XL, Dev Set	66.45	63.79	30
QA-XL, Test Set	64.18	60.58	30
Fine-tuned DistilBert	87.357	78.874	6

Table 1: Best Experiment Score, Non-PCE

- Training epochs is not long enough, by comparing figure 2,3 and figure 4,5, we could see QA-XL's NLL curve is still keep going down after 30 epochs while QANet's NLL curve gets flat. I think QA-XL could get slight better result if we increase to the training epochs to 40.

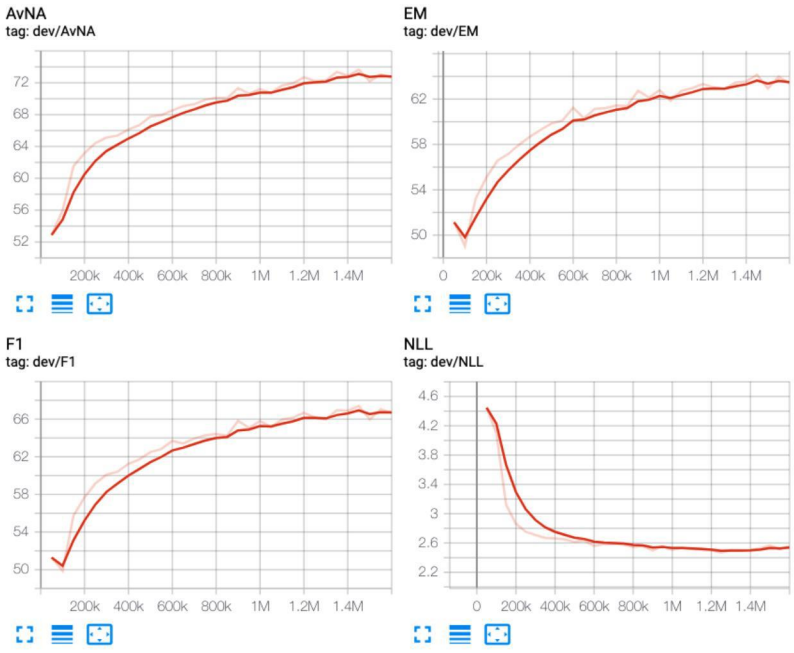

Figure 2: QANet tensorboard visualization - Dev

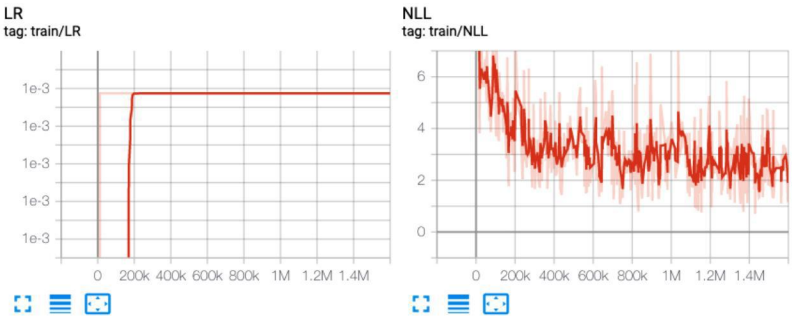

Figure 3: QANet tensorboard visualization - Train

6 Analysis

Although QANet-XL underperformed QANet, the TensorBoard visualization (figure 2,3,4,5) shows that its early epoch learning rate was slower than QANet. The QANet's NNL essentially plateued after 1.5 million steps, while QANet-XL was still improving. This may indicate that given enough training time (ex: 40 train epochs), QANet-XL could outperform vanila QANet.

From an informal survey of the selected examples in Tensorboard (QA-XL), it appears that, while the AvNA metric improves, it still tend to predict N/A fairly often, even when there is an answer. This bias toward choosing no answer might indicate that, it would be good to use a higher drop out rate or different learning rate to improve the early epoch performance of QA-XL.

Unfortunately, I didn't get enough time to perform hyper parameter tuning, so I don't have quantitative measurement on how much improvement it might gain for different hyper parameter combinations (batch size, number of heads, hidden size and embedding size), I would leave this for future exploration.

7 Conclusion

In this project, I implemented QANet architecture and successfully applied TransformerXL method into it. From the experiment, this combination approach (QA-XL) seems promising to slightly outperform QANet model with long enough training time. Due to the limitation of time and computing resources, I didn't finish setup the Katib[8] hyperparameter tunning environment to fine tune the hyper parameters.

I'm intrigued to apply different hyper parameter tuning algorithms to QANet and QA-XL, I hope to figure out what kind impact for the model performance a good set of hyper parameters can have.

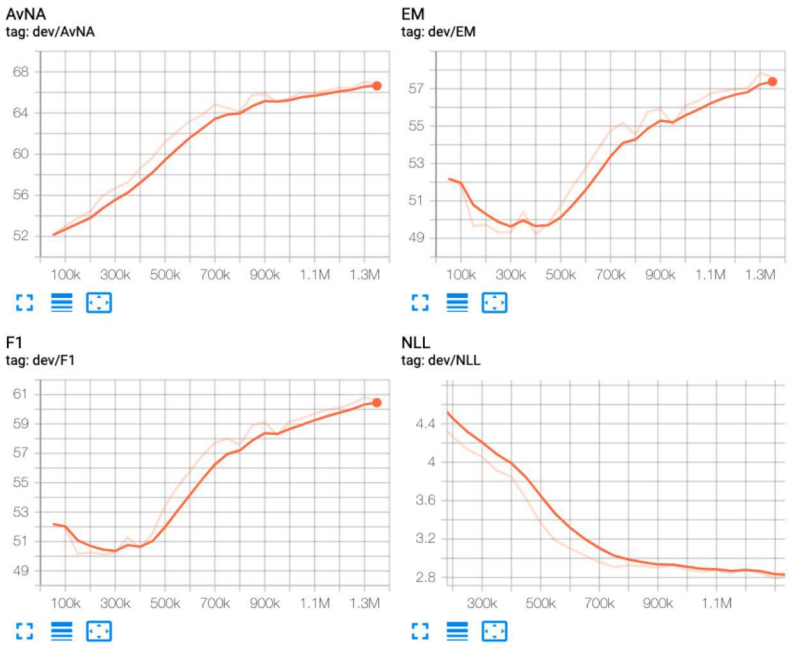

Figure 4: QA-XL tensorboard visualization - Dev

References

[1] 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, 2018.

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[4] heliumsea/qanet-pytorch: https://github.com/heliumsea/qanet-pytorch.

[5] andy840314/qanet-pytorch-: https://github.com/andy840314/qanet-pytorch-.

[6] insam/qa-xl: https://github.com/insam/qa-xl.

[7] huggingface/transformers: https://github.com/huggingface/transformers/.

[8] Katib: Automatic hyperparameter tuning.

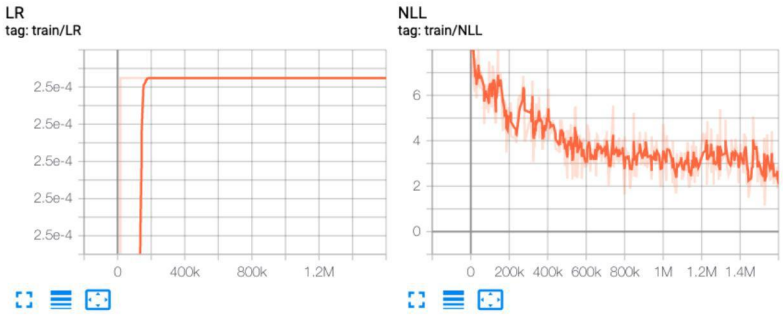

Figure 5: QA-XL tensorboard visualization - Train