Transformer-XL on QAnet

James Chao jbchao@stanford.edu

Problem

QAnet [1] uses transformers that don't suffer from long-term dependancy information loss like RNNs do, but since they process entire sentences at a time, they can only learn from context their fixed input window. Before pre-trained models like BERT were introduced, QAnet was the top performer on SQuAD v1.

My Approach

I will work on adapting QAnet to SQuAD v2, which requires much more precise modelling given the occurrance of "no answer" questions. I will use the techniques introduced in **Transformer-XL** [2] to address the fixed window issue by reintroducing recurrance into transformers and storing previous hidden states in long-term memory (similar to an LSTM) and using them in the encoding of subsequent sentences.

Conclusions

The transformer-XL returned dissapointing results, but this is likely due to an implementation flaw as the base QAnet does start learning after some time. I expect the transformer-XL architecture won't revolutionize anything, but should be able marginally improve performance. I think it's definitely worth pursuing different ways of speeding up classic transformer architectures, especially since increasingly popular pre-trained models like BERT could benefit as well from these improvements. As such, I will continue trying to perfect my transformer-XL architecture, and will be satisfied once it beats my QAnet-based model.

Model Architecture

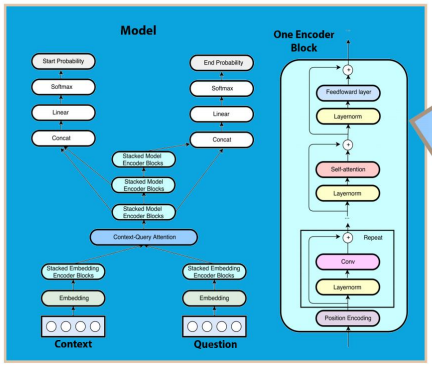

Figure 1: This is the original QAnet architecture [1].

My model uses a similar architecture to the original **QAnet**, with the following major exceptions:

Convolutional Embeddings

Similar to QAnet, I pass character embeddings through a 2D convolutional layer to learn inter-character relations, but I add an adittional 1D convolutional layer on the combined character and word embeddings to learn relations between parts of words. Additionally, instead of a linear feedforward layer in each encoder block, I use convolutional layers to convolve over attention distributions.

Stacked Encoder Blocks

I use the same general architecture for the transformer encoder blocks as QAnet, except for the linear feedforward layer, which I substitute for a convolutional layer to capture any hidden relationships between attention distributions. For self-attention, I use ***nn.MultiheadAttention***.

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.

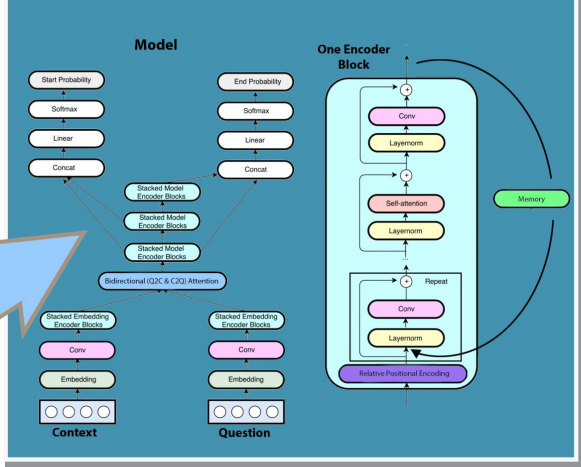

Figure 2: My final model architecture, which draws inspiration from both QAnet [1] and TransformerXL [2]

Applying the following **TransformerXL** strategies, as well as the changes to **QAnet** I mentioned below, gives my final model architecture.

Recurrance with Memory

TransformerXL's main innovation is to store previous hidden states inside a ***mems*** tensor, to be concatenated to the current states in the next sequence's self-attention calculation, similar to implementing residual connections between layers [2].

Relative Positional Encoding

Since concatenating states will make absolute positional embeddings meaningless, I use relative positional encodings [2].

Results

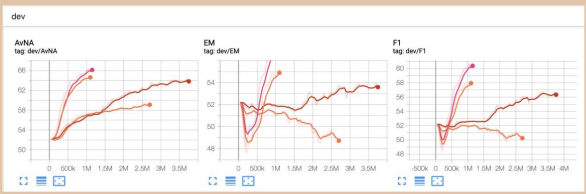

Figure 3: The results of my training. The higher-ending orange line is the baseline BiDAF model, the pink line is the BiDAF model with convolutional embeddings, the red line is my base QAnet model, and the bottom orange line is my transformer-XL model.

Model	Baseline	Conv. Embedding	QAnet	Transformer-XL
AvNA	64.88	**69**	64.31	59.2
EM	55.28	**59.5**	53.44	48.46
F1	58.35	**63**	56.15	50
NLL	3.04	**2.78**	3.34	4.07
Num Iterations	1.1M	2.5M	4M	2.7M

Figure 4: The final results of each model, evaluated on the dev set.

Experimental Details

Dataset: modified SQuAD v2 provided in starter code
Inputs: (Q, C, A) triplets
Output: start, end in C
Task: predict where A is in C (if it's in C at all)
Evaluation metrics: F1, Em, AvNA
Baseline: provided word-embedding BiDAF models

Analysis

The base QAnet model performed well, albeit slowly. Due to time and Azure constraints, I was not able to train the model fully, but it did show much promise with an AvNA score of XXX, an F1 score of XXX, and an EM score of XXX. When I add the transformer-XL modifications, however, the model refuses to learn. This is not due to an incompatibility between the two architectures—they have been successfully merged before—but rather suggests an implementation flaw. I will spend the next couple of days ironing out these details. Some notable observations include:

- Adding just the convolutional embedding layer, however, seemed to have a marked effect on performance and scores well over the baseline on every metric.
- The transformer-XL did continue to improve its AvNA score over time despite the fact that its other scores steadily decreased.