EANet: Enhanced Attention Network for Question Answering System

Insop Song
insop@stanford.edu

Abstract

In this paper, we have implemented Question Answering system using BiDAF (Bi-Directional Attention Flow) [1] based machine comprehension neural network framework. We have replaced RNN (recurrent neural network) based encoders with self-attention based Transformer encoders [2]. We use word and character embedding.

We present two unique approaches: i) an adaptive learning rate scheduler to improve Transformer's notoriously tricky hyperparameter tuning, ii) applying inception layers to the currently proposed Question and Answering system.

1 Introduction

Reading comprehension is one of the important NLP (natural language processing) tasks that can assist human in many applications areas. Question Answering system is a specific NLP task in reading comprehension category. It is defined as follows: a given context and a query, an algorithm predicts start and end word positions, and the span of these two words in the context are the answer sentence to the query. This problem can be defined as follows: a given context $X = \{x_1, ..., x_T | x_i \in X\}$ and a given query $q = \{q_1, ..., q_J | q_i \in Q\}$. A Question Answering system learns to predict start word (s) and end word (e) position, where $s, e \in [0, T)$. Here T denotes the number of words in the context, therefore, a valid answer span should meet this condition, $s < e$.

SQuAD [1] [3] is a reading comprehension data set. Context paragraphs in SQuAD are from Wikipedia, and answers are collected from Amazon Mechanical Turks. We use SQuAD 2.0 test set which include "N/A", answer not available, and around half of the total questions (150k) are in this category.

This paper is organized as follows: Section 2 describes the end-to-end Question Answering system, a model based on BiDAF [1], Transform [2], and QANet [4]. Section 2.6 presents two unique ideas to the project. In section 3, we summarize our experimental results and test configurations. In section 4, we discuss error analysis and experiment results. Finally in section 5, we draw a conclusion and future studies.

2 Approach

We use a commonly used Question Answering framework similar to BiDAF [1]. Figure 1 shows our approach. Encoders in Contextual and modelling layers are consist of self-attention and convolution based instead of RNN based encoder. We replace RNN based encoder, which is difficult to take advantage of GPU's parallel compute power, with self-attention (Transformer encoder [2]). Initially, we use the Transformer encoder as is, later we added

[1] https://rajpurkar.github.io/SQuAD-explorer/

convolutional neural network (CNN) to the input of the Transformer encoder similar to QANet [4].

Here is the summary of each layer of the model 1:

- **Embedding Layer** consists of word and character embedding. Word embedding maps word to a pre-trained word embedding model, Golve [5]. Character embedding maps each word to a character embedding using 1d CNNs that can be concatenated with word embedding.
- **Contextual Layer** finds out the relationship between neighboring words, which is also called an embedding encoder layer. We use self-attention with CNN.
- **Bi-directional Attention Layer** produces a feature vectors that couples context and query, and finds similarity matrix in both directions, context to query and query to context.
- **Modeling Layer** scans the context, and we use stacked Transformer encoders with CNN similar to contextual layer.
- **Output Layer** predicts start and end position of context to form answer.

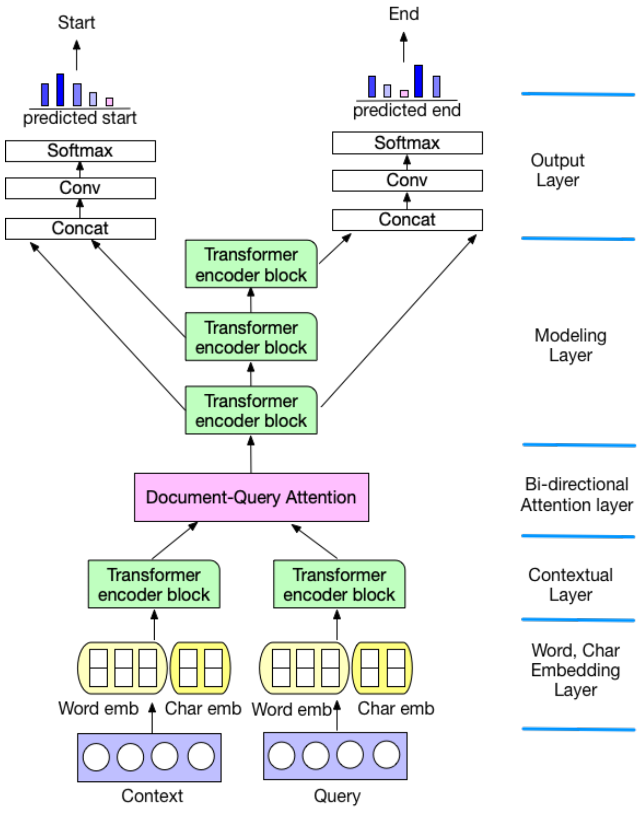
Start
End
predicted start
predicted end
Softmax
Conv
Concat
Softmax
Conv
Concat
Output Layer
Transformer encoder block
Transformer encoder block
Transformer encoder block
Modeling Layer
Document-Query Attention
Bi-directional Attention layer
Transformer encoder block
Transformer encoder block
Contextual Layer
Word emb
Char emb
Word emb
Char emb
Word, Char Embedding Layer
Context
Query

Figure 1: BiDAF based question and answering framework. Replacing RNN encoder with Transformer encoder [2] and QANet type output layer [4]

2.1 Embedding Layer

We use both word and character embedding to represent context and query in high dimensional vector space. We concatenate word and character embedding and use it as a single embedding to the next layer, which is contextual layer.

2.1.1 Character Embedding Layer

In addition to word embedding, we add character embedding to the QA system. Character-level encoding helps to represent **UNK**, out-of-vocabulary words. Unlike pre-trained word embedding, which is freeze-ed during training, a randomly initialized character embedding is used, and it is trained during the forward propagation process.

Each character is represented with an embedding vector with a size of $char\_emb$. As shown in Figure 2, each word with $word\_len$ number of characters with embedding goes through 1D convolutional neural networks (CNN). In this example, "n," "e," "t," "w," "o," "r," "k" are shown as word's characters. 1D CNN kernel ($1 \times char\_emb$) sweeps in $word\_len$ direction, where $char\_emb$ is an input channel size (in_ch). We use $char\_emb$ number of kernel, which is output channel size of (out_ch), to generate output activation. The output dimension of the 1D convolution is $[in\_ch, out\_ch]$. Then max pooling is applied. In this Figure 2, for simplicity, batch size dimension is omitted in the description, which is ($batch\_size \times sequence\_len$) for the character embedding process. This generated character embedding size of $\in \mathbb{R}^c$, which we use character embedding size. This embedding vector is concatenated with word embedding vector to represent each word's vector.

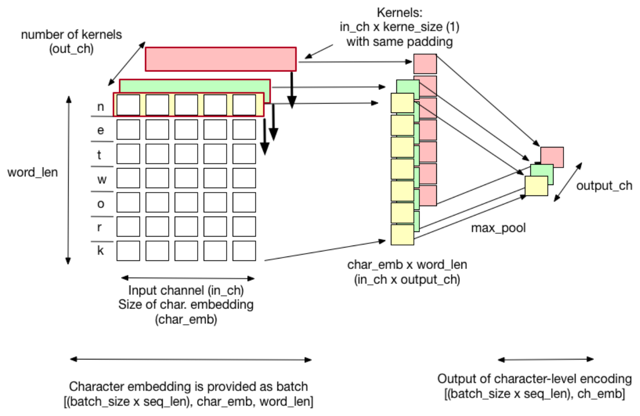

Figure 2: Character-level encoding process. Note that batch dimension is omitted for simplicity.

2.1.2 Word Embedding Layer

We use pre-trained fixed Glove embedding [5], so that each word can be mapped to multi-dimensional vector space.

Let X be context with size of T and q be query with size of J. Let d is the combined dimension of word and character vectors. Output dimension of the embedding layer for context is $\in \mathbb{R}^{d \times T}$ and query is $\in \mathbb{R}^{d \times J}$.

We applied two Highway network, [6], which has a gated shortcut connections, to the embedding outputs.

2.2 Contextual Layer

Contextual layer finds the interaction between words. Instead of using LSTM as original BiDAF paper [1], we use Transformer self-attention encoder [2] initially, and later added depth-wise separable convolution [7] neural network layers with a residual network before applying self-attention of the Transformer.

2.2.1 Transformer Encoder

The Transformer architecture is based on self-attention mechanism without using RNN and utilize multi-head self-attention shown in Figure 3.

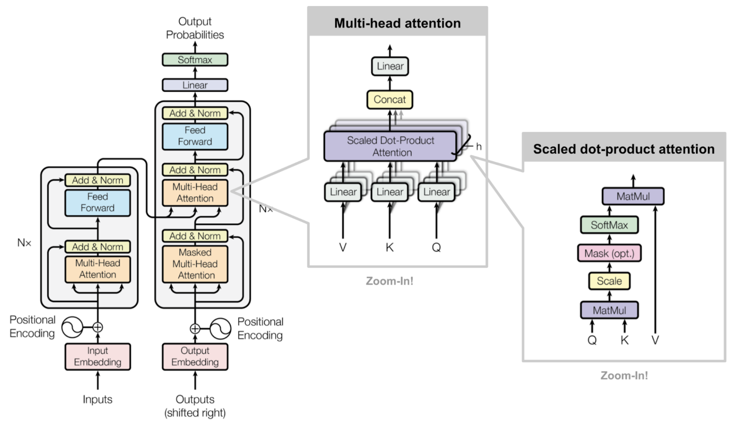

Figure 3: Transformer architecture (Annotated Fig. 17 from `https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html` using the original Image source Fig. 1 and 2 in [2])

RNN is good at learn the sequential relationship in temporal data. However, due to its sequential processing, it is hard to take advantage of GPU's parallel processing, so it is slower than other network types, such as MLP and CNN.

The authors of Transformer [2] introduced an attention module which learns the dependencies between segment of tokens all at once.

Self-attention relates two sequences to compute a representation of the sequence. Transformer is one of the first approaches in neural sequence transduction that uses solely self-attention without using RNN or convolution, which enables the model to be run in parallel.

The authors' of the paper stacked six identical encoder, each encoder is composed of self-attention layer followed by feed forward neural network layer. The decoder has one more layer in addition to self-attention layer and feed forward layer. It has another attention layer to focus which part of the input token.

2.3 Bidirectional Attention (BiDAF) Layer

This layer is for understanding the query and context, and the attention should follow in both directions: from context to query and query to context.

Not all the word token in the contexts are useful for answer to query, i.e. relates to the query. Therefore, in this attention layer, a similarity matrix is calculated between context vectors and query vectors in order to measure their relationship.

BiDAF attention layer calculates attentions in both directions: from context to query and from query to context, which is derived from a shared similarity matrix S, where each element s_{ij} depicts the similarity between the i the context and j the query. We leave the further derivation of BiDAF attention layer to readers, and can be found from [1].

2.4 Modeling Layer

Modeling layer is similar to contextual layer that capture the relationship between concatenated inputs from the previous BiDAF attention layer. Also this layer prepares for the final decoding. Instead of using RNN, which is difficult to parallelize with GPU, we use CNN to capture the long term dependencies between inputs and can summarize information between them. We stacked three Transformer encoders, which internally contains depthwise separable CNNs and self-attentions.

2.5 Output Layer

We have two output layers and each of them predicts start and end word positions for answers respectively. We use a simple softmax layers that takes concatenates two of three modeling layers. Then we calculates the negative sum of the log probabilities against the ground.

2.6 Unique Approach

In this section, we show two unique ideas. First unique approach is an adaptive learning rate scheduling to Transformer network, which successful helped to improve the performance. Second unique approach is to apply Inception layers [8], which is still in testing.

2.6.1 Adaptive Learning Rate Schedule

Transformer hyperpameter tuning is fairly tricky, it was rather difficult for us to tune the right parameters. There is even a report written to provide tips to correctly train Transformer [9]. During the guest lecture [2], Richard Socher mentioned that Transformer very finicky to train.

Transformer paper [2] used Adam optimizer [10] with fixed learning rate schedule. They suggested a linearly increase the learning rate up to *warmup* step, and exponentially decrease from the *warmup* step. We have tested this scheme; however, in many tests, we have little gain performance over many epochs.

We look for alternatives to tune learning rate to improve the training and found this method called "reduce the learning rate on plateau." [3]. In this method, *ReduceLROnPlateau* [4], when there is no performance improvement over configured step, it reduces the learning rate with a constant factor. Then disable for a *cooldown* period, then it repeats the adaptive scheduling.

We found this scheme helped our training, but we observe an instability in early stage of the training, which we suspect that sudden learning rate increase (though it is small) at starting time case this instability. So we updated the original algorithm with by scale the learning rate by factor of $log(steps + 2)/log(warmup\_steps - 1)$, till the *warmup* step then continue a normal adptive behavior. With this change, we see less instability in error plot during the startup time.

We use the learning scheduler code from PyTorch and added our warmup modificaiton [5]. With this change to *ReduceLROnPlateau*, we see less instability in error plot.

Figure 5 shows the adaptive learning rate during the training time.

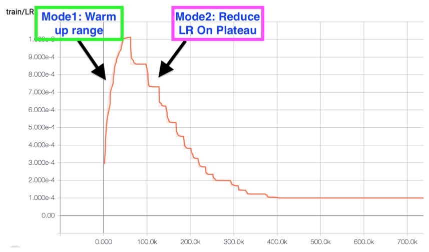

Figure 4: Adaptive learning rate based on validation results.

2.6.2 Applying Inception Layer

We have ran large number of training runs over three different GPU systems, and do not see a significant performance gain by applying Transformer encoder even after re-visiting the implementations and test environment. That's why we conjecture that due to the size of our

[2]`http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture17-multitask.pdf`
[3]`https://bit.ly/2WOV5Am`
[4]`https://pytorch.org/docs/master/optim.html`
[5]`https://pytorch.org/docs/master/optim.html`

network, a gradient update might have trouble flow normally. That's what triggered us to think about a way to help learning process by adding wider network to our current network. Inception network [8] is known for helping deep network learn without suffering gradient flow. We have applied Inception layers before the Transformer encoders. Also we have ran the test with residual connections to ease up the gradient flow in case there is a difficulty. We have not have a conclusion of the effectiveness of this approach yet. With the different batch size due to GPU memory, we see different results from different GPU systems. On a training run with a smaller batch size showed a promising improvement; while, another training run with a larger batch size showed no improvements.

We would continue to investigate further even after the report submission to understand our Transformer network and training behavior.

3 Experiments

3.1 Dataset

The SQuAD [3] dataset consists of three sets: publicly available Train and Dev sets for developers to use, and secretly kept Test set for SQuAD team to evaluate the submitted models. In the default project, 'Kaggle style' is used instead, i.e. test sets are given by the organizer and model generated CSV files are submitted and evaluated.

3.2 Implementation Details

We have developed our network incrementally. First, we have replaced only the LSTM encoder of BiDAF with Transformer encoder. Then, we have replaced modeling layer and output layers with Transformer encoder. For embedding layers, we also started with word embedding and later added character embedding. For the hyper parameter tuning, we started with fixed learning rate, then used the fixed warm-up and exponential decay rate. And finally used the adaptive learning rate reduce on plateau.

We started with the following two code bases and modified significantly to implement the Transformer encoder for Question Answering systems. There are other code bases that are referenced as well, but not significant enough to mentioned in the report, but we have added comments to our code for such a case.

- The project baseline implementation [6]
- Transformer encoder code is based "The Annotated Transformer" [7]

We use 300 dimensions for word embedding and 64 dimensions for character embedding, so total embedding size is 364. In this test, we use three different types of GPUs: 2 x M60 GPU with 8 GB RAM, 2 x V100 GPU with 16 GB RAM, and 1 x RTX GPU with 8 GB RAM. Available memory is the limiting factor to decide the network parameters including batch size, and we have noticed that different batch size influence the performance of the network. It would be worthwhile to re-factor the code to optimize for the memory usage as a future studies to speed up the training, since larger batch size speed up the training.

For adjusting aforementioned adaptive learning rate scheduling. We have adjust initial warm-up step and *patience*[8] steps to determine when there is no validation improvement.

Detail hyper parameters are described in the next subsections.

3.3 Hyper-parameters

We use size of 96 as our hidden layer size for the Transformer encoders, and used size of 384 for feed-forward network size for the Transformer encoder. 12 multi-head is used for the

[6]`https://github.com/chrischute/squad.git`

[7]`http://nlp.seas.harvard.edu/2018/04/03/attention.html`

[8]`https://pytorch.org/docs/master/optim.html`

Network	EM	F1	Batch	epoch
BiDAF (Baseline)	56.2	59.7	64	30
Transformer (Milestone)	54.6	58.1	64	30
Transformer (Final)	56.7	59.4	30	13

Table 1: Test summary for test against dev. test set.

multi-head attention within the Transformer layer. We use 1 encoder layers in the contextual layer, and 4 encoder layers in the modeling layer.

We used 7 depth-wise separable CNN in the Transformer encoder in the contextual layer, and 2 depth-wise separable CNN in the Transformer encoder in the modeling layer.

We use drop probability of 0.1 and max gradient clip to be 5. For the adaptive learning rate scheduling, we use $1e-3$ as initial learning rate, and 0.82 reduction factor when there is no improvement for 10 validation steps (*patience* step). We use *cooldown* steps of 5 till the adaptive reduction starts again.

We use batch size of 30 and run evaluation at every 1000 steps to provide performance metric feedback to the adaptive learning rate scheduler.

3.4 Results

Table 1 shows the evaluation scores from dev. test set. It shows the baseline, milestone, and the current results. Figure 5 shows the learning curve during the current training

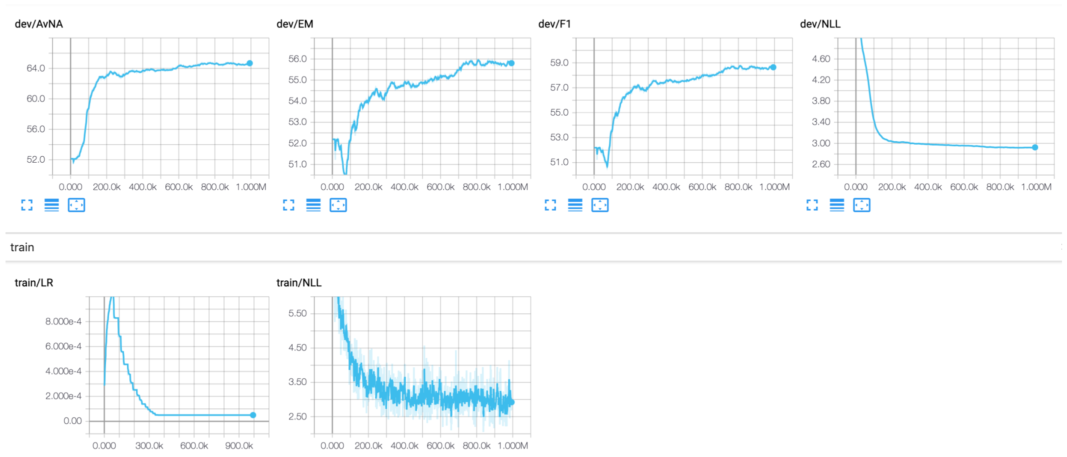

Figure 5: Learning curve from the Final Transformer described in Table Table 1

4 Analysis

We have spend large portion of our time to improve learning curve and it is not yet done. As mentioned in previous section, we have come up with unique ideas to gain more performance.

We have learned that there is a relationship between learning rate and the batch size. Since the larger batch size has less statistical variation and helps the learning and less sensitive to the learning rate. Coincidental, we have come a cross a new ICLR 2018 paper titled *Don't Decay the Learning Rate, Increase the Batch Size* [11].

4.1 Error Analysis

We inspect failed predictions manually, and here are two cases: i) Imprecise word span and ii) ambiguous answer in contextual meaning.

4.1.1 Imprecise word span/boundary

Many of error includes in this type, where an end word span boundary is incorrect, and span is shorter than what it should be. A model with more complexity may fix this since it will understand the context and query relationships.

Question: Which problem consists of both inflationary and deflationary impacts?
Context: The embargo had a negative influence on the US economy by causing immediate demands to address the threats to U.S. energy security. On an international level, the price increases changed competitive positions in many industries, such as automobiles. Macroeconomic problems consisted of both inflationary and deflationary impacts. The embargo left oil companies searching for new ways to increase oil supplies, even in rugged terrain such as the Arctic. Finding oil and developing new fields usually required five to ten years before significant production.

Answer: Macroeconomic problems
Prediction: Macroeconomic

4.1.2 Ambiguous answer in contextual meaning

Some of the answers are ambiguous and contextual meaning could be similar for QA systems; however, it could be clear for human grader to find an exact wording from the question. In the following example, *support* and *encourage* could be in the close proximity in word vector space; hence, it incorrectly predicts answer. This could be fixed by adding more embedding, such as sub-word embedding [12]

Question: What did John Paul II's visits in 1979 and 1983 encourage?
Context: John Paul II's visits to his native country in 1979 and 1983 brought support to the budding solidarity movement and encouraged the growing anti-communist fervor there. In 1979, less than a year after becoming pope, John Paul celebrated Mass in Victory Square in Warsaw and ended his sermon with a call to "renew the face" of Poland: Let Thy Spirit descend! Let Thy Spirit descend and renew the face of the land! This land! These words were very meaningful for the Polish citizens who understood them as the incentive for the democratic changes.

Answer: growing anti-communist fervor

Prediction: budding solidarity movement

5 Conclusion

In this project, we have implemented self-attention based QA systems without using RNN, which allows us to utilize GPU fully in parallel. We have used modified Transformer encoder to QA system based on BiDAF, and successfully perform QA systems. We have two unique approaches to the project, first one is to apply adaptive learning rate schedule that helped our network perform. Second idea is to apply Inception layer together with Transformer encoder. For this idea, we still have not yet conclude its effectiveness, and we plan to continue investigate.

Here are the few things that we have not look in details due to the lack of time. We could apply heuristics and rules to select span of the end words that could potentially improve scores. More embedding, such as sub-word embedding, can be added that can certainly help to boost the performance. Training Transformer network is tricky, so more advance parameter tuning, which we have done with limited scope in this project, would be one of the good topics for the future project. We would like to draw a conclusion of the effectiveness of added Inception layer.

References

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

[2] 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.

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

[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. *CoRR*, abs/1804.09541, 2018.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

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

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015.

[9] Martin Popel and Ondrej Bojar. Training tips for the transformer model. *CoRR*, abs/1804.00247, 2018.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[11] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.