Building a QA system (IID SQuAD track)

Stanford CS224N Default Project

Fenglu Hong
Department of Computer Science
Stanford University
hfenglu@stanford.edu

Han Zhang
Department of Computer Science
Stanford University
hzhang20@stanford.edu

Abstract

Question answering is an interesting NLP task, as it provides a measurement for how well the model can understand and reason about the written human language. This project aims to produce a question answering system that works well on the Stanford Question Answering Dataset 2.0 (SQuAD 2.0). Based off a Bidirectional Attention Flow model, we integrate character-level embedding and self-attention mechanism, along with GRU layers, to boost the model's performance. After hyperparameter search, our best model achieves an F1 Score of **63.408** and a EM Score of **60.456** on CS224N internal test set of SQuAD 2.0.

1 Introduction

Question Answering has been a popular research topic in the field of Natural Language Processing. Given a paragraph (also called a context) and a question, the goal of close-formed question answering is to select a span from the paragraph that answers the question correctly, or indicate that the question cannot be answered using the provided paragraph. Common applications of the Question Answering systems include enabling search engines to provide concise answers for questions searched and developing smart virtual assistants that answer user questions.

In this project, we focus on Stanford Question Answering Dataset 2.0 (SQuAD 2.0) [1], a large-scale dataset for reading comprehension and close-formed question answering. Including both answerable and unanswerable questions, SQuAD 2.0 presents a challenging reading comprehension task. Based on a slight variation of the BiDAF model[2], we investigate the effectiveness of a series of techniques, including character-level embedding, self-matching attention, answer pointer, and transformer blocks, and find that the best performance is achieved with the addition of character-level embedding and self-matching attention, and the replacement of bidirectional long short-term memory (LSTM) layers with bidirectional Gated Recurrent Unit (GRU). We perform hyperparameter search and train final model using the best model architecture and hyperparameters. We also conduct error analysis on the final model to understand its strengths and limitations.

2 Related Work

There exists a large body of work on reading comprehension and question answering. According to the official SQuAD 2.0 leaderboard [3] as of March 2021, most popular models with high performance use pretrained contextual representations (such as BERT [4] and ELMO [5]) and ensemble methods. Specifically, BERT is a bidirectional Transformer encoder pretrained on a large corpus of text with two objectives: masked language model and next sentence prediction. The Transformer [6] Encoder block applies key-query-value attention and multi-head attention for fast, non-recurrent encoding, and utilizes tricks, such as residual connections and layer normalization to speed up training. While such pre-trained language models improve the performance on SQuAD significantly, they are expensive to train.

On the contrary, the BiDAF model is built upon GloVe embedding and the rest of the model needs to be trained from scratch on the training data. The BiDAF model uses recurrent neural networks, such as LSTM, to capture the sequential inputs and scan the context. It also applies query-to-context and context-to-query attention on the learned contextual embedding, to obtain query-aware context representations. These representations can then be used in downstream tasks, such as modeling the probability distribution over the context for answer prediction. However, since recurrent networks can only memorize limited context in practice, the BiDAF model does not have direct way to aggregate information from the entire passage to infer the answer. To address this issue, R-Net [7] proposes a self-matching mechanism that dynamically improves context representations with evidence from the whole passage. The Match-LSTM and Answer Pointer model [8] is another model trained from scratch for question-answering that builds upon LSTM layers and attention mechanism. One of its main differences from the BiDAF model is that instead of predicting start and end position independently, the answer is predicted in an Answer Pointer layer that uses attention mechanism to predict the probability distribution of the end position conditioned on that of the start position.

3 Approach

On top of the provided baseline BiDAF model, we add support for character-leveling embedding, self-attention mechanisms, answer pointer, and transformer block hoping to improve the model's performance in question answering. We code all added features by ourselves, and they can be enabled or disabled independently.

Baseline. The baseline model we use is based on the Bidirectional Attention Flow (BiDAF) model without character-level embedding. The implementation is provided in the default IID SQuAD final project start code Github repository[9].

Character-level Embedding. Besides GloVe word embedding, we implemented character-level embeddings for each word as described in the original BiDAF paper.

First, we compute the character embeddings for all characters in each word and store them in a matrix $C \in \mathbb{R}^{L_c \times H_c}$, where L_c is the maximum number of characters in a token and H_c is the dimension of the character embedding. Then we apply Convolutional Neural Networks (CNN). Each CNN filter converts C into a vector $v_c \in \mathbb{R}^{L_c}$, then v_c is max-pooled over the entire width to produce a single output $\in \mathbb{R}$. Concatenate the output from N_c filters, and we get our fixed-size character-level embedding for each input word.

For each word, we concatenate the corresponding GloVe word embedding $\in \mathbb{R}^{H_w}$ with its character-level word embedding $\in \mathbb{R}^{N_c}$ to get the combined word embedding $\in \mathbb{R}^{H_w+N_c}$, and pass it to the downstream highway network as in the baseline model.

Self-attention. Self-attention mechanism is widely successful on many NLP tasks. We implement a self-attention layer similar to the Self-Matching Attention in [7] after the attention layer in the baseline model. The previous bidirectional attention flow layer output question-aware representations $\{g_t\}_{t=1}^N$. We opt to pass these representations directly to the self-attention layer, without going through another layer of recurrent network. For each context location $t \in \{1, \ldots, N\}$, the self-attention layer produces an attention-pooling vector c_t of the whole context as follows:

$$
\begin{aligned}
s_j^t &= v^T \tanh(W_1 g_j + W_2 g_t) \\
a_i^t &= \exp(s_i^t) / \textstyle\sum_{j=1}^N exp(s_j^t) \\
c_t &= \textstyle\sum_{i=1}^N a_i^t v_i
\end{aligned}
$$

where the dimension of v (self attention dimensionality) is a hyperparameter. An additional gate is applied to $[g_t; c_t]$ as follows:

$$g_t = \text{sigmoid}(W_g[g_t; c_t])$$
$$[g_t; c_t]^* = g_t \odot [g_t; c_t]$$

$[g_t; c_t]^* \in \mathbb{R}^{16H}$ is then fed into the modeling layer as in the baseline.

Transformer Block. Adapting ideas from the Transformer Encoder Block, we further implement a more complex self-attention block, with the use of layer normalization, residual connection, self-matching attention sublayers, and feed-forward sublayers.

Specifically, each transformer block is a stack of the following building blocks: Layer Normalization 1 $\rightarrow$ Self-Matching Attention $\rightarrow$ Layer Normalization 2 $\rightarrow$ Feed-Forward Layer. The Feed-Forward layer has the following structure: Linear(H, 4H) $\rightarrow$ ReLU $\rightarrow$ Linear(4H, H) $\rightarrow$ Dropout.

We apply two residual connections as follows:

- $x = x + \text{Self-Attention}(\text{LayerNorm1}(x))$
- $x = x + \text{Feed-Forward}(\text{LayerNorm2}(x))$

Answer Pointer. In the baseline model, the start and end positions for the answer span are predicted independently. We implement an alternative output layers with answer pointer which predicts the probability distribution of the start position and the distribution of the end position conditioned on the start position. The answer pointer layer is implemented following the Boundary Model introduced in *Machine Comprehension Using Match-LSTM and Answer Pointer* [8].

Let $M \in \mathbb{R}^{N \times 2H}$ be the output of the modeling layer. The answer pointer layer takes M as input, and runs for two timesteps: first predicts the start position distribution and then predicts the end position. In the first timestep, an attention weight vector $p^{\text{start}} \in \mathbb{R}^N$, where p_i^{start} represents the predicted probability that the answer starts at position i in the context paragraph, is calculated as follows:

$$F_0 = \tanh(VM^\top + (W^a h_0^a + b^a) \otimes e_N)$$
$$p^{\text{start}} = \text{softmax}(v^\top F_0 + c \otimes e_N)$$

where $V \in \mathbb{R}^{H \times 2H}$, $W^a \in \mathbb{R}^{H \times H}$, b^a, $v \in \mathbb{R}^H$ and $c \in \mathbb{R}$ are learnable parameters, h_0^a is initialized to a zero vector $\in \mathbb{R}^H$, and the outer product $(c \otimes e_N)$ means repeating scalar c for N times to form a vector $\in \mathbb{R}^N$.

Then we calculate $h_1^a = \text{LSTMCell}(M^\top p^{\text{start}\top}, h_0^a)$ and plug h_1^a in the formulas above in place of h_0^1 to calculate F_1 and p^{end}. p^{start} and p^{end} are used to calculate loss or make predictions the same way as in the baseline model. The prediction of the end position conditions on the prediction of the start position since p^{start} is used to calculate the input to the LSTMCell.

4 Experiments

Data. We train and evaluate our model on the Stanford Question Answering Dataset 2.0 (SQuAD 2.0). The training set contains 129,941 examples. The dev set and test set are obtained by randomly splitting the official dev set, and contain 6078 and 5915 exapmles respectively.

Evaluation method. We evaluate model performance using Exact Match (EM) and F1 score metrics on the dev set, and the final model is also evaluated on the test set. For both metrics, the predicted answer is evaluated against 4 human answers (same or different) for each question, to allow for some flexibility in picking the answers. The highest score among the four is recorded. We also summarize different types of errors made by the model.

Experimental details. First, we examine the performance of different model architectures in the model selection stage. Then, we conduct hyperparameter search on the best model found. Lastly, we train the final model using the most promising hyperparameters and evaluate it on the dev and test set. We also carry out ablation study for analysing our model's performance.

I. Model selection. Table 1 lists the set of experiments that we run for model selection. Each model is trained for 10 epochs on the entire training dataset, with the same hyperparameter setting as the default setting provided in the codebase, except for the batch-sizes, which are included in Table 1.

In the listed experiments, character-level word embedding (Char-Emb.) has character embedding dimension of 64, and uses 100 Conv2d filters with width of 5 and height of 64. GloVe word vectors have a dimension of 300 and are fixed during training. Each final word vector has dimension of 400 (concatenating GloVe vector and character-level representation). Self-attention layer (Self-Attn.) has attention dimensionality 100. For models with transformer blocks (Trans-Blck.), we use a stack of two blocks between the BiDAF Attention layer and the modeling layer. We also experiment with replacing all LSTM layers in the baseline model with GRU (GRU). Dropout is applied on the output of embedding layer and RNN encoder (LSTM/GRU) and the input to BiDAFAttention and Answer Pointer layer. A dropout rate of 0.2 is used. The models are trained with the Adadelta optimizer [10] with a fixed learning rate of 0.5.

Model	RNN	Batch-size	Dev EM	Dev F1
Baseline	LSTM	64	54.02	57.27
w/ Char-Emb.	LSTM	64	57.3	60.51
w/ Char-Emb.	GRU	64	58.66	61.94
w/ Char-Emb. + Self-Attn.	LSTM	32	57.7	60.92
w/ Char-Emb. + Self-Attn.	GRU	32	**58.83**	**62.12**
w/ Char-Emb. + Ans-Ptr.	LSTM	32	58.34	61.56
w/ Char-Emb. + Ans-Ptr.	GRU	32	56.41	59.64
w/ Char-Emb. + Trans-Blck.	LSTM	32	56.49	59.60

Table 1: Model selection experiment results

II. Hyperparameter search. The best model obtained from the previous set of experiments is a model with character-level embedding, self-attention and GRU enabled. We experiment with different values of hyparameters and optimizer choices with this model architecture. In each experiment, we train a model on the entire training dataset for 10 epochs. The same embedding dimensionality and attention dimensionality are used as mentioned in Model selection section. Hyperparameter search details are elaborated below, and model's performances on F1 metrics are demonstrated in Figure 1.

i. learning rates We experiment with three learning rates: fixed learning rate of 0.1, fixed learning rate of 0.5 (the default), and initial learning rate of 0.5 that decays over time (at epoch i, the decayed learning rate is $0.5 * 0.95^i$). The results show that learning rate of 0.5 with decay outperforms the fixed learning rate of 0.5, which outperforms the fixed learning rate of 0.1.

ii. optimizer. We compare the performance of using the Adadelta optimizer (the default) and the Adam optimizer [11], both with a fixed learning rate of 0.5. Results show that Adadelta performs much better than Adam, which reduces loss by very little and incurred no change in F1 and EM score over the training duration. Since Adam optimizer uses momentum for gradient updates and generally the updates are slower and more stable, it is possible that a larger learning rate could help it run better.

iii. dropout rate. We compare the performance of using dropout rate of 0.1 and 0.2 (the default). The model trained with dropout rate of 0.1 achieves higher F1 and EM score than the model with dropout rate of 0.2, especially in early training phrase.

III. Final model and training scheme. Based on the results of model selection and hyperparameter search, a model with character-level embedding, self-attention and GRU layers is optimal, and the training scheme should use a dropout rate of 0.1, a decaying learning rate that starts at 0.5, and the Adadelta optimizer. We will refer to this model as *final model*. After its performance on dev set plateaus, we experiment with finetuning GloVe embedding.

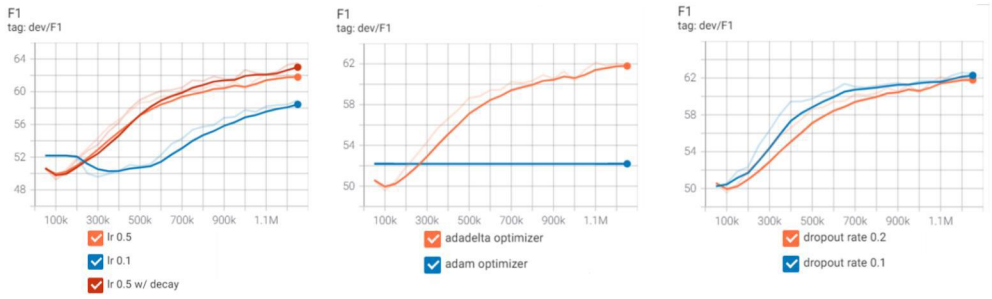

Figure 1: Hyperparameter searching results

For ablation study, we train a model with only character-level embedding and GRU layers under the same condition. Final results are included in Table 3.

Results and Quantitative analysis. Our final model quickly achieves the best F1 and EM scores on the dev set at the 11^{th} epoch. After that, the loss on development set increases and F1 and EM scores stop improving. The training is manually terminated at the 18^{th} epoch, when the model shows no sign of further improvement. We continue to train the model starting from the best checkpoint while finetuning GloVe embedding with a learning rate of 0.1. With limited experiments, it is not effective and leads to worse performance, so we forgo this direction.

We conduct a quantitative error analysis on the final model's performance, as summarized in Table 2. Compared to the performance on training set, the model makes considerably more mistakes on predicting answers for unanswerable questions. Further analysis reveals that some predicted answers for unanswerable questions have low probability scores, indicating relatively low confidence in the prediction. We set 0.14 to be the lower bound for the probability score required for any non-"N/A" prediction, and also require its probability score to exceed the "N/A" prediction's score by 0.04. Adding these thresholds increases the F1 and EM scores on the development set by around 2 points each, though the improvement on the test set is smaller. The performance after applying the thresholds is included in Table 3.

From the listed performance, we can tell that character-level embedding and self-attention mechanism both make contributions to the improvement of model's performance. Character-level embedding improves the baseline performance to a larger extent than self-attention. We will discuss their effects qualitatively in the following section.

Dataset	EM	FP	FN	Gold. $\in$ Pred.	Pred. $\in$ Gold.	Other
Train	73.70	04.76	05.72	05.00	06.73	04.09
Dev	60.39	19.95	09.39	02.71	02.13	05.43

Table 2: Error distribution for the final model. FP is the error of predicting an answer for unanswerable question; FN is the error of predicting 'N/A' for answerable question; Gold. $\in$ Pred. is the error of predicting an answer containing any gold answer but is longer; Pred. $\in$ Gold. is the error of predicting an answer contained by any gold answer but is shorter.

	Dev EM	Dev F1	Dev AvNA	Test EM	Test F1
Baseline	57.13	60.36	67.33	N/A	N/A
Char-Emb.	59.15	62.54	69.13	N/A	N/A
Char-Emb. w/ threshold	61.44	63.92	68.19	N/A	N/A
Final model	60.43	63.97	**70.66**	59.53	63.27
Final model w/ threshold	**62.41**	**65.08**	69.74	**60.46**	**63.41**

Table 3: Models' best performance on dev and test set

5 Analysis

As we can tell from previous analysis, distinguishing between answerable and unanswerable questions is a main difficulty for the models. As the baseline model achieves 67.33 on the Answer vs. No

Answer (AvNA) metric, a model with character-level embedding increases the score to 69.13 (before applying the thresholds), while the addition of self-attention further boosts it to 70.66. Intuitively, self-attention allows a model to attend to longer-range relationships between parts of the context and question, and find answers that are not in close proximity to the related signal words. Figure 2 shows an example, where the model with self-attention succeeds in finding the answer "Adriatic", that is 10 words away from the key word "Dyrrachium", while other models without self-attention fail. However, without a robust logical reasoning, this long-range attention is not always beneficial and can trick the model into selecting seemingly related answers that appear somewhere near the key words. An example is shown in Figure 3.

- **Question:** Where was Dyrrachium located?
- **Context:** The further decline of Byzantine state-of-affairs paved the road to a third attack in 1185, when a large Norman army invaded Dyrrachium, owing to the betrayal of high Byzantine officials. Some time later, Dyrrachium—one of the most important naval bases of the Adriatic—fell again to Byzantine hands.
- **Answer:** the Adriatic
- **Prediction:** Adriatic

Figure 2: The baseline model and the model with only Char-Emb. do not predict an answer, while the model with Char-Emb. and Self-Attn. predicts the correct answer.

- **Question:** Who married Cnut the Great?
- **Context:** The Normans were in contact with England from an early date. Not only were their original Viking brethren still ravaging the English coasts, they occupied most of the important ports opposite England across the English Channel. This relationship eventually produced closer ties of blood through the marriage of Emma, sister of Duke Richard II of Normandy, and King Ethelred II of England. Because of this, Ethelred fled to Normandy in 1013, when he was forced from his kingdom by Sweyn Forkbeard. His stay in Normandy (until 1016) influenced him and his sons by Emma, who stayed in Normandy after Cnut the Great's conquest of the isle.
- **Answer:** N/A
- **Prediction:** Emma

Figure 3: The model with only Char-Emb. predicts "N/A" correctly, while the model with Char-Emb. and Self-Attn. predicts a wrong answer.

6 Conclusion

In this project, we find that character-level embedding and self-attention both improve the the baseline BiDAF model's performance on question answering task. We also explore the effect of modifying the hyperparameter space and the use of GRU layers towards the models' learning curves. Our final model with both techniques and GRU layers achieves an F1 Score of **63.408** and a EM Score of **60.456** on the test set. We analyze the types of errors made by our model, and examine the pros and cons of self-attention. While self-attention is helpful in considering long-range relationships, more robust logical reasoning is required for the model to unequivocally say no to unanswerable questions.

7 Future work

The model's performance could possibly be further improved through more extensive hyperparameter search, on hyperparameters such as hidden layer dimension, self attention dimension and the use of L2 regularization. Also, through our experiments we find that setting the probability threshold for reporting Anwser/No Answer helps the model's performance on dev set. Currently, the threshold is set by inspection of data and trial and error, in the future we can devise a more systematic way of finding a threshold function that fits the data better.

References

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

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[3] `https://paperswithcode.com/sota/question-answering-on-squad20`.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, page arXiv:1810.04805, October 2018.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762, June 2017.

[7] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[8] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer, 2016.

[9] Mina Lee. squad. `https://github.com/minggg/squad`, 2019.

[10] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv e-prints*, page arXiv:1212.5701, December 2012.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.