Diverse Ensembling with Bert and its variations for Question Answering on SQuAD 2.0

Zhen Qin *zhenqin@stanford.edu*
Weiquan Mao *mwq@stanford.edu*
Zhining Zhu *annazhu@stanford.edu*

Abstract

In this paper, we produce a question answering system that works well on SQuAD. Bi-Directional Attention Flow (BiDAF) model is implemented as baseline, which pushes F1 score to 61.508 and EM score to 57.99 on Dev dataset. Then we applied a language representation model called Bidirectional Encoder Representations from Transformers (BERT) on SQuAD dataset. With one additional output layer, we experiment with different hyper-parameters in fine-tuning pre-trained BERT representations. Aiming to improve upon a standard BERT implementation, we have tried adding more additional layers to BERT, applying L1 regularization, freezing the first few layers of BERT, and using BERT embedding on BiDAF. After ensembling all models, we have now pushed SQuAD 2.0 question answering Dev F1 score to 79.944, Dev EM score to 73.643, Test F1 score to 78.841 and Test EM score to 76.010.

1 Introduction

Question-Answering System is one of the most popular natural language process tasks due to the creation of large question answer datasets. This can be used in many practical applications such as virtual assistants and automated customer service. The release of the Stanford Question Answering Dataset [3] has facilitated rapid progress in this field. Our project uses BiDAF as baseline, BERT-based architecture as the core, L1 as regularization. The goal is to answer the question correctly - select the span of text or N/A if there is no answer in the paragraph. Another direction for improvement is to use Ensembling methods, where we combine multiple models into a more robust Question Answering system by several different ensemble mechanisms.

2 Related Work

In the past few years, reading comprehension with neural networks has been studied thoroughly. Most of the high-performing models uses neural attention mechanism to combine the representations for the context and the question. BiDAF[4] is one among them, which represents the context at different levels of granularity and uses a bi-directional attention flow mechanism to achieve a query-aware context representation without early summarization. Besides BiDAF, there are also other attention mechanism such as self-attention[7] and coattention[9]. Since last year, Bidirectional Encoder Representations from Transformers [1] (BERT) has achieved state-of-the-art performance for eleven NLP tasks, like Question Answering[3] and Question Natural Language Inference[8].

Drawing insights from the previous work, we attempt to leverage the performance of BiDAF and BERT models on Question Answering by fine-tuning, architectural changes, and other variations. We also aim at amplifying the effectiveness of all the above model changes by diverse Ensembling methods.

3 Approach

3.1 Basline: BiDAF

As the default project, the baseline model has already been provided, which is a model based on BiDAF [2] but does not include a character-level embedding layer. It is composed of Embedding Layer, Encoder Layer, Attention Layer, Modeling Layer and Output layer.

Specifically, the embedding layer performs an embedding lookup to convert the indices into word embedding for both context and question. A Highway Network is also used to refine the embedded representation. The encoder layer uses a bidirectional LSTM to incorporate temporal dependencies between timesteps of the embedding layer's output. The main idea of attention layer is that attention flows both ways - from the context to the question and from the question to the context. The modeling layer is tasked with refining the sequence of vectors after the attention layer. It integrates temporal information between context representations conditioned on the question. The output layer is tasked with producing a vector of probabilities corresponding to each position in context.

Our loss function for the baseline model is the cross-entropy loss for the start and end locations. We average across the batch and use Adadelta optimizer to minimize the loss.

3.2 Bidirectional Encoder Representations from Transformers

BERT[1] achieves state-of-the-art performance for eleven NLP tasks, like Question Answering and Question Natural Language Inference, through only fine-tuning the last layer. BERT has such noteworthy achievement because it learns a more powerful bi-directional representation than most of the previous approaches. BERT's architecture is mainly multi-layer bidirectional Transformer encoder with bidirectional self-attention mechanism. The encoder of BERT is pre-trained with two tasks, "masked language model" (MLM) and Next Sentence Prediction. These two objectives help encoder to learn both left and right contextual of a word in the sentence and provides significant support for downstream tasks like question answering.

3.2.1 Fine-tuning

We use the pre-trained BERT-Base[6] model, which is cased and has 12 layer with 768-hidden, 12-heads, and 110M total parameters. Cased means that the true case and accent markers are preserved. To integrate the pre-trained encoder of BERT and fine-tune it to solve SQuAD 2.0, a final classification layer is added with weights $W \in R^{H \times K}$, where H is the encoded hidden size and K is the number of classifier labels. The label probabilities $P \in R^K$ are computed with a softmax layer, $P = \text{softmax}(CW^T)$. For SQuAD 2.0, the final layer outputs two probabilities, start probability, indicating whether a token is start of the answer, and end probability, indicating whether a token is end of the answer. Averaged cross-entropy loss of start and end probability prediction is used as training loss. More specifically, Adam optimizer with L2 weight decay is used to minimize the cross-entropy loss.

For fine-tuning, we keep most of the hyper-parameters the same as in pre-training, with the exception of the batch size, learning rate, and number of training epochs. Due to the limitations on memory of our computing resources, when we change the batch size, we need to adjust the maximum sequence length accordingly.

3.3 Variations on BERT

3.3.1 Regularization

Regularization is a powerful tool to prevent over-fitting. The two most common regularization methods are L1 and L2 regularization. L1 regularization penalizes the weight vector for its L1-norm(i.e. the sum of absolute values of the weights), whereas L2 regularization uses L2-norm(i.e. the sum of squared values of the weights). In practice, L1 regularization produces sparsity - many of the weights of the features are set to zero as a result of L1-regularized training. Therefore, the size of the model can be much smaller than that produced by L2-regularization. We mainly experiment with L1

regularization. In details, the training loss with L1 regularization can be expressed as:

$$J_(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i) + \frac{\lambda}{2m}||w||_1$$

where λ is the regularization parameter.

3.3.2 Add more fully connected layers

With just one additional output layer, the pre-trained BERT representations manage to create state-of-the-art models. It is thus natural to change the architecture of BERT by stacking more layers and go "deeper". It is widely believed that deep models are able to extract better features than shallow models and hence, extra layers help in learning features. In the experiment, we add one extra layer before the output layer. And we feed this model into ensembling in the experiment.

3.3.3 Freeze shallow transformer layers

When fine-tuning a pre-trained model, we freeze the weights of the first few layers and prevent updates to their values on Gradient Descents. This technique is called “Freezing". We apply “Freezing" because the first few layers of BERT are very likely to capture universal features relevant to the downstream SQuAD task. More specifically, BERT[1] is composed of an embedding layer and then a sequence of 12 identical self-attention transformers. Its pre-training tasks include masked language model and Next Sentence Prediction, which should have covered learning features relevant to bidirectional contextual embedding.

3.4 Use BERT’s contextual embedding on BiDAF

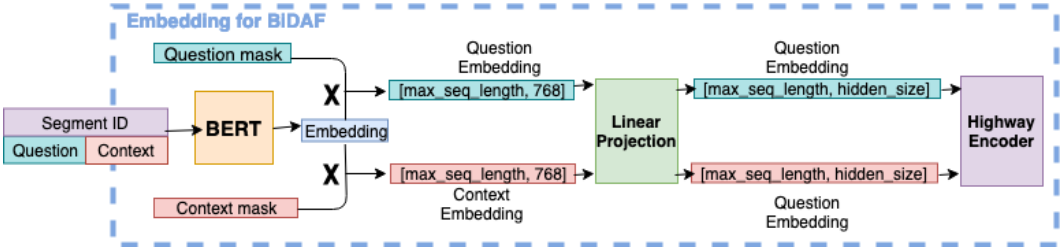


Figure 1: Architecture design: replace BiDAF’s embedding with BERT’s last output layer

BERT could also be used as a word-level embedding method with bidirectional contextual information. In this part we replace BiDAF’s GloVe word embedding with BERT last layer’s output as contextual word embedding. In this way, BERT’s weight won’t be changed at all during training. Figure 1 displays the integrated model architecture. Using BERT embeddings, we embed question and context together, which enhance the contextual correlation between question and context. The segment ID input for BERT is used to generate the question and context input mask for BiDAF. Output of BERT is filtered with question and context masks separately to get the separate question embedding and context embedding of shape [max sequnce length, 768], where 768 is BERT’s transformer’s hidden layer size. Then, after a linear projection layer, the embedding size is transformed into BiDAF’s defined hidden layer size. Finally, the embedding is feeded into a highway network, same as the original BiDAF.

In addition, in order to speed up BiDAF model training, we replace the original LSTM cell in BiDAF’s RNN encoder with Simple Recurrent Unit (SRU)[5], a unit with light recurrence that offers both high parallelism and sequence modeling capacity.

3.5 Ensembling

In modern Machine Learning, Ensembling Methods are extensively used to combine multiple learning algorithms, preferably from different model classes, into an aggregate model with better performance than any single model. Each of the BERT and BIDAF-based model we built make different predictions

on probabilities of start and end positions, and therefore we use Ensembling Methods in hope of utilizing the information extracted from all models. Specifically, we create two different algorithms in combining these models.

(a) Guided Random Search for Weighted Average Ensembling

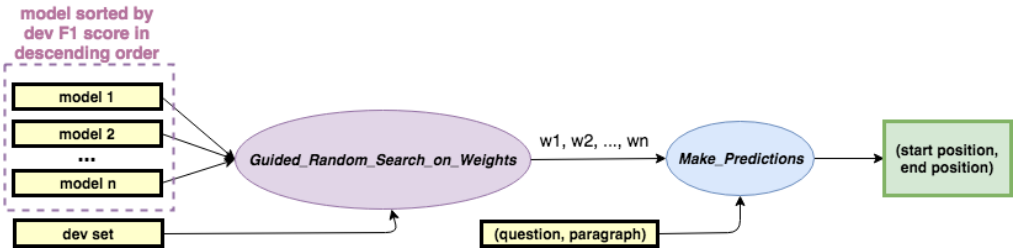


Figure 2: Pipeline for Guided Random Search + Weighted Average Ensembling

Assume that we have a total of n models to ensemble, and for each inputting (Question, Paragraph) pair, model k outputs

$$^{k}p_{start_i}, ^{k}p_{end_i}, 0 \leq i \leq \text{len(paragraph)}$$

where $^{k}p_{start_i}$ is the probability that the i-th position is the start position, and $^{k}p_{end_i}$ is the probability that the i-th position is the end position. Following this notation,

$$\begin{aligned} ^{k}p_{ij} &:= P(\text{start position} = i, \text{end position} = j) \text{ predicted by the k-th model} \\ &= p_{start_i} \times p_{end_i} \\ p_{ij} &:= P(\text{start position} = i, \text{end position} = j) \text{ predicted by the weighted average ensembling model} \\ &= \sum_k w_k \times p_{ij}^k, w \in R^n \end{aligned}$$

$\text{argmax}_{(i,j)} p_{ij} = (\text{start position}, \text{end position})$

So our goal is now reduced to finding the best $w \in R^n$. With this motivation, we develop a pipeline that learns the weights w, and make predictions on the prediction set. The details are described in Algorithm1. In high level, our algorithm randomly assign weights to each model, with the only restriction that a better model should never be assigned a lower weight than a model not as good. With the weights learned, we re-do the predictions by taking the weighted average of the probabilities predicted by each model.

Algorithm 1 Guided Random search + Weighted Average Ensembling

Input:
1. set of k models: $M \in R^k$
2. dev set: $\{(X_l, Y_l = (start\_pos, end\_pos)_l)\}_{l \in 1,2...n_{dev}}$
3. prediction set: $\{X_m\}_{m \in 1,2...n_{pred}}$
4. max_num_iter

Output: Predictions on the prediction set

$best\_weight = Guided\_Random\_Search\_on\_Weight$(M, max_num_iter, dev set)

Return $Make\_Predictions$(M, $best\_weight$, prediction set)

Algorithm 2 $Guided\_Random\_Search\_on\_Weights$: learn weights for Weighted Average Ensembling

Input:
1. set of k models: $M \in R^k$
2. max_num_iter
3. dev set: $(X_l, Y_l = (start\_pos, end\_pos)_l)_{l \in 1,2...n}$

Output: $best\_weight$

Initialize $best\_F1, best\_weight$ = 0.0, None

while *num_iter* < *max_num_iter* **do**
  Initialize $w \in R^k$ as an array of non-increasing random floats
  foreach *dev set example* (X_l, Y_l) **do**
    p_{ij}^k := *P(start_pos = i, end_pos = j) predicted by the k-th model in set M*
    Y_l' = $(start\_pos, end\_pos)_l'$ = *argmax*$_{(i,j)} \sum_k w_k \times p_{ij}^k$
  end
  $F1 = F1(\{Y_l\}_{l \in 1,2...n}, \{Y_l'\}_{l \in 1,2...n})$
  if $F1 > best\_F1$ **then**
    $best\_F1 = F1$
    $best\_weight = w$
  end
  num_iter += 1
end
Return $best\_weight$

Algorithm 3 $Make\_Predictions$: using weights learned from Guided Random Search

Input:
1. set of k models: $M \in R^k$
2. weights $w \in R^k$
3. prediction set: $\{X_l\}_{l \in 1,2...n}$

Output: Predictions on the prediction set

for *each prediction set example* X_l **do**
  $p_{ij}^k := P(start\_pos = i, end\_pos = j)$ predicted by the k-th model in set M
  Y_l' = $(start\_pos, end\_pos)_l' = argmax_{(i,j)} \sum_k w_k \times p_{ij}^k$
end
Return prediction on the prediction set $\{Y_l'\}_{l \in 1,2...n}$

(b) Follow the Most Confident Prediction

Another approach is to always adopt the most confident prediction for each example as the final prediction. This means that if model k outputs a prediction (i,j) with p_{ij}^k, and p_{ij}^k is greater than the prediction probability of this example by any other model, then we follow model k's prediction (i.e. start position = i, end position = j). The details are elaborated in Algorithm 4.

4 Experiments and Analysis

4.1 Data

We use SQuAD 2.0 as the reading comprehension data set. The paragraphs in SQuAD are from Wikipedia. The questions and answers are using labeling from Amazon Mechanical Turk. There are around 150k questions in total, and roughly half of the questions cannot be answered using the provided paragraph. However, if the question is answerable, the answer is a chunk of text taken directly from the paragraph. This means that SQuAD systems don't have to generate the answer text – they just have to select the span of text in the paragraph that answers the question.

Algorithm 4 Follow the most Confident Prediction

Input:
1. set of k models: $M \in R^k$
2. dev set: $\{(X_l, Y_l = (start\_pos, end\_pos)_l)\}_{l \in 1,2...n_{dev}}$
3. prediction set: $\{X_m\}_{m \in 1,2...n_{pred}}$

Output: Predictions on the prediction set

for *each prediction set example* X_l **do**

$p_{ij}^k := P(start\_pos = i, end\_pos = j)$ predicted by the k-th model in set M

$(start\_pos, end\_pos)_l^{k\prime}, p_l^k = argmax_{(i,j)} p_{ij}^k, max_{(i,j)} p_{ij}^k$

Y_l' = prediction of the start and end position = $(\text{start\_pos, end\_pos})_l' = (\text{start\_pos, end\_pos})_l^{argmax_k p_l^k}$,

end

Return prediction on the prediction set $\{Y_l'\}_{l \in 1,2...n}$

The SQuAD dataset has been split into three sets: Train set has 129,941 examples, all taken from the official SQuAD 2.0 training set; Dev set has 6078 examples, randomly selected from the official dev set; Test set has 5921 examples, the remaining examples from the official dev set along with some hand-labeled examples.

4.2 Evaluation Method

We mainly use two types of evaluation metrics, Exact Match and F1 score. Exact Match(EM) is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly. In our evaluation, EM stands for the percentage of outputs that match exactly with the ground truth. F1 is the harmonic mean of precision and recall, more specifically:

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \; ; \; \text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

For questions that do have answers, we take the maximum F1 and EM scores across the three human-provided answers for that question. And for those without answers, both the F1 and EM score are 1 if the model predicts no-answer, and 0 otherwise.

4.3 Experiments and Analysis

4.3.1 Baseline

First, we train the baseline model and compared the loss, AvNA(Answer vs. No Answer), EM, and F1(official SQuAD evaluation metrics) for both train and dev sets. Over 3 million iterations we find that: Firstly, the train loss continues to improve throughout. Secondly, the dev loss begins to rise around 2M iterations(overfitting). Thirdly, the dev AvNA reaches about 68, the dev F1 reaches about 60 and the dev EM score reaches around 57. Although the dev NLL improves throughout the training period, the dev EM and F1 scores initially get worse at the start of training, before then improving.

4.3.2 BERT

(a) Fine-tuning

We visualize the loss curves of all BERT fine-tuning experiments below:

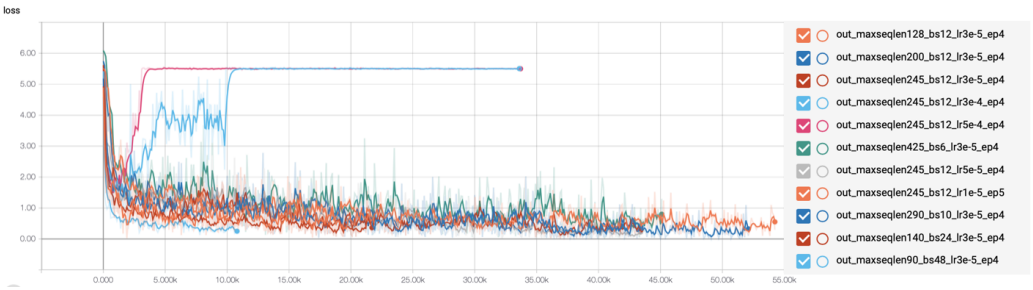


Figure 3: Learning curve of all BERT fine-tuning experiments

We can find that with large learning rates in scale of $e-4$ ($5e-4$ and $3e-4$), learning curves spike after around 5k iterations, and the losses fail to converge. Moreover, when learning rate = $5e-5$ or $3e-5$, the losses converge the fastest. It turns out that when $max\_seq\_length = 245$, and $batch\_size = 12$, $learningrate = 3e-5$, with best performance, Dev F1 score = 77.166.

(b) L1 regularization

To experiment with the effects of L1 regularization, we fix the maximum sequence length to 140, batch size to 24, learning rate to $3e-5$ and epoch to 4, while changing L1 regularization parameter from $1e-4$, $1e-3$ to $1e-2$.

Table 1: Comparison between different L1 regularization parameters

L1 regularization parameter	Dev F1	Dev EM
$\lambda = 0$	74.679	71.915
$\lambda = 1e-4$	75.705	73.001
$\lambda = 1e-3$	76.666	73.824
$\lambda = 1e-2$	76.76	73.955

As seen from the table 1, when we apply L1 Regularization and as we increase λ, the performance becomes better. It is believed that for this experiment where maximum sequence length is 140, batch size is 24, there is a serious over-fitting problem. Thus, when applying L1 Regularization, we naturally use sparsity to eliminate insignificant features. When we increase λ, increasingly more unnecessary features are removed, and thus we manage to alleviate the over-fitting problem.

(c) Freeze shallow transformer layers

Table 2: Comparison between freezing different layers

Description	Dev F1	Dev EM
No freeze	74.679	71.915
Freeze first 1 transformer layers	76.841	73.939
Freeze first 3 transformer layers	74.702	71.8
Freeze first 5 transformer layers	74.306	71.405
Freeze first 11 transformer layers	59.536	56.038

With max sequence length 140, batch size 24, we experiment on freezing the embedding layer and first 1, 3, 5 or 11 self-attention transformer layers while training BERT, and compare the result with the training without freezing. From table 2, we can see that reasonable freezing depth increases speed of training without hurting performance. Freezing first 5 layers increase the training speed by 30% because we stop more backward gradients calculation. In addition, freezing lets BERT focus on learning task specific features in the subsequent transformer layers and linear layer, and thus improves performance.

4.3.3 Use BERT's contextual embedding on BiDAF

Figure 4 represents the training loss for baseline, baseline with SRU, and baseline with SRU and BERT embbeding. With BERT embbeding, training loss converges faster than the other two model. This is because original GloVe embbeding for baseline does not contain any contextual information, which should be learned by BiDAF model during training. Using BERT embedding, BiDAF model can skip many iterations of computations in finding the contextual information.

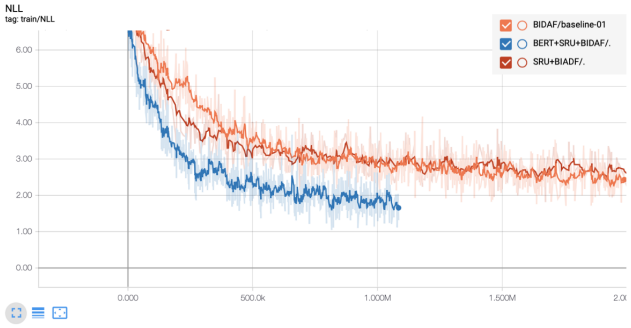


Figure 4: Training loss comparisom for BiDAF, BiDAF with SRU, BiDAF with and BERT embedding

4.3.4 Ensembling

We run the two ensembling algorithms on all 26 models we have (the details of all models are listed in Appendix II), and the performance of the best model produced by both methods is listed in Table 3.

Table 3: Ensembling Results

ID	Ensembling Method	Dev F1	Dev EM	Test F1	Test EM	Testboard Submission
1	Guided Random Search for Weighted Average	79.944	77.081	78.841	76.010	Submission 2
2	Follow the Most Confident Prediction	77.941	75.930	Did not evaluate on the test set		

(a) Weights learned by "Guided Random Search for Weighted Average Ensembling" approach

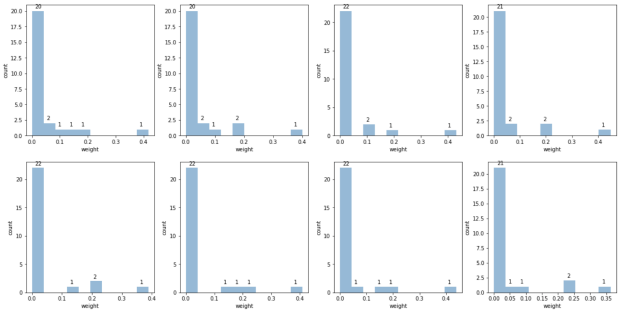


Figure 5: Ensembling Weights for Top 8 Ensembling Models in one run(100 iters) of Guided Random Search of weights

Figure 5 visualizes the weights for the top 8 Ensembling models in one run(100 iters) of Guided Random Search of weights by plotting the distribution in histograms. For all 8 Ensembling models, there is a clear separation between the weight of the top model to ensemble, and the rest of the models to ensemble. Most of the models only bear a weight in the order of 1e-3.

(b) Guided Random Search on Weights

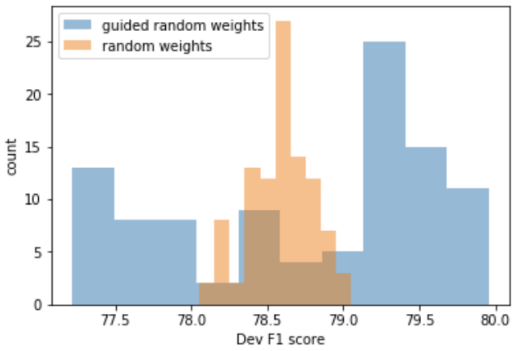


Figure 6: F1 scores in one run(100 iters) of Guided Random Search of weights vs F1 scores in 100 iters of random weights with no guide

In order to enhance the performance and speed up weights learning, we incorporate some domain knowledge in choosing weights. We add a guidance that a better model should never be assigned a lower weight than a model not as good. Figure 6 is a histogram that visualizes the F1 scores in one run(100 iters) of Guided Random Search of weights vs F1 scores in 100 iters of random weights with no guide. If weights are initialized purely randomly, the resulted F1 scores follow a normal distribution. On the other hand, with the guidance, Ensembling is more likely to escape from the expected F1, and pick up the extremes - models with either high or low F1 scores.

(c) Generalization to Test Set

Ensembling Methods are known to be very effective in decreasing variances of the final model, and reducing over-fitting. While a model is unstable if a small change to the training set causes a large change in the output hypothesis, Ensembling smooths out this dramatic shift by averaging the results. Quantitatively, our Ensembling model turns out to generalize well to the test set, with only a 1.3% decrease in both Test F1 and Test EM from Dev F1 and Dev EM, respectively.

(d) Analogy between Weighted Average and Majority Vote

Among all p_{ij}^k predicted by all of 26 non-Ensembling models over all dev set examples, only 1.48% of them are greater than 0.99, and 76.47% are less than 0.001. This highly-skewed distribution of probabilities implies that our "Weighted Average" approach actually resembles "Weighted Majority Vote" in nature because it is very likely that only the top prediction by each model is taken into consideration during Ensembling.

(e) Efficiency of "Follow the Most Confident Prediction" approach

Although "Follow the Most Confident Prediction" approach fails to yield a result as good as the "Weighted Average" approach, it is useful in its efficiency. This method avoids the usage of dev set, as well as a learning process in Ensembling. It outputs the prediction on prediction set directly after a single iteration over the model's predictions. Therefore, this method is both data efficient and time efficient, with a small sacrifice in accuracy.

5 Error Analysis

Our best model is the Weighted Average Ensembling Model (model 27 in Appendix II). In this section, to understand what our best model manages to solve, as well as its limitations, we analyze three typical scenarios in predicting the answer spans. We also compare our predictions with the baseline predictions, and dive into the causes of differences:

5.1 Example 1: Syntactic complications and ambiguities

Context:A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 that is not a prime number is called a composite number. For example, 5 is prime because 1 and 5 are its only positive integer factors, whereas 6 is composite because it has the divisors 2 and 3 in addition to 1 and 6. The fundamental theorem of arithmetic establishes the central role of primes in number theory: any integer greater

than 1 can be expressed as a product of primes that is unique up to ordering. The uniqueness in this theorem requires excluding 1 as a prime because one can include arbitrarily many instances of 1 in any factorization, e.g., 3, $1 \cdot 3$, $1 \cdot 1 \cdot 3$, etc. are all valid factorizations of 3.

Question:What is the only divisor besides 1 that a prime number can have?

Our model prediction: N/A

Baseline prediction: N/A

Analysis: Given the context, the correct answer should be "itself". Both our model and baseline incorrectly predict "No Answer". This might be due to a preposition reference confusion problem. In previous training, when model encounters a preposition, it usually needs to find the noun that this preposition referring to as the final answer. However, in this problem, the preposition "itself" should be the final correct answer.

5.2 Example 2: Paraphrase Problem

Context: The Beroida, also known as Nuda, have no feeding appendages, but their large pharynx, just inside the large mouth and filling most of the saclike body, bears "macrocilia" at the oral end. These fused bundles of several thousand large cilia are able to "bite" off pieces of prey that are too large to swallow whole – almost always other ctenophores. In front of the field of macrocilia, on the mouth "lips" in some species of Beroe, is a pair of narrow strips of adhesive epithelial cells on the stomach wall that "zip" the mouth shut when the animal is not feeding, by forming intercellular connections with the opposite adhesive strip. This tight closure streamlines the front of the animal when it is pursuing prey.

Question: Beroida are known by what other name?

Our model prediction: Nuda

Baseline prediction: Beroida

Analysis: In this example, the baseline model fails in outputting the correct prediction, while our model succeeds in making the exact prediction. Instead of predicting the paraphrase of “Beroida", the baseline model returns “Beroida" itself. Our model mitigates this Paraphrase Problem probably because BERT uses bidirectional self-attention and therefore is more likely to pick up the right contextual information.

5.3 Example 3: Predicting No-Answer

Context: Lake Constance consists of three bodies of water: the Obersee ("upper lake"), the Untersee ("lower lake"), and a connecting stretch of the Rhine, called the Seerhein ("Lake Rhine"). The lake is situated in Germany, Switzerland and Austria near the Alps. Specifically, its shorelines lie in the German states of Bavaria and Baden-Württemberg, the Austrian state of Vorarlberg, and the Swiss cantons of Thurgau and St. Gallen. The Rhine flows into it from the south following the Swiss-Austrian border. It is located at approximately (47°39’N 9°19’E) / (47.650°N 9.31) / 47.650;

Question: How many bodies of water make up the Rhine?

Our model prediction: N/A

Baseline prediction: three

Analysis: In this example, our model succeeds in outputting “No Answer", which is not captured by the baseline BiDAF model. Both of models use a threshold score for whether a span answers a question. Since our BERT model goes through a pass of NA threshold tuning after training, they are more likely to get a more reasonable prediction of NO Answer.

6 Conclusion

In this paper, we have implemented four variants on BERT by adding extra layers before output layer, applying regularization, freezing shallow transformer layers and using BERT’s contextual embedding on BiDAF. Simple Recurrent Unit is also applied to accelerate the training process

as well as improving the performance. Besides, we have proposed two ensembling algorithms to further improve the performance of our models. After fine-tuning and ensemble 26 BiDAF-based and BERT-based models, we can push Test F1 score to 78.841, Test EM score to 76.010 with a relatively small dataset, which achieves competitive performance with other published stat-of-the-art architectures and rank around 30 on the SQuAD leaderboard.

There are multiple other things that can be implemented to further improve the model. One promising work involves synthetic self-training. We could use seq2seq model to generate positive questions from context and answer. Then we could heuristically transform positive questions into negative questions, like "no answer" or impossible. This method is proved to be effective to push 3.0 F1/EM score higher by Google AI Language.

References

[1] 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.

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

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

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

[5] Sida I. Wang Hui Dai1 Yoav Artzi Tao Lei, Yu Zhang. Simple recurrent units for highly parallelizable recurrence. *arXiv preprint arXiv:1709.02755*, 2018.

[6] `https://github.com/google-research/bert`. Tensorflow code and pre-trained models for bert.

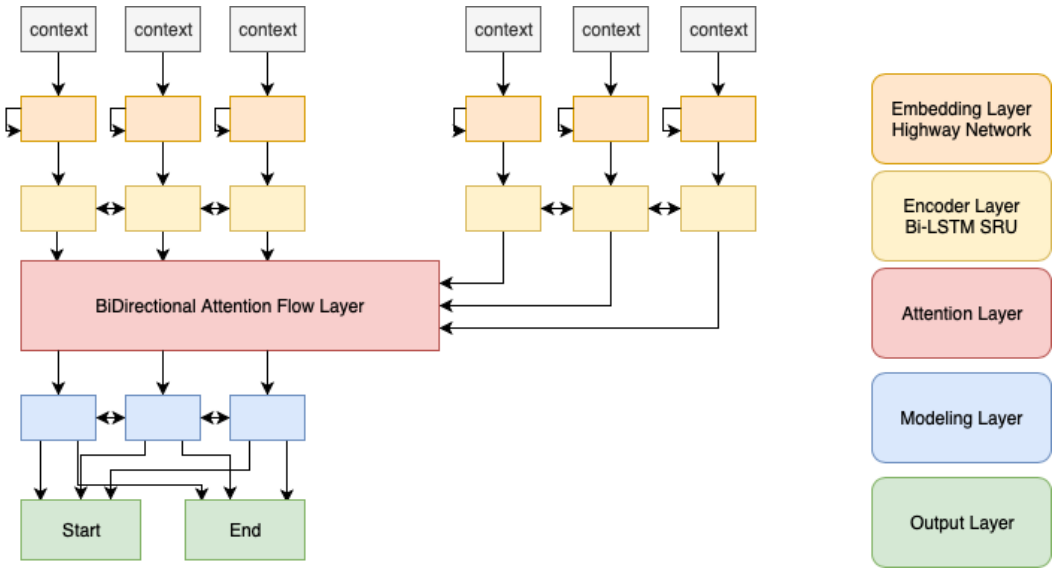
[7] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198, 2017.

[8] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[9] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

Appendix

Appendix I: Architecture of BiDAF Models (LSTM or SRU encoder)



Appendix II: Details of all models

Table 4: All models

ID	Experiment Name							Dev F1	Dev EM
BERT Models		**Pre-trained model**	**Number of Epoches**	**Learning Rate**	**Batch Size**	**Max Sequence length**	**Note**		
1	out_maxseqlen245_bs12_lr3e-5_ep6l1	BERT-Base, Cased	6	$3e-5$	12	245	$\lambda = 1e-4$	77.206	73.478
2	out_maxseqlen245_bs12_lr3e-5_ep4	BERT-Base, Cased	4	$3e-5$	12	245		77.166	73.643
3	out_maxseqlen140_bs24_lr3e-5_epoch4_freeze0_l1_0	BERT-Base, Cased	4	$3e-5$	24	140	freeze 1 layer	76.841	73.939
4	out_maxseqlen140_bs24_lr3e-5_ep4_l11e-2	BERT-Base, Cased	4	$3e-5$	24	140	$\lambda = 1e-2$	76.76	73.955
5	out_maxseqlen140_bs24_lr3e-5_ep4_l11e-3	BERT-Base, Cased	4	$3e-5$	24	140	$\lambda = 1e-3$	76.666	73.824
6	out_maxseqlen245_bs12_lr3e-5_ep6	BERT-Base, Cased	6	$3e-5$	12	245		75.925	72.343
7	out_maxseqlen140_bs24_lr3e-5_ep4_l11e-4_uncased	BERT-Base, Uncased	4	$3e-5$	24	140	$\lambda = 1e-4$	75.899	72.902
8	out_maxseqlen140_bs24_lr3e-5_ep4_l11e-4	BERT-Base, Cased	4	$3e-5$	24	140	$\lambda = 1e-4$	75.705	73.001
9	out_maxseqlen245_bs12_lr3e-5_ep4_l11e-4	BERT-Base, Cased	4	$3e-5$	12	245	$\lambda = 1e-4$	75.671	72.606
10	out_maxseqlen245_bs12_lr3e-5_ep5_l1+	BERT-Base, Cased	5	$3e-5$	12	245	$\lambda = 1e-4$, add one layer	75.354	71.685
11	out_maxseqlen245_bs12_lr3e-5_ep4_uncased	BERT-Base, Uncased	4	$3e-5$	12	245		75.071	71.372
12	out_maxseqlen140_bs24_lr3e-5_epoch4_freeze2_l1_0	BERT-Base, Cased	4	$3e-5$	24	140	freeze 3 layers	74.702	71.8
13	out_maxseqlen140_bs24_lr3e-5_epoch4	BERT-Base, Cased	4	$3e-5$	24	140		74.679	71.915
14	out_maxseqlen290_bs10_lr3e-5_epoch4	BERT-Base, Cased	4	$3e-5$	10	290		74.633	71.372
15	out_maxseqlen245_bs12_lr5e-5_ep4	BERT-Base, Cased	4	$5e-5$	12	245		74.546	71.092
16	out_maxseqlen200_bs12_lr3e-5_ep4	BERT-Base, Cased	4	$3e-5$	12	200		74.356	71.241
17	out_maxseqlen140_bs24_lr3e-5_epoch4_freeze4_l1_0	BERT-Base, Cased	4	$3e-5$	24	140	freeze 5 layers	74.306	71.405
18	out_maxseqlen245_bs12_lr1e-5_ep5	BERT-Base, Cased	5	$1e-5$	12	245		73.885	70.829
19	out_maxseqlen400_bs6_lr3e-5_epoch4	BERT-Base, Cased	4	$3e-5$	6	425		73.725	70.5
20	out_maxseqlen128_bs12_lr3e-5_ep4	BERT-Base, Cased	4	$3e-5$	12	128		73.638	71.142
21	out_maxseqlen245_bs12_lr3e-5_ep4+	BERT-Base, Cased	4	$3e-5$	12	245	add one layer	73.292	69.908
22	out_maxseqlen90_bs48_lr3e-5_ep4	BERT-Base, Cased	4	$3e-5$	48	90		72.954	70.813
23	out_maxseqlen140_bs24_lr3e-5_epoch4_freeze10_l1_0	BERT-Base, Cased	4	$3e-5$	24	140	freeze 11 layers	59.536	56.038
BIDAF Models		**Word Embeddings**	**Number of Epoches**	**Learning Rate**	**Encoder**	**Note**			
24	baseline_sru	GloVe	30	0.5	SRU			64.08	
25	bert_with_bidaf_epoch_8	BERT-Base, Cased	8	0.5	SRU			63.987	60.809
26	baseline	GloVe	30	0.5	LSTM	Baseline		61.508	57.99
Ensembling Models									
27	Guidede Random Search for Weighted Average							79.944	77.081
28	Follow the Most Confident Prediction							77.941	75.930