SQuAD 2.0 Question Answering with Reformer

Stanford CS224N Default Project

Haiyuan Mei
Department of Computer Science
Stanford University
hmei0411@stanford.edu

Abstract

Transformer based models routinely achieves state of the art results. Training these models is prohibitively expensive, limiting it's use mostly in large industrial labs. Various improvements and models have been created to mitigate this problem, including XLNet [1], ALBERT [2], etc. In this course project, I aim to explorer a Reformer [3] based QANet model [4], using pytorch for the question-answering tasks defined in SQuAD2.0 [5]; I will train three models: the baseline BiDAF [6] model without character level word embedding, the QANet model which applies the original Transformer [7] model, and the Reformer based QANet. The project will be non-PCE based, confining the study on the Reformer model and comparison between Reformer and Transformer [7].

1 Introduction

SQuAD 2.0 (Stanford Question Answering Dataset) [5] is a closed-domain, extractive QA dataset which tailors it's answers in such a way to facilitate answering factoid questions. Given a short background passage (context) and a question sentence, the goal is to answer the question by finding a span of text in the background text. The problem is defined in such a way that is easy to be modeled by a computer program, such as the answer is given by two indexes in the context passage (for unanswerable questions, answers are simply given by two '0' index words, which are inserted to the sequence in pre-processing stage), which can be intuitively modeled by softmax probabilities and cross entropy loss.

The BiDAF(Bidirectional Attention Flow for Machine Comprehension) [6] model uses bi-directional LSTMs to encode both the context and question sequences, which are then fed into an attention layer composed of both 'question to context' and 'context to question' attentions to come up with an attention matrix, then through modelling layer and finally an output layer which gives a pair of softmax probabilities that can be used to calculate the loss using two cross entropy terms. This structure is inherited by QANet(Combining Local Convolution with Global Self-Attention for Reading Comprehension) [4] model, in the sense that it applies the same idea of a sequence encoder, an question-context/context-question attention layer(Dynamic coattention networks [8]), a modelling layer and a final output layer for the two softmax probabilities. The only difference is that no LSTM network is used at all in QANet, instead it applies the recent introduced Transformer [7] model as it's encoding layer and model layer network.

Attention Is All You Need [7] is the foundation of almost all of today's state of the art language models. Unlike all the RNN based networks, transformers can attend to any word in the sequence directly, hence it has no diminishing gradients problem, which means it can be theoretically used for arbitrarily long sequences. Almost all of today's state of the art models, including but not limited to BERT [9], XLNet [1], ALBERT [2], etc.

Transformer based models constantly achieves state of the art results in a number of Natual Language Processing tasks; Particularly, as shown in SQuAD 2.0 leaderboard (Stanford Question Answering Dataset) [5], all submissions ranked before 12 have achieved better performance compared to human

answer and they are all transformer based models. However one major problem for transformers is it's demand for computation. Training of $\text{BERT}_{\text{BASE}}$ was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total), Training of $\text{BERT}_{\text{LARGE}}$ was performed on 16 Cloud TPUs (64 TPU chips total); each pretraining took 4 days to complete. XLNet is trained train on 512 TPU v3 chips for 500K steps with an Adam weight decay optimizer, linear learning rate decay, and a batch size of 8192, which takes about 5.5 days. It cannot even fine tune a task on a single TPU with a these pretrained models.

Reformer [3] model aims to improve transformer performance problem by introducing three techniques.

- **Reversible layers**[10]: An N-layer model is N-times larger than a single layer due to the fact that activations need to be stored N times for back propagation purpose. Reversible layers enables storing only a single activation but keeps for an N-layer model.
- **Chunk feed-forward layers**: Split feed-forward activations and propress them in chunks removes the d_{ff} factor and saves memory inside feed forward layers.
- **locality-sensitive hashing attention**: The biggest problem with transformers is for a sequence length L the complexity of self-attention computation takes $O(L^2)$ complexity both for memory and computation. LSH based attention is able to reduce the complexity to $O(L \log L)$ and hence can be used for sequences longer than 1M.

2 Models

In this project I mainly explorer three models for SQuAD 2.0 dataset. The BiDAF [6] baseline as described in default final project handout; the QANet [4] model and a Reformer [3] based QANet, which are described as following.

2.1 BiDAF baseline

The BiDAF(Bidirectional attention flow for machine comprehension [6]) model is a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. The default course project code for BiDAF model(https://github.com/minggg/squad) is used to train the first baseline model. The model is divided into six layers.

- **Character Embedding Layer** maps each word to a vector space using character-level CNNs following [11].
- **Word Embedding Layer** maps each word to a vector space using a pre-trained GloVe word embeddings. The D dimentional GloVe embedding (here D=300) is then projected to H dimention to form the final hidden layer dimention size, and then put through a highway network. The final refined word embedding is concatenated with the character embedding vector as input to the next step - the Contextual Embedding layer.
- **Contextual Embedding Layer** utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context. This layer uses a 2 layer Bi-LSTM to generate two different H dimensional hidden states, which are then concatenated into a single 2H dimensional hidden states as input to the next part of the network.
- **Attention Flow Layer** The core part of the BiDAF model is the bidirectional attention flow layer, it couples the query and context vectors and produces a set of query aware feature vectors for each word in the context. Let $c_i, q_i \in \mathbf{R}^2 H$ be context and question hidden states, we first compute a Similarity Matrix$S \in \mathbf{R}^{NxM}$, with each pair (c_i, q_j) we have a similarity score S_{ij} given by $S_{ij} = w_{\text{sim}}^T[c_i; q_j; c_i \circ q_j]\mathbf{R}$; We then calculate two attentions:
  - **Context-to-Question attention**, given by a_i as below:

$$\begin{aligned} &\bar{S}_{i,:} = \text{softmax}(S_{i,:}) \in \mathbf{R}^M, \forall i \in 1, 2, ..., N \\ &a_i = \textstyle\sum_{j=1}^{M} \bar{S}_{i,j} q_j \in (R)^{2H} \end{aligned} \tag{1}$$

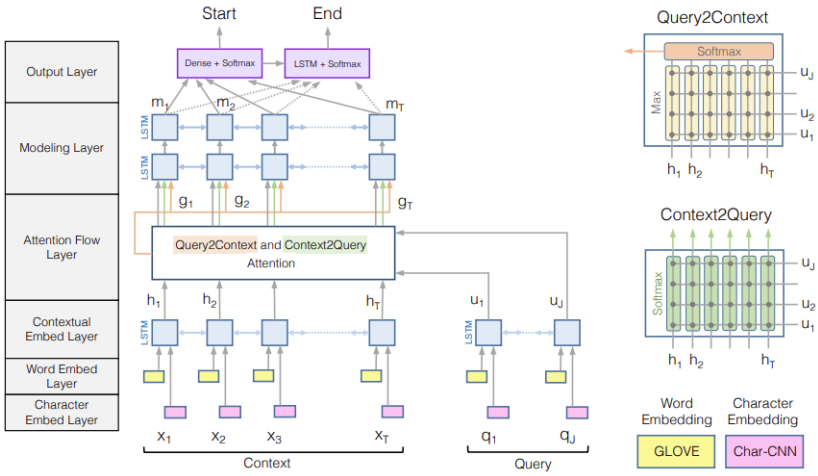

Figure 1: BiDirectional Attention Flow Model

– **Question-to-Context attention**, given by b_i calculated as below:

$$\begin{aligned} \bar{\bar{S}}_{:,j} &= \text{softmax}(\bar{S}_{:,j}) \in \mathbf{R}^N, \forall j \in 1, 2, ..., M \\ S' &= \bar{S}\bar{\bar{S}}^T \in \mathbf{R}^{N \times N} \\ b_i &= \sum_{j=1}^{N} S'_{i,j} c_i \in (R)^{2H}, \forall i \in 1, 2, ..., N \end{aligned} \tag{2}$$

Finally the bidirectional attention output g_i is given by combining the context hidden state c_i together with the two attentions a_i and b_i: $g_i = [c_i; a_i; c_i \circ a_i; c_i \circ b_i] \in \mathbf{R}^{8H}$.

- **Modeling Layer** This is another BiLSTM layer use the above provided attention output g_i as input to refine the sequence of vectors after the attention layer: $m_i = \text{BiLSTM}(g_i) \in \mathbf{R}^{2H}$. Since the modeling layer comes after the attention layer, the context representations are conditioned on the question by the time they reach the modeling layer.
- **Output Layer** provides an answer to the query. Concretely, the output layer takes as input the attention layer outputs g_i, the modelling layer outputs m_i, and applies yet another BiLSTM on top of modelling layer output m_i to produce a vector m'_i for each m_i: $m'_i = \text{BiLSTM}(m_i) \in \mathbf{R}^{2H}$. The final softmax output denoting the start and end index of the answer is given by:

$$\begin{aligned} p_{\text{start}} &= \text{softmax}(W_{\text{start}}[G; M]) \\ p_{\text{end}} &= \text{softmax}(W_{\text{end}}[G; M']) \end{aligned} \tag{3}$$

The training loss sums up the two cross-entropy loss with respect to the true start and end index of theanswer, averaged over all examples:

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log(p^1_{y^1_i}) + \log(p^2_{y^2_i}) \tag{4}$$

2.2 Multi-Heads attention in Transformer

Transformer is first introduced in Attention Is All You Need [7], and is considered a better substitute to RNN networks such as LSTM and GRU ever since. It is the foundation of almost all of today's state of the art language models, including but not limited to BERT, XLNet, ALBERT, etc. We will not attempt to describe the whole original Transformer model, instead will look into the essense of transformer, the so called 'Scaled Dot-Product Attention' and 'Multiple Heads Attention', which are described as follows.

Suppose the word embedding matrix is denoted as $E \in \mathbf{R}^{d \times n}$, where d is the word vector dimention and n is the size of vocabulary. The transformer model first makes 3 linear transformations from E to $K \in \mathbf{R}^{d_k \times n}$, $Q \in \mathbf{R}^{d_k \times n}$, $V \in \mathbf{R}^{d_v \times n}$: $K = W_K E$, $V = W_V E$, $Q = W_Q E$; The 'Scaled

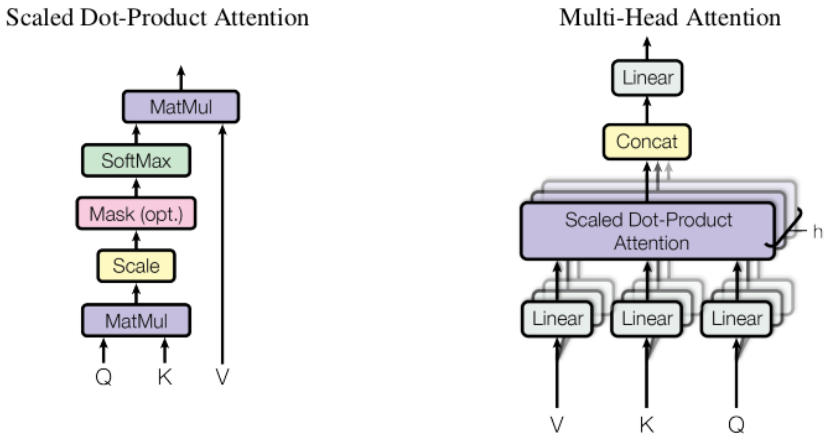

Figure 2: Scaled dot-product attention and Multi-Heads attention

Dot-Product Attention' and 'Multiple Heads Attention' are then built upon the three matrices K, Q, V:

$$\begin{aligned} &\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \\ &\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \text{head}_2, ..., \text{head}_h\right)W^O \\ &\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{5}$$

Where $W_i^Q \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbf{R}^{d_{model} \times d_v}$, and $W^O \in \mathbf{R}^{hd_v \times d_{model}}$. and the output of the multi head attention is a sequence of d_{model} word vectors.

2.3 QANet

The QANet [4] is very similar to the default baseline BiDAF model, in the sense that they all follow the BiDAF six layer structure as described in the BiDAF section. The key component which makes QANet outperform BiDAF is the stacked transformer based embedding encoders and stacked transformer based model encoders; and the transformer used in QANet has an extra type of sublayer, the convolutional sublayer which starts with a layer norm module and a convolutional module.

Of the six BiDAF layers as described in the BiDAF section, QANet replaced the two RNN based layers: Embedding Encoder Layer and Model Encoder Layer, with two transformer based layers: Stacked Embedding Encoder layer and Stacked Model Encoder layer. Instead of using standard transformer encoder, the encoder layer used in QANet is a stack of the following basic building block: [convolution-layer + self-attention-layer + feed-forward-layer], as illustrated in the upper right of Figure 3. We use depthwise separable convolutions-[12] [13] rather than traditional ones, as it is memory efficient and has better generalization. The kernel size used in is 7, with 4 conv layers in each encoder block. Multi-Heads attention mechanism is applied in QANet, with 8 heads through out all layers. Each of these basic operations(conv, self attention and feed forward) is wrapped in a residual block, as shown lower-right in Figure 3.

For all other layers (character/word embedding layers, cq/qc attention and output layer), QANet shares the same structure as BiDAF as described in the BiDAF section. QANet code is adapted from QANet-pytorch for study purpose.

2.4 Reformer

Reformer(the efficient transformer [3]) is designed to handle 1 million words in context windows, while using only 16GB of memory. It combines following techniques in order to enable the self attention mechanism for long sequences: Memory-efficient attention and locality-sensitive hashing to reduce the memory and computations complexity from $O(l^2)$ to $O(l \log l)$, split activations inside feed-forward layers and processing them in chunks to remove the d_{ff} factor and saves memory inside feed-forward layers, and reversible residual layers (RevNets [14])to eliminate the problem storing n

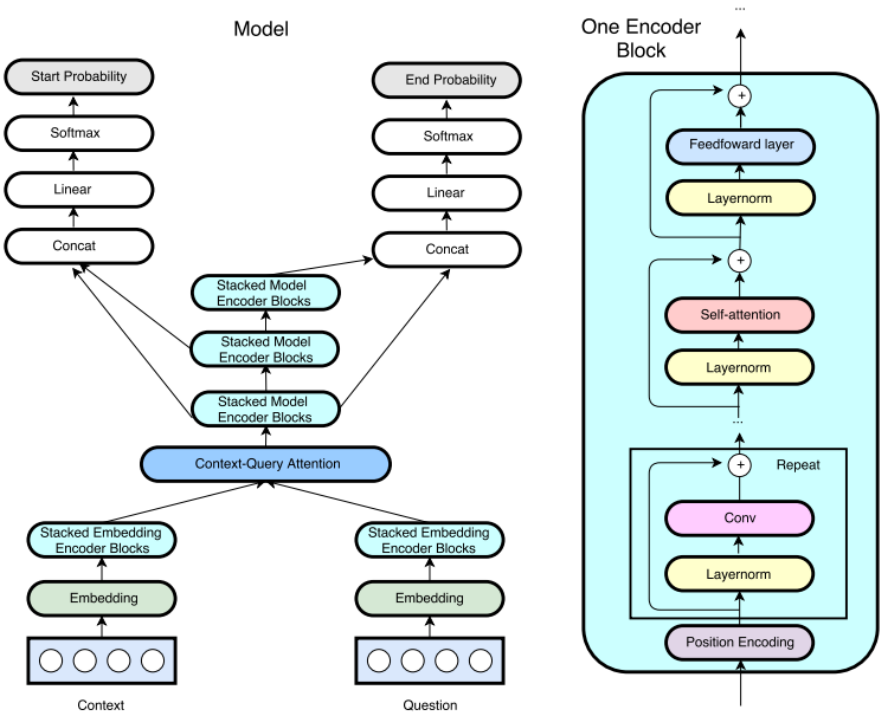

Figure 3: QANet network and the transformer based encoder block

copies of actions for n layer transformer models. We will discuss in detail on how Reformer handles long sequences, and refer the introduction of RevNets to its original paper and Reformer paper. The use of Reformer for SQuAD 2.0 task with a QANet structure is original, in order to quick prototype the course project training, the Reformer model is adapted from reformer-pytorch.

2.4.1 Memory-efficient attention

As can be seen from transformer Scaled Dot-Product Attention equation $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, the term QK^T will have both computational and memory complex $O(l^2)$ in order to calculate the self attention for a length l sequence (in which case the key matrix K and query matrix Q are the same in Reformer, and the attention calculation involves loading a huge $(R)^{l\times l}$ matrix), which makes it very hard to even fine tune a model on a single GPU. Reformer model made use of the fact that in order to calculate self attention, the QK^T does not need to be loaded to memory as a whole, it can be computed for each query q_i separately as:

$$\text{Attention}(q_i, K, V) = \text{softmax}\left(\frac{q_i K^T}{\sqrt{d_k}}\right)V \tag{6}$$

This will make the Reformer model less efficient compared to transformer in a sense that the computation is no long parallel, but it only uses memory proportional to sequence length, makes it plausible to train a self attention model with arbitrarily long sequences.

2.4.2 Locality Sensitive Hashing Attention in Reformer

The use of Locality Sensitive Hashing in the calculation of attention is a major technique in Reformer for self attention calculation. Since we are actually only interested in softmax(QK^T), and softmax is dominated by the largest elements, for each query q_i we only need to focus on the keys in K that are closest to q_i. The idea of hashing attention is to only consider a small subset of closest keys for the l elements in Q, which can be achieved by locality-sensitive hashing (LSH). Figure 4 illustrates the whole process: firstly traverse the l elements and calculate a hash code $h(x)$ for each of the

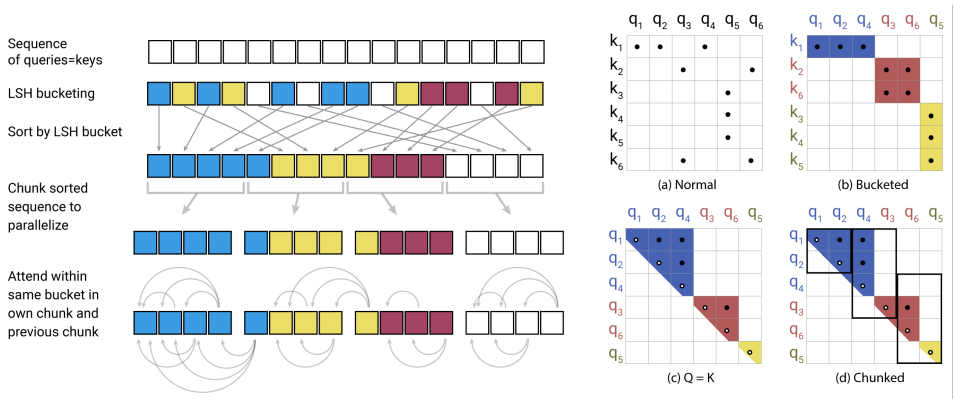

Figure 4: LSH Attention: hash-bucketing, sorting, chunking steps and the resulting causal attentions

input x, which takes $O(l)$ complexity; then sort the elements by the hash value which takes $O(l \log l)$ complexity. The overall complexity before chunking is $O(l \log l)$.

Locality Sensitive Hashing is a hash function that assigns each high dimensional vector x to a hash h(x) with the property that if the vectors are close to each other they tend to get the same hash bucket high probability and otherwise not. Reformer makes use of a well known angular distance based LSH [15], which is illustrated in Figure 5. Illustrating the Reformer gives a simplified explanation on the angular locality sensitive hash used in Reformer. To formalize it, in order to get b hashes, we first fix a random matrix R of size [d_k , b/2]. We then define $h(x) = \arg\max([xR; -xR])$ where [u; v] denotes the concatenation of two vectors. This method is a known LSH scheme[15] and is easy to implement and apply to batches of vectors.

We now formalize the LSH attention calculation used in Reformer. First of all, the normal attention equation $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ is rewritten for a single query position into:

$$o_i = \sum_{j \in \mathbf{P}_i} \exp(q_i k_i - z(i, \mathbf{P}_i)) v_j, \text{where } \mathbf{P}_i = \{j : i < j\} \tag{7}$$

where $\mathbf{P}_i$ denotes the set that q_i attends to, and z is the normalizing term in the attention (which also can be used to omit $\sqrt{d_k}$). For batching purposes, attention over a larger set $\tilde{P}_i = \{0, 1, ..., l\}\mathbf{P}_i$ is used to mask out elements not in $\mathbf{P}_i$:

$$o_i = \sum_{j \in \mathbf{P}_i} \exp(q_i k_i - m(j, \mathbf{P}_i) - z(i, \mathbf{P}_i)) v_j, \text{where } m(j, \mathbf{P}_i) = \infty \text{ if } j \notin \mathbf{P}_i \text{ else } 0 \tag{8}$$

The LSH attention can then be thought of in terms of restricting the set $\mathbf{P}_i$ for which a query position i can attend to, by allowing attention only within a single hash bucket:

$$\mathbf{P}_i = \{j : h(q_i) = h(k_j)\} \tag{9}$$

3 Experiments

3.1 Data

I used SQuAD V2.0 (Stanford Question Answering Dataset) [5] to train all three models. Each input contains a context paragraph sequence no longer than 400 words, and a question sequence no longer than 50 words; and the output is the answer to the question, denoted by a pair of indexes indicating the starting and ending index of the span of text in the context paragraph. There are roughly half of the SQuAD examples are no-answer, in order to support this scenario, each context paragraph will be inserted a starting word(index 0) as the first word, and the index 0 will be used to output an unanswered question. The data is devided into 3 portions: train (129,941 examples), dev (6078 examples) and test (5915 examples).

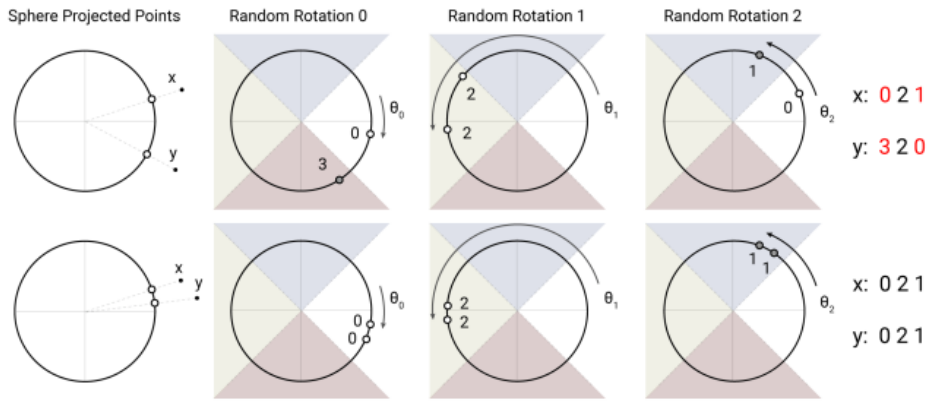

Figure 5: An angular locality sensitive hash uses random rotations of spherically projected points to establish buckets by an argmax over signed axes projections. In this highly simplified 2D depiction, two points x and y are unlikely to share the same hash buckets (above) for the three different angular hashes unless their spherical projections are close to one another (below).

3.2 Evaluation method

Same as the default course model, in this project I will use three metrics to to evaluate each of the models: Exact Match (EM), F1 and AvNA, as described below:

- **Exact Match (EM)** Measures whether the prediction is correct: $\text{EM} = \begin{cases} 1, & \text{correct.} \\ 0, & \text{otherwise.} \end{cases}$
- **F1**: he harmonic average of precision and recall: $\text{F1} = 2 \cdot \frac{precision - recall}{precision + recall}$
- **AvNA (Answer vs. No Answer)**: measures the classification accuracy of your model when only considering its answer (any span predicted) vs. no-answer predictions.

3.3 Experimental details

Unlike the default settings in BiDAF model, in order to be able to use Reformer model, the max context sequence length is changed to be 512, the max question sequence length is set to be 64, to have a length that can be divided by two hash buckets size, which is set to be 16. The batched length is also aligned to be a multiple times length as the double hash bucket size, which in our case is a multiple of 32. The Reformer model is trained with two different hyper-parameter settings, associated with two different baseline comparison as described below.

- **Simple Reformer** For fast prototype purpose, firstly I started from a non character embedding based QANet+Reformer model. With batch size 100, hidden size 100, a single layer reformer input encoder sublayer with 1 convolutional layer which uses kernel size 7, hash bucket size 16, number of hash rounds 8 and a single attention head; and 2 double layer reformer model encoder sublayers with 2 convolutional layer, hash bucket size 16, number of hash rounds 8 and a single attention head. Two baseline models are trained to evaluate the performance of the simple reformer model: the default BiDAF is without character embedding, trained with batch size 64, hidden size 100, 1 layer convolutional embedding encoder and 2 layers model encoder with a single nVidia GTX 1070TI GPU; and the original QANet model without character level embedding layer, using batch size 32, hidden size 100, a single layer transformer input encoder sublayer with 4 cnn modules, and a 7 layer model encoder transformer sublayers each with 2 cnn modules. All models are trained for 30 epochs for about 5 hours for BiDAF, 9 hours for QANet and 15 hours for the simple Reformer model on a single GPU (Figure 6 and Figure 7).
- **Complex Reformer** To get a better performance, I then the added the reformer model with a trainable character embedding layer, with batch size 64, hidden state dimension $d_{\text{model}} = 96$

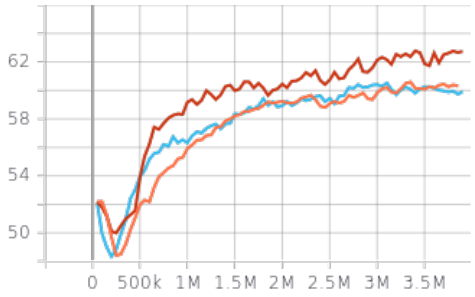

Figure 6: F1 for first round simple reformer model setting. Light orange: BiDAF without character embedding, dark orange: QANet without character embedding, blue: Reformer without character embedding

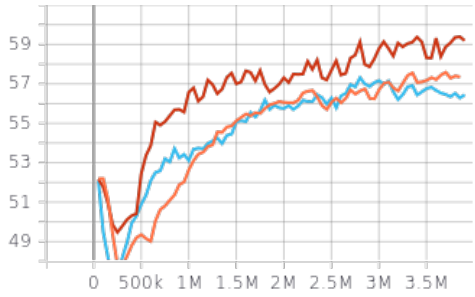

Figure 7: EM for first round simple reformer model setting. Light orange: BiDAF without character embedding, dark orange: QANet without character embedding, blue: Reformer without character embedding

on a single GPU. The model has a single embedding encoder layer with 4 convolutional neural network sublayers; each cnn has a kernel size 7 and filter size same as the hidden state dimension d_{model}; a double layer model encoder block, each has 2 convolutions with kernel size 7 and filter size same as d_{model}. In order to improve the probability of hashing more element to the same bucket, the number of hash rounds is increased to 16, and the bucket size is still 16 as the first reformer; the number of attention heads is 4 for embedding encoder, 8 for model encoder. This time I chose only the original QANet model as baseline, which also contains character embedding layer. I trained it with batch size 32, hidden size 100, a single layer transformer input encoder sublayer with 4 cnn modules, and a 7 layer model encoder transformer sublayers each with 2 cnn modules. The QANet model is trained on with double nVidia GTX 1070TI GPUs for 30 epochs, the training of the complex reformer model takes about 2.5 days for 30 epochs on a single nVidia GTX 1070TI GPU.

3.4 Results and Analysis

The initial results shows that Reformer based QANet can only roughly achieve the baseline model. However it takes about 15 hours to train, compared to only about 4 hours for baseline BiDAF, 9 hours for transformer based QANet. The reduce of speed is because the Memory-efficient attention(computing attention separately for each q_i) in Reformer would reduce computational parallelism but achieve the ability of dealing with 1 million length sequence. For situations such as SQuAD 2.0 where the sequence is limited to be less than 1000, memory efficient attention in Reformer model can be disabled to enable GPU's parallel computation to achieve faster speed theoretically (this experiment is not included due to time limit).

The second attempt of the complex reformer model shows significant improvement on all three performance metrics. As can be seen from table 1, the performance is even slightly better than the second QANet baseline; with F1 score raised from to, and EM score raised from to. The dev curve is still on the rise after 30 epochs which means we can still keep training the complex reformer model to get a better score; whilst the first simple reformer reaches a maximum at about epoch 28(step 3M).

From the comparison of QANet and Reformer based QANet we can draw a conclusion that the LSH based attention mechanism in Reformer model can achieve at least the same performance as the full attention in Transformer model, which means it can be used in calculating attention for arbitrary long sequences. In return for this benefit, the computational cost of a model grows with the number of hashes, so this hyperparameter can be adjusted depending on the available compute budget.

4 Future work

In this project I combined an LSH based attention mechanism with BiDAF structure to experiment the practicality of Reformer model in a real NLP task SQuAD 2.0. The goal of improving the Reformer [3] based QANet model [4] to generate better F1/EM score as the Transformer [7] based QANet has been fulfilled. The result proved to be successful memory wise, however the impact of

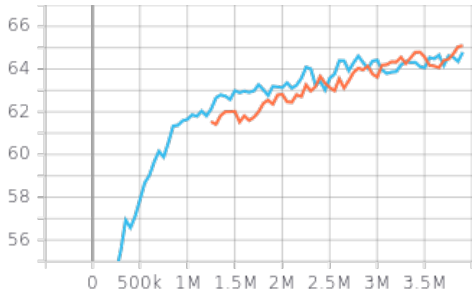

Figure 8: F1 for second round complex reformer model setting. orange: QANet with character embedding, blue: Reformer with character embedding

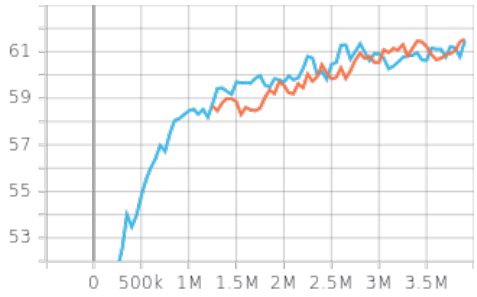

Figure 9: EM for second round complex reformer model setting. orange: QANet with character embedding, blue: Reformer with character embedding

Table 1: Evaluation results for BiDAF, QANet with character embedding and Reformer based QANet with character embedding. From the table it is obvious that the complex reformer based QANet model performs roughly the same as the original QANet(Figure 8 and Figure 9), however the training is drastically slower than QANet.

Experiments	BiDAF	QANet	QANet with Reformer
F1	< 60.5	65.1	64.78
EM	< 57.5	61.54	61.47
AvNA	-	71.89	70.71
Training time	-	<8 hrs	60hrs

computation parallelism brought by reformer needs further study in order to provide improvement to this model. Nonetheless the ability of Reformer to handle arbitrarily long sequences opens the way for the use of the Reformer on many generative tasks. Currently it achieves the better evaluation results as QANet, but it's a lot slower compared to QANet. Future explorations will be to use Reformer in pretrained models such as BERT [9], ALBET [2] or XLNet [1], etc., to enable the training of very large pretrained models on only a single GPU.

5 Acknowledgements

Thanks to Professor Manning and the teaching team for providing a wonderful Natural Language Processing course, I not only learned so much cutting edge technologies in this field, but also get plenty of practical practice in training/debugging deep learning models through the 5 assignment and the course project. Thanks to my project mentor Prerna for providing guidance at every stage of the project, which guarantees the project to be on the right track all the time.

References

[1] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019.

[2] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[3] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[4] 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.

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

[6] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[8] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2016.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[10] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations, 2015.

[11] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[12] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[13] Lukasz Kaiser, Aidan N. Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation, 2017.

[14] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

[15] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance, 2015.