SQuAD: Morpheme infused Mnemonic Reader for Question Answering

Vishal Anand
Columbia University
New York, New York
va2361@columbia.edu

Rakshita Nagalla
Columbia University
New York, New York
rn2439@columbia.edu

Asha Jacob
Columbia University
New York, New York
aj2782@columbia.edu

Abstract

In this paper, we aim to describe the method used to develop a model in order to create a Question Answering System. This system would answer a query corresponding to a given context document. This system is based on the Mnemonic Reader for Machine Comprehension [4]paper. In this paper, several techniques have been proposed to overcome some of the shortcomings faced by the base paper. A special kind of preprocessing has been done to infuse morphemes into the BiDirectional LSTMs to handle the 'why' questions. Experiments on Stanford Question Answering Dataset show that our model performs reasonably well.

1. Introduction

From a long time efforts are being made to bring Artificial Intelligence and Natural Language Processing together in order to find a state-of-the-art solution to the Question Answering Problem. In this problem, a passage and a question that has it's answer within the passage is given. The system aims to find an answer to the question from the document at hand.

Owing to the extensive research done on developing such robust question answering systems in addition to the cutting edge deep learning techniques, neural networks have proven to efficiently give very good results for this Question Answering problem. Most of these neural networks work with the concept of encoding - interacting - pointing [2]. Words are converted into their corresponding encodings by means of RNNs after words in the paragrapha nd the question are projected into distributed representations. The interaction between the given paragraph and it's question is handled by the Attention Mechanism and the beginning and end spans of the answer to the question in the document is given by implementing the Pointer Network [5]. Several problems were posed by the basic encoder-interaction-pointer framework such as not being able to capture part-of-speech and named entities as a part of word and character embeddings, missing long distance interaction between different parts of the same answer, failure of single-hop prediction mechanism to give a proper response to the given question and fuzzy nature of the answers to 'why' type of questions. The Mnemonic Reader [4] dealt with these problems by including additional features along with word embeddings such as part-of-speech and named-entity tags to capture important concepts in the given document. In addition, the paragraph is aligned with itself as well as the question to obtain a complete document representation. Following this, the pointing mechanism points to the boundary of the answer in the context.

In this paper, we propose infusing Morphemes into the Mnemonic Reader Model in order to better capture the overall context of the given document and question. This would help reduce the fuzziness of the 'why' type of questions and give better answers overall.

Section 2 gives an overview of the System, Section 3 gives an idea about the training mechanism used, followed by Evaluation of our system and other related work in sections 4 and 5 respectively.

2. System Overview

The overall system includes preprocessing the given data such as deriving word and character embeddings, infusing morphemes, normalizing the data, generating features that the model needs to be trained on, and training the actual model on these features.

2.1. Preprocessing and Question Treatment

1. From the literature review, we decided to infuse attention while attempting to extract answers. While experiments have been performed on abstractive summarization techniques, the same is not explored for extractive summarization techniques. The words in the questions were modeled over the articles[1] , each of which have multiple contexts. The set of topic models with the maximum similarity to the question-words' were picked up and injected into the question. This updated question is then fed into the LSTM model.

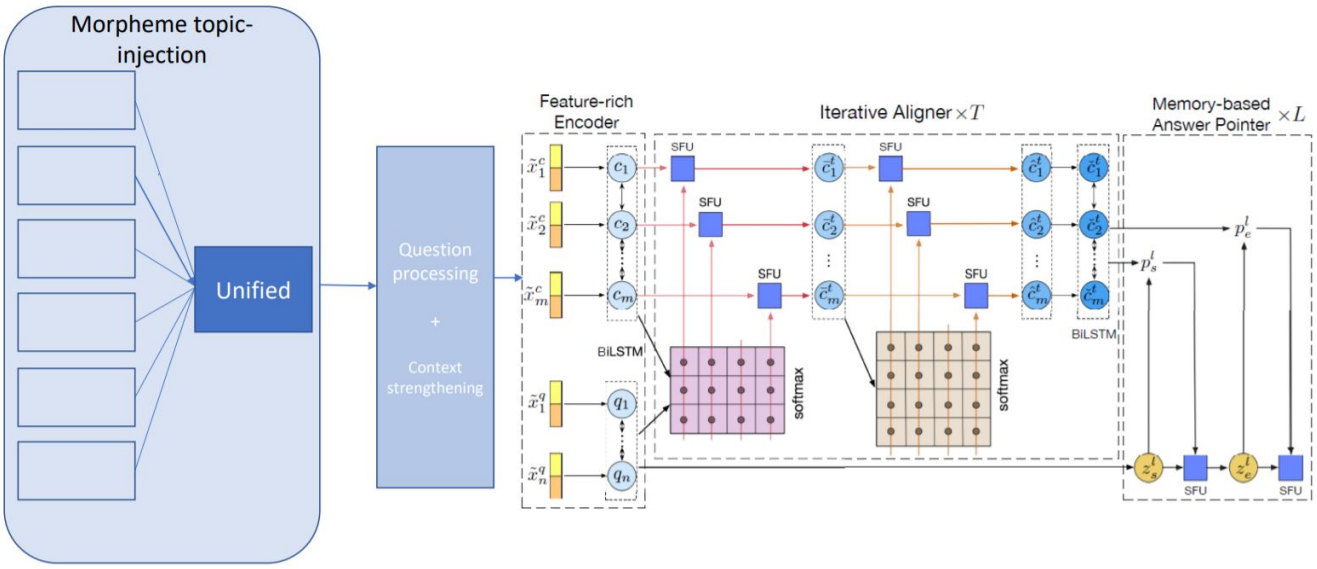

Figure 1. Overall Architecture of Morpheme Infused BiDirectional LSTMs

2. For the bulk of text pre-processing, we have levergaed the NLTK libraries. There were challenges encountered in varied ways of tokenizing and lemmatization processes. The work-arounds and delta in performance are discussed in the results section.

3. There have been cases when the number of words produced to be appended to the query at hand were smaller than the expected upper-limit of 10 words. While one could persist the same count for vanilla attention increase, however for model structure where batches are pre-defined to accentutate the word vectors of specific words, they could be tiled across to reach the upper-limit count of injected words.

2.2. Model Overview

The solution to a Question Answering Problem is to find an answer to a question given the document, which contains the answer. This answer exists in the document as a continuous sequence of words. We reimplemented the Mnemonic Reader to obtain the probability distributions of the start and end spans that point to the answer in the document. The three main components of this model are the encoder, aligner and the memory-based answer pointing mechanism.

2.2.1 Encoder

The encoder, which is the first part of the model converts the words in the document and the question into embeddings which are then further encoded for further processing. For the Question Answering task, two types of embeddings are used - word-level embedding and character-level embedding to which more features are added.

Embedding Let's say the given document contains n words whereas the question contains m words. The encoder converts each of the words in the document as well as the question into it's corresponding word embedding. We used 300-dimensional glove vectors pre-trained on Common Crawl dataset to lookup embeddings for words and characters in our corpus. In addition to this, a Bi-Directional LSTM [6] with 50 hidden units is also used in order to encode the characters in each of these words to capture the context better. Therefore the final encoded representation consists of a d-dimensional sequence, which is nothing but the combination of the word embeddings as well the character embeddings generated by the Bi-LSTMs.

Features Instead of stopping at just the word and character-level embeddings for encoding the document and question, several other features are also sued in order to make the encoding more robust. Part-Of-Speech tags specify the part-of-speech that each word belongs to (eg. Noun, Adjective, Verb, Adverb etc.), and Named Entity tags capture the presence of Named Entities (usually Proper Nouns) in the given question and document for which embedding matrices are created. In addition, the Exact - Match (EM) feature, which denotes if there are exact overlaps between words in the document and question is also used as an added feature. The normalized term frequency calculated as the the count of the the word in the question relative to the total length was added a sfeature for each of the question and context words. These additional features, i.e. the embeddings of the tags from the lookup matrices and EM are concatenated to the combined word embedding (word + char-

acter level embeddings) for each word.

3 exact match features were engineered indicating whether there is a match as it is or after making them case-insensitive or in the lemmatized form. The nltk tagger generated 40 unique Parts-of-Speech and NER tags. Including the term frequency, we engineered a total of 44 features.

The final encoded representation, including the additional features would be an *n*-dimensional embedding for the document and an *m*-dimensional embedding for the question where *n* and *m* are the number of words in the document and the question respectively.

Encoding Once the embeddings for the document and the question are created and more features are added to it, they are encoded. In order to model the words (from the document and question) with respect to the context provided, a Bi-Directional Long Short Term Memory (Bi-LSTM) is used to create the corresponding encodings. This is done in the following manner:

If h is the number of hidden states, c_i is the hidden state of the LSTM for the i^{th} word in the context and q_i is the hidden state of the LSTM for the j^{th} word in the question then c_i and q_i are each of $2h$ dimensions (since the LSTMs are bi-directional) and are given by-

$$c_i = \text{BiLSTM}(c_{i-1}, x_i^{context}) \text{ where } \forall i \in [1,...,n]$$
$$q_j = \text{BiLSTM}(q_{j-1}, x_j^{question}) \text{ where } \forall j \in [1,...,m]$$

Finally, after encoding, we obtain two matrices which contain the encoded versions of the document and the question. The encoded matrices of the document and question would be of the orders $2h*n$ and $2h*m$ respectively. These encoded matrices are then sent to the Iterative Aligner as input.

We used bidirectinal LSTM with 100 hidden units with a dropout of 0.2 to encode both the questions and the document.

2.2.2 Iterative Aligner

The iterative Aligner consists of an Interactive aligner, a Semantic Fusion Unit (SFU), and an Aggregator. This aligner works in a sequence of hops where, in each hop, word representations are updated by paying attention to the document as well as the question. Following this, the SFU performs the actual task of updating the representations. At the end, the Aggregating Unit urges the information to go through the entire document at the end of each hop.

Interactive Aligner Over the course of T hops, the iterative aligner described above reads the entire document as well as the question. In a given t^{th} hop, the above aligner attends to the document as well as the question and generates the corresponding representation once it summarizes how they interact between each other. With this, a co-attention matrix between the document and the question of order $n*m$ is created. The values in this matrix B_t is given as -

$$B_t = q_i^T . c_j^{t-1}$$

where q_i is the embedding of the question and c_j^{t-1} is the representation of the j^{th} word in the document in the previous hop. Each value in this matrix represents the similarity between the i_{th} word in the question and the j_{th} word in the document.

An attended question vector is calculated for each word in the document and combined with the already existing word in the document in order to find the word from the question that is most similar to the word in the document that is currently under consideration. Assume b_j^t be the n-dimensional vector that represents the normalized attention distribution for the j^{th} word in the document. Similarly, let q_j^t be a $2h$ dimensional vector of the query vector for the j^{th} word in the question. This value b_j^t is generated by applying a non-linearity (Softmax) activation to all the row values of the j^{th} column of the B_t matrix. The attended query vector is computed by taking dot product between the encoded question matrix and the above calculated b_j^t. m such question attended vectors are generated for each of the m words in the question. Similarly, in order to calculate a question-involved representation for the j^{th} word in the context, the representation of the same word from the previous hop, i.e. c_j^{t-1} is combined with the above calculated attended question vector and passed to the SFU. SFU takes four parameters, the document representation from the previous hop (c_j^{t-1}), the attended vector for the question (q_j^t), value obtained as a result of element-wise multiplication of c_j^{t-1} and q_j^t, and lastly, the result of element-wise subtraction of c_j^{t-1} and q_j^t.

Semantic Fusion Although the information gained from the attention mechanism has been generated, it is necessary to combine this with the original word. In order to achieve this task, Semantic Fusion Unit is used, which takes a set of k vectors called fusion vectors along with another vector r as an input and gives output o.

If the input vector r is d-dimensional, then the output vector as well as all the fusion vectors are also d-dimensional. The Semantic Fusion Unit mainly consists of two parts - composition and a gate. Composition generates a hidden state, which is nothing but the combination of the input vector r and the fusion vectors, given by $\tilde{r}$. This uotput is then passed as input to the 'gate', which controls the extent to which the hidden state is exposed, by means of an update gate g. Let W_r and W_g be weight matrices and b_r

and b_g be biases. Each of the matrices and biases are trainable. The final output vector that is generated from the SFU is then represented as-

$$\tilde{r} = \tanh(W_r([r; f_1, ..., f_k]) + b_r)$$
$$g = \sigma(W_g([r; f_1, ..., f_k]) + b_g)$$
$$o = g \text{ o } \tilde{r} + (1 - g) \text{ o } r$$

where $f_1, ..., f_k$ is the fusion vector and σ stands for sigmoid activation. Both tanh and σ are non-linear activation functions. o represents element-wise multiplication.

Once interactive aligning is done, we use iterative aligning in order to capture the long distance dependencies. In this phase, the representation of the document combined with question awareness are aligned with the representation of the document. This helps overcome the limited functionality of recurrent Neural Network of not being able to model long term dependencies that exist within the document [7]. Every word that exists is only aware of the words that exist around it and has no idea about the overall context of the document. The alignment that is done helps provide added insight which can be included as features along with the representation of the document. As done in interactive aligning, a co-attention matrix is created. However, this co-attention matrix reveals the similarity of the context words among themselves. This self co-attention matrix is given as-

$$\tilde{B_{ij}^t} = \bar{c_i^t}^T . \bar{c_j^t}, \text{ when } i \neq j$$
$$\text{else } 0$$

Now, an attended document-based vector can be created by computing dot product between the encoded matrix for the document with the normalized attention distribution of the j^{th} word of the document in the t^{th} hop. This document vector is then passed through an SFU where parameters would be the generated attended document vector, the question - aware document representation, result of element-wise multiplication between the two and result of element-wise subtraction between the two.

Aggregating After every hop, a Bi-Directional LSTM, similar to the one used in the Encoding stage is used to model the context representation generated in the above stage with it's document-related interaction. Here, the short-term information between the words in the document is described, out of the words in the document trained on all words in the question as well as the document. The final output therefore is a fully (context + question) representation of the document, which contains n values, one for each word out of n words in the document. This representation then acts as input to the next hop, or the answer pointing module, if it is the last hop.

2.2.3 Answer Pointing

We need to find the span of words in the document in order to find the answer to the question given along with the document. This can be generated by making predictions of the start (i) and end position (j) of the phrase containing the answer in the document, that is trained on the context as well as the question

The answer-pointer maintains a memory-based vector that keeps track of knowledge necessary for refining the predictions about the spans continuously. This vector is first set to the summary of the question and is updated regularly during the process of predicting. We set it to the last state of the encoded question summary. We used a total of 2 hops. In each l^{th} hop, the mechanism first attempts to attend over the fully aware representation of the document with the above specified memory-based vector, which produces the probability distribution for the start position of the answer span suing the pointer network specified by [5]. To merge the memory vector and fully aware context we use a 2-layer fully connected neural network with ReLU nonlinearity and dropout on the first hidden layer. Then Softmax is applied to generate the probability distribution for start position.

Let the memory vector be z_s^l. This memory vector can gain information from the evidence vector u_s^l in order to keep making itself better. The SFU used for refinement takes this evidence vector and memory vector as input and produces an updated memory vector given by

$$z_e^l : z_e^l = \text{SFU } (z_s^l, u_s^l)$$

Now, the probability distribution for the end position of the answer span is generated in the same way as generation of probability distribution of the start position of the answer span. If the l^{th} hop is not the last hop, then the output generated is the memory vector z_s^{l+1} which acts as input to the next hop.

3. Training

Our goal is to train the network to minimize the F1 score and Exact Match. F1 measures the degree of overlap between the answer generated by the model and the target answer provided as a part of the training set. As opposed to the Exact Match metric, which is a 'hard' metric, F1 score is a 'soft' metric.

To achieve this goal, we train minimize the sum of negative log-likelihood loss of start and end position i.e we are minimizing the following objective function:

$$J(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij}^s log p_{ij}^s + y_{ij}^e log p_{ij}^e$$

where N is the number of samples and K is the number of valid positions and y_i^s is one-hot encoded vector of true start

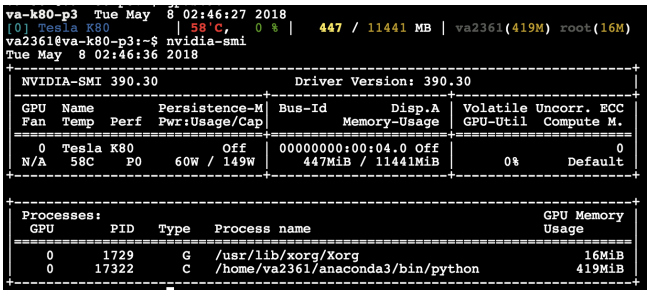

Figure 2. Final GPU details: K80

position of the i^{th} sample and p_{ij}^{s} is the predicted probability that the answer to the i^{th} question ends at j^{th} position

The gradients of all the parameters in the network were computed using Adamax optimizer with a learning rate of 0.002. Adamax is a variant of Adam optimizer based on infinity norm which is more stable when there are a large number of parameters. [3]. Further, we clip the parameter gradients to a maximum value of 10.

Once the network is trained, we choose the right answer to be the valid start and end position pair with maximum joint probability for start and end positions. We train the model for 14 epochs with a batch size of 45.

4. Evaluation

4.1. Dataset

We used the SQuAD dataset to train and evaluate our model. SQuAD is a Reading Comprehension dataset and contains questions obtained by means of crowd sourcing on articles from Wikipedia. An important thing to note is that answers to all questions exists in the passages as a sequence of words, i.e. a span. SQuAD has more than 100,000 pairs of questions and their corresponding answers taken from more than 50 articles and therefore is an exhaustive dataset to train the model on. The SQUAD dataset has multiple crowd - sourced answers for each of the questions, during training we only choose the first answer as the ground truth answer span but consider all the answer spans during evaluation.

4.2. Experimental Configuration

1. Python system configuration: The pip freeze captured our system state: as mentioned in requirements.txt

2. Conda version: 4.5.1

3. GPU server Server We experimented on GOU servers and found the convergence on NVIDIA P100 servers to be remarkably faster. Since it was more expensive, we shifted to Tesla K80 GPU on Ubuntu 16.04 Operating System, with CUDA-9.1 and cuDNN-7.1. Refer details in figure 2

3. The approach taken for tokenization and lemmatization of the questions and contexts played a huge role in shaping the F1 and EM scores of the experiments. Having selected NLTK as the choice of preprocessing library, there were challenges faced in parsing the unicode characters abounding the training file. The `nltk.word_tokenize` would create trouble in some special characters encountered. In those situations, we switched over to `nltk.wordpunct_tokenize`. However, if the the latter was used for parsing the entire dataset, the F1 and EM scores on the training dataset would decline by ∼2%. This was surprising given that tokenization is the very first window of actions performed on the dataset before the LSTM begins learning. We could potentially have tried other libraries, which might change the accuracies and give more insights into how the tokenization, lemmatization and tagging affects the system's ability to learn. Thus, we have evaluated the systems in 2 ways as given in the 2 tables below. Table 1 reports scores only on development examples that could be tokenized and Table 2 assigns a score of zero to examples that could not be tokenized (this is from the official evaluation)

4.3. Results

Method	F1	EM
with topics	**77.56**	**68.10**
without topics	76.42	67.00

Table 1. Results on data that could be tokenized

Method	F1	EM
with topics	**76.99**	**67.61**
without topics	76.42	67.00

Table 2. Results from official eval script

The original reinforced Mnemonic Reader has an EM score of 81.2 and F1 score of 87.9 on development dataset. Without the reinforcement learning approach, by training the model to minimize the total negative log likelihood loss we achieve best F1 and EM score of 77.56% and 68.103% respectively. Figure 3 shows how the training loss reduces over time and Figure 4 depicts how the F1 and EM score improve over each iteration.

We compare the performance of Mnemonic reader with and without topics in tables 1 and 2, and observe that adding topics slightly improves the performance of the model. We also observe that the answers generated by the topic-infused model in figure 6 are more verbose while those by the model without topic are crisp and to the point.

5. Conclusion

In this paper, we propose morpheme-infused bi-directional LSTMs which is an enhanced Mnemonic Reader

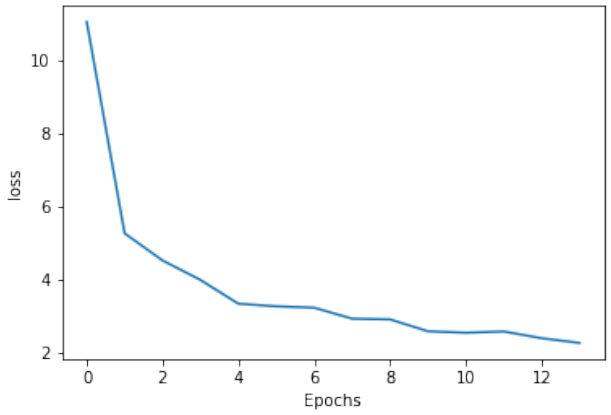

Figure 3. Negative log-likelihood loss on train data

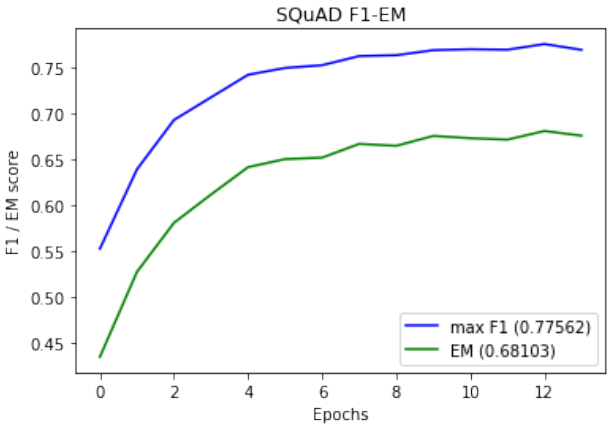

Figure 4. EM and F1 scores vs epoch on development data

Passage	In January 1880, two of Tesla's uncles put together enough money to help him leave Gospić for Prague where he was to study. Unfortunately, he arrived too late to enroll at Charles-Ferdinand University; he never studied Greek, a required subject; and he was illiterate in Czech, another required subject. Tesla did, however, attend lectures at the university, although, as an auditor, he did not receive grades for the courses.
Question	Why was he unable to enroll at the university?
Answer	he arrived too late to enroll at Charles-Ferdinand University; he never studied Greek, a required subject; and he was illiterate in Czech
Prediction – morpheme MR	he never studied Greek, a required subject
Prediction – MR	he never studied Greek, a required subject

Figure 5. Prediction on the questions - 1

used for Machine Comprehension for the purpose of question answering. The model takes care of context-based information such as overall summary, parts-of-speech tags, named-entity tags and then combines the maximum likelihood loss along with Adam Optimizer, which performs reasonable well on the SQuAD dataset. The morpheme enhancement novelty proposed is promising, as it improves both the F1 and EM score. Moreover, the answers are more verbose suggesting that it might be more useful for certain applications or as a valuable ensemble for certain question-types.

Passage	'In anglophone academic works, theories regarding imperialism are often based on the British experience. The term "Imperialism" was originally introduced into English in its present sense in the late 1870s by opponents of the allegedly aggressive and ostentatious imperial policies of British prime Minister Benjamin Disraeli. It was shortly appropriated by supporters of "imperialism" such as Joseph Chamberlain. For some, imperialism designated a policy of idealism and philanthropy; others alleged that it was characterized by political self-interest, and a growing number associated it with capitalist greed. Liberal John A. Hobson and Marxist Vladimir Lenin added a more theoretical macroeconomic connotation to the term. Lenin in particular exerted substantial influence over later Marxist conceptions of imperialism with his work Imperialism, the Highest Stage of Capitalism. In his writings Lenin portrayed Imperialism as a natural extension of capitalism that arose from need for capitalist economies to constantly expand investment, material resources and manpower in such a way that necessitated colonial expansion. This conception of imperialism as a structural feature of capitalism is echoed by later Marxist theoreticians. Many theoreticians on the left have followed in emphasizing the structural or systemic character of "imperialism". Such writers have expanded the time period associated with the term so that it now designates neither a policy, nor a short space of decades in the late 19th century, but a world system extending over a period of centuries, often going back to Christopher Columbus and, in some accounts, to the Crusades. As the application of the term has expanded, its meaning has shifted along five distinct but often parallel axes: the moral, the economic, the systemic, the cultural, and the temporal. Those changes reflect – among other shifts in sensibility – a growing unease, even squeamishness, with the fact of power, specifically, Western power.'
Question	According to Lenin why must capitalistic countries have an imperialistic policy?
Answer	need for capitalist economies to constantly expand investment, material resources and manpower in such a way that necessitated colonial expansion.
Prediction – morpheme MR	material resources and manpower in such a way that necessitated colonial expansion
Prediction – MR	constantly expand investment

Figure 6. Prediction on the questions - 2

For future work, a more detailed study would allow us to explore how the morpheme enhanced model compares with the baseline model for varied questions categories like 'why','how', which result in different lengths of answers. We would also like to explore possibly more robust ways of injecting morphemes into the model. Rather than appending them to the question directly, the word embedding of the topics can be concatenated to the encoded question and context representations like we have done in the case of other features.

6. Acknowledgements

We thank Prof. Sameer Maskey, Prof. Steven Rennie and our TAs, who have guided us throughout the project with their inputs and suggestions. Their insights have been very crucial in working on the problem at hand. We would also like to acknowledge the support of Columbia, Google and our instructors that made it easier to work with GPUs employed for this project.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003. 1

[2] L. S. Dirk Weissenborn, Georg Wiese. Making neural qa as simple as possible but not simpler. *arXiv:1703.04816*, 2017. 1

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[4] X. Q. Minghao Hu, Yuxing Peng. Reinforced mnemonic reader for machine comprehension. *arXiv:1705.02798*, 2017. 1

[5] N. J. Oriol Vinyals, Meire Fortunato. Pointer networks. *arXiv:1506.03134*, 2017. 1, 4

[6] J. S. Sepp Hochreiter. Long short term memory. *Neural Computation 9(8): 1735-1780*, 1997. 2

[7] P. F. Yoshua Bengio, Patrice Simard. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks 5(2):157166*, 1994. 4