

# Question Answering on SQuAD 2.0 using QANet with Performer FastAttention

Stanford CS224N Default Project

**Árpád József Tamási**

Department of Computer Science  
Stanford University  
arpad@stanford.edu

## Abstract

This project reproduced the QANet model using Performer Fast Attention for the SQuAD 2.0 reading comprehension task. Fast Attention approximates self-attention in linear complexity, provable and tuneable accuracy, making Transformers suitable for large-scale tasks. We substituted self-attention layers with Fast Attention in QANet encoder blocks and evaluated the model with different feature counts, gaining 18% faster training and keeping accuracy.

## 1 Introduction and related work

Reading comprehension and question answering is a fundamental problem for NLP model benchmarking. Answering questions from a text requires the reader to understand both the text and question and the relationship between them, which is a crucial skill of humans. The SQuAD challenge [1] aims to evaluate this by providing context passages, questions about them, and answers selected by humans. The SQuAD 2.0 dataset [2] raises the bar by adding unanswerable questions.

Early deep neural NLP models used recurrent neural networks (RNNs) to encode input sequences. An attention mechanism improved their performance by identifying complex dependencies between tokens in the sequences. Transformers [3] introduced positional encoding to enable parallel processing and solve long-range dependency issues and self-attention to compute similarity scores. While Transformers become state-of-the-art in machine learning, their attention mechanism scales quadratically with the input sequence's length, precluding their usage for large-scale problems.

Performers [4] introduce Fast Attention Via positive Orthogonal Random features (FAVOR+), a provably accurate and practical approximation of regular attention, with linear space and time complexity. They are the first linear architectures fully compatible with regular Transformers, providing theoretical guarantees.

Models that became successful on the SQuAD challenge employed two key ingredients: a recurrent model to process sequential inputs and an attention component to cope with long-term interactions. The Bidirectional Attention Flow [5] (BiDAF) achieved strong results, but it is slow due to its recurrent nature, especially for long texts.

QANet [6] proposed convolutions and self-attentions without recurrence as the building blocks of encoders that separately encode the query, the context, and interactions between them learned by standard attentions.

In this project, we implement the QANet model substituting its self-attention blocks with Performer Fast Attention and evaluating its speed and accuracy. Recent models achieve higher scores than QANet by using pre-trained contextual embeddings. Instead of competing with them, we focus on QANet training performance with Performer Fast Attentions.

## 2 Approach

We implement the QANet model, based on the original paper but learning a lot from heliumsea/QANet-pytorch<sup>1</sup> and naver/claf<sup>2</sup>, both designed for SQuAD 1.1 [1] implementations. To ensure that our implementation is able to learn actually, we extract a few subsets of the SQuAD training and dev set, containing 950-1300 samples..

- small: contains 3-4 topics from the training and dev set.
- small-overfit: the same set, using the training set for validation
- small-answered, small-answered-overfit: the same sets, answerable questions only

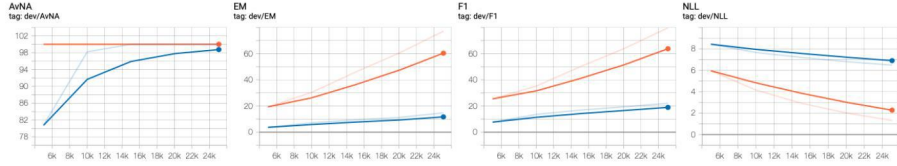


Figure 1: Dev metrics on the small-overfitting dataset - QANet (orange) vs Bidaf (blue)

### 2.1 Problem

The reading comprehension task considered in this paper, is defined as follows. Given a context paragraph with  $n$  words  $C = c_1, c_2, \dots, c_n$  and the query sentence with  $m$  words  $Q = q_1, q_2, \dots, q_m$ , output a span  $S = c_i, c_{i+1}, \dots, c_{i+j}$  from the original paragraph  $C$ . In the following, we will use  $x$  to denote both the original word and its embedded vector, for any  $x \in C, Q$ .

### 2.2 QANet Model

#### 2.2.1 Input embedding

We reuse the provided BiDAF embedding based on  $p_1 = 300$  dimensional pre-trained GloVe [7] vectors and extend it with  $p_2 = 200$  dimensional character embedding based on the QANet paper. We take the provided characted embedding vector sequences, apply a 2D convolution over them then take maximum value of each row of this matrix to get a fixed-size vector representation of each word. Finally, we concatenate this vector to the token's word embedding vector to get  $[x_w; x_c] \in \mathbb{R}^{p_1+p_2}$ , where  $x_w$  and  $x_c$  are the word embedding and the convolution output of character embedding of  $x$  respectively. We apply a convolution to resize this representation for the  $d = 128$  hidden size of encoder layers (mentioned in the embedding encoder layer in the paper). We also adopt a two-layer highway network on top of this representation. For simplicity, we also use  $x \in \mathbb{R}^d$  to denote the output of this layer. Instead of using -OOV- to represent "no answer" prediction, we added a -NOANSWER-token, initialized randomly in the embedding vectors.

#### 2.2.2 Embedding encoder layers

The encoder layer is a stack of depthwise separable convolutions, followed by a self-attention (replaced later with Fast Attention) and a feed-forward-layer. Each of these operations are placed inside a residual block, including a normalization layer. We also adopt a stochastic layer dropout [8] with  $p_l = 1 - \frac{l}{L} (1 - p_L)$  where  $l$  is the layer index,  $L$  is the last layer and  $p_L$ . We use one input encoder layer with 4 convolutions, kernel size 7, 8 heads in self-attention and  $p_L = 0.9$ .

<sup>1</sup>codebase: <https://github.com/heliumsea/QANet-pytorch>

<sup>2</sup>codebase: <https://github.com/naver/claf>

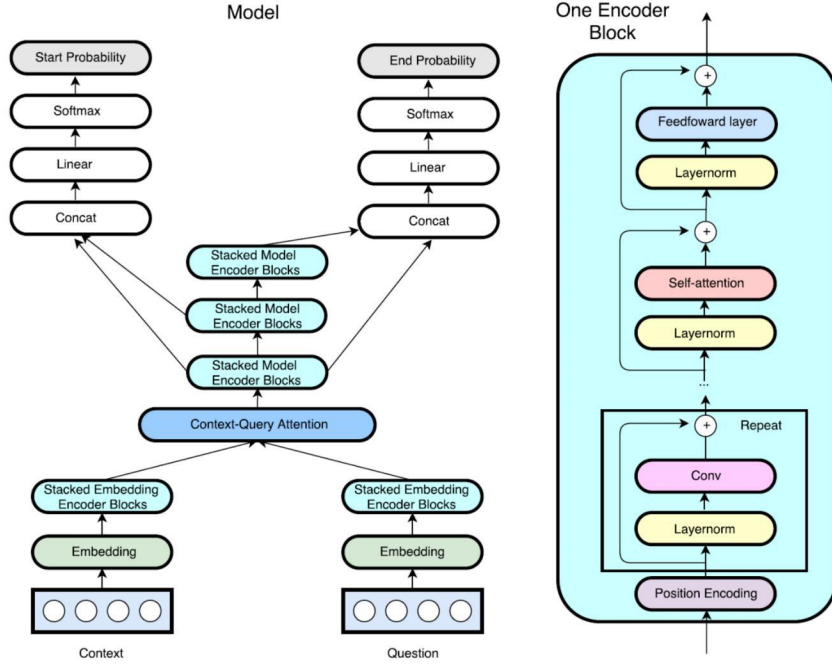


Figure 2: QANet architecture

### 2.2.3 Context-query attention

Since the attention implemented in the provided BiDAF model is an improved version of the one defined in the QANet paper, we use it unchanged in the model. We use a convolutional layer to resize the model to  $d = 128$

### 2.2.4 Model encoder layers

Model encoder layers follow are identical to embedding encoder layers. We apply 7 model encoder layers with 2 convolutions, kernel size 7, 8 heads in self-attention and  $p_L = 0.9$ . We repeat this layer 3 times with shared weights, resulting  $M_0$ ,  $M_1$  and  $M_2$ .

### 2.2.5 Output layer

For each token we compute the probability  $p^1$  and  $p^2$  to be the first and last token of the span, using  $W_1$  and  $W_2$  trainable weights.

$$p^1 = \text{softmax}(W_1[M_0, M_1]) \quad (1)$$

$$p^2 = \text{softmax}(W_2[M_0, M_2]) \quad (2)$$

The score of a span is the product of its start and end position probabilities. We use the provided objective function since it is identical to the one used in QANet.

$$L(\theta) = -\frac{1}{N} \sum_i^N \left[ \log(p_{y_i}^1) + \log(p_{y_i}^2) \right] \quad (3)$$

## 2.3 Data augmentation

We decided not to implement data augmentation due to time limitations.

## 2.4 Performer Fast Attentions

### 2.4.1 Bidirectional attention mechanism

Bidirectional dot-product attention used in self-attention mechanisms has the following form:

$$Att(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{D}^{-1} \mathbf{A} \mathbf{V}, \text{ where} \quad (4)$$

$$\mathbf{A} = \exp(\mathbf{Q} \mathbf{K}^\top / \sqrt{d}) \in \mathbb{R}^{L \times L} \text{ is the attention matrix} \quad (5)$$

$$\mathbf{D} = \text{diag}(\mathbf{A} \mathbf{1}_L), \text{ where } \mathbf{1}_L \text{ is the all-ones vector of length } L \quad (6)$$

Time and space complexity of computing 5 are  $O(L^2 d)$  and  $O(L^2 + Ld)$ , since  $\mathbf{Q} \mathbf{K}^\top$  has to be computed before applying  $\exp$ . FAVOR+ method, introduced by Performers, approximates  $\mathbf{A}$  in  $O(Ld^2 \log(d))$ .

FAVOR+ works for attention blocks using matrices  $\mathbf{A} \in \mathbb{R}^{L \times L}$  of the form  $\mathbf{A}(i, j) = K(\mathbf{q}_i^\top, \mathbf{k}_j^\top)$  with  $\mathbf{q}_i^\top, \mathbf{k}_j^\top$  standing for the  $i^{th}, j^{th}$  key vector of  $\mathbf{Q}$  and  $\mathbf{K}$  and kernel  $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$ .

It introduces a randomized feature mapping  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}_+^r (r > 0)$ , as

$$K(\mathbf{x}, \mathbf{y}) = \mathbb{E} [\phi(\mathbf{x})^{top} \phi(\mathbf{y})] \quad (7)$$

The attention approximation is

$$\hat{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \hat{\mathbf{D}}^{-1} (\mathbf{Q}' ((\mathbf{K}')^\top \mathbf{V})) \quad (8)$$

$$\hat{\mathbf{D}}' = \text{diag}(\mathbf{Q}' ((\mathbf{K}')^\top \mathbf{1}_L)) \quad (9)$$

Figure 3 illustrates the complexity of computing  $Att$  and  $\hat{Att}$ .

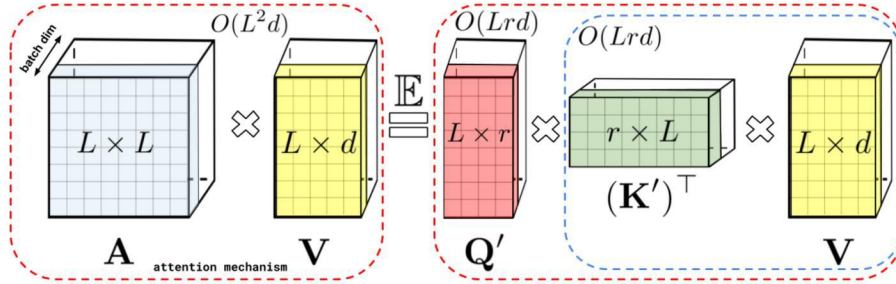


Figure 3: Approximation of the regular attention mechanism via random feature maps.

We use lucidrains/performer-pytorch<sup>3</sup>. It is designed to minimize model surgery, so we simply call its FastAttention instead of our dot product attention. The approximation accuracy is tuneable by the `nb_features` argument which sets  $r$  in randomized feature mapping. Its default value is 256, we test different values during the evaluation.

## 2.5 Hyperparameter search

To simplify hyperparameter search, we implement typed configurations in python data classes, a dataclass loader that reads configurations from json files and a simple GridSearch class that reads the same files but processes hyperparameter search attributes.

```
// ...
"performer": {
  "hyperparam-search": [
    {"nb_features": 1},
```

<sup>3</sup>codebase: <https://github.com/lucidrains/performer-pytorch>

```

        {"nb_features": 16},
        {"nb_features": 128},
        {"nb_features": 256}
    ]
},
// ...

```

### 3 Experiments

#### 3.1 QANet baseline

We use the provided dataset and the initial QANet implementation for evaluation, using  $p_2 = 200$  dimension char embeddings and the hyperparameters defined by the QANet paper.

#### 3.2 Performer baseline

We substitute the encoder self-attentions with Performer fast attentions, using  $r = 2$  in randomized feature mapping.

#### 3.3 Performer no-answer

We encode no answer with a randomly initialized -NOANSWER- token.

#### 3.4 Performance tests

We test performer training performance and accuracy with different  $r$  settings (1, 128, 256) on a small dataset and train for in 30 epochs to measure its impact on speed and the loss.

#### 3.5 SQuAD challenge test results (IID SQuAD track)

- EM: 62.164
- F1: 65.761

#### 3.6 SQuAD challenge dev results

|                           | Train NLL    | Dev NLL      | Dev AvNA     | Dev F1       | Dev EM       | Training time |
|---------------------------|--------------|--------------|--------------|--------------|--------------|---------------|
| <b>BiDAF baseline</b>     | 2.839        | 3.268        | 66.49        | 59.76        | 56.2         | <b>04:09</b>  |
| <b>QANet baseline</b>     | <u>1.687</u> | 3.118        | 73.59        | 66.79        | 63.19        | 06:30         |
| <b>Performer baseline</b> | <u>1.793</u> | 2.992        | 74.04        | 67.84        | 64.05        | 05:42         |
| <b>Performer noanswer</b> | <b>1.667</b> | <b>2.674</b> | <b>76.13</b> | <b>69.93</b> | <b>66.46</b> | <u>05:31</u>  |

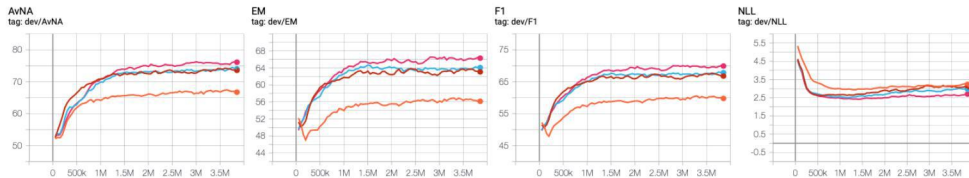


Figure 4: Experiment training history. Orange: BiDAF, Burgundy: QANet baseline, Blue: Performer baseline, Pink: Performer no-answer

### 3.7 Performance results

|                          | Train NLL | Training time in m ss |
|--------------------------|-----------|-----------------------|
| <b>QANet</b>             | 2.222     | 5:30                  |
| <b>Performer (r=1)</b>   | 2.486     | <b>4:40</b>           |
| <b>Performer (r=128)</b> | 2.574     | 6:06                  |
| <b>Performer (r=256)</b> | 2.844     | 8:04                  |

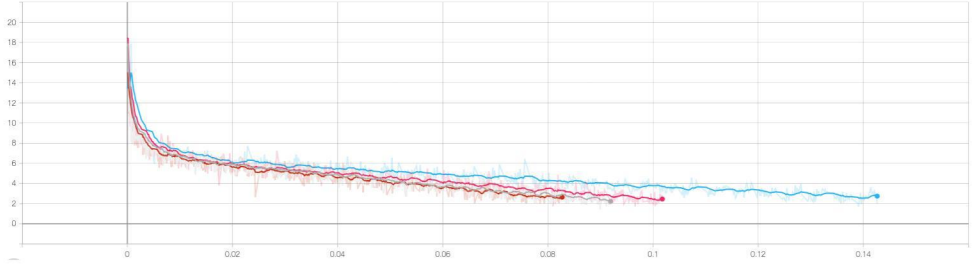


Figure 5: Experiment training history. Gray: QANet, Burgundy:  $r = 1$ , Pink:  $r = 128$ , Blue:  $r = 256$

## 4 Analysis and conclusion

The initial QANet model performs significantly better on the challenge but its training takes 30% more time. While SQuAD is not a large-scale problem, since its contexts and questions are not long, it is exciting to see that we reached similar scores with Performer Fast attentions, with 18% faster training.

We see that  $r$  has a high impact on training speed, with high values it perform slower than the original self attention at the given sequence length. Fortunately, even with a low  $r = 2$  value it reached the same score.

## References

- [1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.
- [2] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.
- [3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.
- [4] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.
- [5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.
- [6] 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.
- [7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [8] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016.