BERT for Question Answering on SQuAD 2.0

Yuwen Zhang
Department of Materials Science and Engineering
yuwen17@stanfrod.edu

Zhaozhuo Xu
Department of Electrical Engineering
zhaozhuo@stanford.edu

Abstract

Machine reading comprehension and question answering is an essential task in natural language processing. Recently, Pre-trained Contextual Embeddings (PCE) models like Embeddings from Language Models (ELMo) [1] and Bidirectional Encoder Representations from Transformers (BERT) [2] have attracted lots of attention due to their great performance in a wide range of NLP tasks. In this project, we picked up BERT model and tried to fine-tune it with additional task-specific layers to improve its performance on Stanford Question Answering Dataset (SQuAD 2.0). We designed several output architectures and compared their performance to BERT baseline model in great details. So far, our best-proposed single model built an LSTM Encoder, an LSTM decoder and a highway network on top of the BERT base uncased model and achieved an F1 score of **77.96** on the dev set. By applying ensemble technique with selected models, our final version model currently ranks 12th on the Stanford CS224N SQuAD 2.0 test leaderboard with an F1 score **77.827** (name: Pisces_BERT).

1 Introduction

In this paper, we focus on reading comprehension question answering which aims to answer questions given passage or document. This QA task is always challenging since it requires a comprehensive understanding of natural languages and the ability to do further inference and reasoning. For our project, we mainly focus on Stanford Question Answering Dataset (SQuAD 2.0) which approximate the real reading comprehension circumstances. Models that have great performance on SQuAD can be regarded as a solid benchmark to solve RC and QA problems. Although rapid progress on this dataset has already been made by many teams, most researchers are still trying to work out a better model that (1) can adapt to no-answer questions very well and (2) can outperform human oracle (F1: 89.452). Our work mainly develops a model on top of the google-released BERT model. By replacing the linear BERT output layer with an encoder-decoder architecture, we successfully implemented the task-specific layers that can deal with the SQuAD 2.0 problems quite well. Currently, our best single model has achieved a F1 score of **77.96** on the dev set, while the ensemble version achieved **79.44** tested by us on the dev set. The leaderborad result is **77.827** on the test set.

2 Related work

Language model pre-training has shown to be effective for improving many natural language processing tasks. Among different models, the most recent google-released Bidirectional Encoder Representations from Transformers (BERT) [2] is a conceptually simple but empirically powerful one. It performs very well in a wide range of tasks, including our task, the SQUAD 2.0 competition. According to the paper [2], the pre-trained BERT representations can be fine-tuned with additional architectures to succeed in specific tasks. Therefore, here we are going to build our own output network on top of BERT pre-trained model.

Preprint. Work in progress.

In our project, we simply adapted BERT model with a linear output layer (described in paper[2]) as our baseline model. We designed several modules on top of it as the task-specific output layers. Generally, we aim at building a subsequent encoder-decoder architecture as post-processing to improve BERT model's performance on the SQUAD 2.0 challenge. For our main encoder-decoder architecture, we adapted RNN-based bi-directional long short-term memory layer (LSTM) [3] and gated recurrent units (GRU) [4] as the encoder and decoder, which are commonly used in sequence to sequence [5] translation task. We also tried a CNN-based encoder block, which is implemented in QANet [6] and CharCNN[7] networks. For multi-layer state transitions in our recurrent neural networks, we used highway network [8] to adaptively copy or transform representations. And for the final output layer, we compared the original linear layer and the QA output layer from Bi-Directional Attention Flow paper [9]. The details are explained in the approach section below.

3 Approach

In our project, we picked up the BERT model released by Google research in 2018 as our starting point [10, 11]. We first trained the BERT-Base-Uncased model with one additional linear output layer (default implementation in BERT paper [2]) as the baseline and evaluated their performance on the SQuAD 2.0 dataset. Then, we designed our new task-specific output architectures and fine-tuned the pre-trained BERT base model. We adapted our output layers based on insights gained from other networks and came up with several operations ourselves. By evaluating those models on the dev set, we were aiming to improve the performance of our designed models for better accuracy on SQuAD 2.0 dataset. We also tried to ensemble our model with BERT-Large-Cased models for better performance to succeed on Leaderboard.

3.1 Pre-trained BERT Baseline Model

BERT is a bidirectional encoder representations from transformers [2]. By training deep transformers on a carefully designed bidirectional language modeling task, the pre-trained BERT representations can be fine-tuned later with one additional output layer to perform well in a wide range of tasks without substantial task-specific architecture. The BERT model architecture is a multi-layer bidirectional transformer encoder, and it is discussed in great details in paper [12]. During the training process, for a given token, first, we convert it to the input embeddings, a sum of the token embeddings, the segmentation embeddings, and the position embeddings. Then we pre-trained the BERT model on bidirectional masked LM task and next sentence prediction task. With our pre-trained BERT model, we can easily adapt it to span prediction tasks, i.e. question answering on the SQuAD 2.0. To achieve this, we first represent the input question and paragraph as a single packed sequence (question with A segmentation embeddings and paragraph with B segmentation embeddings). And then we introduce two new parameters for the fine tuning: a start vector $S \in \mathbb{R}^H$ and end vector $E \in \mathbb{R}^H$. Let's denote the final hidden vector from BERT for the ith input token is $T_i \in \mathbb{R}^H$. We can calculate the probability of word i being the start of the answer span as a dot product between T_i and S followed by a softmax over all of the words in the paragraph. And the training objective is the log-likelihood of the correct start and end position. If the question has no answer, we will simply predict both the start and end positions as 0.

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

3.2 Modules on Top of BERT

In this section, we discuss modules we implemented as the output architecture. We gained these insights from existing networks and implemented them in our own way on top of the pre-trained BERT-base model.

3.2.1 Encoder and Decoder Blocks

We explored several encoders and decoders discussed in other NLP works and here is a summary:

Bidirectional Long Short-Term Memory (LSTM) Layer Encoder/Decoder: Long Short-Term Memory [3] (LSTM) is an RNN architecture aimed to solve vanishing gradients problem. On each

time step t, we have a hidden state $h^{(t)}$ and a cell state $c^{(t)}$. The cell can store long-term information, and the LSTM can erase, write and read information from the cell controlled by three gates (forget gate $f^{(t)}$, input gate $i^{(t)}$ and output gate $o^{(t)}$). On each time step, we can update our hidden state and cell state as following:

$$\widetilde{\mathbf{c}}^{(\mathbf{t})} = tanh(\mathbf{W_c h^{t-1}} + \mathbf{U_c x^t} + \mathbf{b_c})$$
$$\mathbf{c}^{(\mathbf{t})} = \mathbf{f}^{(\mathbf{t})} \circ \mathbf{c}^{(\mathbf{t-1})} + \mathbf{i}^{(\mathbf{t})} \circ \widetilde{\mathbf{c}}^{(\mathbf{t})}$$
$$\mathbf{h}^{(\mathbf{t})} = \mathbf{o}^{(\mathbf{t})} \circ tanh\mathbf{c}^{(\mathbf{t})}$$

By adding an RNN LSTM encode/decoder on top of the BERT model, we can integrate temporal dependencies between time-steps of the output tokenized sequence better.

Gated Recurrent Units (GRU) Encoder/Decoder: Gated Recurrent Units [4] is a simpler alternative to the LSTM. And on each time step t, we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state), and we used two gates (update gate $u^{(t)}$ and reset gate $r^{(t)}$) to control the states.

Convolutional Neural Network (CNN) Encoder: Current end-to-end machine reading and question answering (Q&A) models are primarily based on RNN encoders and decoders. However, there are innovative architectures that use CNN in this task for effectiveness and efficiency. Inspired by QANet [6] and CharCNN [7], we tried a CNN-based encoder which consists of convolution operations after RNN units, where convolution models local interactions and RNN models global interactions.

The first layer, as usual, is a typical Bidirectional LSTM. After that, for the CNN part, we use 2D convolution on the embedded sequence. Given a tensor with size as $(batch\_size \times seq\_length \times hidden\_state)$, we first unsqueeze it to $(batch\_size \times 1 \times seq\_length \times hidden\_state)$ and then do 2D convolution on the last two dimension with output size as $(batch\_size \times hidden\_state \times seq\_length \times 1)$. Finally, we squeeze the last dimension and swap the last two dimension to get the same dimension as input. This convolution extracts the relationship of nearby word embeddings in a sequence without change the dimension of input tensor.

3.2.2 Self-attention Layer

For some NLP tasks, people propose to use attention mechanism on top of the CNN or LSTM model to introduce an extra source of information to guide the extraction of sentence embedding. Here we also implemented a self-attention layer discussed in paper [12], and it allows each position of the output token to attend to all positions up to and including that position. In our implementation, we used the simplest basic dot-product attention. This can help for better interpreting the inference between different positions in the output sequence.

3.2.3 Highway Network

Highway network is a novel architecture that enables the optimization of networks with virtually arbitrary depth [8]. By applying a gating mechanism, a neural network can have paths along which information can flow across several layers without attenuation. Given an input $x \in \mathbb{R}^H$, a one-layer highway network computes:

$$\mathbf{x_{gate}} = \sigma(\mathbf{W_{gate}x} + \mathbf{b_{gate}}) \in \mathbb{R}^H$$
$$\mathbf{x_{proj}} = ReLU(\mathbf{W_{proj}x} + \mathbf{b_{proj}}) \in \mathbb{R}^H$$
$$\mathbf{x_{highway}} = \mathbf{x_{gate}} \odot \mathbf{x_{proj}} + (1 - \mathbf{x_{gate}})$$

$\mathbf{W_{proj}}$, $\mathbf{W_{gate}}$, $\mathbf{b_{gate}}$ and $\mathbf{b_{proj}}$ are learnable parameters. We choose a highway network to train multi-layer state transitions in our recurrent neural networks. Instead of traditional neural layers, it can allow the network to adaptively copy or transform representations. So it can help to refine the tokenized sequence.

3.2.4 Output Layer

For the output layer, we tried the original BERT linear output layer and the QA output layer from Bi-Directional Attention Flow (BiDAF-Out).

BERT Output Layer: A simple linear output layer that converts the dimension of the output sequence from $(batch\_size, seq\_len, hidden_state)$ to $(batch\_size, seq\_len, 2)$. And we split it to get the start and end logits. Finally, we compute the cross-entropy loss with the start and end position vectors

QA output layer from Bi-Directional Attention Flow (BiDAF-Out): To replace the linear output layer in default BERT model, we add a QA output layer adapted from BiDAF model [9]. First we apply a LSTM to the BERT tokenized output sequence $o_1, o_2, ..., o_N \in \mathbb{R}^H$ and get a model output sequence $(m_1, m_2, ..., m_N \in \mathbb{R}^{2H})$. Then we apply a bidirectional LSTM to the model output sequence, producing a vector $m_i^{'}$ for each m_i:

$$\mathbf{m^{'}_{i,fwd}} = LSTM(\mathbf{m^{'}_{i-1}}, \mathbf{m_i}) \in \mathbb{R}^H,\ \mathbf{m^{'}_{i,rev}} = LSTM(\mathbf{m^{'}_{i+1}}, \mathbf{m_i}) \in \mathbb{R}^H$$

$$\mathbf{m^{'}_i} = [\mathbf{m^{'}_{i,fwd}}; \mathbf{m^{'}_{i,rev}}] \in \mathbb{R}^{2H}$$

Now, let $O \in \mathbb{R}^{H \times N}$ be the matrix with columns $o_1, o_2, ..., o_N \in \mathbb{R}^H$. And let $M, M^{'} \in \mathbb{R}^{2H \times N}$ be matrices with columns $m_1, m_2, ..., m_N$ and $m_1^{'}, m_2^{'}, ..., m_N^{'}$. Then we calculate the start logits and end logits as following:

$$\mathbf{S_{logits}} = \mathbf{W_{start}}[O; M], \mathbf{E_{logits}} = \mathbf{W_{end}}[O; M^{'}]$$

Finally we compute the cross entropy loss with the start and end position vectors.

3.3 Proposed Models

Our main idea is to add an encoder-decoder architecture on top of the PyTorch implementation of BERT baseline [10]. This idea may come from the computer vision area. For multi-view synthesis task [13], we always use a general auto-encoder to generate the sketch of other views and an additional auto-encoder for texture level reconstruction.

Here, we propose a branch of models based on combinations of different modules. The architecture of our models can be found in Figure 1.

We divide our models mainly by different encoder-decoder structures. The first branch of models are based on **BiLSTM Encoder** and **BiLSTM Decoder**. They contain a Bidirectional LSTM (BiLSTM) encoder/decoder with 3 LSTM units. All recurrent units are associated with dropout with probability 0.2. Each LSTM unit takes BERT output as input and generates hidden states recurrently. For the second branch of models, we have **GRU Encoder** and **GRU Decoder**, which both contain a GRU unit. We also introduce an **CNN encoder**, which contains a BiLSTM layer and a 2D convolution layer described in Section 3.2.1. The BiLSTM layer also contains 3 bidirectional LSTM units.

Then, we may pass the output of encoders or decoders into a **Highway** network described in section 3.2.3. Besides that, we also use the **Self-attention** modules described in Section 3.2.2 to see if it improves the performance of encoder-decoder structure.

For the output layer, we have two types. The first type is standard **BERT-SQUAD-Out**, which is just a linear layer that obtains start and end logits from the sequence representation. The second type is the **BiDAF-Out**, which is described in Section 3.2.4.

We tried different combinations among these modules to get better performance than BERT baseline model. The details of representative models are presented below.

a. **BiLSTM Encoder + BiDAF-Out**
   This model uses the output layer of BiDAF instead of BERT output layer. We add a BiLSTM encoder before it to improve the performance.

b. **BiLSTM Encoder + Highway + BERT-SQUAD-Out**
   In this model, we modify the encoder-decoder structure by removing the BiLSTM decoder and alternate it with a highway network.

c. **BiLSTM Encoder + Highway + BiLSTM Decoder + BERT-SQUAD-Out**
   This is another typical encoder-decoder architecture where Highway network is used as a bridge for encoder and decoder.

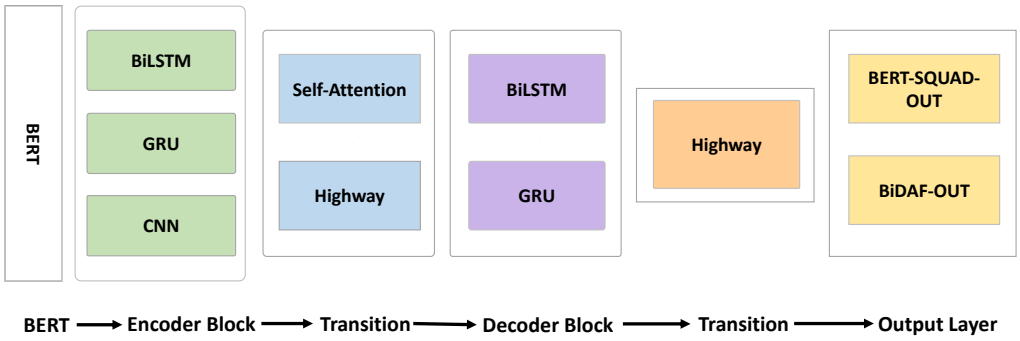

Figure 1: Schema of model architecture

d. **BiLSTM Encoder + BiLSTM Decoder + Highway + BERT-SQUAD-Out (our best model so far)**
   In this model, instead of using Highway network as a bridge between the encoder and decoder, we use it as a post-processing layer to improve the feature representation of BERT's output layer for SQUAD.

e. **GRU Encoder + Self-attention + GRU Decoder + BERT-SQUAD-Out**
   A GRU encoder with 1 GRU unit, after that, we pass it through a self-attention layer described in Section 3.2.2. Finally, we have a GRU decoder with 1 GRU unit. This is a typical architecture used in seq2seq model [5].

f. **GRU Encoder + GRU Decoder + BERT-SQUAD-Out**
   Simple GRU encoder-decoder. Both encoder and decoder contain 1 GRU units. We use this compared with the previous model to evaluate the performance of attention layer in the post-processing encoder-decoder network.

g. **CNN Encoder +BERT-SQUAD-Out**
   In this model, we use a CNN encoder, which contains a BiLSTM module as the first layer. The BiLSTM module has 3 LSTM units. Then we use the 2D convolution layer presented in Section 3.2.1 to extract the relationship between different embeddings in the sequence.

h. **CNN Encoder +Self-attention +BERT-SQUAD-Out**
   As typical GRU encoder-decoder with attention performs not so good, we combine the CNN encoder described above with self-attention layer to form an architecture on top of BERT. This is a typical structure used in QANet[6].

i. **CNN Encoder + BiLSTM Decoder + Highway + BERT-SQUAD-Out**
   In this model, we add a BiLSTM Decoder after the CNN encoder described above. Then, we use a Highway network before the finally BERT output layer as post-processing.

During the experiments, we compare the performance of those models on the Dev set. Then, we try to ensemble each two or more models for the final performance on the leaderboard.

All models are trained with pre-trained weights from BERT base model. **The code for the pre-trained BERT model is borrowed from Github [11], and all top implementations are realized by us.**

3.4 Ensemble

We finally picked up several best models for ensemble. The detailed approach can be discussed as voting on the questions that have no answer. Given a branch of models, for each question, if one model says it has no answer, we will make a prediction as "no answer". For the questions that have answers, we set a master model. The answer of the master model on these questions will be the final prediction. The intuition behind this is that, based on our analysis in section 4.5.2, bad models we tried are bad because they always have poor performance on questions that have no answer.

4 Experiments

4.1 Data

We used Stanford Question Answering Dataset (SQuAD 2.0) [14] to train and evaluate our models. Samples in this dataset include (question, answer, context paragraph) tuples. The paragraphs are from Wikipedia. The questions and answers were crowdsourced using Amazon Mechanical Turk. And we have around 150k questions in total, and roughly half of the questions are not answerable (this is new in SQuAD 2.0). However, if a question is answerable, the answer is guaranteed to be a continuous span in the context paragraph.

4.2 Evaluation method

We apply two metrics to measure the performance of our model: Exact Match (EM) score and F1 score.
Exact Match is a binary measure (true/false) of whether the system output exactly matches the ground truth answer exactly. This is a fairly strict metric.
F1 is a less strict metric, and it is the harmonic mean of precision and recall. $F1 = \frac{2 \times precision \times recall}{precision + recall}$.
The system would have 100% precision if its answer is a subset of the ground truth answer and 50% recall if it only includes one out of the two words in the ground truth output.
When a question has no answer, both the F1 and EM score are 1 if the model predicts no-answer, and 0 otherwise.

4.3 Experimental details

We set up several experiments with the default BERT-base model's configurations [11]. And we kept the training hyperparameters the same for all our proposed models: train_batch_size=24, learning_rate=3×10^{-5}, num_train_epochs=2.0, max_seq_length=384, doc_stride=128. We ran every experiment on server with 2 GPUs. It usually took 8-12 hours for a model to train.

4.4 Results

We evaluated our proposed models on the dev set, and the results are summarized in Table 1. In total, we have 7 models that exceed the baseline BERT-base model with one default output layer (PyTorch and Tensorflow implementation) in F1 or EM. Among all these models, model **BiLSTM Encoder + BiLSTM Decoder + Highway + BERT-SQUAD-Out** has the best F1 score **77.96** and EM score **74.98** on the Dev Set.

For ensemble models, we tried to ensemble every two models by voting on the questions with no answer. Then, we select one of the models as master who give predictions on questions that has answers. From the result we can see that model 11 and 7 have the best ensemble performance. We also tried to ensemble model 11 and 7 with BERT large case model and got a 79.443 F1 score and a 76.996 EM score on the Dev Set. Compared to the pure BERT large cased model's performance (F1 79.393, EM 76.917), the ensemble model improves a bit. We used our evaluation file to get scores for fair comparison. Our final model's performance on the Test Set also got a good result, with a **77.827** F1 score and a **74.472** EM score on the leaderboard.

5 Analysis

5.1 Model Performance Analysis

From the results, we can see that encoder-decoder architectures added on top of BERT is generally effective. It conducts post-processing on BERT output representations to improve their quality. After comparing different models, we find that:

Recurrent units such as BiLSTM and GRUs may help improve performance on top of BERT. All proposed models that outperform BERT baseline contain at least one recurrent encoder or decoder. It verifies our proposed idea that RNN encoder-decoder architecture can help to integrate temporal

ID	**Architecture on Top of BERT**	**F1**	**EM**
1	BERT-base PyTorch Implementation	76.70	73.85
2	BERT-base Tensorflow Implementation	76.07	72.80
3	GRU Encoder + Self-attention + GRU Decoder + BERT-SQUAD-Out	73.59	69.87
4	BiLSTM Encoder + BiDAF-Out	76.37	73.05
5	CNN Encoder +Self-attention +BERT-SQUAD-Out	76.49	73.23
6	CNN Encoder + BERT-SQUAD-Out	76.56	73.64
7	GRU Encoder + GRU Decoder + BERT-SQUAD-Out	76.85	73.77
8	CNN Encoder + BiLSTM Decoder + Highway + BERT-SQUAD-Out	77.07	73.87
9	BiLSTM Encoder + Highway + BERT-SQUAD-Out	77.41	74.32
10	BiLSTM Encoder + Highway + BiLSTM Decoder + BERT-SQUAD-Out	77.66	74.87
11	BiLSTM Encoder + BiLSTM Decoder + Highway + BERT-SQUAD-Out	**77.96**	**74.98**
12	Ensemble of 11 and 7	**78.35**	**75.60**
13	Ensemble of 11 and 7 and BERT large case model	**79.443**	**76.966**

Table 1: F1 and EM scores for different model architectures (All our implementations are done in PyTorch).

ID	**HasAnsF1**	**NoAnsF1**	**hasAnsEM**	**NoAnsEM**
1	79.29	74.33	73.33	74.34
2	80.75	71.78	73.92	71.78
3	82.47	65.43	74.71	65.44
4	81.26	71.875	74.33	71.88
5	80.48	72.82	73.68	72.82
6	79.70	73.67	73.61	73.67
7	80.80	73.30	74.26	73.30
8	81.56	72.95	74.85	72.95
9	80.73	74.37	74.26	74.37
10	74.68	80.40	68.87	80.40
11	78.72	77.27	72.47	77.27

Table 2: Has-Answer and No-Answer F1 and EM scores for different model architectures.

dependencies between time-steps of the output tokenized sequence better, thus refine the output sequence for following operations.

Self-attention may not help improve SQUAD performance when it is added on top of BERT. We can see that model 3 cannot behave as well as model 6 when self-attention is added between the GRU encoder and GRU decoder. It makes sense because the architecture of the BERT model alone is built on lots of masked attention layers and pre-trained with specific tasks. Adding attention on top of it as a task-specific layer may not improve its performance further.

CNN encoder can improve the performance on SQUAD when added on top of BERT with other structures. However, the benefits of CNN encoder is limited. When we replace the BiLSTM encoder with a CNN encoder in our best model, we have an F1 score dropped from 77.96 to 77.07. However, from model 5 we can see that, by adding a self-attention layer on top of the CNN encoder, we can improve the performance of our model. As we know that, CNN always models the local interactions and the following RNN or self-attention layer can model the global interactions. Although the performance of CNN encoder is not as good as an RNN one, it's always faster than RNN for getting rid of its iterative nature. So it is promising to combine the CNN encoder with a simple data augmentation technique to enrich the training data to improve the model's performance further.

BiDAF output layer cannot outperform the BERT output layer. We see that even adding an additional encoder, the output layer of BiDAF still cannot beat the baseline (Model 4). This may because, for BiDAF model, we actually pass the query-to-context and context-to-query attention to the output layer (modeling layer + an additional output layer). But in our BERT model, although the tokenized sequence we pass to the output includes both the query and the context information, we do not necessarily interfere with their relations further before passing to the output. We suppose that by adding an attention layer in between may help to improve this model's performance.

Highway network is generally effective when adding between the encoder and decoder or after encoder-decoder network. We can see that our best single model uses the highway network as post-processing. The second and third best model also use the highway, but they put the highway in the middle of encoder and decoder. So highway is always promising to serve as a multi-layer state transition.

Generally, adding encoder and decoder network with recurrent units on top of BERT with highway networks can have better performance than BERT baseline. We think the reason may be that the additional encoder-decoder serve as a post-processing step in fine-tuning. The encoder-decoder network further polishes the language representation for SQUAD task as the pre-trained BERT model is more general.

5.2 Has-Answer and No-Answer Analysis

In this section, we analyze the F1 and EM score on both questions that have answers and questions that have no answer. As shown in Table 2, for each model, we present their Has Answer F1, EM, and No Answer F1, EM. We find that:

Models that cannot beat BERT baselines have poor performance on questions that have no answer. For model 3, we can see that when self-attention is added between the GRU encoder and decoder, the F1 for questions that have no answer drops a lot. BiDAF output layer also has the same effect. When CNN encoder is added before the self-attention layer, the poor performance on no answer questions is improved a bit. To sum up, for the models that cannot beat the baselines, they increase the performance on questions with answers while sacrificing no answer question's performance. In fact, self-attention has been demonstrated to be effective in SQuAD 1.1[6, 12, 9]. This experiment may suggest that the promising structure for SQuAD 1.1 may not have good performance on questions with no answer.

Some outperforming models have a more balanced score in questions with or without answers. We can see that starting from model 7, some models that have higher total F1 or EM have more balanced F1 or EM on both questions with or without answers. For our best model, which is model 11, we can see a very balanced F1. However, model 8 with CNN encoder is still less balanced. Model 10 also shows a very imbalanced and almost inverted F1 in questions with/without answers.

We present the examples of predictions from the baseline (model 1), the best single model (model 11) and the worse model (model 3). All examples come from dev set. For each example in Table 3, there is no answer. However, model 3 always predict the wrong answer. Based on these analyses, instead of predicting both the start and end positions to be 0 for no-answer questions, we'd better figure out a more effective way to deal with no-answer questions. Besides, we need to take a deeper look into the trade-off between the accuracy for has-answer and no-answer accuracy.

Questions	**model 1**	**model 3**	**model 11**
Who made fun of the Latin language?	NA	Geoffrey Chaucer	NA
Who led Issacs troops to Cyprus?	NA	Guy de Lusignan	NA
Who began a program of church reform in the 1100s?	NA	the dukes	NA

Table 3: Examples

6 Conclusion

In this paper, we designed several task-specific architectures on top of the BERT model according to insights gained from other networks. By comparing their performance to BERT baseline model and doing a deep analysis, we finally implemented our best model with an RNN encoder-decoder structure followed by a highway network as the output layers. With ensemble techniques, our model achieves an F1 score of **79.44** on the Dev Set and **77.827** on the Test Set. We can further improve the performance of our model by fine-tuning the parameters in each layer. Also based on the error analysis, we can see that there is still a gap in has-answer and no-answer accuracy which we tried to compensate with ensemble on no-answer predictions. The underlying mechanism should be understood better to help balance the model's performance on both has-answer and no-answer predictions.

References

[1] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Sean Robertson. `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`, 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. *arXiv preprint arXiv:1804.09541*, 2018.

[7] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[8] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[9] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[10] GitHub. `https://github.com/huggingface/pytorch-pretrained-BERT`, 2018.

[11] GitHub. `https://github.com/google-research/bert`, 2018.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[13] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C Berg. Transformation-grounded image generation network for novel 3d view synthesis. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 3500–3509, 2017.

[14] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.