Reproduce Simple QANet on SQuAD 2.0

Zhengqi Zhu stevezhu@stanford.edu
Stanford Center for Performance Development

BACKGROUND:

Recently, there is a growing interest in the tasks of machine reading comprehension and automated question answering. By the time the QANet paper was published, most of the end-to-end machine-reading and question answering models are primarily based on two key ingredients: recurrent neural networks (RNNs) and attention mechanisms. However, despite the success of accuracy of these models, they are comparatively slow to train and inference due to their recurrent nature. This weakness leads to a high turnaround time for experimentation and limits researchers from rapid iteration. Also, it prevents the models from being used for larger datasets and being deployed in real-time machine comprehension systems applications.

Figure 1: example dataset

PROBLEM:

The main goal of this project is to successfully implement QANet with PyTorch and build a Question-Answering system using this model. Then achieve a higher EM and F1 score on the SQuAD 2.0 dataset than the given baseline model using the implemented model. The secondary goal is to apply a masking mechanism on the attention layer, modify the model size, the number of layers, or other improvements to analyze if these changes could make the implemented model performs better.

METHODS: QANET

As described in the QANet paper, the higher-level architecture contains 5 layers:

1. **Input Embedding Layer**: The embedding of each word w is obtained by concatenating its word embedding and character embedding. The word embedding is fixed during training and initialized from the p1 = 300 dimensional pre-trained GloVe word vectors. All the out-of-vocabulary words are mapped to a <UNK> token, whose embedding is trainable with random initialization. To obtain the character embedding, each character is represented as a trainable vector of p2 = 200 dimensions. The word length is truncated or padded to 16. The output from this layer of a word x is [xw; xc] ϵR^(p1+p2), xw is the word embedding and xc is the convolution output of character embedding of x.
2. **Embedding Encoder Layer**: The structure of the encoder layer is [n × convolution-layer + self-attention-layer + feed-forward-layer]. The convolution layer has a kernel size of 7, d = 128 filters, and each block has 3 convolution layers. The self-attention layer adopts the multi-head attention mechanism. The number of heads is 8. The total number of encoder blocks is 1. The output of this layer is d = 128 dimensions.
3. **Context-Query Attention Layer:** C denotes the encoded context and Q denotes the encoded query. First compute the similarities between each pair of context and query words, rendering a similarity matrix S ϵR^(n×m). Then apply softmax function to each row of S to get a matrix ¯S. The context-to-query attention is computed as A = ¯S ·Q^T ϵR^(n×d). The query-to-context attention is computed as B = ¯S · ¯S^T · C^T , where ¯S is column normalized matrix of S computed by softmax function.
4. **Model Encoder Layer:** The input of this layer at each position is [c,a,c ⊙a,c ⊙b][4], where a and b are respectively a row of attention matrix A and B. The number of convolution-layer is 2 and total number of blocks are 5. Other parameters are the same as the Embedding Encoder Layer.
5. **Output Layer:** Each example in SQuAD is labeled with a span in the context containing the answer. The probabilities of the starting and ending position are modeled asp1 = softmax(W1[M0; M1]), p2 = softmax(W2[M0; M2]), where W1 and W2 are two trainable variables and M0, M1, M2 are the outputs of the three model encoders from button to top. Compute the product of its start position and end position probabilities to get the score. Finally, the objective function is defined as the negative sum of the log probabilities of the predicted distributions indexed by true start and end indices, averaged over all the training examples: $L(\theta) = -\frac{1}{N}\sum_i^N[\log\left(p_{y_i^1}^1\right) + \log(p_{y_i^2}^2)]$ where y_i^1 and y_i^2 are respectively the ground truth starting and ending position of example i. At inference time, the predicted span (s,e) is chosen such that p1sp2e is maximized and s < e.

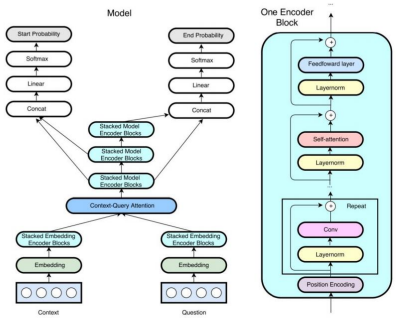

Figure 2: QANet model

EXPERIMENT:

- **Data:** As described in the default project handout, the dataset is SQuAD 2.0. This dataset was being pre-processed as given and split into three parts: train, dev, and test. The train set contains 129,941 examples, the dev set contains 6078 examples, and the test set contains 5915 examples.
- **Evaluation method:** As introduced in the default project handout, the evaluation metrics are EM and F1 scores. Exact Match is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly. F1 is a less strict metric – it is the harmonic mean of precision and recall.
- **Experimental details:** For hardware, I used Google CoLab NVIDIA Tesla P100 GPU with 16 GB memory. Due to the limitation of GPU memory, I have to decrease the number of blocks of each encoder layer. About 12.58 GB was used in the experiment. The epoch number is set to 15 and batch size of 16. The training time is around 15 hours to get the result. The learning rate is 0.5 as default.
- **Results:** As attached below, the blue line refers to QANet implemented, and the orange line refers to the baseline model. After 15 epochs, our QANet model got EM 63.855 and F1 67.675 on dev set and the baseline, after 30 epochs, got EM 56.02 and F1 59.57. QANet model also got EM 62.418 and F1 65.749 on test set.

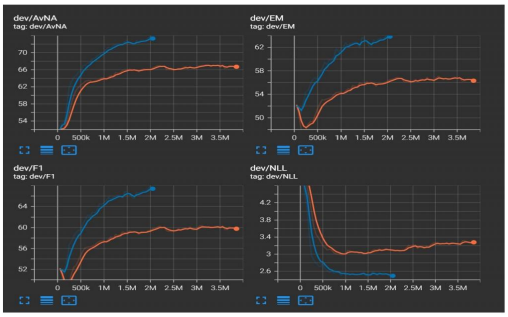

Figure 3: Results

Analysis:

The result shown above is not as good as the QANet paper. Excepting the limitation of model size, the paper also used a data augment skill that translates the dataset into French and then translates it back. In this way, the original dataset would be doubled. In this project, I did not make any improvement on the input, and that could be future work.

Conclusion:

This project successfully reproduced a simple QANet, and the performance has achieved the target that is higher than the baseline model. However, the EM and F1 score is still not as expected as in the paper. Also, in the paper, the researchers highlighted their comparatively short training time but during implementation, the training time is not shortened as expected. This may be both a model implementation and hardware (GPU) issue, and I still need to do more research on this area.

References:

1. Minh-Thang Luong Rui Zhao Kai Chen Mohammad Norouzi Adams Wei Yu, David Dohan and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In International Conference on Learning Representations (ICLR), 2018.
2. Percy Liang Pranav Rajpurkar, Robin Jia. Know what you don't know: Unanswerable questions for squad. In Association for Computational Linguistics (ACL), 2018