

Transformer Exploration

Stanford CS224N Default Project - Squad 2.0

Aidan Donohue

Department of Computer Science
Stanford University
aidanjd@stanford.edu

Blake Brandon

Department of Computer Science
Stanford University
bbrandon@stanford.edu

Abstract

In this project we build a question answering model that given an input sentence and paragraph outputs either the correct answer or indicates that it can not be found. We attempt to maximize the F1 score of our model on the SQuAD 2.0 dataset. We began by expanding our baseline to include phrase embeddings to match the original BiDAF model ([1]). We then attempt to expand the model by utilizing Transformer blocks in our encoder and modeling layers. Our highest performing model achieved F1 scores of 62.430 on the test set and 63.05 on the dev set.

1 Key Information to include

- Mentor: Mandy Lu
- External Collaborators: N/A
- Sharing project: No

2 Introduction

In the task of reading comprehension or question answering (QA) a model must generate an answer based on an input paragraph and question. Reading comprehension allows us to evaluate both how well a model understands both natural language and its understanding of natural language [2]. The Stanford Question Answering Dataset (SQuAD) is a large dataset of question answer-pairs was created in order to advance natural language understanding. Performance on the SQuAD dataset is evaluated using two metrics. One metric the Exact Match (EM) score is a binary metric that checks if the prediction matches the correct answer exactly. The F1 score, on the other hand, is a more robust metric that compares the precision in recall of a model's output.

Currently, modern approaches to the QA task are likely to use pre-trained contextual embeddings (PCE). PCE is a common tool in many different NLP tasks that allows the model to predict relationships between sentences. These embeddings are applied to downstream task either using a feature-based approach, such as in ELMo ([3]), or via fine-tuning as done in BERT ([4]). These models have proven to be highly effective and have rendered their non-PCE counterparts almost entirely obsolete in the field. Due to non-PCE models no longer being state-of-the-art there has been much less interest in expanding and improving non-PCE approaches. Prior to the popularization of PCE models, non-PCE Transformer models were making great strides in completing QA tasks on the SQuAD dataset. One such model, QANet [5], achieved state of the art F1 scores in 2018 by incorporating convolution into a Transformer model. In this work we hope to build upon where they left off and improve the Transformer architecture without the use of pre-trained contextual embeddings.

3 Related Work

Seo et al. (2017) introduced the Bi-Direction Attention Flow (BiDAF) model for Machine Comprehension [1]. The BiDAF like many other models consisted of an RNN encoder and decoder layers as well as an attention layer. The attention layer is used to tell us what part of the context paragraph is most useful for answering the question. The BiDAF model used a unique attention layer that unlike its predecessors was Bi-Directional, did not use a fix sized vector attention vector, and the attention layer at each timestep did not depend on previous timesteps. At the time, RNN neural networks were the common framework used for the QA task and the BiDAF model was the best of those models. The BiDAF model achieved a state-of-the-art F1 score of 81.1%.

With RNN as the most common approach to QA tasks, the BiDAF model pushed the RNN architecture one step further. However, one issue of recurrent networks that the BiDAF model failed to solve was the training time of such models. RNN models require the hidden state of the previous timestep to calculate the current hidden state and therefore prevents parallelization. As a result, RNN models take a long time to train which limits rapid iteration and the amount of data they can process. In response to this a new architecture, the Transformer, was introduced that was more parallelizable and trained faster than the BiDAF model.

Vaswani et al. (2017) proposed a new recurrence-free model based entirely on attention, called the Transformer [6]. Typically, RNNs use an attention mechanism to work in conjunction with the model. The transformer model removes the recurrence entirely and as a result allows for more parallelization and therefore increased training speeds. The transformer model utilizes self-attention specifically due to the lower computation complexity, the amount of computation that can be parallelized, and the ability to learn long range dependencies. Each layer in the encoder and decoder layer use multi-head scaled dot product self attention in order to attend to positions in other layers. Each encoder layer consists an attention sub-layer and fully connected feed-forward network sub-layer. The decoder layer contains both sub-layers (with some masking applied to the attention layer) as well as an additional sub-layer that attends to the output of the encoder layer. The resulting model is highly parallelizable model that generalizes well to many tasks.

A year after the original Transformer model was produced the QANet model achieved an 84.6% state-of-the-art F1 score [5]. The QANet model adds an additional convolution layer sub-layer that works in conjunction with the self-attention and feed-forward sub-layers when encoding embeddings. As a result saw an increase of 2.7 to the F1 score when compared to the original transformer model. The model utilized it's speed to train on more data and used data augmentation process to train on paraphrased sentences as well as original sentences.

4 Approach

We begun with a baseline BiDAF model with a common architecture for QA tasks. The model consisted of an embedding layer, and an encoder layer, an attention layer, a modeling layer, and an output layer. We began by extending the baseline model to include character embeddings to match the BiDAF model described in Seo et al 2016 [1]. Next, we replaced the LSTM encoder and modeling layers with Transformer encoder blocks. The resulting architecture is as follows:

4.1 Embedding Layer

The first extension to the baseline BiDAF model was to implement a feature omitted from the distributed handout code, character embeddings. In the original BiDAF paper [1], character embeddings are concatenated with pre-trained word embeddings and encoded in a recurrent layer. The original paper describes these as "phrase embeddings." Our implementation of the BiDAF model with character embeddings, differs slightly from original. One difference is the use of LSTMs as opposed to GRUs for the various recurrent architectures. We experimented with GRUs in order to see if we could achieve a faster training. However, our experiments concluded that the speed increase as well as the performance decrease of GRUs over LSTMs were negligible. We referenced Seo et al. (2017) to accomplish this.

In order to use the Transformer encoder, which operates on sets and thus doesn't maintain sequence order, we must manually inject position information into our model. We add sinusoidal positional

encodings with our phrase embeddings to add relative position. Key step in adding the positional encodings is to scale up the magnitudes of the phrase embeddings so that the positional encodings do not obscure the encoded properties of the phrase embeddings. We followed the positional encodings used in Vaswani et al. (2017) [6], who found that sinusoidal positional encodings performed just as well as learned positional embeddings.

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i + 1) = \cos(pos/10000^{2i/d_{model}})$$

4.2 Encoder Layer

Given the baseline model with an LSTM encoder we experimented with utilizing both a GRU and and Transformer encoder. Ultimately, we choose the Transformer encoder as it is recurrence free and therefore more parallelizable and faster to train. In all models our encoder is applied to both the context and query embeddings separately.

Our encoder layer consists of a single Transformer block. Each Transformer block consists of a convolution layer, a self-attention layer, and feed forward layer. The convolution layer in the encoder contains a single convolution block that performs depthwise seperable convolutions. As noted in Yu et al. (2018) this form of convolution generalizes well and is memory efficient [5]. The next layer performs multi-headed scaled dot product self-attention on the embeddings. Finally, we apply a feed forward network to the self attention result. We apply layer normalization to each convolution block as well as to the self attention and feed forward layer outputs.

$$SepConv(W_p, W_d, y)_{(i,j)} = PointwiseConv_{(i,j)}(W_p, DepthwiseConv_{(i,j)}(W_d, y))$$

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

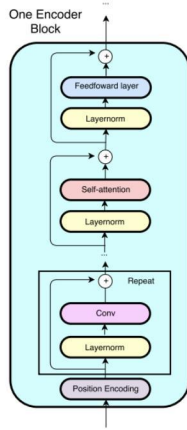


Figure 1: Encoder block [5]

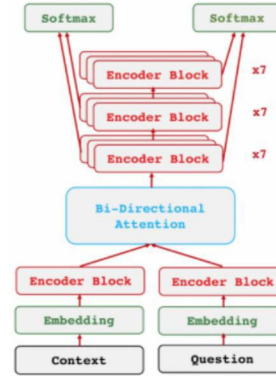


Figure 2: Model Architecture [7]

4.3 Attention Layer

For the attention layer we utilize the Bidirectional Attention Flow mechanism from the baseline model. Bi-Directional attention flow is proven to work in QA tasks because it allows us to encode the context and query separately and then attend to their outputs. As described in Seo et al. (2017) we utilize a memory-less bi-directional attention where representations are allowed to flow between layers.

4.4 Modeling Layer

Unlike our encoder layer, the modeling layer only needs to be applied once. Our transformer modeling layer utilizes three identical Transformer blocks that are applied to the attention output one after another. The blocks used in the modeling layer are the same as the encoder blocks with the exception that we utilize 7 convolution blocks rather than just one.

4.5 Output Layer

The transformer output layer uses the outputs of the 3 transformer modeling layers to predict a start and stop index. The outputs of each modeling layer are projected from two dimension (not including the batch dimension) down to a single dimension corresponding to the sequence length. The projections from modeling layers 1 and 2 are added and passed into a masked softmax function to get the start position probabilities over the input sequence length. The outputs from modeling layers 1 and 3 are added and passed into a masked softmax function to get the end position probabilities over the input sequence length.

5 Experiments

5.1 Data

We utilize a modified version of the SQuAD 2.0 dataset. Since the true SQuAD 2.0 test set is hidden, we create our custom dev and test sets that are created by splitting the official dev set in half. Our model is trained and tuned entirely based on the custom dev split and the official squad train set. Final evaluation is done using the modified test set.

5.2 Evaluation method

We evaluate based on binary Exact match and F1 harmonic mean of precision and recall. We primarily hoped to maximize F1 score because it is a more robust way to evaluate QA models.

5.3 Experimental details

Both the baseline model and the extended char embeddings model were tested using three different model configurations:

name	hidden size	encoding layer	modeling layer	drop probability
default	100	lstm (1 layer)	lstm (2 layer)	0.2
big lstm	100	lstm (2 layer)	lstm (3 layer)	0.2
gru	100	gru (1 layer)	gru (2 layer)	0.2

Our Transformer model had the following configuration:

name	Transformer
embedding layer	embedding size 100, projection size 128
encoding layer	Transformer (1 block, 1 convolution block, 8 attention heads)
modeling layer	Transformer (3 blocks, 7 convolution block, 8 attention heads)
drop probability	0.2

5.4 Results

The results are shown in figures 2, 3, and four below. The dark blue color represents our Transformer model which failed to learn. The light blue represents our baseline model, and the two green models are the character embeddings extended LSTM with different parameters. Our big LSTM configuration, with more LSTM layers in the encoding and modeling layers, learned faster than our default LSTM. In the first 2.5 million iterations the big LSTM outperforms the default parameters, but ultimately they converge at only slightly different scores.

The big LSTM, our highest performing model, achieved an F1 score of 63.05% and an EM score of 59.47% on the dev dataset. On the test dataset our model scored an F1 and EM scores of 62.43% and 58.81% respectively.

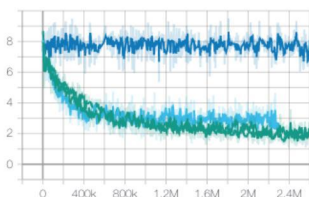


Figure 3: Train NLL

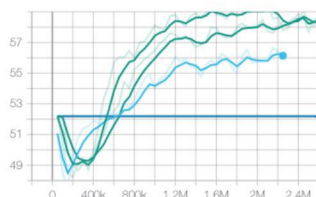


Figure 4: Dev EM

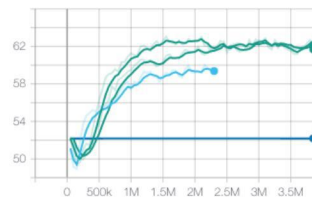


Figure 5: Dev F1

6 Analysis

In our attempts to extend the baseline model we produced two separate models. One model utilized character embeddings to successfully extend the baseline, while the other utilized transformer blocks and failed to reach the baseline standard. We will first discuss what went wrong with our first extension, before exploring why our second extension failed.

Since languages do not have fixed vocabularies there is always the possibility that at test time we will see novel words that didn't appear during training. Subword models such as character embeddings help account for novel words in our test sets. Character embeddings allow us to add information into our word vectors so that even out-of-vocabulary words contain some information about the word's characters. These models were able to train more efficiently than our baseline model and ultimately scored better when tested. This is captured by the loss function in figure 3. By retaining information about subwords we reduced the amount of cross entropy loss during training.

Besides showing that our BiDAF model with character embeddings both outperformed the baselines, the graph also shows an interesting relationship between the two different parameter sets. By comparing the two different parameter configurations (default/big LSTM) it became clear that it was time for another architectural change to the model. Although the big LSTM outperformed the default configuration the impact was only marginal. Figures 4 and 5 show the dev F1 and EM scores for the two models overtime. The big LSTM starts off performing better, but eventually both models converge to a similar score. This suggests that we were unlikely to improve our model very much by adding more layers.

With our character embedding working properly, we attempted to make another extension to the resulting model architecture. We hoped to replace the LSTM encoder and modeler with a Transformer in order to increase training efficiency. By increasing training speed we could have more time to iterate and make additional improvements to our model. However, our Transformer model failed to learn and thus we weren't able to move past our first iteration. Our model still failed to learn, however, and as shown in figure 4 and 5 failed to improve EM or F1 scores overtime.

It's unclear exactly what prevented our Transformer model from successfully training on the QA task. However, there are a few issues with our approach that prevented us from taking full advantage of the Transformer architecture. The first issue is that our model was simply too big. Since Transformer models train faster they can take advantage of having more layers. However, our original implementation was too big to run on our virtual machine. Unfortunately, attempting to navigate this ultimately insurmountable issue ate up a large portion of the time we spent on the transformer model. As a result we used linear projections to project down the output of our embedding and attention layer. However, it seems most likely that the issue lies in the convolutional layer of the transformer encoders. It's possible that in applying convolutions layers and then reshaping them that we lost information necessary for learning.

7 Conclusion

We successfully extended the baseline model with character embeddings, but failed to properly integrate Transformer blocks into the model. Due to this we were unable to iterate and make

improvements to the Transformer architecture as we had hoped. This, however, should not discourage future attempts to build and extend non-PCE Transformer models. In fact, due to the current popularity of PCE models, we believe its increasingly important that non-PCE models aren't forgotten. Currently PCE models are more likely to yield state-of-the-art results, but that doesn't mean non-PCE models aren't incredibly powerful. QANet performed just slightly under the performance of a human and we believe the architecture could be improved upon to equal or even surpass human F1 scores. With less people focusing on non-PCE it becomes easier for any single researcher to make a novel advancement in the field. These advancements are important because they might apply to both PCE and non-PCE models and because there's no certainty that non-PCE models can't become state-of-the-art again in the future.

References

- [1] Minjoob Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *Association for Computational Linguistics (ACL)*, 2017.
- [2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Richard Socher. Squad: 100,000+ questions for machine comprehension of text. In *Association for Computational Linguistics (ACL)*, 2016.
- [3] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Association for Computational Linguistics (ACL)*, 2018.
- [4] Jacob Devlin, Ming-Wang Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Association for Computational Linguistics (ACL)*, 2019.
- [5] 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. In *Association for Computational Linguistics (ACL)*, 2018.
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomen, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Association for Computational Linguistics (ACL)*, 2017.
- [7] <https://github.com/chrischute/squad-transformer>.