Composing Sparsity in Match-LSTM for Machine Comprehension

Gurpreet Singh 150259
Saransh Bhatnagar 150637
Siddhant Garg 150711
Divyansh Singhvi 150238
Megha Agarwal 150403
Dushyant Kumar 150242

Abstract

Machine Comprehension is a fundamental problem in Natural Language. There are many datasets which provide wide range of corpus for this task. We intend to explore the most recently released dataset, the Stanford Question Answering Dataset (SQuAD, Rajpurkar et al., 2016), which offers a large number of real crowdsourced questions and passages. We try to explore a number of approaches to this task, particulary Match-LSTMS along with PointerNets, as proposed by Wang and Jiang (2016b) for the task of machine comprehension. Or main goal is to improve the scalability of the models by targetting the sparsity of the learned networks in an attempt to reduce the hypothesis space of our model, while trying to minimize loss of accuracy.

1. Introduction

Question Answering (or Machine Comprehension) is a fundamental problem in Natural Language Processing where we want the computer system to be able to understand a corpus of human knowledge and answer questions provided in Natural Language. This involves seemingly simple, but complex tasks such as understanding human language enough to interpret provided questions, and searching for an appropriate answer.

A distinguishing characteristic between humans and naive models is the ability of humans to quickly relate the passages to focus on subsets of the passage with demarcating words related to the task at hand. There exists many approaches to this problem.

There have been many approaches to solving the machine comprehension, however most lack the scalability and the lightness of the model. In our project, we intend to explore the high maintenance characteristic of these models, and attempt to apply sparsity inducing techniques in order to reduce the heaviness of the models.

2. Problem Statement

In this course project, we aim to explore the problem of Machine Comprehension (or Question Answering) by using Deep Learning based models and see how they perform on the Stanford Question Answering Dataset (SQuaD).

We state our problem statement similar to as stated in (Rajpurkar et al., 2016).

For each question-passage-answer tuple $(\mathbf{Q}, \mathbf{P}, \mathbf{A})$, where $\mathbf{Q} = (q_1 \dots q_M)$ are the tokens for the question, $\mathbf{P} = (p_1 \dots p_N)$ are the tokens for the passage, and $\mathbf{A} = (a_b, a_e)$ is the answer span, a_b and a_e representing the beginning and ending indices in the passage. The task is maximising the conditional probability $\mathbb{P}\left[\, \mathbf{A} = (a_b, a_e) \,\middle|\, \mathbf{Q}, \mathbf{P} \,\right]$, *i.e.* the probability of answer being correct, given the question and the passage or instead estimating $\mathbb{P}\left[\, \mathbf{A} \,\middle|\, \mathbf{Q}, \mathbf{P} \,\right]$ and thereby finding the answer

$$\mathbf{A}^* = \arg\max_{\mathbf{A}} \mathbb{P}\left[\, \mathbf{A} \,\middle|\, \mathbf{Q}, \mathbf{P} \,\right].$$

3. Dataset

We will use the SQuad (Rajpurkar et al., 2016) dataset, for the goal of studying techniques in Machine Comprehension.

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension consisting of 100,000+ question-answers corresponding to passages extracted from over 500 wikipedia articles, where the answer to each question is a continuous segment of text from that passage.

The dataset is roughly portioned in 80k test set, 10k training set and 10k validation set. SQuAD does not provide a list of answer choices for each question, as compared to other datasets. Also, unlike some other contemporary datasets whose questions and answers are created automatically in cloze style, the question and answers in SQUAD were created by humans through crowdsourcing making the dataset more realistic.

Another difference between SQUAD and cloze style queries is that answers to cloze style queries are single words or entities while answer in SQAD can be non-entities and can contain long phrase.

The squad dataset has a varied length of questions although most of them lie in the 5-15 word range. The context length is mostly distributed (evenly) between $20 - 250$ word range. The variety of the context length, makes fitting to some piece of the context difficult, which is a good characteristic of the dataset.

In additon, the span-based QA setting is quite natural. For many user questons into search engines, open-domain QA systems are often able to find the right documents that contain the answer. The challenge is the last step of "answer extracton", which is to find the shortest segment of text in the passage or document that answers the question [1].

[1] https://rajpurkar.github.io/mlx/qa-and-squad/

4. Relevant Background

4.1 Long Short Term Memory

Long Short Term Memory (LSTM, Hochreiter and Schmidhuber, 1997) networks are a special kind of Recurrent Neural Networks (RNNs), capable of learning long-term dependencies. LSTMs are explicitly designed to avoid long-term dependency problem, for example, for predicting a word after observing a sequence of words history, we might need the context of the words observed much before the 'to be predicted' word.

This requires a memory model, which vanilla RNNs are incapable of handling. Remembering information for long periods of time is default behavior of the LSTM networks, which is handled using states and the controlling the flow of information is handled using gating networks. LSTMs, therefore, have the ability to remove or add information to the cell state, regulated by gates or gated networks. The structure of a simple LSTM network is given in Figure 1

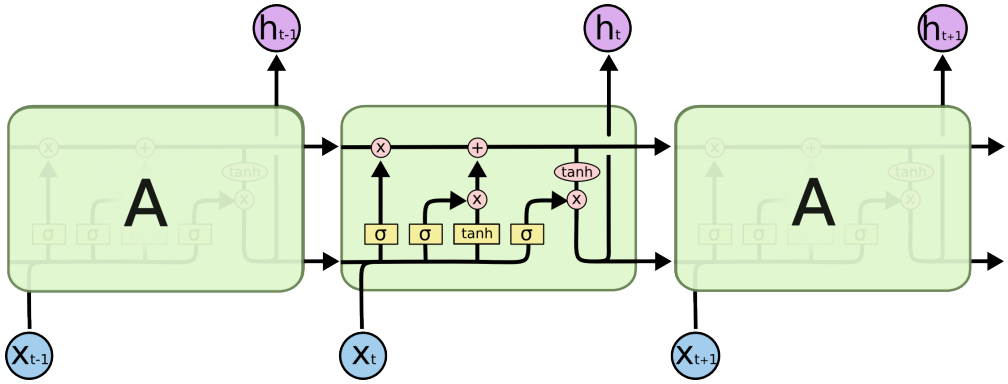


Figure 1: Insides of a LSTM Network. *(Source: Olah, 2015)*

Note. For the further discussion, we borrow the notations and description from Olah (2015)

4.1.1 Layers of a LSTM

We define the state of the LSTM at a time state t as C_t. A LSTM as described above consists of four layers that we describe as below.

1. **Forget Gate Layer** *Forget Layer* decides what information we are going to throw away from cell by looking at previous cell state and the current data point (or word for the task of machine comprehension). The forget layer uses a sigmoid function to determine the amound of information to pass or to forget. The computed vector (or matrix) is represented by f_t

2. **Input Gate Layer** What new information we are going to store in the cell state is decided by a sigmoid and a tanh layer. Sigmoid layer called the *input gate layer* decides which values we'll update, and to what extent, and the tanh layer creates a vector of new candidate values, $\widetilde{C}_t$, that will be added to the state.

3. **Update Layer** Updation of the state is carried out by multiplying the old state C_{t-1} by f_t, the output from the forget layer, and adding the update values $\widetilde{C}_t$ to get the new state, *i.e.* C_t.

4. **Ouput Layer** The *Output Layer* handles the updation of the output label, using the new state, the input data and the old label. First, a sigmoid layer is run throuth the *concatenation* of the old label h_{t-1} and the data point. Then, the current (updated) cell state is passed through a tanh layer and multiplied to the output of the sigmoid gate to get the new label / output h_t.

A detailed look at the states of the LSTM is given in Figure 2.

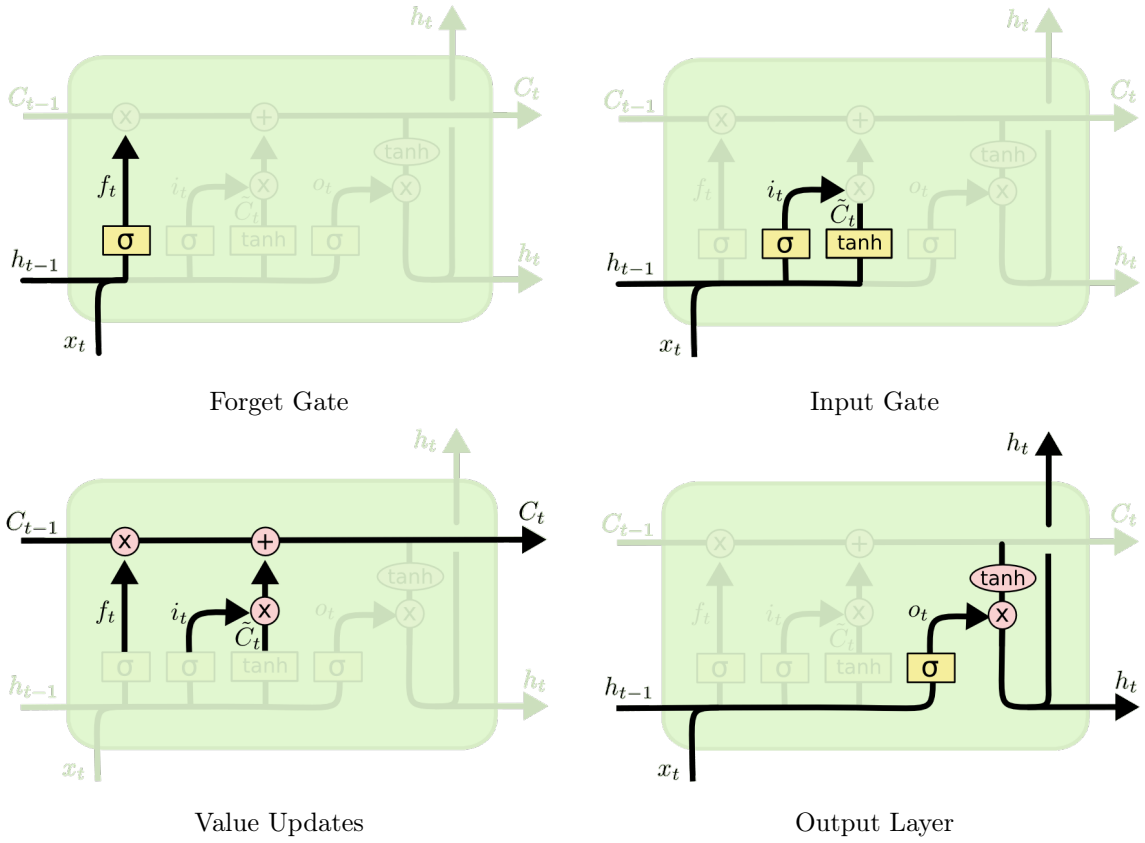
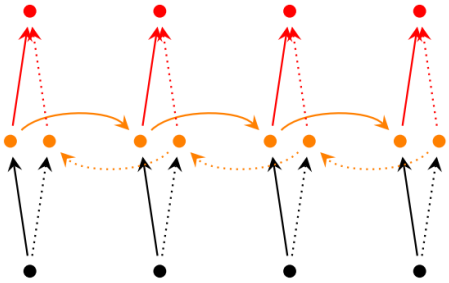


Figure 2: Working of a simple LSTM. *(Source: Olah, 2015)*

4.1.2 Bidirectional LSTMs

Bidirectional RNNs (BRNNs, Britz, 2015), which are the base of bidirectional LSTMs, are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements, for example, word prediction, such as filling in a blank within a sentence, might benefit from including the post blank words into consideration while prediction the current word.

Bidirectional RNNs, in their essence, are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs. Combing BRNNs with LSTM gives bidirectional LSTM which can access long-range context in both input directions.



(a) Bidirectional RNN

includes/bidirectional-lstm.png

(b) Bidirectional LSTM

Figure 3: Biderectional RNNs (left) and LSTMs (right) *(Source: Britz, 2015)*

4.1.3 Match-LSTMs

Match-LSTMs (Wang and Jiang, 2016a) are an extension to basic LSTMs, introduced by *Wang and Jiang* for the purpose of textual entailment, and later used the same in machine comprehension to achieve state-of-the-art results on the SQuAD (at the time of publication).

Match-LSTM attempts to recover an sequence of positions within a context paragraph using the contents of the context and an associated question (Graczyk, 2017).

To verify textual entailment, match-LSTM goes through the tokens of the hypothesis sequentially. And for each position, attention mechanism is used to obtain a weighted vector representation of the premise. This weighted premise combined with current token of hypothesis fed into match-LSTM.

The match-LSTM model essentially sequentially aggregates the matching of the attention weighted premise to each token of the hypothesis and uses the aggregated matching result to make a final prediction.

Details of the model are given in the papers Wang and Jiang (2016a) and Wang and Jiang (2016b)

5. Previous Works

Traditional solutions question answering tasks relied on NLP pipelines that involved multiple steps of linguistic and lexical analysis, including syntactic parsing, named entity recognition, question classification, semantic parsing, etc. In these approaches each layer of parsing added its own set of errors or loss which propogated over pipelines to subtle failures that required a lot of manual tweaking to get to the right results.

Existing end-to-end neural models assume that the answer is a single token making them unsuitable for use in SQuAD, which expects a sequence based answers. Hence, we require new and more advanced models for

machine comprehension tasks. We have described a few of them below.

Most of the state-of-the-art approaches have the following settings in common, which we, as well, will extend in our approach as an attempt to solve the problem of question answering in SQuAD.

1. Pre trained GLoVe (Pennington et al., 2014) vectors are used as embeddings for each word, therefore formign the word embedding layer.
2. This word embedding layer is connected through LSTM which could of different types like vanilla LSTM, bidirectional LSTM, match-LSTM, etc. to develop a context based embedding layer.
3. This layer is followed by attention flow layer which is used to make the model aware of the query. The paragraph and query are both processed through the attention mechanism to generate an interdependent context, therefore learning a query aware latent representing of the passage.
4. This latent representation is therefore used to predict the answer spans using various strategies, some of which we discuss briefly in the following sections.

5.1 Bi-Directional Attention Flow for Machine Comprehension

The BiDAF (Seo et al., 2017) model is a hierarchical multi-stage process which consists of six layers. The comprehensive model is given in Figure 4.

1. **Character Embedding Layer.** Character embedding layer is responsible for mapping each word to a high-dimensional vector space. We obtain the character-level embedding of each word using Convolutional Neural Networks (CNN) (Kim, 2014).
2. **Word Embedding Layer.** Word embedding layer also maps each word to a high-dimensional vector space. Pre-trained GloVe vectors, are used to obtain the fixed word embedding of each word.
3. **Contextual Embedding Layer.** This layer model the temporal interactions between words. For this, LSTM is placed in both directions and concatenate the outputs of the two LSTMs, thus obtaining $\mathbf{H}$ from context word vectors $\mathbf{X}$ and $\mathbf{U}$ from query word vectors $\mathbf{Q}$.
4. **Attention Flow Layer.** Attention flow layer is responsible for linking information from context and query words. The inputs to the layer are contextual vector representations of the context $\mathbf{H}$ and the query $\mathbf{U}$. The outputs of the layer are the query-aware vector representations of the context words, $\mathbf{G}$, along with the contextual embeddings from the previous layer. Shared similarity matrix, $\mathbf{S}$ is used to obtain the attentions and the attended vectors in both direction.

   The similarity matrix $\mathbf{S}$ is used to obtain the attentions and the attended vectors in both direction, *i.e. Context-to-Query* Attention and *Query-to-Context* Attention.
5. **Modeling Layer.** The input to the modeling layer is $\mathbf{G}$, which encodes the query-aware representations of context words by using two layered bi-directional LSTM.

   The output of the modeling layer captures the interaction among the context words conditioned on the query. The output of the modelling layer is $\mathbf{M}$.
6. **Output Layer.** This layer predict the start and the end indices of the phrase in the paragraph to answer the query. It outputs probability distribution of the start index, p^1 over the entire paragraph by using $\mathbf{G}$ and $\mathbf{M}$ and predicts the probability distribution of the end index, p^2 by using $\mathbf{G}$ and M^2, where M^2 is calculated by passing $\mathbf{M}$ through another bidirectional LSTM.

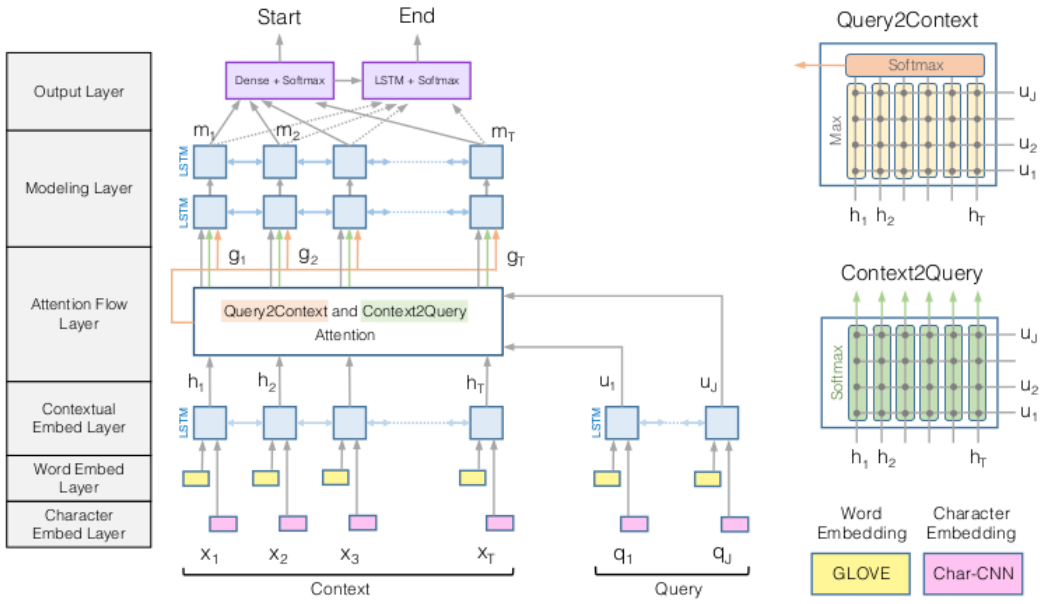


Figure 4: Bi-Direction Attention Flow Model *(Source: Seo et al., 2017)*

5.2 FastQA

The FastQA (Weissenborn et al., 2017) model consists of three basic layers, namely the embedding, encoding and answer layer. The FastQA model is given in 5

1. **Embedding Layer.** It computes the embedding of tokens by concatenating lookup-embedding and char-embedding.
2. **Encoding Layer.** This layer computes the query aware context embedding by concatenating earlier embedding, word-in-question features. The word-in-question features determine whether the context words are present in query or not and how similar they are to question tokens. Then these query aware context embedding is fed to a bidirectional RNN to allow for interaction between the features accumulated in the forward and back-ward RNN.
3. **Answer Layer.** After encoding context and question tokens, the probability distribution p_s for the start location of the answer is computed by a feed-forward neural network and then the conditional probability distribution p_e for the end location conditioned on the start locations is computed similarly by a feed-forward neural network.

   The overall probability p of predicting an answer span (s, e) is $p(s, e) = p_s(s) \cdot p_e(e|s)$. The model is trained to minimize the cross-entropy loss of the predicted span probability $p(s, e)$.

5.3 R-Net: Machine Reading Comprehension with Self-Matching Networks

R-Net (Research Asia, 2017) is an end-to-end neural network model for reading comprehension and question answering. This model consists of four parts:

1. The recurrent network encoder to build representation for questions and passages separately,

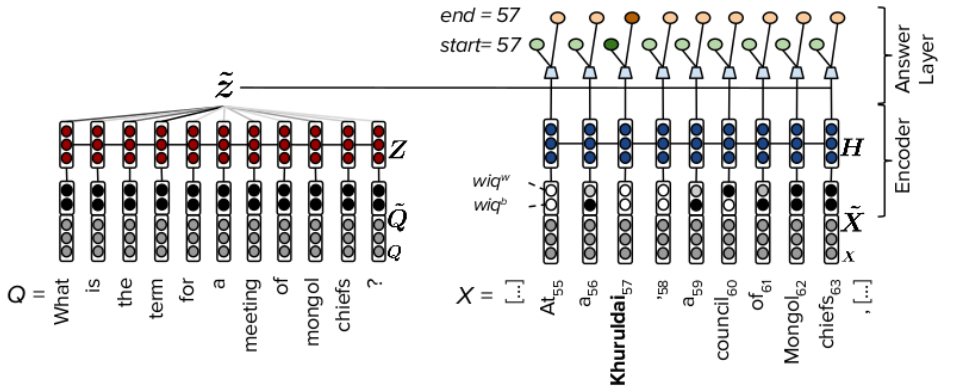


Figure 5: Illustration of FastQA system *(Source: Weissenborn et al., 2017)*

2. the gated matching layer to match the question and passage,
3. the self-matching layer to aggregate information from the whole passage, and
4. the pointer-network based answer boundary prediction layer.

A view on the R-Net model is given in Figure 6.

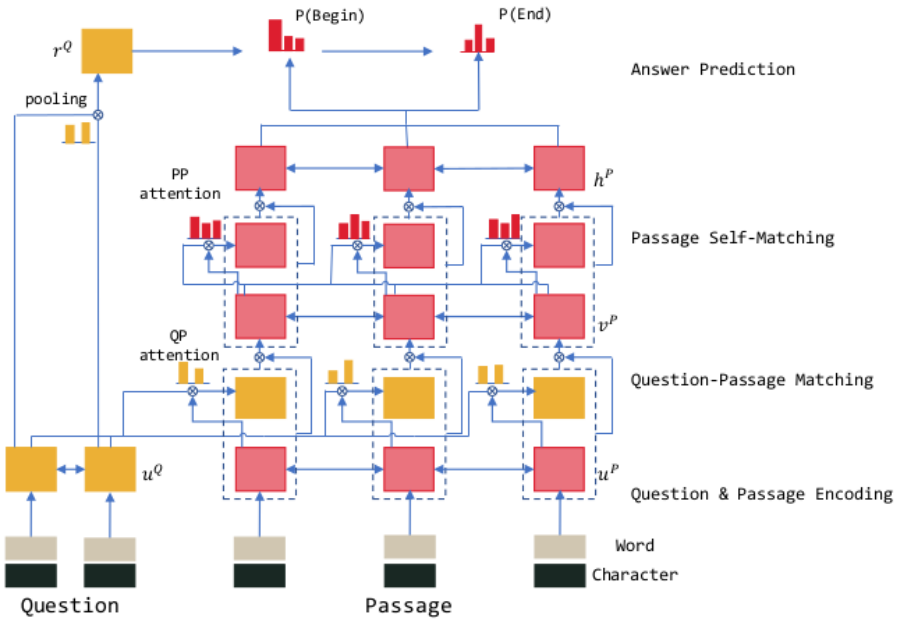


Figure 6: R-NET structure overview *(Source: Research Asia, 2017)*

5.3.1 R-Net Structure

First, the question and passage are processed by a bidirectional recurrent network separately. We then match the question and passage with gated attention-based recurrent networks, obtaining

question-aware representation for the passage. On top of that, we apply self-matching attention to aggregate evidence from the whole passage and refine the passage representation, which is then fed into the output layer to predict the boundary of the answer span.

5.3.2 Gated Attention Based RNNs

A gated attention-based recurrent network is used to incorporate question information into passage representation. It is a variant of attention-based recurrent networks, with an additional gate to determine the importance of information in the passage regarding a question. Different from the gates in LSTM or GRU, the additional gate is based on the current passage word and its attention-pooling vector of the question, which focuses on the relation between the question and current passage word.

5.3.3 Self-Matching Attention

It is a directly matching the question-aware passage representation against itself. It dynamically collects evidence from the whole passage for words in passage and encodes the evidence relevant to the current passage word and its matching question information into the passage representation. Self-matching extracts evidence from the whole passage according to the current passage word and question information.

5.3.4 Output Layer

Pointer networks are used to predict the start and end position of the answer. r. In addition, we use an attention-pooling over the question representation to generate the initial hidden vector for the pointer network. Given the passage representation, the attention mechanism is utilized as a pointer to select the start position p^1 and end position p^2 from the passage.

The R-Net model is the current state-of-the-art model for machine comprehension on SQuAD. However, we refrain from studying the model due to its complexity and the paper's involved nature.

5.4 Question Answering using Match-LSTM and Answer Pointer

Wang and Jiang (2016b) introduced the Match-LSTM (Wang and Jiang, 2016a) model for textual entailment, however, it proved to be useful for the task of machine comprehension as well, with an extra extension using PointerNet (Vinyals et al., 2015).

In the model proposed by *Wang and Jiang*, the architecture consists of three main layers

1. **LSTM Preprocessing Layer.** The purpose of the *LSTM Preprocessing Layer* is to incorporate contextual information into the representation of each token in the passage and the question. This is done by passing the passage matrix, $\mathbf{P} \in \mathbf{R}^{d \times Q}$ and the question matrix, $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ through a one-directional standard LSTM. The authors represent this as

$$\mathbf{H}^p = \overrightarrow{\text{LSTM}}(\mathbf{P}), \qquad \mathbf{H}^q = \overrightarrow{\text{LSTM}}(\mathbf{Q})$$

   The matrices $\mathbf{H}^p$ and $\mathbf{H}^q$ represent the hidden representations of the passage and the question matrices, respectively.

   Note. Q and P are the sizes of or the number of tokens in the question and the passage, respectively

2. **Match-LSTM Layer.** Treating the question as a premise and the passage as the hypothesis, we can apply the Match-LSTM model. The model first uses the standard word-by-word attention mechanism to obtain attention weight-vector $\boldsymbol{\alpha}_i \in \mathbb{R}^Q$ using detailed steps over an additional layer of LSTM. The detailed steps to obtain the attention vector are given in the paper, Wang and Jiang (2016b).

   This attention is iteratively used to estimate the bidirectional hidden representation of the passage, which in turn is used to compute the attention itself. This iterative procedure ensures that the representation we have for the passage is query aware, and is built using the attention obtained from the query (question). This representation is denoted by the matrix $\mathbf{H}^r$

3. **Answer Pointer Layer.** The final layer is the *Answer Pointer Layer*, which uses the idea of Pointer Networks Vinyals et al. (2015). This model takes in, as input, the hidden representation $\mathbf{H}^r$, and tries to identify the answer within the passage using two approaches, the *sequence model*, which tries to determine the complete sequence and probabilistically models the whole sequence, and the *boundary model*, which is only concerned with the starting and ending tokens of the answer. However, we are only interested in the boundary approach, as it performs better, and better suits the nature of PointerNets.

   The passage (hidden representation $\mathbf{H}^r$) is first passed through the attention mechanism to compute a attention vector $\boldsymbol{\beta}_k \in \mathbb{R}^{P+1}$, where $\beta_{j,k}$ is the probability of selecting the j^{th} passage token as the k^{th} token in the answer. The probability of the answer is then modelled as

$$\mathbb{P}\left[\mathbf{a} \,\middle|\, \mathbf{H}^r\right] \quad = \quad \mathbb{P}\left[a_s \,\middle|\, \mathbf{H}^r\right] \mathbb{P}\left[a_e \,\middle|\, a_s, \mathbf{H}^r\right]$$

   where

$$\mathbb{P}\left[a_k = j \,\middle|\, a_1, a_2 \ldots a_{k-1}, \mathbf{H}^r\right] \quad = \quad \beta_{j,k}$$

   Hence, the objective is to maximize $\mathbb{P}\left[\mathbf{a} \,\middle|\, \mathbf{H}^r\right]$, and the answer corresponding to the maximum answer is reported as the answer to the query.

Although the Match-LSTM model works well, the problem with this approach (rather problem with using PointerNets) is that answers which are long, *i.e.* have relatively more number of tokens, are predicted with lesser accuracy. However, one of the key benefits of using Match-LSTM is that it is a simpler model (lesser model complexity) than the other state-of-the-art models, and therefore, might be more scalable than other models.

5.5 Comparision of Different Approaches

In Table 1, we have given the EM (Exact Match) and F1 scores [2]

Note. The scores are given for single models only, and do not involve ensembles, however for most models, ensembles perform better that single models.

6. Goals of the Project

In our project, we will explore the various approaches that have been used for machine comprehension, and try to study the scalability and the performance of each model. The problem with most approaches discussed is the low speed of convergence, given the enormous number of weights in the models.

[2] The description of the scores is given by Rajpurkar et al. (2016)

Model	Training Set		Test Set	
	EM	F1	EM	F1
LR Baseline (Rajpurkar et al., 2016)	40.0	51.0	40.4	51.0
BiDAF (Seo et al., 2017)	68.0	77.3	68.0	77.3
FastQA (Weissenborn et al., 2017)	-	-	68.4	77.1
R-Net (Research Asia, 2017)	72.3	80.6	72.3	80.7
Match-LSTM (Wang and Jiang, 2016b)	64.1	73.9	64.7	73.7

Table 1: Comparision of different models for Machine Comprehension *(Source: Research Asia, 2017)*

We'll try to adopt model compression based on removing redundant structure in original DNNs (Wen et al., 2017), to provide scalability and wide adoption corresponding to limited resources. We will attempt to extend the Match-LSTM and Answer-Pointer model used by *Wang and Jiang*, considering it is a relatively simple model, and apply Sparsity Reduction on them, hoping to significantly reduce the training time, and to make the model simpler.

Pruning approaches produce non-structurally sparse LSTMs which provide comparatively lower speedups in comparison to structurally sparse lstms. A visual comparision is given in 7. Therefore we'll use group lasso regularization in order to obtain structurally sparse LSTMs, as explained by Wen et al. (2017). It can both reduce parameters in models and obtain regular nonzero weights for fast computation (non structural / irregular pattern of non-zero weights may not be friendly for computation efficiency.)

We also aim to reduce the number of basic structures simultaneously during learning such that the original schematic and dense structure of the lstms are retained along with reducing sizes of these basic structure.

To start we'll try to Identify structure inside the LSTMs that shall be considered as a group to most effectively explore sparsity in these basic structure. The basic structures interweave with each other, independently removing these structures can result in mismatch of their dimensions and then inducing invalid recurrent units. Therefore to deal with this we include Intrinsic Sparse Structure within LSTMs.

To learn sparse ISS, we turn to weight sparsifying. We can remove one component of ISS by zeroing out all associated weights, lets say k-th hidden state of h is removable, now all the connection receiving these stages can be removed. However, the number of weights in one ISS weight group can be very large, for avoiding overfitting and maintaining original recognition we can independently sparsify each weight using l1-norm regularization. Decreasing the size of ISS simultaneously reduce the dimensions of basic structures.

Wen et al. have explained this pruning technique in detail in their paper, where they analyze the BiDAF model (Seo et al., 2017). We wish to explore the application of the same approach on a heavier and simpler model, and find the trade off between the performance drop, and the speed boost.

If time allows, we will explore further extensions to the model, such as exploring techniques to find better word embeddings, to allow more information embedding in the word vectors, for example, including the genre of the passage provided to us, or adding semantic features within the word embeddings.

Another problem with Match-LSTMs (which R-Net handles efficiently) is that all question types have the same attention mechanism.If time allows, we will also look into tweaking the model to deal with different types of questions differently, particularly tweaks in the attention mechanism, *i.e.* for different question types, such as 'Why?', 'What?', etc. This is relevant as different question types demand different format of the answers, for example a 'Who?' question would focus more on the subject or the object of the sentence. This might increase the performance of the Match-LSTM model, however, will also add to the complexity of the model, and therefore the training time.

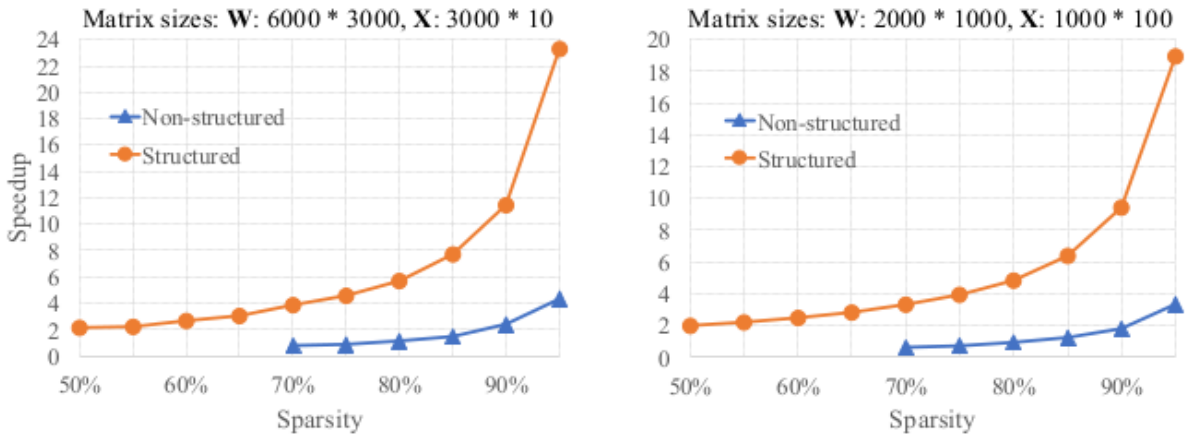


Figure 7: Speedups of Matrix Multiplication using Non-Structured and Sparsity *(Source: Wen et al., 2017)*

References

Denny Britz. Recurrent neural networks tutorial, 2015. URL `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns`.

Michael Graczyk. Implementation and analysis of match-lstm for squad, 2017. URL `https://web.stanford.edu/class/cs224n/reports/2761882.pdf`.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Yoon Kim. Convolutional neural networks for sentence classification. *EMNLP*, 2014.

Christopher Olah. Understanding lstm networks, 2015. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv*, abs/1606.05250, 2016.

Microsoft Research Asia. R-net: Machine reading comprehension with self-matching networks, 2017. URL `https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf`.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhad, and Hananneh Hajishirzi. Bi-direction attention flow for machine comprehension. *arXiv*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *ArXiv e-prints*, 2015. URL `https://arxiv.org/abs/1506.03134`.

Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. *CoRR*, abs/1512.08849, 2016a. URL `http://arxiv.org/abs/1512.08849`.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016b. URL `http://arxiv.org/abs/1608.07905`.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural qa as simple as possible but not simpler. *arXiv*, 2017.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning intrinsic sparse structures within long short-term memory. *CoRR*, abs/1709.05027, 2017. URL `http://arxiv.org/abs/1709.05027`.