CS224N Default Final Project Report: Question Answering on SQuAD 2.0

Chaonan Ye
Stanford University
yec0214@stanford.edu

Wanling Liu
Stanford University
liuwl@stanford.edu

Abstract

Machine reading comprehension has become a central task in natural language understanding. In this work, we aim to propose a competitive question answering system to approximate the true language understanding when deployed on the Stanford Question Answering Dataset (SQuAD 2.0). The final system is a weighted ensemble with Bidirectional Encoder Representations from Transformers (BERT), and Bi-Directional Attention Flow (BiDAF) networks extended with character-level embeddings and self-attention layers in two ways: a self-Matching Network introduced in R-Net and self-attention layers used in BiDAF++. Our experimental evaluations show that our current models produce well-calibrated confidence scores on SQuAD 2.0. Our work contributes to mainly two parts. Firstly, we combine the self-attention mechanism with the BiDAF baseline model. Secondly, we make an ensemble of BERT and the extended BiDAF models, yielding state-of-the-art results against baselines. Overall, our final model achieves scores of 77.707 EM and 80.275 F1 on SQuAD 2.0 on the dev set as well as 77.312 EM and 79.984 F1 on the test set, which is a large improvement from about 55 EM and 58 F1 of the starter code on dev set.

1 Introduction

Reading and comprehending human languages is a challenging task for machines, which requires the understanding of natural languages and the ability to do reasoning over various clues. Question answering (QA) is one of the popular problems in this field and has been actively researched in Natural Language Processing. QA has gained great significance and popularity since it has a wide range of applications, such as web search, e-learning, and interactive voice response. One of the most popular challenges for QA is SQuAD. Ever since SQuAD 2.0 has been released, there has been intense competition among research groups and individuals.

In this project, we focus on designing a question answering system that has promising performance on SQuAD 2.0. Our starter baseline model is a BiDAF variant model as described in the default project handout[1]. We extend the BiDAF baseline model mainly by adding a character-level embedding layer and two types of self-attention layers inspired by BiDAF++ and R-NET. We also change the original Long Short-Term Memory (LSTM) to Gated Recurrent Unit[2] (GRU), since they have similar accuracy but GRU is computationally cheaper. At last, we combine our extended BiDAF models with several BERT models to make an ensemble, further boosting the performance.

2 Related work

Recently, the tasks of machine comprehension (MC) and question answering (QA) have gained significant popularity within natural language processing. One of the key factors to achieve promising

[1]Default project handout: http://web.stanford.edu/class/cs224n/project/main.pdf

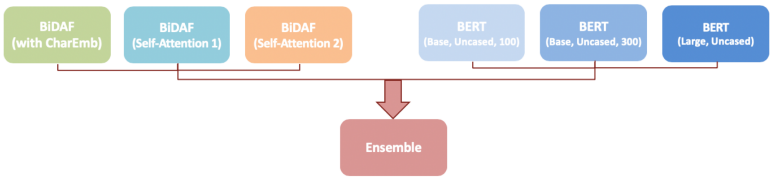

Figure 1: High-level architecture of the final ensemble model

results is the use of attention-based neural network [1], which has the ability to learn the "importance" distribution over the inputs. However, typically most methods use attention to focus on an only small portion of the context. Therefore, any works have introduced different ways to solve this problem. For example, Minjoon Seo, Aniruddha Kembhavi, et al introduces the Bi-Directional Attention Flow (BIDAF) network[14], a multi-stage hierarchical process. Another popular way to handle this issue is by using self-attention. It is an attention mechanism, relating to different positions of a single sequence to compute a representation of the sequence. It has been used successfully in conjunction with Recurrent neural networks(RNN) to solve the QA task [3],[6],[8],[10]. Additionally, Google has introduced Transformer in 2017, the first transduction model relying entirely on self-attention to compute representations of its input and output and gain distinguished achievements [16]. Inspired by these successes, we extend BiDAF starter code with self-attention layers to improve the performance.

Another key factor is to improve the representation of language. Recently, Google has proposed a new language representation model called BERT [5], which is designed to pre-train bidirectional representations conditioned on both left and right context. Combined with an additional output layer, BERT can be widely used in various tasks including QA. So, we combine BERT with several of our models to make an ensemble to improve the performance.

3 Approach

Our final model is a weighted ensemble with fine-tuned BERT models including BERT-Large-Uncased and BERT-Base-Uncased (BERT-Large model contains more parameters), and self-matching BiDAF networks with pretrained GloVe vectors[9]. The main architecture is shown in Figure 1.

Specifically, our first step is to replace LSTM with GRU to accelerate the computational speed, since GRU gives similar accuracy but faster speed (in our experiment, it can save around 2 minutes for each epoch compared with LSTM). To achieve better performance, we add a character-level embedding layer on the starter code of the default final project [15]. In addition, we add self-attention layers on the BiDAF starter model in several ways to further improve the performance of BiDAF. Finally, we ensemble fine-tuned BERTs and modified BiDAF networks with different weights to boost performance. The ensemble function is written by ourselves.

3.1 BiDAF Baseline Model

The BiDAF baseline model in the starter code is based on BiDAF[14](we call it BiDAF starter model here) but without character embedding layers. Therefore, it only consists of five layers: Word Embedding Layer, Contextual Embedding Layer, Attention Flow Layer, Modeling Layer, and Output Layer. We extend this starter model mainly in the following two ways to improve the performance.

3.1.1 Character-level embedding layer

Unlike the original BiDAF model[14], the implementation of the starter code does not include a character-level embedding layer. Following Kim et al.'s work[7], we extend the starter code with a character-level embedding layer using character-level Convolutional Neural Networks (CNN) to yield better results. The character embedding layer maps each word to a high-dimensional vector space. Let x represent a word and x_{char_emb} represent a reshaped character embedding for x after character vocabulary lookup. Then we apply 1D CNN:

$$x_{conv} = Conv1D(x_{char\_emb})$$

After that, we apply the ReLU function to x_{conv} and apply max-pooling across the window to obtain the final character embedding x_{conv_out}:

$$x_{conv\_out} = MaxPool(ReLU(x_{conv}))$$

Finally, we concatenate the character embedding x_{conv_out} with word embedding calculated via starter code to get the final embedding for a word x. The character embedding layer is written by ourselves.

3.1.2 Self-Attentions

As we mentioned in session 2, we extend BiDAF baseline model with self-attention due to the success of many works. We have tried two different types of self-attention layers: one is inspired by BiDAF++ [3]; the other is from R-net [8].

Self-Attention layers from BiDAF++

For addressing the challenges that come with training on document level data, BiDAF++ uses a layer of residual self-attention. The input of this self-attention layer is a query-aware context representation vector. First, the input is passed through a linear layer with ReLU activations. Then, it passes through a bi-directional GRU. After that, we apply attention mechanism between the context and itself as following:

Let h_i be the vector for context word i and n_c be the lengths of the context. Then compute attention between context word i and context word j as:

$$a_{ij} = w_1 * h_i + w_2 * h_j + w_3 * (h_i \odot h_j)$$

where w_1, w_2, and w_3 are learned vectors and $\odot$ is element-wise multiplication. In this case, we set $a_{ij} = -inf$ if $i = j$. Then, compute an attended vector c_i for each context token as:

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{n_c} e^{a_{ij}}}$$

$$c_i = \sum_{j=1}^{n_c} q_j p_{ij}$$

We also compute a context-to-context vector C_c:

$$m_i = max_{1 \leq j \leq n_q} a_{ij}$$

$$p_i = \frac{e^{m_i}}{\sum_{i=1}^{n_c} e^{m_i}}$$

$$C_c = \sum_{i=1}^{n_c} h_i * p_i$$

The final vector computed for each token is built concatenating h_i, c_i, $h_i \odot c_i$ and $q_c \odot c_i$. This layer is applied residually, so this output is additionally summed with the input.

We modified the self-attention layers based on reference from Allennlp [2].

Self-Matching Attention from R-NET

Inspired by the self-matching network consisting of a gated attention-based recurrent network and self-matching attention in R-NET [8], we add self-matching attention layers on the query-aware representations of context words given by BiDirectional Attention Flow layer. The self-matching attention mechanism would refine the representation by matching the passage against itself, which effectively encodes information from the whole passage.

Let v_t^P represent a passage representation dynamically incorporating aggregated matching information from the whole question. The self-attention layer is to solve the problem that v_t^P generated from the Question-to-Context Attention layer has limited knowledge of context. It allows the question-aware passage representation being matched against itself in order to collect evidence relevant to the current

[2] Allennlp: https://github.com/allenai/allennlp/blob/master/allennlp/models/reading_comprehension/dialog_qa.py

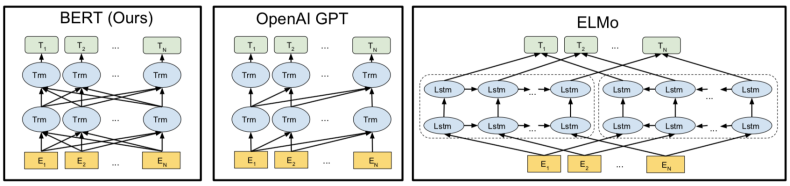

Figure 2: Difference between three language representation models[5].

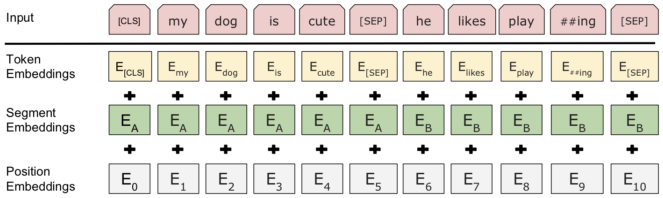

Figure 3: BERT input representation [5].

passage word and the question from the entire passage. After that, the self-matching attention layer encodes the evidence into passage representation h_t^P:

$$h_t^P = BiRNN(h_{t-1}^P, [v_t^p, c_t])$$

where $c_t = att(v^P, v_t^P)$ is an attention-pooling vector of the v^P:

$$s_t^j = v^T \tanh(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P)$$

$$a_i^t = \exp(s_i^t)/\Sigma_{i=1}^n \exp(s_j^t)$$

$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

Besides, an additional gate is applied to $[v_t^p, c_t]$ to control the input of RNN. The gate focuses on the relation between the question and current passage word.

The code for self-matching layers was modified by ourselves based on a GitHub repo [3].

3.2 BERT

Based on our token-level task, it is significant to consider both left and right context information in which the word appears in the text. Thus, we used a PCE (pre-trained contextual embeddings) model called BERT[5], which is a new language representation model and can be widely used in different tasks, as one of our baselines to improve the performance. Unlike ELMo[11] and OpenAI GPT[12] that use unidirectional language models, BERT uses a bidirectional Transformer to pre-train representations, which makes it condition on both left and right context in all layers (see Figure 2). The input embeddings of BERT are the sum of three kinds of embeddings: token embeddings, segment embeddings and position embeddings, which is shown in Figure 3.

The language model (LM) is pretrained by using two unsupervised prediction tasks: masked LM (in which it randomly masks some percentage of input tokens and then train the LM to predict those masked tokens) and next sentence prediction (given two sentences A and B, train the model to predict whether B is the next sentence of A). Masked LM is important for training deep bidirectional representation and next sentence prediction is helpful for tasks like question answering(QA) since the model needs to understand sentence relationship. Pre-training is essential since pre-trained word embeddings perform significantly better compared with learning embeddings from scratch. Finally, according to different tasks, parameters are fine-tuned for maximizing the log-probability of the correct labels.

We use the TensorFlow version of BERT model provided by Google [4], and fine-tune BERT-base-uncased model and BERT-large-uncased model. We then combine those BERT models with our extended BiDAF models to make an ensemble.

[3]RNET-Pytorch: https://github.com/hengruo/RNet-pytorch/blob/master/models.py

Model	EM	F1
BiDAF with CharEmb	58.192	61.389
BERT-Base-Uncased (100)	71.356	73.859
BERT-Base-Uncased (300)	73.462	76.604
BERT-Large-Uncased	75.502	78.496

Table 1: The performance of baseline models.

Model	EM	F1
SMA v1	55.369	58.626
SMA v2	55.402	58.623
SMA v3	**56.125**	**59.548**

Table 2: The performance of BiDAF with Self-Matching Attention (SMA) inspired by R-NET trained for 10 epochs (refer to 4.3.3 for explanation of three different versions of SMA).

4 Experiments

4.1 Data

We mainly focus on SQuAD 2.0 dataset [13], which contains around 150,000 questions in the format of <question, context, answer> triples. The dataset includes about 50% of the questions that can be answered using the provided paragraph, and the rest of the questions are not answerable using the given paragraph. The paragraphs are from Wikipedia, and the questions and answers were crowdsourced using Amazon Mechanical Turk.

4.2 Evaluation method

We use the evaluation metrics: Exact Match (EM) and F1 scores for the quantitative evaluation. We compared and analyzed our models using EM and F1. In addition, we found and analyzed some typical examples of our results for qualitative evaluation. For incorrect predictions, we further classified them into different error categories.

4.3 Experimental details

4.3.1 BiDAF

As a preparation step, we replaced LSTM with GRU. We compared the runtime and results of models using LSTM and GRU and found that GRU gave similar accuracy but faster speed. In our experiment, we found that it saved about 45 minutes for 30 epochs compared with using LSTM. Therefore, in our further experiments, we decided to use GRU instead of LSTM to accelerate the speed of computation.

4.3.2 Character-level embedding layer

We trained the BiDAF baseline model with a character-level embedding layer written by ourselves, using parameters given by the starter code[15]. We trained 30 epochs. For each epoch, it took 30 minutes to train.

4.3.3 Self-Attentions

To obtain the best performance of combining self-attention layers with the BiDAF baseline, we tried adding two self-attention layers separately in different places.

Self-Attention layers from BiDAF++

First, since the intuition for using self-attention was to effectively aggregate evidence from the whole passage to infer the answer, we added self-attention layers on the query-aware context outputted from Attention Flow Layer. Then, we passed the output of self-attention layers to Modeling Layer in order to refine the context vectors further. The experiment ran for 30 epochs and each training epoch took around 20 minutes.

The second try was adding the same self-attention layer after the Modeling Layer in BiDAF starter model. So, in this case, the input of the layer was the refined context representations after the attention layer. And the output was passed to the Output Layer to produce a vector of probabilities corresponding to each position in the context.

The last trial we did was similar with the second one. We took the sum of the same self-attention layer with the output of Modeling layers based on what we did in the second trial, then used the sum as input for the Output Layer.

Self-Matching attention layer from R-NET

We also tried to add self-matching attention layer in several ways. Firstly, we applied Self-Matching attention on query-aware representations of context words produced by the Attention Flow Layer of BiDAF (see Table 2: SMA v1 for the result in dev set). Secondly, we tried to apply self-matching attention after the modeling layer, and then replaced the BiDAF attention with self-matching attention for the output layer of BiDAF (see Table 2: SMA v2 for result in dev set) or combined self-matching attention with BiDAF attention (added them together as the final attention for the model, see Table 2: SMA v3 for result in dev set). We trained those models for 10 epochs, and the training time for each epoch is around 1 hour on GPU.

4.3.4 BERT

As for BERT, we trained the TensorFlow version of BERT-Base-Uncased (trained for about 6 hours on Azure GPU) and BERT-Large-Uncased (trained for about 1 hour on Google Cloud TPU) models provided by Google[4]. As for BERT-Base-Uncased model, our training batch size is 4, learning rate is 3×10^{-5}, number of training epochs is 2, maximum sequence length [4] is 100 or 300 and document stride [5] is 128. As for BERT-Large-Uncased model, our training batch size is 24, the learning rate is 3×10^{-5}, the number of training epochs is 2, the maximum sequence length is 384 and document stride is 128. We chose uncased models since question answering was not case sensitive so uncased models usually give better results.

4.3.5 Ensemble

Since there are many feasible ways to do ensembling, we tried two ways using different models and methodologies and found out that both models can boost the performance against all baselines.

Ensemble of One BiDAF + One BERT entended model

We first used predictions from our extended BiDAF models to "modify" the results of BERT-Large-Uncased model: for each question, if BiDAF's best prediction was the same as any of the top 20 predictions of the BERT model, then we assigned some weight (a hyperparameter) to the probability of this prediction in BiDAF and then added it to the probability of this prediction in BERT:

$$P_{overall} = weight * P_{BiDAF} + P_{BERT}$$

P was the new probability for those predictions, and other predictions' probabilities remained the same. At last, we chose one prediction with the highest probability in BERT top 20 predictions for each question. The best result was EM 75.683 and F1 78.552 on the dev set, and it was from ensemble of BiDAF starter model with character embeddings and BERT-Large-Uncased model, and the weight was 0.7. The result is shown in Table 4: Ensemble 1.

Ensemble of All BERTs + Multiple BiDAFs extended models

As we had 3 BERT models (BERT-Large-Uncased, BERT-Base-Uncased (100), BERT-Base-Uncased (300)), we first made an ensemble of all BERT models as a sub-ensemble to improve the model performance. More specifically, for each question-id in each BERT, we had 20 top candidate answers with probabilities in each BERT model (so the total was 3 * 20 = 60). Then for each question, we accumulated probabilities among all 60 answer candidates if they were the same answers. Finally, the answer to this question was picked by the highest accumulated probability.

Next, we combined this sub-ensemble with multiple models (3 BERTs + 3 BiDAFs) to generate our final model. The 3 BiDAF entended models are one with character embedding and GRU; one with

[4]Maximum sequence length: the maximum total input sequence length after WordPiece tokenization: sequences longer than this will be truncated, and sequences shorter than this will be padded

[5]Document stride: when splitting up a long document into chunks, how much stride to take between chunks

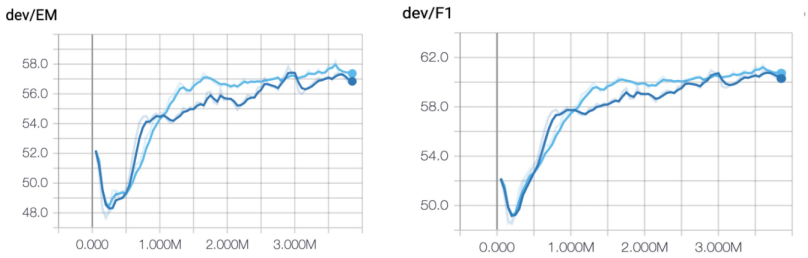

Figure 4: EM and F1 for BiDAF models: light-blue: BiDAF w/ CharEmb; blue: BiDAF w/out CharEmb

self-attention layers from BiDAF++ added after Modeling layer; and the last one is with summing up self-attention layers from BiDAF++ with modeling output, then inputting the sum to the Output layer. The reason for choosing these three models was that they included all extensions done on BiDAF starter code and performed better than others.

The main idea was the same as ensembling BERTs but we used weighted sum to obtain our answer for each question. The weights are: for each question,

$$p_{ans} = \sum_{i=1}^{2} 0.5 * P_{BERTBase_i} + 1 * P_{BERTLARGE} + \sum_{i=1}^{3} 0.3 * P_{BiDAFBaseline_i} + 2 * P_{SubEnsemble}$$

where p means probability.

4.4 Results

Our experimental results are shown in Table 1. Our final model is submitted on PCE test leaderboard. The baselines are submitted on PCE dev leaderboard. Note that all scores shown in all tables were tested on dev set except for model called 'Ensemble 2 (test)' (i.e. the final model).

Character Embedding

According to Table 1 and Figure 4, we can conclude that the baseline BiDAF model with a character-level embedding layer performs a little better than the one without. Besides, our result for original BiDAF is about one point less than the reference on leader board for both EM and F1, but still comparable. The reason for this small difference might be that we have only trained the original BiDAF 30 epochs or have not fine-tuned the model enough. If the model were trained more epochs and fine-tuned, the result might be as good as the reference. Moreover, adding character embedding doubles the training time around 6 hours to 12 hours. So we chose not to include the character embeddings when we trained other models.

Self-attention from BiDAF++

For self-attention layer inspired by BiDAF++, the results are shown in Table 3. It is obvious that all models with the BiDAF++ self-attention layers yielded better performance than the BiDAF starter code. However, since we used 30 epochs for each model instead of 25 mentioned in Handout, our results might slightly higher since we ran more epochs. But the increase for both EM and F1 are not trivial. So, we think self-attention layers do help model represent the question-related-context vector more suitably.

The model added self-attention layers after Attention layer performs the best, which matched the intuition.

Self-Matching attention from R-NET

As we have described in 4.3.3, we tested three versions of self-matching attention. We can see from Table 2 that even only trained for 10 epochs, those results are still comparable or even slightly higher than the BiDAF baseline model. The best result obtained from self-matching attention was EM 56.125 and F1 59.548.

BERT

Model	EM	F1
SA(after Modeling layer)	55.08	59.98
SA(after Attention layer)	**57.60**	**61.33**
SA(Addition after Modeling layer)	56.36	59.92
Ref: BiDAF Starter Code	55	58

Table 3: The performance of BiDAF with Self-Attention (SA) inspired by BiDAF++ trained for 30 epochs. All use biGRU for Encoder besides reference baseline model.

Model	EM	F1
Ref: BiDAF Starter Code (dev)	55	58
Ensemble 1: One BERT + One BiDAF (dev)	75.683	78.552
Ensemble 2: Three BERTs + Three BiDAFs (dev)	**77.706**	**80.285**
Ensemble 2:(test)	**77.312**	**79.984**

Table 4: The performance of two ensembles on dev set and the final model on test set.

It is obvious that the BERT-Base-Uncased model performs significantly better than BiDAF, and BERT-Large-Uncased model performs better than BERT base model, which are expected. We tried maximum sequence lengths 100 and 300 for BERT-Base-Uncased model (see Table 1: BERT-Base-Uncased (100) and BERT-Base-Uncased (300) respectively). We can conclude from the results that when we increase the maximum sequence length, the result may get better.

Ensemble

As we mentioned in 4.3.5, we used two ways to do ensembling. The result is shown in Table 4. We chose the ensemble model with the highest dev score as our final model and tested it on the test set. This final ensemble model with **77.312** EM and **79.984** F1 on the test set is a promising system.

5 Analysis

Since our final model is an ensemble, the performance is highly dependent on its sub-models. We chose one of its typical sub-models, a BiDAF model with self-attention inspired from BiDAF++ after Modeling layer, to do qualitative evaluation through dev examples shown on TensorBoard. Additionally, this model has the highest scores among extensions of BiDAF starter code models. From the results in the TensorBoard, we can see that it answers straightforward questions with high accuracy. More specifically, if the question has keywords which are able to be found in the context, and the answer is near the keywords, the model will correctly answer the question with high probability. However, if the answer needs not only the context but also some logic, the model will return N/A for most of the time. For example, if the question is asking: "who was person A's grandfather?", but the context only contains some equivalent information such as "B's grandson A refused to submit...", instead of same key words as "person A's grandfather", the model would fail to answer it and return a N/A. Our further plan to fix this issue is either developing a deeper model via adding more layers or training model to learn some common sense. However, training model to learn logic or do reasoning by itself is still hard today.

6 Conclusion

In this paper, we introduce an ensemble of self-attention BiDAF models with character embeddings and three fine-tuned BERT models. We tried two types of self-attention layers for BiDAF in different positions of the model. The experimental quantitative evaluations show that our model achieves the state-of-the-art results in SQuAD2.0. The analysis of qualitative evaluation also shows that our model has learned a suitable representation of QA but not learned complex logic appropriately. Overall, our model is able to answer non-trivial questions by attending correct locations in the given context and we have successfully achieved most of our goals.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] 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.

[3] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*, 2017.

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

[5] 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.

[6] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. *arXiv preprint arXiv:1702.00887*, 2017.

[7] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[8] Microsoft Research Asia Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. *Work-in-progress technical report posted on Microsoft.com*, 2017.

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[10] 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.

[11] 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.

[12] Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. Technical report, Technical report, OpenAI, 2018.

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

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

[15] CS224N Staff. Starter code for stanford cs224n default final project on squad 2.0. `https://github.com/chrischute/squad.git`.

[16] 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.