Simplified Dynamic Neural Network for SQuAd Question Answering

Mengying Zhang, Alicia Ge

Abstract

One of the challenging tasks in Natural Language Processing is how to obtain useful information from given text message. In this project, we use two models to tackle Question Answering using Stanford Question Answering Dataset (SQuAD). Our baseline model follows traditional question answering steps where we divide the tasks into different subtasks. The baseline model achieves exact accuracy of 8.2% and adjusted accuracy of 14.7% on test data. Our second model Simplified Dynamic Memory Network(SDMN) comprises 4 neural network modules that work together to do Question Answering task. It first reads in the passage and the question, and then iteratively searches for correct sentence that contains the answers and then spits out exact word for its output. SDMN achieves 14.3% on our test data using only the first word in answer as golden answer. Given the time constraint, we were not able to use full answers in our network, but we believe the full version will demonstrate more significant learning abilities of the model.

1 Introduction

There has been an increasing demand in making the machine to comprehend large corpus of text data and retrieve useful information based on the query. The task is formally formulated as Question Answering(QA) in Natural Language Processing. Different from traditional data analysis, QA task requires pre-processing of text data, and finding a way to make the machine understand words meaning. QA also requires more complex machine learning algorithms to handle query formulation, passage understanding, and answer retrieval. The most challenging part comes from how to piece together different sub-tasks into one framework that machine can proceed to do the entire task on its own. Traditional approach mainly includes window-slicing and rule-based methods, while in recent years many of the machine learning algorithms have been adopted to address the task as well. The triumph of Recurrent Neural Network has brought QA into a new stage where it asks for less human labor in finding the features that are mostly required by supervised machine learning algorithms. Progress in QA research can lead to improvement in optimal search engine result. It further indicates potential ways for computers to understand human words. In this project, we develop two baseline models that do not require any learning and one Neural Network Model that does not rely on any external knowledge to deal with QA task using Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016). We hope our baseline models can provide insight to how we can deal with complex task using different Natural Language Processing (NLP) techniques and our second model as a supporting application of existing Dynamic Memory Network(Kumar et al.).

2 Problem Definition and Data

Our task is to develop two baseline models and one Neural Network model to do QA. In our project, Question Answering task is specifically defined as to find the correct answer, which is a constituent segment of the context, to the proposed question from a Wikipedia article. We use SQuAd as our data set to conduct QA.

SQuAD is a highly cleaned and curated dataset that includes 107,785 reading comprehension problems based on wikipedia articles that are manually collected and answered by crowdsource workers. Example data is shown in Figure1. There are 23,215 paragraphs extracted from 536 articles from different domains ranging from architecture,

Paragraph	Question	Answer
Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend \"Venite Ad Me Omnes\". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary.	To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France	Saint Bernadette Soubirous
	What is in front of the Notre Dame Main Building?	a copper statue of Christ
	The Basilica of the Sacred heart at Notre Dame is beside to which structure?	the Main Building
	What is the Grotto at Notre Dame?	a Marian place of prayer and reflection

Figure 1: Example data from SQuAD

Answer type	Percentage	Example
Date	8.9%	19 October 1512
Other Numeric	10.9%	12
Person	12.9%	Thomas Coke
Location	4.4%	Germany
Other Entity	15.3%	ABC Sports
Common Noun Phrase	31.8%	property damage
Adjective Phrase	3.9%	second-largest
Verb Phrase	5.5%	returned to Earth
Clause	3.7%	to avoid trivialization
Other	2.7%	quietly

Figure 2: Different types of answers from SQuAD

music to abstract concepts. Each paragraph has corresponding questions and golden answers are usually long phrases that are entailed in the passage. The answers include a variety of entities such as location, date, person and verb/adjective phrases as shown in Figure2. We have randomly partitioned the data into the training set(80%), development set(10%) and the test set(10%).

3 Related Work

Many different approaches have been taken for QA tasks from the beginning of the 21st century. Riloff and Thelen hand-crafted a set of Wh based rules, named Quarc, to find the answers(Riloff and Thelen, 2000). Quarc is built upon morphological analysis, part-of- speech tagging, semantic class tagging and entity recognition and results in 40% of accuracy on their reading comprehension test set. Deepak et all tackles the problem by first building surface text patterns using regular expression from the external web (Ravichandran and Hovy, 2001) and then use suffix trees to obtain answers. Early approaches rely either on hand-crafted rules or external system to perform QA and these usually do not use machine learning to exploit information from data itself. Hwee Ton Ng, Leong Hwee Teo, Jennifer Lai Pheng Kwan. Ng et all are the first to apply machine learning approaches to QA problems (Ton et al.). They search for useful feature representation vector and use logistic regression to tackle QA problems. The features include number of overlapping words(DMWM), number of not overlapping verbs(DMVM), and corresponding DMWM/DMVM from previous/next sentence, and three other binary variables that look for keywords and specific text positions. They have achieved competitive result on Remedia dataset as opposed to handcrafted rule-based approach. Recently, more exciting algorithms been proved to successfully tackle large variety of QA systems. For example, a three sub-model approach is introduced to solve the question answering problem (Hu et al., 2017). They use feature-rich Encoder, Iterative Aligner and Memory-based Answer Pointer to first locate the passage, and understand it and find the answer. More recently, there has been a major breakthrough in QA using so-called Memory Neural Network first proposed by Weston et all (Weston et al., 2015), then further developed by Sukhbaatar et all(Sukhbaatar et al.) to make the Memory Network end-to-end so that its fully differentiable. Kumar et all (Kumar et al.) expanded the idea to a more general Dynamic Memory Network(DMN) that is applicable to a variety of NLP tasks. Our Neural Network model largely builds upon this framework which will discuss further in the following section.

4 Methodology

We have built two models in our project: Randomized Parsed Tree Search, which is our second baseline model and Simplified Dynamic Memory Networks. The traditional QA tackling process includes three stages: question processing, passage retrieval and answer processing. Our baseline model follows this strategy. However, our second Neural Network model will integrate these into one and solve the problem in one phase.

4.1 Baseline Model

4.1.1 Random Search

The first baseline model is random search. We first use window slicing method to extract the most plausible sentence that includes the target answer and then returns the first word in the sentence as our answer. Specifically, we first find the count of overlapping unigram and bigram between the question and each sentence in the passage, which

will yield a weighted similarity score that is defined by

$$Score = \frac{\frac{1}{3}N_1 + \frac{2}{3}N_2}{L} \quad (1)$$

where N_1 means count of overlapping unigram , N_2 means count of overlapping bigram and L represents length of the question. Notice that we put more weights on bigram overlapping because bigram is a stronger signal of a real match. For example, in unigram, there is a high chance the word *the* will be in both question and the context sentence, which does not really contain useful information. However, in bigram, we have a higher chance of seeing meaningful phrases such as *Nortre Dome* which sends a stronger signal of relevance between the question and the context. Then we return the first word of the sentence that has the highest score, which is our final answer.

4.1.2 Randomized Parsed Tree Search

Our second baseline model Randomized Parsed Tree Search follows the traditional 3-step strategy in dealing with QA tasks. First in passage retrieval stage, we use unigram and bigram to find the highest overlapping sentence between the question and each sentence in the passage. We use exactly the same similarity score schema in the first baseline model to score each sentence in the passage. In question processing stage, we use a pre-defined mapping to determine the answer type we want to extract (Figure 3). Then in answer processing stage, we parse the highest score candidate sentence retrieved from question processing stage and find the corresponding phrase that matches the part of speech(POS) tag required by the answer type. Instead of using all combinations of phrases that matches the required POS tag, we first search for the highest level of matched node and compare each of the satisfied phrase with the question. Phrase(s) that does not contain any of the questions non-stopwords will be our candidate answer(s). If None is found, we go to the bottom level of matched node and repeat the procedure stated above (Figure 4). In the end, we end up with a list of candidate answers and we randomly pick our final answer from this list. If none of the candidate answers are found from the current sentence, the algorithm searches for the next highest score candidate sentence.

```
# ____________________________________________
# a dictionary maps question type to answer type
# ____________________________________________
QA_TYPE_MATCH = {'what': 'NP', 'when': 'CD', 'where': 'NP', 'whom': 'NP', 'why': 'NP',
                 'who': 'NP', 'which': 'NP', 'whose': 'NP', 'name': 'NP', 'example': 'NP',
                 'how many': 'CD','how much': 'CD','what percentage': 'CD','how often': 'CD',
                 'what year':'CD','location':'NP'}
```

Figure 3: Answer Type Mapping

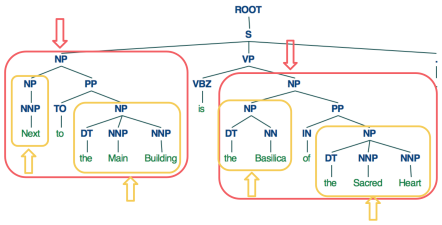

Figure 4: Example of Parsed Tree for a Candidate Sentence

Parse the candidate sentence "Next to the main building is the Basilica of the Sacred Heart". Suppose the question is "What's next to the main building?" and the desired answer type is NP. We first select the highest level of NP marked by the red box which is "Next to the main building" and "the Basilica of the Sacred Heart". Since "Next to the main building" contains question's non-stopwords, we choose "the Basilica of the Sacred Heart" as our candidate answer. If none is found, we go to bottom level which is marked as yellow box to retrieve candidate answers using the same approach.

The reasoning behind this algorithm is to divide the QA task into different sub-tasks. First we find the most plausible sentence in the passage that might contain the answer by comparing its relevance to the question, then we pinpoint the answers location within the sentence using the predefined mapping and the information provided by the parsed tree. Then we eliminate potential answers using not-in-question strategy to narrow down the candidate answers. Finally, the final answer is chosen by random from the candidate answers, thus the name "Randomized Parsed Tree Search". Notice that our second baseline model extends the first baseline model by further extracting useful information from the retrieved candidate sentence, thus improving the model accuracy.

4.2 Simplified Dynamic Neural Network Model

Different from baseline models where no learning is needed, our Simplified Dynamic Neural Network(SDNM) is a Recurrent Neural Network(RNN) based learning model. What resembles with the previous model is that it divides into sub-tasks by chaining 4 pieces of RNN that performs different tasks. We use a simplified version of the original papers idea (Kumar et al.). SDNM has 4 Gated Recurrent Unit(GRU) based modules that mimic how human perform QA(Figure 5). GRU is a special kind of RNN and a simplified version of LSTM which makes the computation a lot faster. We include the full mathematical expression in the following description.

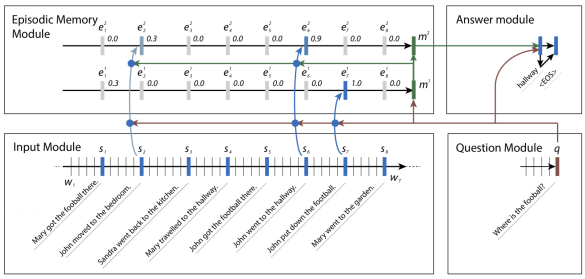

Figure 5: Visualization of 4 module neural network architecture. Credit: (Kumar et al.)

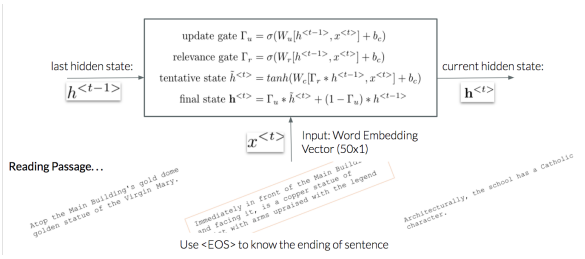

Figure 6: Input and Question Module GRU

1. **Input Module.** A GRU that read in the paragraph by encoded sequence of word embeddings. The hidden states collected at the end of each sentence are the output of this module, which encapsulates the sentence meanings(Figure 6).

2. **Question Module.** Same GRU as Input Module except that it processes the questions instead of context paragraph. The final hidden state of question is the output of this module, which encapsulates the meaning of the question.

3. **Memory Module.** In the memory module, we use attention mechanism to iteratively search for the candidate sentence. It takes the result of Input Module and Question Module as input and then calculate the similarity between each of the sentence hidden state and the question hidden state, in order to finding the most possible sentence that may contain the answer. The last memory hidden state is fed into the second iteration as the initial hidden state.

   Attention is obtained by first calculating a similarity matrix using sentence hidden state, question hidden state, memory hidden state and their element-wise product and then use this as input to two layers feed-forward Neural Network to learn the weights of attention. Then the output g_t^i, where i represents the number of iteration and t represents the number of sentence will determine how much the real attention is given to the current sentence$t$. Intuitively, if this is an important sentence that contains the right answer g_t^i will make the GRU output higher, and conversely if not important at all, it will not even go through this sentence in GRU and retain the previous hidden state as the current final hidden state. The update formula for each time step's hidden state with attention mechanism will be

$$\mathbf{h}^{<t>} = g_t^i GRU(C, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i \tag{2}$$

   where h represents the hidden state. Iteration times are referred to as number of passes later on.

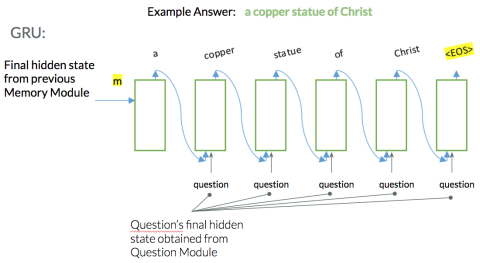

Figure 7: Answer Module GRU

4. **Answer Module.** The answer module takes the result of Memory module, which is its final memory hidden state and the question's hidden state as input and use a recurrent neural network to spit out the answer(Figure 7). This resembles the decoder part of seq2seq model in machine translation. However, in our simplified version of DMN, we fail to implement this part. We instead uses a two layer feed-forward Neural Network that takes in the previous memory hidden state and question hidden state then output a single word as final answer. In order to calculate loss, we only treat the golden answer's first word as correct answer. This significantly impedes the network to learn meaningful things as will be discussed in the next section.

5 Evaluation and Results

5.1 Baseline Model

Changing the hyper-parameters to get a better result

We implement the random baseline at the beginning of the project. The accuracy we get is less than 1% and really close to 0. At the beginning of improving the baseline, we divided the problem into two steps.

First, we used unigram and bigram method to calculate the similarity between each sentence in the paragraph and the question. in this way, we find the sentence that have the most similarity score so that it has the most possibility to contain the final answer span. The result for sentence retrieving is pretty good, we get 88.5% accuracy.

Second, we need to select word span from the candidate sentence. We detect the type of the question by its question words of it. We link each question word into a type of part of speech, then using parsing technique to retrieve all that kind of POS span. At beginning, we select top two candidate sentences to assure there is an output span. There is no phrase selection mechanism and just randomly generate output.

The accuracy is under 3% at the beginning and we decide to find a way to improve the result.We find out some of the outputs are so long that they are highly possible to contain the golden answer. However, they will lower down the answer retrieval accuracy. We then turn into the strategy to first search for top level of the target POS tag,and rule out those who contains the same word with question(means it may be too long and contains too many words). We still want to keep the rest since the top level spans are more possible to contain the true answer. if there is no more top level answer, we will seek to lowest level POS spans. The process will be similar, we rule out those contains same words with question and then randomly generate output.

Evaluation

The second baseline is what we called Randomized Parsed Tree Search. This is the baseline model we mainly focused on. We used several methods to evaluate this model.

$$Proportional\ Accuracy = \frac{N_{Exact\ right} + N_{Type\ 1\ Error} + N_{Type\ 2\ Error}}{N_{Questions}}$$
$$= 22\%$$

Figure 8: Equation of Proportional Accuracy

$$Answer\ Quality = \frac{\left(1 + \frac{len(correct\ answer\ in\ Type\ 1\ Error)}{len(our\ Type\ 1\ answer)} + \frac{len(our\ Type\ 2\ answer)}{len(correct\ answer\ in\ T2\ Error)}\right)}{3}$$
$$= 66.7\%$$

Figure 9: Equation of Answer Quality

1. Proportional Accuracy

   The Proportional Accuracy counts all the answer that exists overlap between generated answer and gold standard answer. Equation and result showed in Figure8.

2. Answer Quality

   There are two types of error exists in the proportional accuracy. One is the output answer contains the gold standard answer. Another is the output answer is part of the gold standard answer. Under this two circumstance, the answer only partially correct. We used the equation below to measure the quality of the answer.Equation and result showed in Figure9.

3. Adjusted Answer Accuracy

   The adjusted answer accuracy combined the two scores above. It adjusted the proportional score by the answer quality.Equation and result showed in Figure10.

5.2 Simplified Dynamic Memory Network model

Since the response of the network is only one words, there is no situation of partial correct. We mainly use the ratio of exact correct to evaluate our model. We have two separate datasets and used training dataset to train the model and used test data to evaluate the performance of the model.

We played around with several hyper-parameters, and find the best result with word embedding of 100 Dimensions, Memory episode with 2 passes, and a low dropout value 0.3.The main results

$$Adjusted\ Answer\ Accuracy = Proportional\ Accuracy * Answer\ Quality$$
$$= 14.7\%$$

Figure 10: Equation of Adjusted Answer Accuracy

Dimensions	Passes	Dropout	Train	Test
50	2	0.5	5.47%	5.70%
100	4	0.5	13.98%	Gradient Explode
100	2	0.3	11.25%	11.72%
300	2	0.5	13.75%	Gradient Explode

Figure 11: SDMN Training results with different parameters

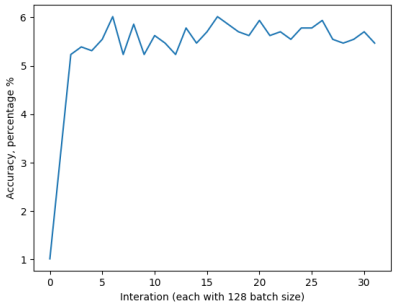

Figure 12: Accuracy with word vector 50 Dimension overtime, preliminary results

and behaviors of different experiments are showed from Figure11 to 15.

6 Discussion

6.1 Baseline Model

Two type of errors

There are two type of errors mentioned above during evaluating the result t=of the baseline model. We become understand distinguish these two types of errors can help us to evaluate the result more accurately since we get some of the answers partially correct. Examples of two types of error showed in Figure16.

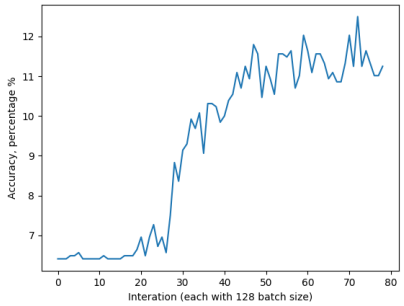

Figure 13: Accuracy with word vector 100 Dimensions and 2 Passes overtime

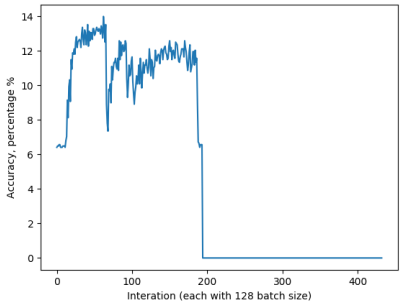

Figure 14: Gradient Explode! Accuracy with word vector 100 Dimensions and 4 Passes overtime, the accuracy first become unstable and then goes to 0.

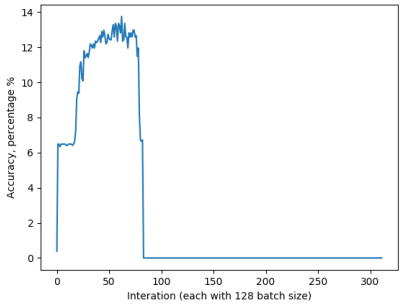

Figure 15: Gradient Explode! Accuracy with word vector 300 Dimensions and 2 Passes overtime, the accuracy goes up and then suddenly become 0.

Question	Correct Answer	Our Answer	Correct Type
How many seconds were left in the game when the Patriots failed their 2-point conversion?	17	17	1
What Carolina player was injured in the NFC Championship Game?	Thomas Davis	Thomas Davis , an 11-year veteran who had already overcome three ACL tears in his career ,	2
Who was limited by Denver's defense?	Newton was limited by Denver's defense	Newton	3

Figure 16: Example of two types of errors.Correct type 1 is exact correct; type 2 means output contains the gold answer and type 3 means another way around.

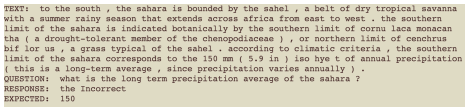

Figure 17: Example of the testing result.

6.2 DMN model

Gradient Exploding

As we can see in Figure17, The response of our mechanism is sometimes meaningless. This is because we only let the module learn the first word of the answer, which may be meaningless so that the network is confused and can not learn much information from the training as we expected.Under this circumstance, since there isn't too much to learn for the network in the first word of answer, the error gradient accumulate results in an unstable network.

Based on different training parameters we set, we picked up several interesting findings.

6.2.1 Dimension of the word embedding

We tried different dimensions of word embeddings to test how this influence the result. Our hypothesis is, with bigger word embedding vectors, the result will become better. However, We later find out this isn't the truth, and when the word embedding is 300 dimensions, Gradient exploding happens.

6.2.2 Iterations

The iteration(or epoch) of the training processing seemed becomes the parameter that constraint most since there is not enough time for us to train the model. At the beginning we treat one question as a input and tried 400000(only 4 epoch in fact) only to receive a poor result(11.4% for train data and 10.5% for test data,even poor than the baseline). We realize this is because we misunderstand the definition of iteration. After that, we become understand why we need a long time to train the model.

6.2.3 Multiple Passes

The number of passes influence the ability of the Episodic Memory Module understand the logic between sentences within one paragraph. However, due to the feature of our dataset that it does not contain too many logic chains between sentences, so we anticipate that adding more passes does not help more with the accuracy. However, what we didn't expected is that the increase of pass not only helpless of the prediction, but also causes gradient explode.

7 Other Things We Tried

The part that is thought as unfinished most is the Answer module of the DMN. The ideal result is retrieve answers that contain multiple answers.

We tried to modify the answer module so that it can generate a sequence of answer instead of one-word answer. The goal is to build up language model by using a GRU cell which receive the previous output as input. We tried to find out a existed function that can called within TensorFlow, but fail to find out one. What we current get is a dynamic RNN cell in TensorFlow, which can not guarantee the coherency of the output phrase.

8 What You Would Have Done Differently or Next

For the Randomized Parsed Tree Search methods, the potential improvement is to let the machine learn how to choose a phrase from several candidates. We can use logistic regression to find out features to help machine distinguish the difference between phrases and improve its understanding based on back propagation.

We enjoy our exploration during the building process of baseline. However, we might spend more time on learning NLP python library such as TensorFlow. This is because as we begin with the DMN model, we find out the implementation of Neural network needs more time than we think. We need time to fully understand the model of DMN that has been build up upon a different dataset; we also need more time to learn the structure and way of thinking of TensorFlow, so that we can fully achieve the goal we initially set up.

Notice that because of many meaningless thus random gold standard answers such as *the, an* , our model performs badly. If there is more time, we will try to figure out how to build a answer

module that can generate multiple-words answer. That means a new decoder unit of GRU cell that can output sequence with EOS signals the ending of output.

After that, we may also want to improve the SDMN by changing the scoring function G in Attention Mechanism. This mechanism controls whether one sentence is considered important or not.

9 Group Effort

Alicia Ge:

Baseline model: Unpacked the data structure and prepared for the algorithm. Implemented the sentence retrieval processing, including unigram and bigram similarity calculation and get 88.5% accuracy. Implemented the calculation the retrieval accuracy, including proportional accuracy, answer quality and adjusted answer accuracy.

Simplified DMN: Feed the original data into Glove dataset to get the already trained word embedding. Implemented the Input module, Question module and Episodic Memory module under Steven Hewitt's tutorial(Steven Hewitt, 2017) with TensorFlow.

Mengying Zhang:

Baseline model: Came up with the initial algorithm and improved the algorithm performance. Implemented parse tree and extract POS tag function.

Simplified DMN: Read through different papers on QA and investigated the Neural Network Architecture for the original paper. Understood the primary code for the project. Implemented Glove loading and pickle set up. Helped debugging and running on GPU. Coordinated with team members on project plan.

Acknowledgments

We want to thank Zhenlin Wang for insightful discussion on neural network and providing of GPU. We are also extremely thankful to Steven Hewitt for his primary code for Dynamic Memory Neural Network.

References

Minghao Hu, Yuxing Peng, and Xipeng Qiu. 2017. Reinforced Mnemonic Reader for Machine Comprehension .

Ankit Kumar, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. ???? Ask Me Anything: Dynamic Memory Networks for Natural Language Processing .

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text https://doi.org/10.18653/v1/D16-1264.

Deepak Ravichandran and Eduard Hovy. 2001. Learning surface text patterns for a Question Answering system. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*. https://doi.org/10.3115/1073083.1073092.

Ellen Riloff and Michael Thelen. 2000. A rule-based question answering system for reading comprehension tests. In *ANLP/NAACL 2000 Workshop on Reading comprehension tests as evaluation for computer-based language understanding sytems -*. https://doi.org/10.3115/1117595.1117598.

Steven Hewitt. 2017. Question answering with TensorFlow. https://www.oreilly.com/ideas/question-answering-with-tensorflow.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. ???? End-To-End Memory Networks .

Hwee Ton, Ng Leong, Hwee Teo, Jennifer Lai, and Pheng Kwan. ???? A Machine Learning Approach to Answering Questions for :Reading Comprehension Tests .

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. MEMORY NETWORKS .