QANet for SQuAD 2.0

Stanford CS224N Default Project - IID SQuAD Track

Venkat Yerramsetti
Department of Computer Science
Stanford University
venkaty@stanford.edu
Mentor: Elissa Li

Abstract

Question Answering task is a very useful NLP task for evaluating an NLP system's language understanding capability. SQuAD is one of the benchmark datasets for the Question Answering task, and QANet model produced state of the art results on the SQuAD 1.1 dataset. In this project, I re-implemented QANet to evaluate it's performance on the SQuAD 2.0 dataset which is a more challenging dataset due to the presence of unanswerable questions. I also showed that reducing QANet's size does not negatively impact performance.

1 Introduction

Question Answering is an actively researched topic in NLP both as a standalone task and as an evaluation method for other NLP tasks. Earlier successful models for this task relied on recurrent networks (RNNs/LSTMs) and attention to handle sequential input and to encode long term interactions respectively. However, the inherent sequential nature of such models slows down training and inference. Subsequently, transformer based models were introduced to address this concern. QANet [1] model is one such model that produced state of the art results on the SQuAD 1.1 dataset [2]. The SQuAD dataset has since been updated to SQuAD 2.0 [3] by adding unanswerable questions, which makes the task more challenging. In this project, I re-implemented the QANet model to evaluate its performance on the SQuAD 2.0 dataset. Compared to the base BiDAF [4] model and its variants as described in this paper, QANet was able to achieve a high F1 score (67.54) and EM score (63.99) on the dev set, and did so in only about one third of the training steps taken by the baselines. On the test set, the model acheived an F1 score of 65.40 and EM score of 61.98. I also empirically showed that a smaller-sized QANet could still achieve high scores. Lastly, using QANet's Encoder block, I introduced a new model that can emulate bi-directional attention using Self-Attention.

2 Related Work

SQuAD 1.1 [2] was one of the large scale benchmark datasets for Question Answering models. However, a limitation of this dataset is that each question is guaranteed to have an answer within the context paragraph, so a model simply needs to learn to predict one out many posssible answer spans. In order to address this limitation, the dataset was updated to SQuAD 2.0 [3] with additional unanswerable questions.

Several RNN-based models were developed to tackle the SQuAD 1.1 and 2.0 challenges. One of the earlier models to perform well on these challenges, BiDAF [4] introduced a context-to-query and query-to-context attention mechanism . Match-LSTM [5] used Pointer Networks [6] to condition end token prediction on start token. However, these RNN-based models are very slow during training and inference due to the inherent recurrent nature of the RNNs. QANet [1] was a transformer based model that aimed to address this limitation, but it was only evaluated on the SQuAD 1.1 dataset. Later models proposed various mechanisms to determine whether a question is answerable or not. For instance, the U-Net model [7] used to multi-task learning setup to address the challenging

unanswerable questions in the SQuAD 2.0 dataset. The current state-of-the-art results on SQuAD 2.0 dataset are achieved by large pretrained models that are finetuned on the SQuAD task.

3 Models

In this section, I will describe all of the models I implemented in this project. I will first reiterate, from the project handout, the details of the provided base **Bid**irectional **A**ttention **F**low (BiDAF) model since it forms the basis for all of the baselines and parts of QANet. In the following sections, let N be the length of the context passage, M be the length of the question, H be the hidden size, d_w be the size of word-level embeddings, and d_c be the the size of character-level embeddings.

3.1 Base-BiDAF Model (Baseline 1)

This is the model provided in the Default project's (SQuAD IID track) starter code. It is equivalent to the original BiDAF model [4] without character-level embeddings. The model is made up of the following five layers.

Embedding Layer

This layer converts a set of word indices, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k \in \mathbb{N}$ to embedding vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^{d_w}$ through a lookup. Then, the embeddings are linearly projected to dimensionality H.

$$\boldsymbol{h}_i = \boldsymbol{W}_{proj}\boldsymbol{v}_i \in \mathbb{R}^H \qquad \text{where } \boldsymbol{W}_{proj} \in \mathbb{R}^{H \times d_w}$$

Finally, a two-layer Highway Network [8] is applied to the projected embeddings $\boldsymbol{h}_i$, where each Highway Network layer performs the following computation.

$$\begin{aligned}
\boldsymbol{g} &= \sigma(\boldsymbol{W}_g\boldsymbol{h}_i + \boldsymbol{b}_g) \in \mathbb{R}^H \\
\boldsymbol{t} &= \text{ReLU}(\boldsymbol{W}_t\boldsymbol{h}_i + \boldsymbol{b}_t) \in \mathbb{R}^H \\
\boldsymbol{h}'_i &= \boldsymbol{g} \odot \boldsymbol{t} + (1 - \boldsymbol{g}) \odot \boldsymbol{h}_i \in \mathbb{R}^H
\end{aligned}$$

Here, $\boldsymbol{W}_g, \boldsymbol{W}_t \in \mathbb{R}^{H \times H}$ and $\boldsymbol{b}_g, \boldsymbol{b}_t \in \mathbb{R}^H$ are learnable parameters. The same layer (thus same parameters) is used to generate both context embeddings $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N \in \mathbb{R}^{d_w}$ and question embeddings $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_M \in \mathbb{R}^{d_w}$.

Encoder Layer

This layer takes the outputs from the Embedding layer and encodes them with temporal dependency information using a bidirectional LSTM. The output from this layer is the concatenation of the forward and backward hidden states of the LSTM.

$$\begin{aligned}
\boldsymbol{h}'_{i,fwd} &= \overrightarrow{\text{LSTM}}(\boldsymbol{h}'_{i-1}, \boldsymbol{h}_i) \in \mathbb{R}^H \\
\boldsymbol{h}'_{i,rev} &= \overleftarrow{\text{LSTM}}(\boldsymbol{h}'_{i+1}, \boldsymbol{h}_i) \in \mathbb{R}^H \\
\boldsymbol{h}'_i &= [\boldsymbol{h}'_{i,fwd}; \boldsymbol{h}'_{i,rev}] \in \mathbb{R}^{2H}
\end{aligned}$$

Bidirectional Attention Layer

This is the core layer in the model and is the reason the model is called Bidirectional Attention Flow model. This layer computes context-to-query and query-to-context attentions. First, using the context encodings $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N \in \mathbb{R}^{2H}$ and question encodings $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_M \in \mathbb{R}^{2H}$ from the Encoder Layer, we compute a Similarity Matrix $\boldsymbol{S} \in \mathbb{R}^{N \times M}$ such that

$$\boldsymbol{S}_{ij} = \boldsymbol{w}_{sim}^\top[\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \odot \boldsymbol{q}_j] \in \mathbb{R} \qquad \text{where } \boldsymbol{w}_{sim} \in \mathbb{R}^{6H}$$

Then, we compute the Context-to-Query attention by applying *softmax* to each row of $\boldsymbol{S}$. This computation tells us the most important question word for each context word.

$$\begin{aligned}
\bar{\boldsymbol{S}}_{i,:} &= \text{softmax}(\boldsymbol{S}_{i,:}) \in \mathbb{R}^M \\
\boldsymbol{a}_i &= \sum_{j=1}^{M} \bar{\boldsymbol{S}}_{i,j}\boldsymbol{q}_j \in \mathbb{R}^{2H}
\end{aligned}$$

Similarly, we compute Query-to-Context attention by applying *softmax* to each column of $\boldsymbol{S} \in \mathbb{R}^{N \times M}$. This computation tells us the most import context word for each question word.

$$\bar{\bar{\boldsymbol{S}}}_{:,j} = \text{softmax}(\boldsymbol{S}_{:,j}) \in \mathbb{R}^N$$
$$\boldsymbol{S}' = \bar{\boldsymbol{S}}\bar{\bar{\boldsymbol{S}}}^\top \in \mathbb{R}^{N \times N}$$
$$\boldsymbol{b}_i = \sum_{j=1}^{N} \boldsymbol{S}'_{i,j}\boldsymbol{c}_j \in \mathbb{R}^{2H}$$

Finally, for each context word $\boldsymbol{c}_i$, we output a bidirectional attention vector $\boldsymbol{g}_i \in \mathbb{R}^{8H}$, which is defined as follows

$$\boldsymbol{g}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \odot \boldsymbol{a}_i; \boldsymbol{c}_i \odot \boldsymbol{b}_i]$$

Modeling Layer

The output of the attention layer is then passed through a two-layer bidirectional LSTM to incorporate temporal information into the context word encodings **conditioned** on the question words.

$$\boldsymbol{m}_{i,fwd} = \overrightarrow{\text{LSTM}}(\boldsymbol{m}_{i-1}, \boldsymbol{g}_i) \in \mathbb{R}^H$$
$$\boldsymbol{m}_{i,rev} = \overleftarrow{\text{LSTM}}(\boldsymbol{m}_{i+1}, \boldsymbol{g}_i) \in \mathbb{R}^H$$
$$\boldsymbol{m}_i = [\boldsymbol{m}_{i,fwd}; \boldsymbol{m}_{i,rev}] \in \mathbb{R}^{2H}$$

Output Layer

Finally, this layer produces two probability distributions $\boldsymbol{p}_{start}, \boldsymbol{p}_{end} \in \mathbb{R}^N$ corresponding to the start and end answer locations in the context passage i.e. $\boldsymbol{p}_{start}(i)$ is the probability that answer starts at the i^{th} context word and $\boldsymbol{p}_{end}(j)$ is the probability that the answer ends at the j^{th} context word. Specifically, the probability distributions are computed as follows:

$$\boldsymbol{p}_{start} = \text{softmax}(\boldsymbol{W}_{start}[\boldsymbol{G}; \boldsymbol{M}])$$

$$\boldsymbol{m}'_{i,fwd} = \overrightarrow{\text{LSTM}}(\boldsymbol{m}'_{i-1}, \boldsymbol{m}_i) \in \mathbb{R}^H$$
$$\boldsymbol{m}'_{i,rev} = \overleftarrow{\text{LSTM}}(\boldsymbol{m}'_{i+1}, \boldsymbol{m}_i) \in \mathbb{R}^H$$
$$\boldsymbol{m}'_i = [\boldsymbol{m}'_{i,fwd}; \boldsymbol{m}'_{i,rev}] \in \mathbb{R}^{2H}$$

$$\boldsymbol{p}_{end} = \text{softmax}(\boldsymbol{W}_{end}[\boldsymbol{G}; \boldsymbol{M'}])$$

where

- $\boldsymbol{G} \in \mathbb{R}^{8H \times N}$ is a matrix with $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_N \in \mathbb{R}^{8H}$ as columns
- $\boldsymbol{M} \in \mathbb{R}^{2H \times N}$ is a matrix with $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_N \in \mathbb{R}^{2H}$ as columns
- $\boldsymbol{M'} \in \mathbb{R}^{2H \times N}$ is a matrix with $\boldsymbol{m'}_1, \ldots, \boldsymbol{m'}_N \in \mathbb{R}^{2H}$ as columns

3.2 Char-BiDAF Model (Baseline 2)

This model is equivalent to the original BiDAF model [4]. In this model, the Embedding Layer of the Base-BiDAF model is augmented with character-level embeddings, which are obtained as follows: First, the length of each input word is either padded or truncated to a maximum length (16 in our case). Then, through a lookup, the characters in the adjusted word are converted to trainable vectors, which are initialized with pre-trained vectors of dimension 64. The resulting matrix ($\mathbb{R}^{16 \times 64}$) is passed through a 1D convolution followed by max-over-time pooling (row-wise max in our case) to obtain a character-level embedding of a fixed size, which is a hyperparameter. Then, we concatenate these character-level embeddings with pre-trained GloVe word embeddings. Finally, just as in the Base-BiDAF model, we project the resulting vector to dimensionality $\boldsymbol{H}$ and pass them through a two-layer Highway Network. The output of this layer is a unified embedding vector $\in \mathbb{R}^H$ for each input word. I implemented this component from scratch based on the details from 3 papers: QANet [1], BiDAF [4], and CNNs for Sentence Classification [9].

3.3 AnsPtr-BiDAF Model (Baseline 3)

This model is the Char-BiDAF model with a modified output layer. Specifically, the Output Layer is replaced with the Boundary-model Answer Pointer layer introduced in [5], which was in turn inspired from Pointer Networks [6]. This layer is similar to the decoder in seq2seq models, but, instead of producing a probability distribution over the entire vocabulary, it produces a probability distribution over the indices of the words in the context paragraph. Given an encoding for the context $\boldsymbol{H}^r = [\boldsymbol{h}_1 \dots \boldsymbol{h}_N] \in \mathbb{R}^{L \times N}$ and trainable parameters $\boldsymbol{V} \in \mathbb{R}^{H \times L}, \boldsymbol{W} \in \mathbb{R}^{H \times L}, \boldsymbol{b} \in \mathbb{R}^H, \boldsymbol{v} \in \mathbb{R}^{1 \times H}, \boldsymbol{c} \in \mathbb{R}$, we compute $\boldsymbol{p}_{start}$ and $\boldsymbol{p}_{end}$ as follows:

$$\boldsymbol{F}_{start} = \tanh(\boldsymbol{V}\boldsymbol{H}^r + (\boldsymbol{W}\boldsymbol{h}_0 + \boldsymbol{b}) \otimes \boldsymbol{e}_N) \in \mathbb{R}^{H \times N}$$
$$\boldsymbol{p}_{start} = \text{softmax}(\boldsymbol{v}^\top \boldsymbol{F}_{start} + \boldsymbol{c} \otimes \boldsymbol{e}_N) \in \mathbb{R}^N$$

$$\boldsymbol{h}_1 = \overrightarrow{\text{LSTM}}(\boldsymbol{H}^r \boldsymbol{p}_{start}, \boldsymbol{h}_0) \in \mathbb{R}^H$$

$$\boldsymbol{F}_{end} = \tanh(\boldsymbol{V}\boldsymbol{H}^r + (\boldsymbol{W}\boldsymbol{h}_1 + \boldsymbol{b}) \otimes \boldsymbol{e}_N) \in \mathbb{R}^{H \times N}$$
$$\boldsymbol{p}_{end} = \text{softmax}(\boldsymbol{v}^\top \boldsymbol{F}_{start} + \boldsymbol{c} \otimes \boldsymbol{e}_N) \in \mathbb{R}^N$$

Here, $\otimes \boldsymbol{e}_N$ is the broadcasting operation, L is the dimension of the context encodings and H is the hidden size. I tested four variants of this layer with different $\boldsymbol{H}^r$ and $\boldsymbol{h}_0$. Only Variant 1 achieved performance improvement over the Char-BiDAF model. Others either under-performed the Char-BiDAF model or barely reached its performance level. I implemented all of these variants from scratch.

Variant 1: H^r is the output of the Modeling Layer and $\boldsymbol{h}_0 = \boldsymbol{0} \in \mathbb{R}^H$.

$$\boldsymbol{H}^r = \boldsymbol{M} = [\boldsymbol{m}_1 \dots \boldsymbol{m}_N] \in \mathbb{R}^{2H \times N}$$

Variant 2: H^r is the output of the Modeling Layer and $\boldsymbol{h}_0$ is the hidden state of the LSTM with average of the columns of $\boldsymbol{H}^r$ as input.

$$\boldsymbol{H}^r = \boldsymbol{M} = [\boldsymbol{m}_1 \dots \boldsymbol{m}_N] \in \mathbb{R}^{2H \times N}$$
$$\boldsymbol{h}_0 = \overrightarrow{\text{LSTM}}(\text{mean}(\boldsymbol{H}^r), \boldsymbol{0}) \in \mathbb{R}^H$$

Variant 3: H^r is the concatenation of the outputs of the Attention Layer and Modeling Layer and $\boldsymbol{h}_0 = \boldsymbol{0} \in \mathbb{R}^H$.

$$\boldsymbol{H}^r = [\boldsymbol{G}; \boldsymbol{M}] = [\boldsymbol{g}_1; \boldsymbol{m}_1 \dots \boldsymbol{g}_N; \boldsymbol{m}_N] \in \mathbb{R}^{10H \times N}$$

Variant 4: H^r is the concatenation of the outputs of the Attention Layer and Modeling Layer and $\boldsymbol{h}_0$ is the hidden state of the LSTM with average of the columns of $\boldsymbol{H}^r$ as input.

$$\boldsymbol{H}^r = [\boldsymbol{G}; \boldsymbol{M}] = [\boldsymbol{g}_1; \boldsymbol{m}_1 \dots \boldsymbol{g}_N; \boldsymbol{m}_N] \in \mathbb{R}^{10H \times N}$$
$$\boldsymbol{h}_0 = \overrightarrow{\text{LSTM}}(\text{mean}(\boldsymbol{H}^r), \boldsymbol{0}) \in \mathbb{R}^H$$

3.4 QANet

QANet has the same high-level structure as the BiDAF model [4], but the layers are built using a transformer-style building block instead of RNN/LSTMs. This building block is called the Encoder Block and is shown in Figure 1 (Right). It is made up of some number of 1D convolutions, followed by a multi-headed self-attention layer (with 8 heads), followed by a feed-forward layer. Layer-normalization is applied to the input to each of these three components, and each component is placed inside a residual block. A sinusoidal positional encoding [10] is added to the Block's input. For memory efficiency and better generalization, depthwise separable convolutions were used. These convolutions function as follows: given an input of shape $H \times W \times C$ (C = num channels), first we individually apply a (depthwise) convolution with a kernel of size $K \times K \times 1$ to each input channel. Then, another (point-wise) convolution is applied with D kernels of size $1 \times 1 \times C$ to produce the final output - here D is the desired number of output channels. Self-attention and feed-forward layers

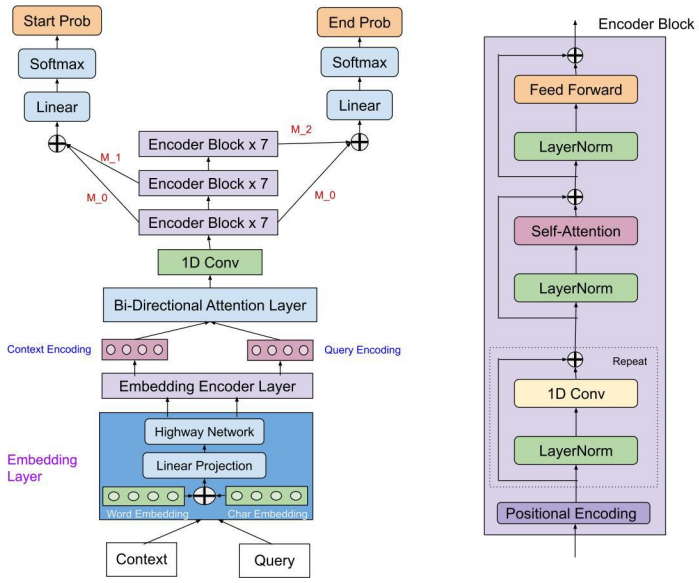

Figure 1: **Left:** QANet Architecture; **Right:** QANet's Encoder Building Block

were implemented as in a standard transformer module as described in [10]. I implemented most of the encoder block from scratch, but I adapted an efficient implementation of Self-Attention from an online tutorial [11]. The full architecture of the QANet model [1] is shown in Figure 1, and consists of five layers similar to those of the BiDAF model.

Embedding Layer produces embedding vectors for both the context and query. For this model, I adapted the Embedding Layer implemented for Char-BiDAF model [see note on PyTorch implementation below].

Embedding Encoder Layer encodes the incoming embeddings. It is made up of one Encoder Block with 4 1D-convolutions (each containing 128 filters of kernel size 7).

Attention Layer computes Context-to-Query and Query-to-Context attention. I reused the Bi-directional Attention Layer from the Base-BiDAF model.

Model Encoder Layer consists of 7 Encoder Blocks with 2 1D-convolutions (each containing 128 filters of kernel size 5). The set of 7 blocks is reused 3 times to produce three encodings $\boldsymbol{M}_0, \boldsymbol{M}_1, \boldsymbol{M}_2$. Also, the input to this layer is of dimensionality $4H$, so, in order to make residual connections, it is first projected to dimensionality H using a single 1D convolution.

Output Layer computes the probabilities of the answer start and end positions using the three Encoder Block outputs $(\boldsymbol{M}_0, \boldsymbol{M}_1, \boldsymbol{M}_2)$ from the Model Encoder Layer. With $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{1\times 2H}$,

$$\boldsymbol{p}_{start} = softmax(\boldsymbol{W}_1[\boldsymbol{M}_0; \boldsymbol{M}_1]), \quad \boldsymbol{p}_{end} = softmax(\boldsymbol{W}_2[\boldsymbol{M}_0; \boldsymbol{M}_2])$$

Note on PyTorch Implementation: In PyTorch, convolutions expect the input channels in the first dimension, but Linear layers apply to the last dimension. In order to avoid constantly switching back and forth between these two data formats, all linear projections are implemented using 1D-convolutions as suggested in [10] and as implemented in the github repository [12].

3.5 QANet-NoAtt

This model is derived from the QANet model and was implemented to test whether the bidirectional attention can be emulated with self-attention. There are three main differences from the QANet model:

1. The Embedding Layer's outputs for context ($\boldsymbol{c}_i$) and question ($\boldsymbol{q}_i$) are concatenated to form a single, combined sequence of length $N + M$.

$$\boldsymbol{g} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_M]$$

2. The Bidirectional Attention Layer is replaced with two Encoder Blocks, each with 2 1D convolutions of kernel size 5.
3. The outputs of the Model Encoder Layer are of length $N + M$, so they are truncated to length N by discarding the last M items.

4 Experiments

4.1 Data

The dataset for this project is a subset of the official SQuAD 2.0 dataset [3]. Each example in the dataset is a (context, question, answer) triplet. The dataset consists of both answerable and unanswerable questions. For answerable questions, the answer is a span of text within the context and is given as two indices into the context. For unanswerable questions, the answer is "NA" (Not Answerable). The dataset is divided into three splits: a **train set** of 129,941 examples, a **dev set** of 6078 examples, and a **test set** of 5915 examples. Additionally, the dev and test sets contain three ground truth answers per example. The goal of a Question Answering model is to extract the correct answer from the context paragraph for a given question.

4.2 Evaluation method

The models are evaluated based on two metrics: **Exact Match (EM)** which is a binary value indicating whether a prediction exactly matches the ground truth ; **F1 score** which is the harmonic mean of precision and recall. Here, precision is the number of correctly predicted answer words divided by the total number of predicted words, and recall is the number of correctly predicted answer words divided by the total number of words in the ground truth answer. For dev and test sets, we use the maximum of the F1 and EM scores across the three ground truth answers for each example.

4.3 Experimental details

The Base-BiDAF, Char-BiDAF, AnsPtr-BiDAF (Variant 1) were each trained for 30 epochs with a learning rate of 0.5 using Adadelta optimizer. The batch size was 64, and the hidden size (output of the embedding layer) was 100. The character-level embedding size was also 100. AnsPtr-BiDAF Variants 2-4 were trained for 10 to 20 epochs, but due to poorer performance than Char-BiDAF, the models were not explored further.

QANet and QANet-NoAtt models were trained for 30 epochs using Adam optimizer with $\beta_1 = 0.8, \beta_2 = 0.999$, and $\epsilon = 10^{-7}$. Learning rate was 0.001 and batch size was 12. Also, L2 weight decay with $\lambda = 3 \times 10^{-7}$ was applied to the parameters. The dimension of character-level embeddings was 200. Dropout was applied to character and word embeddings with rates 0.05 and 0.1 respectively. Unlike the original QANet model, dropout was applied after every Layer-Norm and before every residual connection inside the Encoder block (original QANet applied dropout only once for every two layers). Finally, I used layer dropout in the Encoder Block. This method randomly drops entire layers as opposed to individual hidden units, thus stochastically varies the model's depth. As described in [1] and [13], each layer is retained in the model with a probability of $p_l = 1 - \frac{1}{L}(1 - p_L)$ where L is the total number of layers and $p_L = 0.9$ is the survival probability of the last layer. All of the hyperparameters values were taken directly from the QANet paper [1].

4.4 Results

The SQuAD IID track dev set and test set performances of all models are shown in Table 1. The Answer Pointer model has a marginal improvement over Char-BiDAF model, which I believe is because the Output Layer of Char-BiDAF uses an LSTM to compute the end location probability, thus, in a way, already conditioned end probability on start probability. As we can see, QANet model clearly outperformed all baseline models. However, these scores are substantially lower than the ones

reported in the paper [1]. This discrepancy is due to the presence of unanswerable questions in the current dataset. Note that QANet's performance is substantially lower than the current state-of-the-art results on SQuAD 2.0 dataset, which are achieved using large pretrained models.

	Dev Set			Test Set	
	AvNA	EM	F1	EM	F1
Base-BiDAF	67.92	57.75	61.08	-	-
Char-BiDAF	69.55	60.23	63.33	-	-
AnsPtr-BiDAF 1	69.58	60.21	63.43	-	-
QANet	73.92	63.99	67.54	61.98	65.40
QANet-small	74.14	64.51	68.11	-	-
QANet-medium	**75.13**	**65.28**	**68.88**	**63.01**	**66.72**
QANet-NoAtt	70.79	61.13	64.56	57.06	60.53

Table 1: Dev and Test set results for SQuAD IID track

5 Analysis

5.1 Effect of Model Encoder's Size

I tested two derivatives of QANet model with smaller Model Encoder Layers: QANet-medium with 5 Encoder Blocks instead of 7, and QANet-small with 3 Encoder Blocks. As shown in Table 1, these models surprisingly outperformed the base QANet model. This suggests that the base QANet model might have been underfitted, and that training it longer could result in improved performance. However, the larger base QANet model is approximately 2x-3x slower than QANet-medium and QANet-small, so there is a trade-off.

5.2 Self-Attention v. Bi-directional Attention

I used the QANet-NoAtt model described in section 3.5 to determine the importance of an explicit Bi-directional Attention Layer in a transformer-style architecture. As we can see in Table 1, this model outperformed all of the baselines, suggesting that self-attention is capable of capturing bi-directional attention. However, it's performance is significantly lower than that of the base-QANet. Given the model started overfitting after about 15 epochs, I believe the performance gap could be reduced with further finetuning and regularization.

5.3 Analysis of Model Predictions

Figure 2 shows the distributions of incorrect predictions on the dev set by the QANet model (left) and the Char-BiDAF model (right). The horizontal axis is the gold answer length and the vertical axis is the number of incorrect predictions. As we can see, the majority of errors (red bars) occurred when the models predicted an answer for "Non-Answerable" questions. The next highest number of errors (sum of blue bars) occurred when the models predicted "Not Answerable" for answerable questions. The other errors (orange bars) occurred when the models predicted entirely different answers from the ground truth. Clearly, the models were having difficulties in determining the answerability of a question. Also, while both models were failing in similar scenarios, the base QANet model made fewer mistakes which could be attributed to its much larger size than that of Char-BiDAF. Perhaps adding an answer verifier module like the one proposed in the U-Net model [7], could help improve QANet's performance even further. The data for these plots were obtained using a modified version of the evaluation function - eval_dicts() - provided in the starter code.

6 Conclusion

In this project, I re-implemented the QANet model, evaluated it on the SQuAD 2.0 dataset, and achieved substantial performance improvement over the provided baseline BiDAF model and other baselines that I implemented. I also showed that high performance could still be achieved with fewer number of Encoder Blocks, which also gives a 2-3x speedup to the model. Finally, I demonstrated that Self-Attention Layer in the Encoder Block could implicitly perform bi-directional attention.

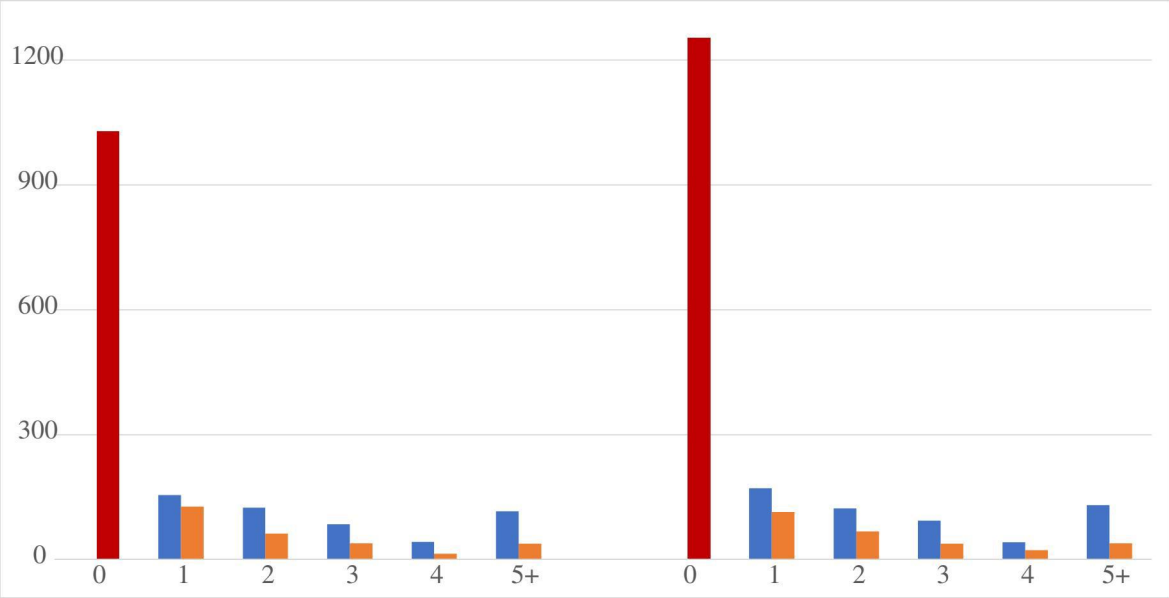

Figure 2: **Left:** Incorrect predictions by QANet; **Right:** Incorrect predictions by Char-BiDAF. Horizontal axis corresponds to gold answer length; Vertical axis corresponds to the number of wrong predictions for a given length. Red bars show the number of incorrect predictions for Non-Answerable questions. Blue bars show the number of incorrectly predicted NAs. Orange bars show other incorrect predictions for answerable questions.

However, the results achieved by all of the models in this project are still substantially lower than the state-of-the-art models. As noted in section 5.3, QANet model has a difficulty in determining the answerability of a question. As a potential improvement, an answer verifier module could be added to the model to help reduce "Non-Answerable" errors. Other possible avenues of future work include 1) investigating whether the base QANet model is underfitted, and if so, train the model longer to achieve higher performance; and 2) determining the minimum size of the QANet model that is sufficient to produce good results on the SQuAD 2.0 dataset.

Acknowledgments

As noted in the Models section, I implemented majority of the work by myself. I adapted/reused the following components from the specified Github repositories:

- Sinusoidal Position Encoding code from Nvidia's Sentiment-discovery repository [14] - *https://github.com/NVIDIA/sentiment-discovery*.
- Self-Attention module from University of Amsterdam's Deep Learning course repository (Tutorial 6) [11] - *https://github.com/phlippe/uvadlc_notebooks*.
- Inspiration for implementing linear projections using 1D convolutions from a QANet implementation [12] at *https://github.com/BangLiu/QANet-PyTorch*.

References

[1] 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. *arXiv:1804.09541 [cs]*, April 2018. arXiv: 1804.09541.

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv:1606.05250 [cs]*, October 2016. arXiv: 1606.05250.

[3] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[4] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *arXiv:1611.01603 [cs]*, June 2018. arXiv: 1611.01603.

[5] Shuohang Wang and Jing Jiang. Machine Comprehension Using Match-LSTM and Answer Pointer. *arXiv:1608.07905 [cs]*, November 2016. arXiv: 1608.07905.

[6] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. *arXiv:1506.03134 [cs, stat]*, January 2017. arXiv: 1506.03134.

[7] Fu Sun, Linyang Li, Xipeng Qiu, and Yang Liu. U-Net: Machine Reading Comprehension with Unanswerable Questions. *arXiv:1810.06638 [cs]*, October 2018. arXiv: 1810.06638.

[8] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks. *arXiv:1505.00387 [cs]*, November 2015. arXiv: 1505.00387.

[9] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882 [cs]*, September 2014. arXiv: 1408.5882.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv: 1706.03762.

[11] Phillip Lippe. Uva deep learning tutorials - tutorial 6: Transformers and multi-head attention.

[12] Bang Liu. Qanet-pytorch github repository.

[13] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep Networks with Stochastic Depth. *arXiv:1603.09382 [cs]*, July 2016. arXiv: 1603.09382.

[14] Nvidia. Sentiment discovery gihub repository.

A Appendix

The following plots show the progress of different models on dev set as they were being trained. The breaks in the plots are due to interruptions during training due to Azure credit exhaustion - training was later resumed using model checkpoints. For clarity, only 4 models are shown:

- QANet-medium in **red**
- QANet in **light blue**
- Char-BiDAF in **dark blue**
- Base-BiDAF in **orange**

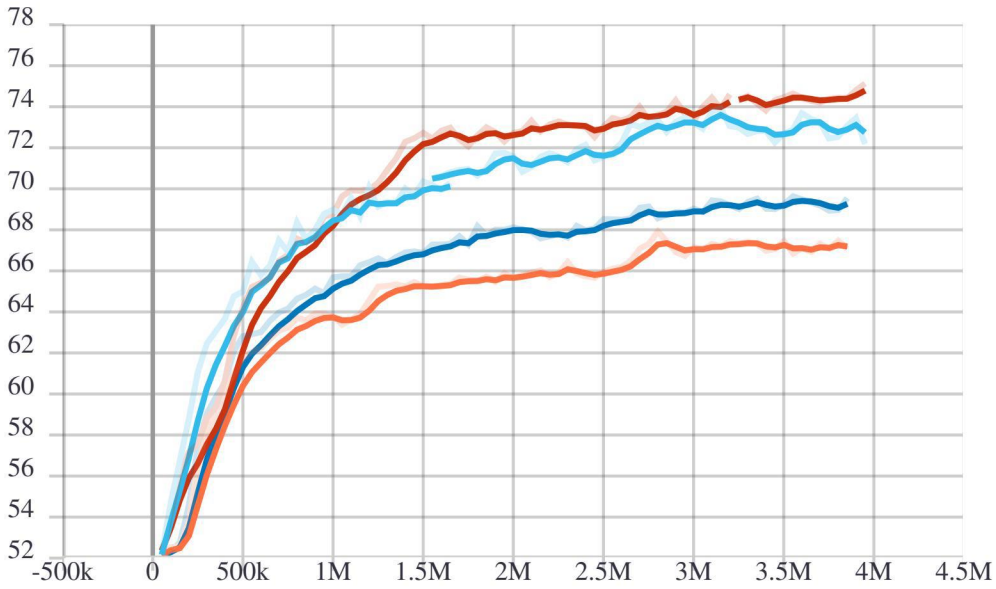

Figure 3: Dev set Answer v. No-Answer Accuracy

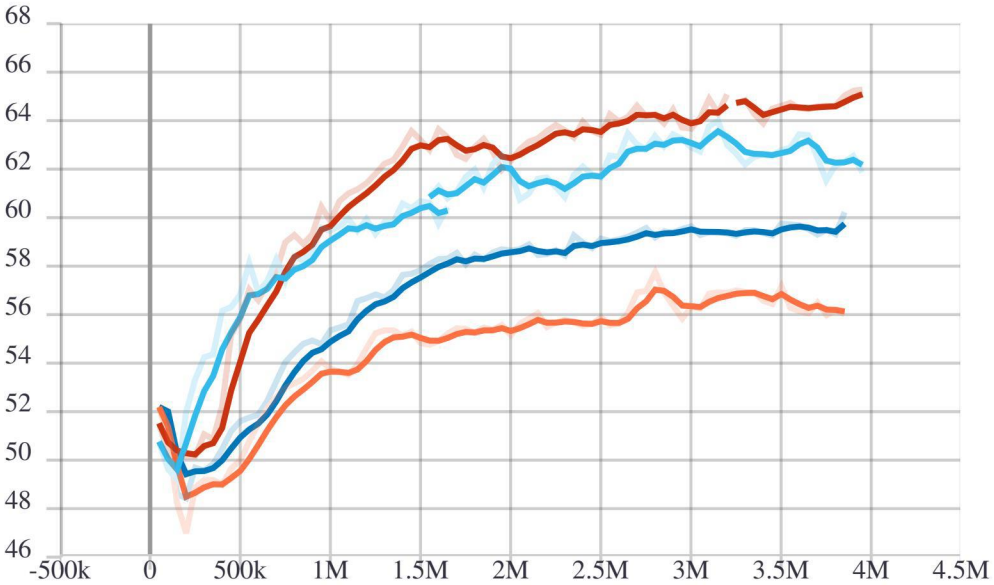

Figure 4: Dev set Exact Match (EM) scores

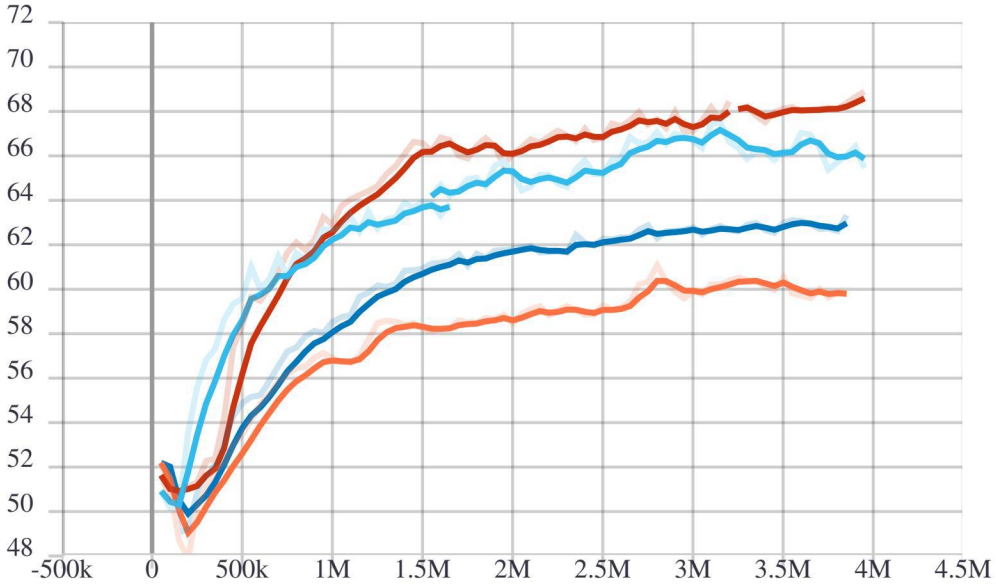

Figure 5: Dev set F1 scores

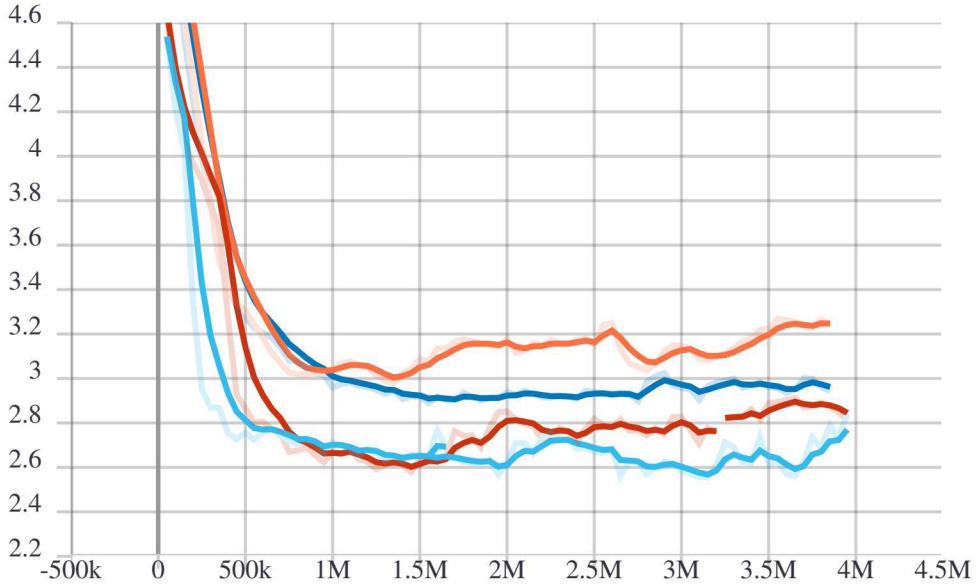

Figure 6: Dev set Negative Log Likelihood loss