Machine Comprehension with BiDAF and Answer Pointer

Zehui Wang
Department of Computer Science
wzehui@stanford.edu

Xiaoxue Zang
Department of Computer Science
xzang@stanford.edu

Abstract

In this project, we implemented a deep learning system for SQuAD challenge. In the input layer, we utilized character-level CNN embeddings and some manual features in addition to the word embeddings. In the attention layer, we used Bi-Directional Attention Flow (BIDAF) [7] to obtain query-aware context representations. To predict the start and end location, we reimplemented the boundary model used in "Match-LSTM with Answer Pointer" [8] and used dynamic programming to search for the start and end location with the maximum joint probability. Our model achieved F1 score of 71.6% and EM score of 60.38% on the test leaderboard.

1 Introduction

Machine Comprehension is an important task in natural language processing. A machine comprehends a passage of text if, for any question regarding that text that can be answered correctly by a majority of native speakers, that machine can provide a string which those speakers would agree both answers that question, and does not contain information irrelevant to that question. [2] Stanford Question Answering Dataset (SQuAD), a new reading comprehension dataset, is focusing on this problem. The goal of this project is to produce a reading comprehension system that works well on SQuAD.

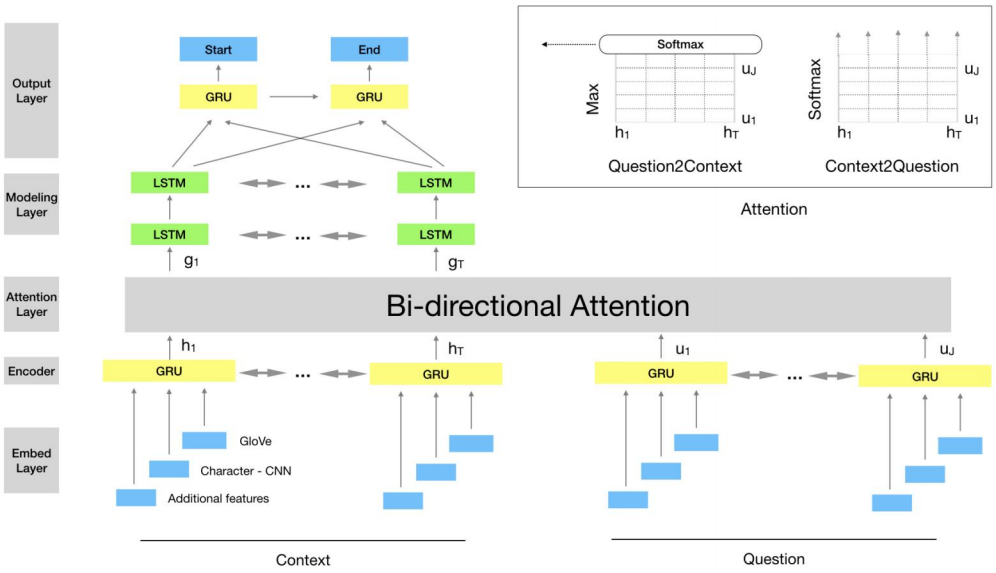
2 Related Work

Traditional work regarding to machine comprehension involves multiple stages of linguistic analysis and feature engineering. With the advances of deep learning, there has been a trend of building neural network architectures for various natural language processing tasks, including machine comprehension problem.
Convolutional Neural Network, which has achieved remarkable results in computer vision, has subsequently been applied to solving NLP tasks and have achieved excellent results in semantic parsing. In our project, we obtained the character-level embedding of each word using CNN, following [5]. The concept of attention also gained popularity recently in training neural networks [9] [1] because it allows models to learn alignments between different modalities. In our project, we adopted Bi-Directional Attention Flow (BIDAF) [7] as the attention layer to obtain query-aware context representations. For output we follow Pointer Net model [8], that is, instead of predicting the start location and the end location independently, we explicitly condition the calculation of the end location on the predicted start location.

3 Model

Our deep learning model is composed of multiple layers to capture different levels of contextual information in both questions and contexts. A conceptual architecture of our model is described in Figure 1. The details of each layer are illustrated as below.

Figure 1: A conceptual architecture of our model.



3.1 Embedding Layer

1. **Word embeddings**: We map each word into a pre-trained 100-dimensional Glove [6] embeddings and keep these pre-trained embeddings as fixed constants in all our experiments.
2. **Character level embeddings**: Given a word w consisting of characters $c_1, \cdots c_L$, we represent the word using trainable character embeddings $e_1, \cdots, e_L$. We take the embeddings into a dropout layer, then into a 1-layer CNN followed by max pooling to obtain the final character-level encodings for the word w.
3. **Manual features**: We follow the approach used in DrQA model [3]. There are two types of the features that we include.
   - Exact match: We include three binary features which indicate whether the original context token, the lower case of the context token, or the lemma form of the context token appears in the question sentence.
   - Token features: We include part-of-speech (POS) tags and named entity recognition (NER) tags of the context tokens to capture more aspects of the context information. We implement labeling using the taggers provided in NLTK packages. Then, we map every POS tag into a 12-dimensional trainable POS embeddings and every NER tag into a 8-dimensional trainable NER embeddings.

3.2 Attention Layer

We incorporate Bi-directional Attention Flow (BiDAF) [7] as the attention layer into our model. The inputs to the layer are vector representations of the context H and questions U. In this layer, we compute attentions in two directions: from context to query and from query to context. First we compute a similarity matrix $S \in R^{T \times J}$, where T is the length of the context representations and J is the length of the question representations. S is computed by

$$S_{ij} = w_{sim}[H_{:t}, U_{:j}, H_{:t} \circ U_{:j}] \in R,$$

where w_{sim} is a trainable weight vector, $H_{:t}$ is the *t*-th column of H and $U_{:j}$ is the *j*-th column of U. Now, the context-to-question is computed as following. Let $a_t = softmax(S_{t:}) \in R^J$. Then the attended query vector is

$$\hat{U}_{:j} = \sum_j a_{tj} U_{:j}$$

For question-to-context attention, we first compute the weights b =softmax$(max_{col} S) \in R^T$. Then the attended context layer is

$$\hat{h} = \sum_t b_t H_{:t}.$$

This vector indicates the weighted sum of the most important words in the context with respect to the query. Finally, we concatenate bidirectional attentions as

$$G = \left[h; \hat{u}; h \circ \hat{u}; h \circ \hat{h}\right],$$

which is our output.

3.3 Modeling Layer

Modeling layer takes the output of attention layer into two layers of bi-directional LSTM to capture the interaction among the context words conditioned on the query.

3.4 Output Layer

The implementation of the output layer is inspired by the decoder mechanism utilized in sequence-to-sequence model, that the prediction of the end position should be hinted by the prediction of the start position. The Boundary Model proposed by Shuohang et al. [8] incorporate a similar idea. Thus, we implement their algorithms and combine it into our model. In order to generate the index of the start and end token a_s, a_e, we use attention mechanism to attend to the high-level contextual representations generated by the previous layer. In details, we first obtain an attention weight factor $\beta_k \in R^d, k \in \{s, e\}$ from $H_r \in R^{T \times 2L}$ which is a matrix representation from the previous layer where T is the context length and $2L$ is the dimension size. Because the matrix of the previous layer usually comes from some hidden states of a bi-directional RNN model, the dimension size is presented as an even number. The computation of β_k is modeled as Equation 1 where $V \in R^{2L \times L}, W^a \in R^{L \times L}, b_a \in R^L, v \in R^L, c \in R$ are parameters to be learned, $h^a_{k-1} \in R^L$ is the hidden vector at position $k-1$ of an answer GRU cell (when $k = s$, h_{k-1} is initialized as a zero vector), and $\otimes e_T$ expands the first dimension of its left vector by repeating the left vector T times. Equation 1 computes variables $F_k \in R^{T \times L}$ and $\beta_k \in R^T$.

$$\begin{aligned} F_k &= \tanh(H_r V + (h^a_{k-1} W^a + b^a) \otimes e_T) \\ \beta_k &= softmax(F_k v + c \otimes e_T) \end{aligned} \tag{1}$$

The hidden vector h^a_s of the start token is computed as the output of a GRU cell and is modeled as below:

$$h^a_s = \overrightarrow{GRU}(H_r \beta_s). \tag{2}$$

Because the probability of generating an answer span is:

$$p(a|H_r) = p(a_s|H_r)p(a_e|a_s, H_r) = \beta_s \times \beta_e,$$

we train the model by minimizing the following negative loss function:

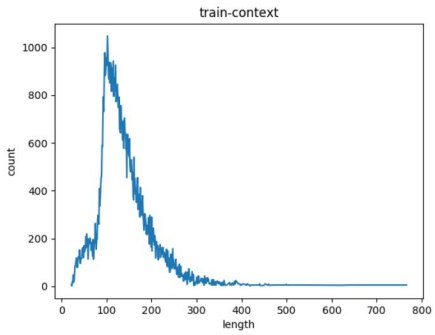
$$-(\log p(a_s|H_r) + \log p(a_e|a_s, H_r)).$$

4 Dataset

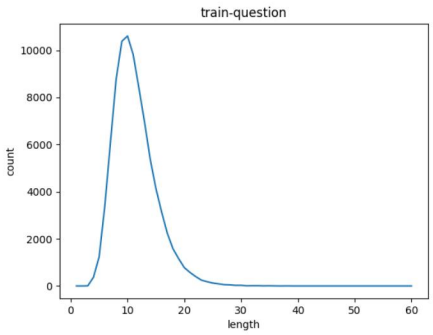
SQuAD collect question-answer pairs from 536 Wikipedia articles. The answers come from the given contexts and are presented as a pair of the start token location and the end token location in the corresponding context. We use a training set of 86,318 pairs and a development set of 10,391 pairs in our experiments. The distributions of the context, question, and answer lengths are shown

in Figure 2. From the figure, we find that most answers primarily locate within the first 200 tokens in the context. The majority of the contexts have less than 400 words and the majority of the questions have less than 20 words. Based on this statistics, we truncate the contexts and questions to a maximum length of 200 tokens and 20 tokens respectively to both reserve most of information in the contexts and questions and to keep the model size within the memory restrictions (56GB) of the device (Azure NV6).

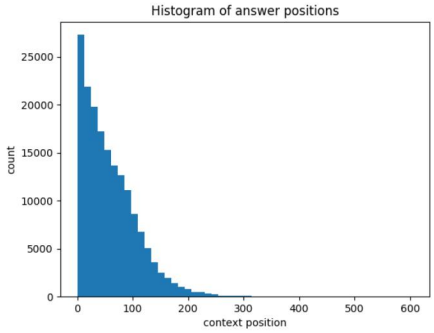
Figure 2: Data length distribution



(a) Context length distribution



(b) Question length distribution



(c) Answer position histogram

5 Experiments

5.1 Implementation details

In the embedding layer, we applied 100 1D filters of width 5 to capture a 100 dimensional character embeddings for each word. We also used 100 dimensional pre-trained Glove word embeddings, 3-dimensional exact-match features, 12 dimensional POS embeddings and 8-dimensional NER embeddings.
The values of the other parameters we chose are hidden state size = 150, batch size = 100, maximum context length = 200, maximum question length = 20, optimizer = Adam optimizer, learning rate = 0.001, regularization = 0.0001, dropout rate = 0.2.

5.2 Results

We gained F1 score of 68.1% and Exact Match of 53.2% on our own dev set and F1 score of 70.7% and EM score of 59.8% on the test set (dev-1.1v.json). We followed and combined the techniques used in three related works (BiDAF, DrQA and Match-LSTM models). A detailed summary of our results is shown in Table 1. We found that all the components that we implemented helped to improve the performance. An ablation study is illustrated in section5.3 to analyze the effectiveness

of all the elements.

We make some slight modifications to our final model to improve the performance. We add an additional dropout layer after computing the intermediate value F_k (Equation 1) compared to the original Answer Pointer proposed in Match-LSTM model. This technique helps to reduce overfitting and boosts F1 score by 1 %. We also used three modeling layers as our default setting after the bi-directional attention layer compared to the original architecture of BiDAF, which uses two. Increasing the modeling layer helps to raise the F1 score by 1 % (from 63.7% t0 64.7%) and EM score by 1.6% (from 48.7% to 50.3%) as Table 1 suggests.

We also implemented dynamic programming in the final prediction, that we search for the start and end location s and e where $s \leq e$ and the product of their probabilities $p_s \times p_e$ achieves the maximum value. It can be solved in linear time using dynamic programming. This technique elevates F1 score by 0.7% .

5.3 Ablations

From Table1, we found that a bi-directional attention layer alone helps to improve the F1 score by 4% compared to the baseline and a combination with modeling layers further boost its power. F1 score increases from 39% to 64.7% when using BiDAF followed by three modeling layers. Adding character-level embeddings further elevates the F1 score by 1.4% to 66.1%. In the original paper, BiDAF is reported to achieve F1 score of 68.0% and EM score of 77.3% . We suspect that the performance gap may results from the lack of Highway layer, the different optimizer, the different parameter settings and the different training strategies that we use.

We added Answer pointer model in the output layer and additional manual features in the input layer to the baseline model. This helped to raise the F1 score by about 20%. Furthermore, we observed that changing the baseline's output layer to Answer Pointer makes the model learn fast at an early stage. After 500 iterations, a BiDAF+ Modeling + Char-CNN model with Answer Pointer can achieve F1 score of 46%, while the same model without the Answer Pointer only achieves F1 score of around 15%. Combining Answer Pointer and adding manual features to the BiDAF model helps to further improve model's performance by 1%.

Table 1: Experiment results on development and test set

Experiment	Dev (%)		Test (%)	
	F1	EM	F1	EM
Baseline	39.6	28.9	43.4	34.6
BiDAF	43.7	31.6	-	-
BiDAF + Modeling (2 layers)	63.7	48.7	-	-
BiDAF + Modeling	64.7	50.3	-	-
BiDAF + Modeling + Char-CNN	66.1	51.4	-	-
BiDAF + Modeling + Char-CNN (no share weights)	63.3	48.1	-	-
Baseline + Answer Pointer + Manual Features	59.5	45.2	-	-
BiDAF + Modeling + Char-CNN + Answer Pointer	66.2	52.2	-	-
BiDAF + Modeling + Char-CNN + Answer Pointer + Manual Features	66.5	52.1		
BiDAF + Modeling + Char-CNN + Answer Pointer + Manual features + regularization(0.0001)	66.4	52.4	69.7	58.3
BiDAF + Modeling + Char-CNN + Answer Pointer (with additional dropout layers) + Manual features	67.5	52.8	70.0	59.4
BiDAF + Modeling + Char-CNN + Answer Pointer (with additional dropout layers) + Manual features + DP	**68.1**	**53.2**	**70.7**	**59.8**

5.4 Comparison

1. Regularization
   Most of our models started overfitting around 6000 $\sim$ 8000 iterations. We added L2 regu-

larization to the final model (BiDAF + Modeling + Char-CNN + Answer Pointer + Manual Features) to address this issue. We compared how regularization changes models' performance and summarized models' performance in Table 2. We trained all the model for 18000 iterations at maximum and stopped the training once it overfitted. The number of iterations in the table means the iterations when the best performance is recorded. As Table 2 suggests, adding 0.0001 regularization only helps to improve our model's EM score slightly and the model still suffers from overfitting. Using higher regularization weights makes the model learn slowly. Although models does not overfit after 18000 iterations when using 0.001 and 0.01 regularizations, the learning speed is much sacrificed.

Table 2: Experiment results using different regularization parameters on the final model

L2 regularization weight	**number of iterations**	Dev (%)	
		F1	EM
0.01	16,500	51.6	35.5
0.001	17,000	65.2	51.0
0.0001	8,000	66.4	52.4
0	8,000	66.5	52.1

2. Weight sharing
   In order to analyze the effect of sharing weights between contexts and questions to the model's performance, we trained a BiDAF+ Modeling + Char-CNN model using different GRU encoders for contexts and questions. The performance dropped by 2.8% (from 66.1% to 63.3%). It agrees with our expectation that sharing weights is better in this case because question and context embeddings exist in the same vector space. Sharing weights helps to enrich the model with more data and thus to learn more accurate representations. However, if context and question embeddings exist in different vector space, for instance they are written in different languages, then sharing weights would be inappropriate.

6 Analysis

6.1 Error Analysis

We analyzed the different types of the errors our model makes in dev-1.1v.json and visualized the distribution in Figure 3. Some predictions does not overlap with the ground truth, which indicates that the model probably fails to understand the semantics. We name this kind of error as location error and 62% of the total errors belongs to this type. Another 38% of the errors results from the imprecise boundary predictions that the model finds the correct area but fails to predict the correct boundaries. In different types of the boundary errors, we found that our model's prediction tends to have a longer tail than the ground truth (Boundary error type 2 and Boundary type 3 in Figure 3). An example is shown below:

Context Radical Islamist organizations like al-Qaeda and the Egyptian Islamic Jihad, and groups such as the Taliban, entirely reject democracy, often declaring as kuffar those Muslims who support it (see takfirism), as well as calling for violent/offensive jihad or urging and conducting attacks on a religious basis.b

Question On what basis do the radical Islamist organizations conduct their attacks?

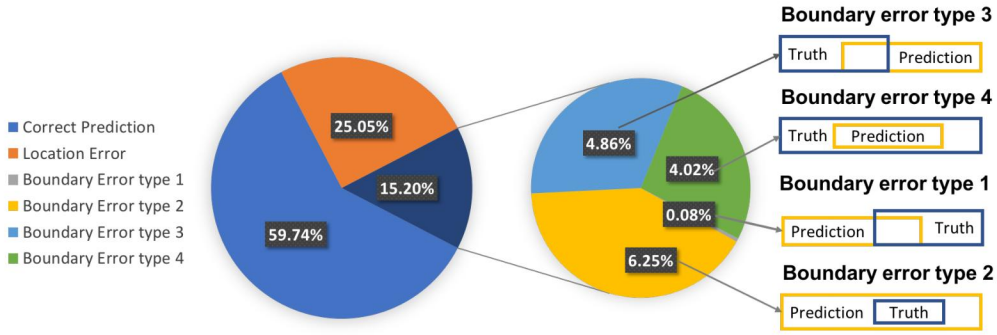
Prediction 'religious basis'

Ground Truth ['religious']

In some cases, the predictions are reasonable answers to humans even though they are different from the ground truths. One example is,

Context After the bill has been passed, the Presiding Officer submits it to the Monarch for royal assent and it becomes an Act of the Scottish Parliament. However he cannot do so until a 4-week period has elapsed,

Figure 3: Statistics of the type of errors model makes



Question What is the minimum amount of time before a bill can go into law?

Prediction '4-week'

Ground Truth ['a 4-week period', '4-week period']

6.2 Visualization

First, we visualized the attention layer in Figure 4. To observe which words the layer actually attends to, we visualized the weight matrix of an example from dev data. The vertical tokens come from the context and the horizontal from the question. The tokens in the bracket are the words from context with high attention weights. From the figure we can see that 'silver' in the question attends to 'silver' and 'gold' with high weights, while for symbols or punctuations in the question, the values in these columns are very low. Also, words in the question tend to attend to the same words in the context, which can help locate the answer span in the context.

We also visualize the output of the encoder to observe the embedding results in 5. We use 1-layer bidirectional GRU to encode the concatenating embedding vectors of character-level CNN and GloVe. We choose the same example in attention visualization. After using PCA to project word representations to the plane, we can see that words are surrounded by ones with similar meanings.

Figure 4: Attention visualization

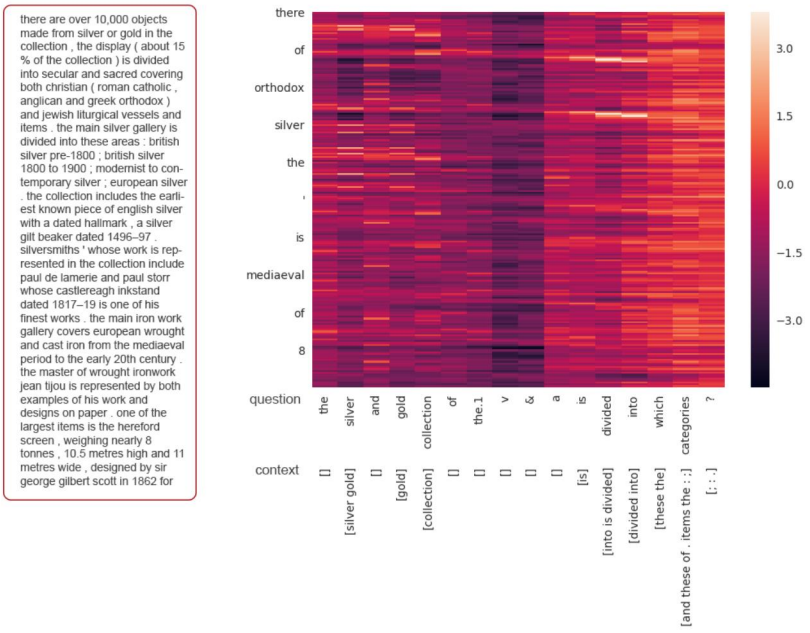
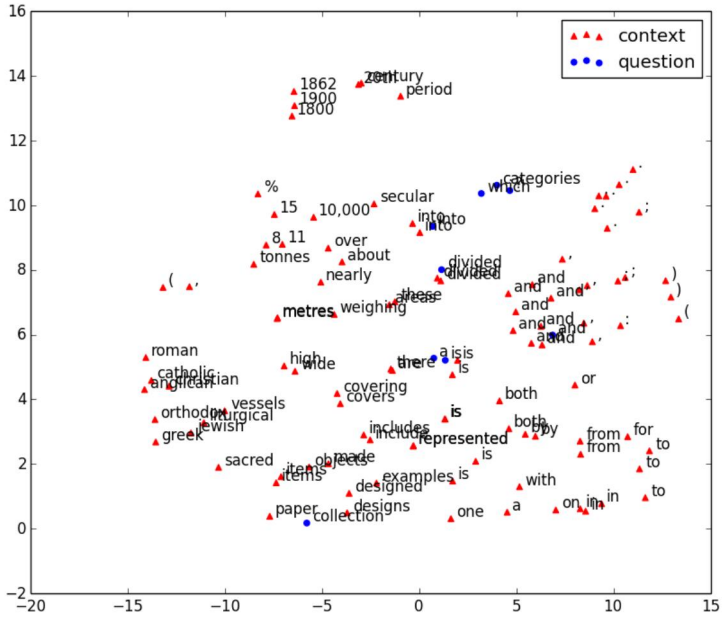


Figure 5: Embedding visualization



7 Conclusion

We combined multiple techniques to build a Question-Answering system for SQuAD challenge. We added a bidirectional attention flow layer, modeling layers, character level embeddings, answer pointer, and additional manual features on the baseline model. Our model achieved F1 score of 0.716 and EM score of 0.604 on the test leaderboard. We analyzed how different component contributes to model's performance. All of the approaches suggested above are useful. In particular, we found that an effective attention layer followed by modeling layers contributed most to the performance boost. In addition, we also found that sharing weights between contexts and questions encoders are not useful and adding regularization reduces overfitting but sacrifices learning speed. We also implemented attention visualization to see if our model produced reasonable results. Some future works includes implementing self-attention or fully-aware attention as are used in R-net and Fusion Net [4] to gain more precise prediction.

References

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision ICCV*, pages 2425–2433, 2015.

[2] Christopher J. C. Burges. Towards the machine comprehension of text: An essay, 2013.

[3] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*, 2017.

[4] Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. In *International Conference on Learning Representations*, 2018.

[5] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[6] 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.

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

[8] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[9] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *International Conference on Machine Learning*, pages 2397–2406, 2016.