Final Report for CS224N Default Project

Zhihan Jiang
Stanford University
Stanford, CA 94305
zhihanj@stanford.edu

Wensi Yin
Stanford University
Stanford, CA 94305
wsyin@stanford.edu

Abstract

Machine comprehension is gaining popularity over recent years and is fueled by the creation of many large-scale datasets. The Stanford Question Answering Dataset (SQuAD) 2.0 dataset [1], which contains both answerable and unanswerable questions, is especially challenging and high-quality. In this milestone, we present our neural network model which incorporates ideas from some high-performing SQuAD or machine reading comprehension models such as BiDAF [2], QANet [3], DrQA [4] and R-Net [5]. Our model achieves F1 score of 63.66% and EM score of 60.42% on the test set. The ensemble model achieves F1 score of 64.70% and EM score of 61.83% on the test set.

1 Introduction

Machine comprehension, or the ability to read the context and answer questions about it, has been a challenging task, since it requires both understanding about natural language and knowledge about the world. SQuAD 2.0 [1] is one of the datasets that leads to a huge advancement in the area of machine comprehension. It consists of 150,000 questions posted by crowd-workers, each of which is relevant to a certain passage on Wikipedia. Different from SQuAD 1.0 [6], nearly half of the questions are unanswerable using the given context. This further enhances the difficulty, since the model must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering.

In this project, on top of the given baseline, we propose an end-to-end deep learning model which combines ideas from some of the best performing SQuAD models. It consists of a multi-level embedding layer which includes word embedding, character embedding and linguistic feature embedding, a transformer-like encoder block for embedding encoder layer, a bi-directional attention layer between the query and context representations and a self-matching-attention layer to refine the BiDAF attention. We report our results in section 4, and we also provide a thorough error analysis towards the model performance and suggest several directions for further research.

2 Related work

In this section, we discuss several architectures and attention mechanisms that have been found to perform well on the question answering task, and have inspired our work.

BiDAF Bi-Directional Attention Flow (BiDAF) [2] network is a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. In particular, The bi-directional attention layer combines the context-to-question attention and the question-to-context attention, which boosts the performance on SQuAD dataset. We use the BiDAF model as our baseline.

QANet QANet [3] proposes a non-recurrent network to speedup the training process but achieve same level of performance as other recurrent models.

DrQA In the work [4], extra linguistic features are added to the embedding vector to help converge and achieve better performance.

R-Net Self-matching attention layer is proposed in [5] to concatenate the representation with the scaled dot-product attention of context against itself, so that the model can effectively encodes information from the whole passage.

3 Approach

The baseline model is the provided model in the starter code which is built based on BiDAF [2]. The baseline model differs from the original BiDAF in that it does not use a character-level embedding layer. The architecture of our model is shown in Figure 1, and we will cover the details of each layer in the following sections.

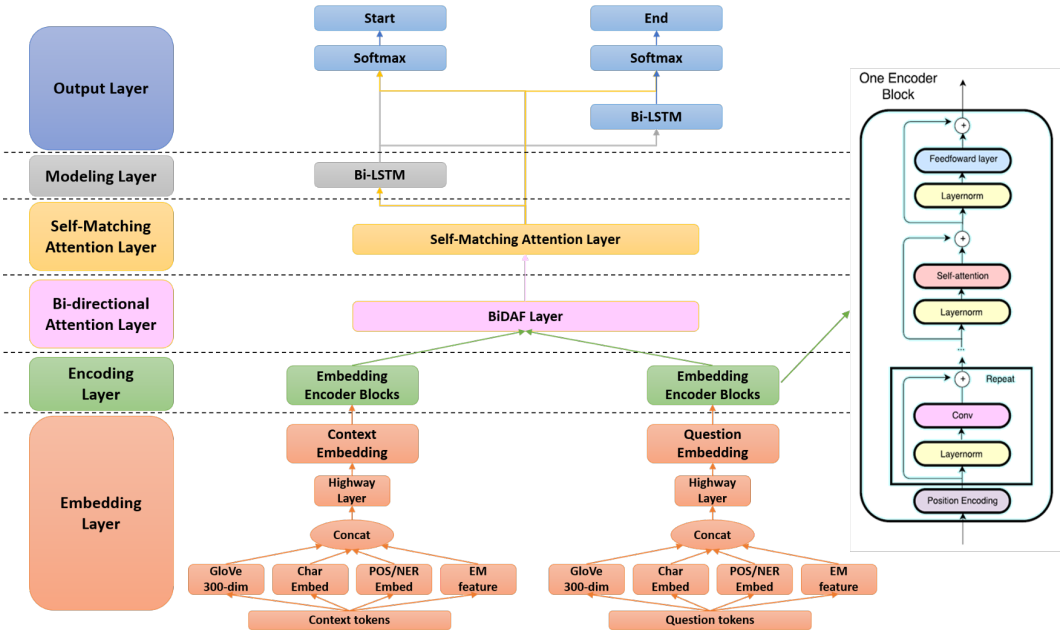

Figure 1: Model architecture of our model.

3.1 Embedding layer

Our embedding vectors of both the context and the question tokens consist of several parts as illustrated below:

- **Pretrained GloVe vectors**: we map each individual word token using the pre-trained GloVE word vector. The word embedding will not change during the training process.
- **Character-based embedding**: we also compute the character-level embedding [7] by first mapping each character of a word token using a character embedding vector, then going through a convolutional network to form word representation.
- **Token symbolic features**: following the idea in [4], we construct 2 additional embeddings using the part-of-speech (POS) and name-entity recognization (NER) of each word. We use the NLTK package [8] to extract and pre-compute these properties on the original context and questions.
- **Exact match (EM) features**: we also use the exact-match feature adopted from [4], which simply adds three binary features to the context embedding vector, indicating whether a token in the context can be exactly matched to one token in the question, either in its original, lowercase or lemma form. The corresponding position in the question embedding vector is padded with 0. This feature turns out to be very helpful.

We concatenate all the embedding and features mentioned above, and run them through a highway layer [9] to generate the final embedding output.

3.2 Embedding encoder layer

For the encoder layer, we adopt the design from QANet [3]. which replaces traditional recurrent encoder with transformer-like encoder block. As shown in figure 1, one encoder block contains the following parts:

- **One position encoding layer [10]**: Given the input, the positional encoding is added which allows the model to easily learn to attend by relative positions.

$$PE_{pos,2i} = sin(pos/10000^{2i/hid-dim})$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/hid-dim})$$

- **Four depthwise-separable convolution layers [11]**: depthwise-separable convolution is used in consideration of memory efficiency. We use 7 for kernel size and 200 for output channel.
- **One self-attention layer [10]**: we use the standard multihead-attention as described in [10]. The number of heads we use is 8.
- **One feed-forward layer**: this is simply a fully-connected layer to generate the output of the encoder layer.

Each of these layer is also accompanied with a layer-normalization layer [12] before it and a stochastic depth dropout layer [13] after it. The total number of blocks is 1. To make later similarity-based attention layers more effecitve, we share the weights of embedding encoder between context and question.

3.3 Attention layer

For the attention layer, we adopt the Bi-directional Attention Flow layer introduced in [2] to combine the context representation with the question representation. Given context hidden states $c_1, ..., c_N$ and question hidden states $q_1, ..., q_N$, we can compute the similarity between context token i and question token j, S_{ij}, C2Q attention for context token i, a_i and Q2C attention for question token i, b_i based on following formula:

$$S_{i,j} = w_{sim}^T[c_i; q_j; c_i \circ q_j]$$

$$\bar{S}_{i,:} = softmax(S_{i,:}) \quad \bar{\bar{S}}_{i,:} = softmax(\bar{S}_{:,j}) \quad S^{'} = \bar{S}\bar{\bar{S}}^T$$

$$a_i = \sum_{j=1}^{M} \bar{S}_{i,j} q_j \quad b_i = \sum_{j=1}^{N} S^{'}_{i,j} c_j$$

Finally, we concatenate the context hidden state c_i, the C2Q attention a_i and the Q2C attention b_i to output $g_i = [c_i; a_i; c_i \circ a_i; c_i \circ b_i]$

3.4 Self-matching attention layer

Inspired by R-Net[5], we use a self-matching attention layer to match the question-aware context representation G against itself, so that each context word is aware of the information from the entire context.

In order to compute the self-matching attention vectors, we first project attention vector G to query vector $Q \in \mathbb{R}^{N \times d}$ and key vector $K \in \mathbb{R}^{N \times d}$ gated by ReLU.

$$Q = ReLU(W_Q G) \quad K = ReLU(W_K G)$$

Then we compute scaled dot-product attention scores s using query Q and key K, and compute self-attention vector v_t based on the score S. While original R-Net implementation use additive attention

here, we find that scaled dot-product attention can avoid heavy computation and also maintain good performance.

$$S = \frac{QK^T}{\sqrt{d}} \quad a_t = softmax(S_{t:})$$

$$v_t = \sum_{i=1}^{N} a_{ti} G_i$$

Finally, we concatenate the original question-aware context representation g_i along with new self-attention vector, and let them go through a bi-directional LSTM layer to form our final self-attended context representation $\tilde{g}_i$.

3.5 Modeling layer

Following the BiDAF paper [2], we use two layers of bi-directional LSTM for the modeling layer in order to refine the sequence of vectors after the self-matching attention layers.

3.6 Output layer

The output layer generates a vector of probabilities with a length equal to the context sentence. We adopt the same technique as in [2], which takes as input the attention layer outputs and modeling layer outputs, then produce p_{start} and p_{end} using following formula

$$p_{start} = softmax(W_{start}[G; M]) \quad p_{end} = softmax(W_{end}[G; M^{'}])$$

where W_{start} and W_{end} are trainable parameters and $M^{'}$ is the output of a bidirectional LSTM to the modeling layer outputs.

3.7 Loss function and prediction methods

The loss function is the sum of negative log-likelihood (NLL) loss for the start and end locations. At test time, we choose the pair (i, j) of indices that maximizes $p_{start}(i) \cdot p_{end}(j)$ subject to $i \leq j$ and $j - i + 1 \leq L_{max}$, where L_{max} is a hyperparameter which specifies the maximum length of a predicted answer.

To predict no-answer questions, we follow [14], in which a OOV token is prepended to the beginning of each sequence. At prediction time, if $p_{start}(0) \cdot p_{end}(0)$ is greater than any predicted answer span, the model predicts no-answer.

4 Experiments

4.1 Data & evaluation metrics

We use 224n-customized version of SQuAD 2.0 dataset [1], which is a machine learning comprehension on Wikipedia articles with more than 100000 both answerable and unanswerable questions. We split the dataset roughly by 90%, 5% and 5% for train, validation and test set.

To better understand our dataset, we examine the data by plotting the distribution of the context length, question length, answer length and question types of training dataset (see Figure 2). The figure shows that the majority number of questions have context length less than 400 and question length less than 30. Therefore we set maximum length limit for context to be 400 and for question to be 30, which leads to faster training speed but minimal performance loss. It is also shown in the figure that "what" question type accounts for the largest proportion in the training dataset and "why" question type accounts for the smallest proportion.

Our evaluation metrics are F1 score and exact match score. The human performance on this dataset is 89.452 (F1) and 86.381 (EM).

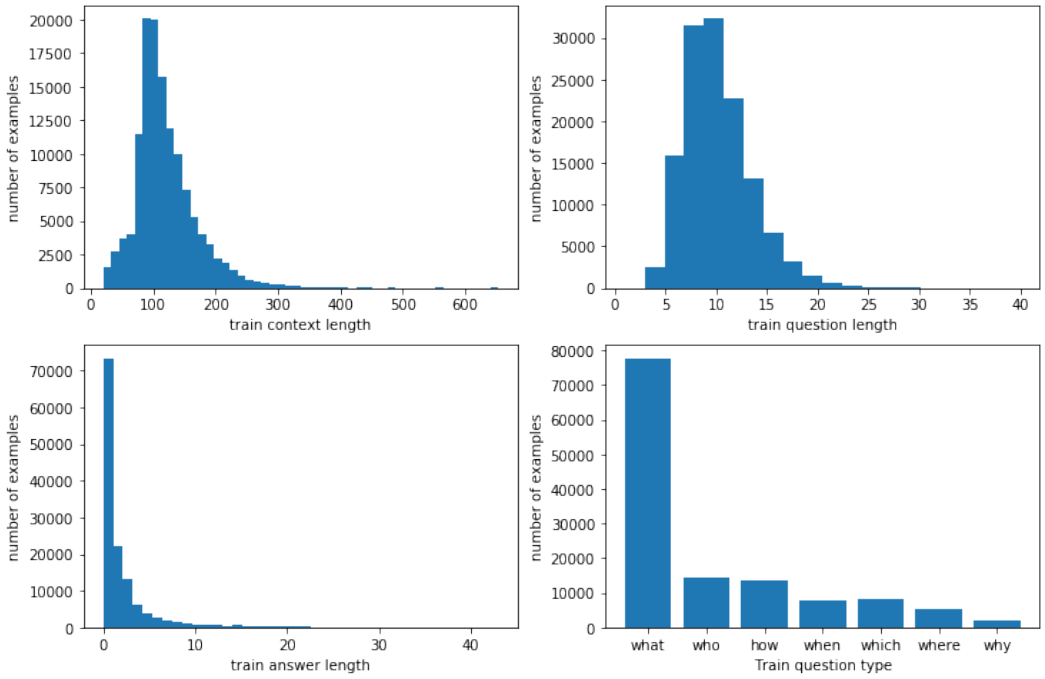

Figure 2: Statistics of the SQuAD 2.0 training set

4.2 Experiment details

For our embedding vetor, we use 300-dimensional GloVe pre-trained vectors for word embeddings, 300-dimensional character embedding, 16-dimensional POS embedding, 8-dimensional NER embedding and 3-dimensional EM features. For our charCNN, we use a convolution kernel size of 5. Hidden size in our model is set to 100. We train our model in a batch size of 64 on a single NVIDIA GPU (P100) for 15 epochs using the following parameters: learning rate: 0.5, dropout layer probability: 0.2. We use Adadelta [15] optimizer, and we also maintain moving averages of all the trainable parameters with an exponential decay rate of 0.999. To avoid gradient exploding problems in our LSTM layers, we employ gradient clipping with a maximum gradient norm of 5. We early-stop the model and save the checkpoint for the model that achieves the best results, and the training process takes around 6 hours to complete 15 epochs.

To further improve the model performance, we build an ensemble model using 6 models of the same architecture but different random seeds for initialization. We calculate the average of the probabilities p_{start} and p_{end} associated with each position from the 6 models, and compute the start and end positions accordingly. This boosts our F1 score by 1.04 and EM by 1.39 on the test set. We also try majority vote, but the performance is not as good.

4.3 Results and effects of components

We submit our result on the test non-PCE leaderboard. A single model of our implementation achieves **F1 score of 63.66** and **EM score of 60.42** on the test set. An ensemble of six models with different initialization achieves **F1 score of 64.70** and **EM score of 61.83** on the test set.

The cumulative results of our model on dev set can be found in Table 1. We can see that the char embedding, QANet encoder blocks, exact match feature and self-matching attention are very helpful to our model, while POS/NER features alone are of little help. Intuitively, character embedding is helpful because it will let our model learn those tokens without pretrained word embeddings. Exact match feature is also very helpful because it "directs" our model to the positions in the context where question tokens are shown, thus highly likely to be the answer span. POS/NER features + exact match work while POS/NER by themselves don't work might be because POS/NER will help the

model figure out which exact match will be helpful (the fact that words like "is", "the", etc. in context might not be very useful). For the QANet encoder block part, we capture both local interaction and global interaction between pair of words within context and question while the gradient flows more smoothly. The self-macthing attention layer also helps the model fuse relevant information from the entire context into each context word thus gaining deeper understanding and generating better answer.

Model	F1	EM
Baseline BiDAF (w/o char emb)	60.61	57.52
Add char-emb	62.66	59.35
Add POS/NER features	62.78	59.10
Add QANet Encoder block	63.90	60.28
Add exact match (EM) feature	65.44	62.31
Add self-matching attention	**66.88**	**63.75**
Ensemble	**68.18**	**65.35**

Table 1: Cumulative model results on dev set

Also, it's worth mentioning that after adding exact match feature into our model, the training process is significantly accelerated. Before adding it, it approximately took 20 epochs to achieve the best performing model, while it only took around 13 epochs to achieve the best performing model with the EM feature. This means small but proper feature engineering will help the training process of our model. After we add self-matching attention layer, the training process is further accelerated and now it takes around 10 epochs to achieve the best performing model.

5 Analysis

5.1 Quantitative error analysis

We perform error analysis for our model on dev set based on the question type and answer length. Figure 3 shows the F1/EM/AvNA scores for different question types and Figure 4 shows the F1/EM/AvNA scores for different context/question/answer lengths.

For the question type breakdown, it can be seen from Figure 3 that AvNA scores of different question types are similar, which means question type doesn't affect much the model performance of this pseudo binary classification task. For the EM/F1 score, we notice that our model is better at answering "who", "when" and "where" questions, even though questions with these question types only make up 20% of training set. That's probably because these kind of questions are straightforward. On the contrary, our model performs poorly on "why" question type. This could be explained by the fact that "why" question type requires a deeper level of understanding, as the relationship between context and question is complicated and requires logical reasoning. Also, there is only 1.5% "why" questions in the training set, which makes it even harder for our model to learn from experience.

For the context/question/answer length breakdown, we find that context length and question length do not have significant influence on our model performance. However, the model performance drops as the answer length becomes longer. This might be due to queries with long answers are often more complicated and require the model to characterize a longer term of dependency.

5.2 Qualitative error analysis

We also provide some concrete examples about how our model performs. In Figure 5, we see that the model is allured to pay attention to the content before "prize", but fails to link "the problem" (in the question) to "The P versus NP problem" (in the context), thus giving the wrong answer. This error might be due to the exact match (EM) feature, which focuses on the word that both occurs in the context and the question. In this case, the model could easily step in a trap that is deliberately designed by the human, where some of the question words match a certain part of the context, while the rest is irrelevant which makes the question unanswerable.

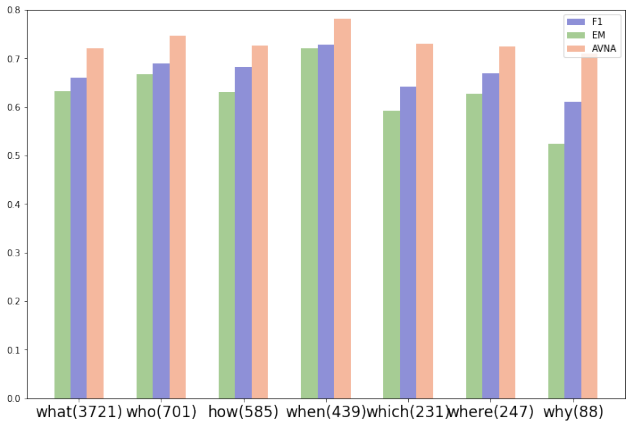

Figure 3: F1, EM and AvNA score of different question types

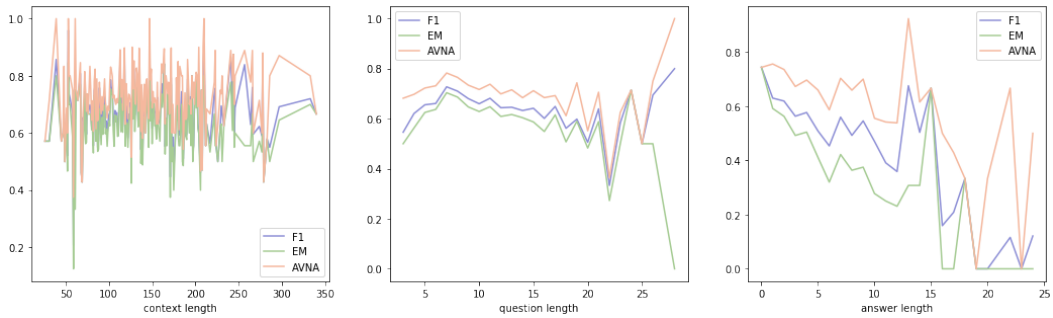

Figure 4: F1, EM and AvNA score w.r.t. context(left)/question(middle)/answer(right) length

Context: The question of whether P equals NP is one of the most important open questions in theoretical computer science because of the wide implications of a solution. If the answer is yes, many important problems can be shown to have more efficient solutions ... The P versus NP problem is one of the Millennium Prize Problems proposed by the Clay Mathematics Institute. There is a US $1,000,000 prize for resolving the problem.
Question: What was the prize for finding a solution to P=NP at the Alpha Prize Problems?
Answer:
Predict: US $1,000,000

Figure 5: Wrong prediction due to insufficient understanding of connections between words.

In Figure 6, the prediction result "Bannow Bay" is a place instead of a "country" as asked in the question. Since the the word "country" never appears in the context, the model has to infer that 1) "Bannow Bay" is not a country name, 2) "Irish culture" is a part of "Ireland", 3) "Bannow Bay" is a part of "Ireland". This could seem easy to the human, but the machine struggles at understanding the logical reasoning behind it, and chooses to predict the "Bannow Bay" without "thinking too much".

Context: The Normans had a profound effect on Irish culture and history after their invasion at Bannow Bay in 1169. Initially the Normans maintained a distinct culture and ethnicity. Yet, with time, they came to be subsumed into Irish culture to the point that it has been said that they became "more Irish than the Irish themselves." The Normans settled mostly in an area in the east of Ireland, later known as the Pale, and also built many fine castles and settlements, including Trim Castle and Dublin Castle. Both cultures intermixed, borrowing from each other's language, culture…
Question: What country did the Normans invade in 1169?
Answer: Ireland
Prediction: Bannow Bay

Figure 6: Wrong prediction due to lack of word interpretation and logic reasoning

5.3 Attention error analysis

We also plot the attention similarity matrix of the BiDAF layer for example showed in Figure 5. It can be seen that the model tries hard to find the connection of word "prize" in question to each word in context, but fails to link "the problem" (in the question) to "The P versus NP problem" (in the context), thus giving the wrong answer.

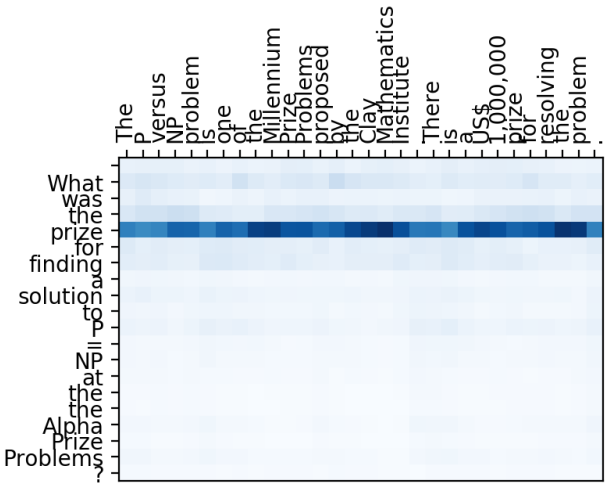

Figure 7: The BiDAF attention heatmap of example in figure 5.

5.4 Potential improvement

Since both quantitative error analysis and qualitative error analysis point to the problem - lacking the ability to understand sophisticated logic, a potential improvement for our model would be to use multi-turn reasoning model [16] to revisit the context, refine the answer and reason beyond the basic semantics meanings of the sentence.

6 Conclusion

In this project, we propose a model architecture to perform machine reading comprehension on SQuAD 2.0 dataset. Our approach combines ideas from some of the state-of-the-art models such as multiple-level embeddings, transformer-like encoder block, bi-directional attention flow and self-matching attention. Our ensemble model achieves results of 64.70% F1 score and 61.83% EM score on hidden test set.

Based on the error analysis results, we propose several directions for improvement. Building a more complicated attention layer might help our model pay attention to the correct answer span. Using multi-turn iterative reasoning method should also help our model improve the ability of logical reasoning as human. Due to non-PCE division constraint we are not allowed to use pre-trained embeddings, but pre-trained embeddings like BERT would definitely boost the performance of our model.

References

[1] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," *CoRR*, vol. abs/1806.03822, 2018.

[2] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *CoRR*, vol. abs/1611.01603, 2016.

[3] A. W. Yu, D. Dohan, M. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," *CoRR*, vol. abs/1804.09541, 2018.

[4] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," *CoRR*, vol. abs/1704.00051, 2017.

[5] W. Wang, N. Yang, F. Wei, B. Chang, and M. 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)*, vol. 1, pp. 189–198, 2017.

[6] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *CoRR*, vol. abs/1606.05250, 2016.

[7] X. Zhang and Y. LeCun, "Text understanding from scratch," *CoRR*, vol. abs/1502.01710, 2015.

[8] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[9] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, 2015.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[11] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[13] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," *CoRR*, vol. abs/1603.09382, 2016.

[14] O. Levy, M. Seo, E. Choi, and L. Zettlemoyer, "Zero-shot relation extraction via reading comprehension," *CoRR*, vol. abs/1706.04115, 2017.

[15] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[16] Y. Shen, P. Huang, J. Gao, and W. Chen, "Reasonet: Learning to stop reading in machine comprehension," *CoRR*, vol. abs/1609.05284, 2016.