Machine Comprehension

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

Abstract

Machine Comprehension is a fundamental problem in Natural Language. A recently released dataset, the Stanford Question Answering Dataset (SQuAD), offers a large number of real questions and their answers created by humans through crowdsourcing. SQuAD provides a challenging testbed for evaluating machine comprehension algorithms, partly because compared with previous datasets, in SQuAD the answers do not come from a small set of candidate answers and they have variable lengths. In this project we implemented two models First we implemented Match-LSTM model using answer pointer using boundary model. Then we also implemented another model based on Scaled-Dot Product Attention and Multi-Head Attention. The later model gives better results.

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. And to solve this problem Attention mechanism is used which allows modeling of dependencies without regard to their distance in the input or output sequences.

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 incorporate

In this work we propose the Match-LSTM based on clustering of question type which optimizes the choice of hyper parameters and subsequently also modelled a different architecture for Q&A based on Multi-Head attention. Further, to deal with the challenge of varying answer length Answer Pointer technique is incorporated in the output layer.

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 (5).

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 have used the SQuAd (5) 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 SQuAD can be non-entities and can contain long phrase.

In order to understand the dataset in more detail and motivate further design and hyper parameter decision, We perform some basic analysis on the dataset in the form of histograms of the context length, question length, and the answer length in the training set.

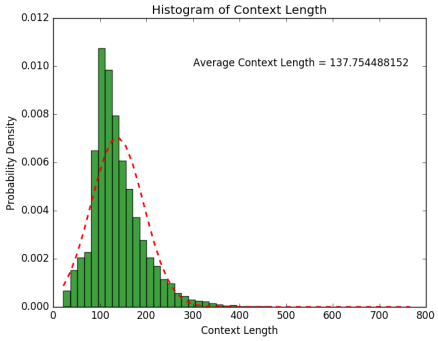


Figure 1

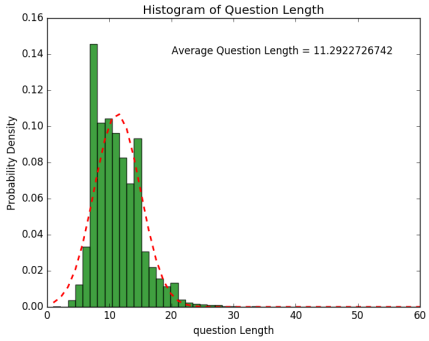


Figure 2

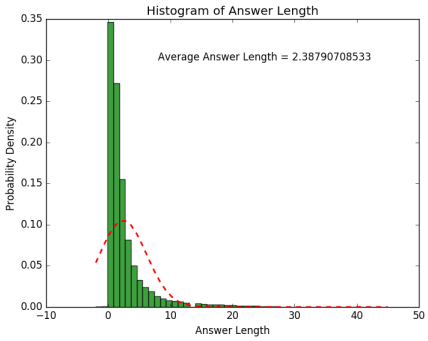


Figure 3

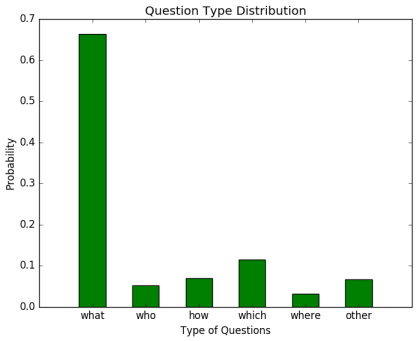


Figure 4

From the histogram of context lengths(Figure 1), the majority of context are shorter than 300 words. Hence it is possible to set the maximum context length to around 300 without losing to much information but gaining a lot in terms of memory and speed efficiency. The maximum question length can be set to 30 and maximum answer length of 10 words in the same way.

The histogram for answer lengths(Figure 2) show that the majority of answers are less than 10 words. This helps us determining answer span at test time as we could limit the search for answer span less than 10 words thus preventing model to predict unnecessarily long span.

We also examine the distribution of different question types like 'what', 'who', 'how', 'which', 'where', and other type include 'when' and 'why' type question. As we can see from the graph(Figure 4) 'what' type question account for largest proportion in 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].

4. Relevant Background

4.1 Long Short Term Memory

Long Short Term Memory (LSTM, 3) 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.

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

4.1.1 Bidirectional LSTMs

Bidirectional RNNs (BRNNs, 1), 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.

4.1.2 Match-LSTM

Match-LSTMs (10) 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 (2).

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 (10) and (11)

4.2 Attention Mechanism

Most of the sequence transduction models comprise of encoder-decoder configuration that are based on complex recurrent or convolutional neural networks. However, The efficiency and model performance can be boosted through adding attention mechanism between encoder and decoder layers.

In the case of machine comprehension, attention mechanism involves creating an attention weighted representation of the context using the question as the test and correspondingly comprehension as hypothesis. The attention weighted context representation acts as input for decoder and subsequently the pointer network also uses the same attention mechanism to compute the probabilistic expected answer tokens to the query.

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, did extend in our approach as an attempt to solve the problem of question answering in SQuAD.

1. Pre trained GLoVe (4) vectors are used as embeddings for each word, therefore forming 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 FastQA

The FastQA (12) model consists of three basic layers, namely the embedding, encoding and answer layer. The FastQA model is given in **??**

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 $p(s, e)$.

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

R-Net (6) 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,
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.

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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.

5.3 Question Answering using Match-LSTM and Answer Pointer

(11) introduced the Match-LSTM (10) model for textual entailment, however, it proved to be useful for the task of machine comprehension as well, with an extra extension using PointerNet (9). In textual entailment, two sentences are given where one is a premise and the other is a hypothesis. To predict whether the premise entails the hypothesis, the match-LSTM model goes through the tokens of the hypothesis sequentially. At each position of the hypothesis, attention mechanism is used to obtain a weighted vector representation of the premise. This weighted premise is then to be combined with a vector representation of the current token of the hypothesis and fed into an LSTM, which we call the match-LSTM. The matchLSTM 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.
Detailed analysis and implementation of layers have been discussed in section 6.1.

5.4 Comparision of Different Approaches

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

Model	Training Set		Test Set	
	EM	F1	EM	F1
LR Baseline (5)	40.0	51.0	40.4	51.0
BiDAF (7)	68.0	77.3	68.0	77.3
FastQA (12)	-	-	68.4	77.1
R-Net (6)	72.3	80.6	72.3	80.7
Match-LSTM (11)	64.1	73.9	64.7	73.7

Table 1: Comparision of different models for Machine Comprehension *(Source: 6)*

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. Our Approaches

In our project, we explored various approaches that have been used for machine comprehension, and analysed 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.

Another problem with Match-LSTMs (which R-Net handles efficiently) is that all question types have the same attention mechanism. Subsequently we transformed the proposed 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. We now give the details of the Match-LSTM model that we implemented

6.1 Match-LSTM with Answer Pointer

We adopted implementation of Match-LSTM and Answer Pointer 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.

[2] The description of the scores is given by (5)

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. At the position i, of the passage, the model first uses the standard word-by-word attention mechanism to obtain attention weight-vector $\boldsymbol{\alpha}_i \in \mathbb{R}^Q$ as follows:

$$\boldsymbol{G_i} = tanh(\mathbf{W}^q\mathbf{H}^q + (\mathbf{W}^p\mathbf{h}_i^p + \mathbf{W}^r\mathbf{h}_{i-1}^r + \mathbf{b}^p) \otimes e_{\mathbf{Q}})$$
$$\overrightarrow{\alpha_i} = softmax(\mathbf{w}^T\overrightarrow{\mathbf{G}_i} + b \otimes \mathbf{e}_Q)$$

where $\mathbf{W}^q$, $\mathbf{W}^p$, $\mathbf{W}^r$, $\in \mathbb{R}^{l \times l}$, $\mathbf{b}, \mathbf{w} \in \mathbb{R}^l$ and $\mathbf{b} \in \mathbb{R}$ are the parameters to be learned. $\overrightarrow{\mathbf{h}_{i-1}^r} \in \mathbb{R}^l$ is the hidden vector at position i-1, and the outer product $(\cdot \otimes e_Q)$ produces a matrix or row vector by repeating the vector or scalar on the left for Q times.

Essentially, the resulting attention weight $\overrightarrow{\alpha_{i,j}}$ above indicates the degree of matching between the i^{th} token in the passage with the j^{th} token in the question. The attention weight vector α_i is used to weighted version of the question and combine it with the current token of the passage to form a vector $\overrightarrow{\mathbf{z}_i}$:

$$\overrightarrow{\mathbf{z}_i} = \begin{bmatrix} \mathbf{h}_i^p \\ \mathbf{H}^q\overrightarrow{\alpha_i}^T \end{bmatrix}$$

This vector $\overrightarrow{\mathbf{z}_i}$ is fed into a standard one-directional LSTM to form our so-called match-LSTM:

$$\overrightarrow{\mathbf{h}_i^r} = \overrightarrow{\text{LSTM}}(\overrightarrow{\mathbf{z}_i}, \overrightarrow{\mathbf{h}_{i-1}^r}),$$

where $\overrightarrow{\mathbf{h}_i^r} \in \mathbb{R}^l$.

A Similar Match-LSTM layer is build in the reverse direction. The purpose is to obtain a representation that encodes the contexts from both directions for each token in the passage. To build this reverse match-LSTM, we first define

$$\overleftarrow{\mathbf{G}_i} = tanh(\mathbf{W}^q\mathbf{H}^q + (\mathbf{W}^p\mathbf{h}_i^p + \mathbf{W}^r\overleftarrow{h}_{i+1}^r\mathbf{b}^p) \otimes \mathbf{e}_Q),$$
$$\overleftarrow{\alpha_i} = softmax(\mathbf{w}^T\overleftarrow{\mathbf{G}_i} + b \otimes \mathbf{e}_Q),$$

where the parameters are $(\mathbf{W}^q, \mathbf{W}^p, \mathbf{W}^r, \mathbf{b}^p, \mathbf{w}\ and\ b)$ $\mathbf{z}_i$ and $\mathbf{h}_i$ are defined in similar manner. Let $\overrightarrow{\mathbf{H}^r} \in \mathbb{R}^{l \times P}$represents the hidden states $[\overrightarrow{\mathbf{h}_1^r}, \ldots, \overrightarrow{\mathbf{h}_P^r}]$. Similarly $\overleftarrow{\mathbf{H}^r} \in \mathbb{R}^{l \times P}$ represents $[\overleftarrow{\mathbf{h}_1^r}, \ldots, \overleftarrow{\mathbf{h}_P^r}]$. Define $\mathbf{H}^r \in \mathbb{R}^{2l \times P}$ as the concatenation of the above two:

$$\mathbf{H}^r = \begin{bmatrix} \overrightarrow{\mathbf{H}^r} \\ \overleftarrow{\mathbf{H}^r} \end{bmatrix}$$

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 (9). 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 $\boldsymbol{\beta}_k \in \mathbb{R}^{P+1}$,

$$\begin{aligned}\boldsymbol{F}_k &= \tanh\left(\mathbf{V}\mathbf{H}^r + (\mathbf{W}^a \boldsymbol{h}^a_{k-1} + \boldsymbol{b}^a) \otimes \boldsymbol{e}_P\right)\\ \beta_k &= \text{softmax}\left(\boldsymbol{b}^{\mathrm{T}} \boldsymbol{F}_k + c \otimes \boldsymbol{e}_P\right)\end{aligned}$$

   Where $\boldsymbol{h}^a_{k-1}$ represents the last hidden state of the answer pointer, projections and parameter matrices dimensions are given by $\mathbf{H}^r \in \mathbb{R}^{2l\times(P+1)}, \mathbf{V} \in \mathbb{R}^{l\times 2l}, \mathbf{W}^a \in \mathbb{R}^l, \mathbf{b}^a, \mathbf{v} \in \mathbb{R}^l$ and $c \in \mathbb{R}$. $\beta_{j,k}$ 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.

Modifications in Answer Pointer Layer In order to differentiate the question types i.e. - what? where? why? how? etc., we setup different Answer Pointer layers for different question types. Training the model based on the clusters on query level allows us to modify the hyper parameters to increase the efficiency of the model.

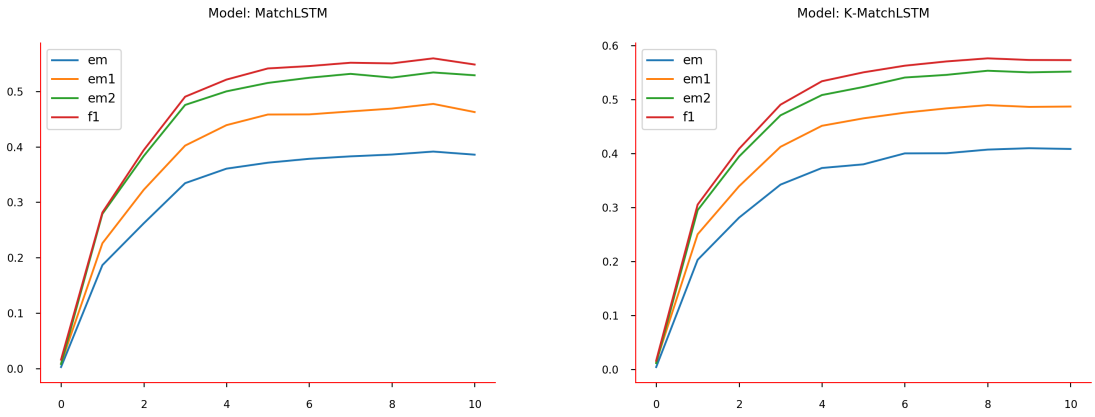
Analysis of the efficiency scores in the figure 5 provides an evidence that without any increase in the training time our model performed better in comparison to base model.

Another heuristic approach is applied in order to improve the test accuracy Since the loss for start and end tokens is computed individually, we weight each loss with the weights proportional to the inverse of the exact match at that token. This significantly improved the convergence and accuracy. The new loss is given as

$$-w_1 \cdot \log(p)\,(a_s \,|\, \mathbf{P}_n, \mathbf{Q}_n) - w_2 \cdot \log(p)\,(a_e \,|\, \mathbf{P}_n, \mathbf{Q}_n)$$

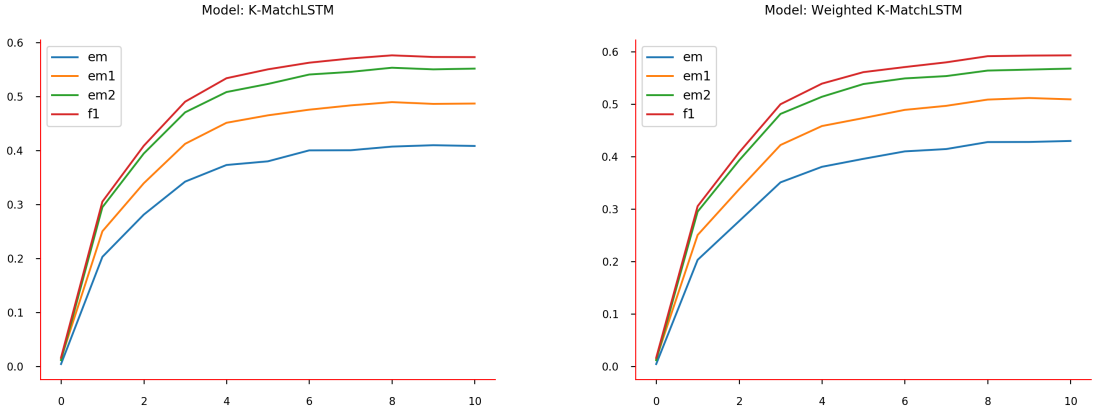
where

$$w_1 = 2\frac{em2}{em1 + em2} \quad w_2 = 2\frac{em1}{em1 + em2}$$



(a) Match-LSTM (b) K-Match-LSTM

Figure 5: Accuracy Analysis - Match-LSTM (left) and K-Match-LSTM (right)



(a) K-Match-LSTM (b) Weighted-K-Match-LSTM

Figure 6: Accuracy Analysis - K-Match-LSTM (left) and Weighted-K-Match-LSTM (right)

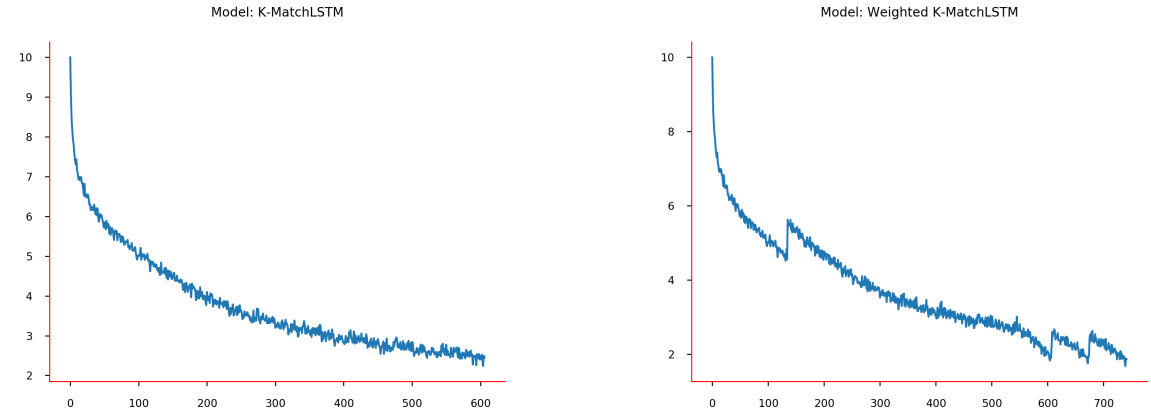


Figure 7: Convergence analysis for Different Models

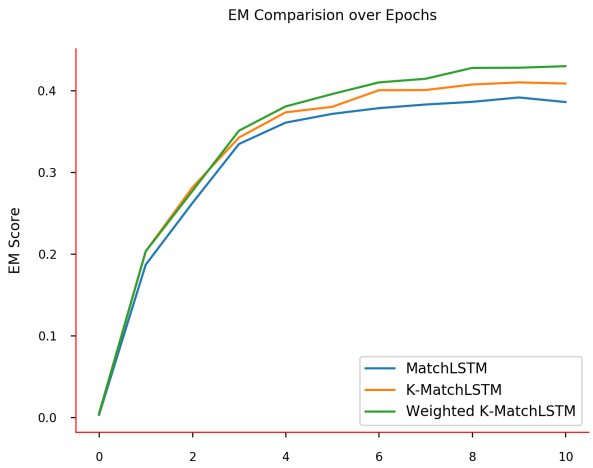


Figure 8: EM scores for Different Models

Further adoption of weighted errors significantly improve the accuracy in terms of various efficiency scores in comparison to K-Match-LSTM as evident from figure 6.

Figure 8 provides empirical evidence of both weighted and cluster intrinsic model outperforming the base model as the number of epochs become large under some regulation.

6.2 Attention Model

Model Description Our model has three components: a **RNN encoder layer**, that encodes both the context and the question into hidden states, **an attention layer**, that combines the context and question representations, and an **output layer**, which applies a fully connected layer and then two separate softmax layers (one to get the start location, and one to get the end location of the answer span).

RNN Encoder Layer : For each SQuAD example (context, question, answer), the context is represented by a sequence of d-dimensional word embeddings $\boldsymbol{x_1}, \ldots, \boldsymbol{x_N} \in \mathbb{R}^d$, and the question by a sequence of d-dimensional word embeddings $\boldsymbol{y_1}, \ldots, \boldsymbol{y_M} \in \mathbb{R}^d$. These are fixed, pre-trained GloVe embeddings. The embeddings are fed into a 1-layer bidirectional GRU (which is shared between the context and the question):

$$\{\overrightarrow{\boldsymbol{c_1}}\overleftarrow{\boldsymbol{c_1}}, \ldots, \overrightarrow{\boldsymbol{c_N}}, \overleftarrow{\boldsymbol{c_N}}\} = biGRU(\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}\})$$
$$\{\overrightarrow{\boldsymbol{q_1}}\overleftarrow{\boldsymbol{q_1}}, \ldots, \overrightarrow{\boldsymbol{q_M}}, \overleftarrow{\boldsymbol{c_M}}\} = biGRU(\{\boldsymbol{y_1}, \ldots, \boldsymbol{y_M}\})$$

The bidirectional GRU produces a sequence of forward hidden states $\overrightarrow{\boldsymbol{c_i}} \in \mathbb{R}^h$ for the context and $\overrightarrow{\boldsymbol{q_j}} \in \mathbb{R}^h$ for the question) and a sequence of backward hidden states $\overleftarrow{\boldsymbol{c_i}}$ and $\overleftarrow{\boldsymbol{q_j}}$. We concatenate the forward and backward hidden states to obtain the context hidden states $\boldsymbol{c_i}$ and the question hidden states $\boldsymbol{q_j}$ respectively:

$$\boldsymbol{c_i} = [\overrightarrow{\boldsymbol{c_i}}, \overleftarrow{\boldsymbol{c_i}}] \in \mathbb{R}^{2h} \ \forall i \in \{1, \ldots, N\}$$
$$\boldsymbol{q_j} = [\overrightarrow{\boldsymbol{q_j}}, \overleftarrow{\boldsymbol{q_j}}] \in \mathbb{R}^{2h} \ \forall i \in \{1, \ldots, M\}$$

Attention Layer We used **Scaled Dot-Product Attention** and **Multi-Head Attention** for our task. Our main aim was to decrease the training time that the current state of art model takes and provide at par accuracy. We were pretty successful in decreasing the training time to 20 min for 1 epoch and model took 20 epoch to reach it's maxima. For the further discussion, we have denoted keys as the context hidden states and the values as the question hidden states.

.

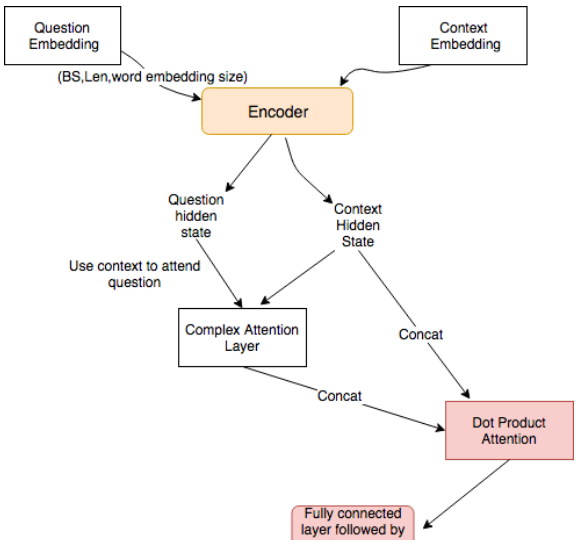


Figure 9: Overview of second Model

6.2.1 Scaled Dot-Product Attention

This attention mechanism was proposed in citeVaswani et al. 2017, in involves multiple heads each of which first linearly down-projects the keys and the values to a lower dimensional space and computes a standard attention between them. The output across the head is then concatenated. The benefits of this approach lie in its ability to compute attention from various perspectives and its also benefits from parallelizability.

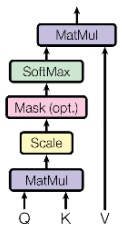


Figure 10: Scaled Dot-Product Attention. *(Source: 8)*

The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values. In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into

matrices K and V. We compute the matrix of outputs as:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\Big)V$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$

6.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively.

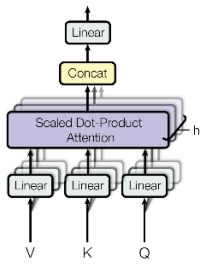


Figure 11: Multi-Head Attention. *(Source: 8)*

On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Fig. Multi-head attention allows the model to jointly attend to information from different representation sub spaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(head_1, ..., head_h)\mathbf{W}^O$$
$$where\ head_i = Attention(\mathbf{Q}\mathbf{W}^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

Where the projections are parameter matrices $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_k}$ and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$. In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

Output Layer: Next, each of the blended representations $\boldsymbol{b_i}$ are fed through a fully connected layer followed by a ReLU non-linearity:

$$\boldsymbol{b_i}\text{`} = ReLU(\boldsymbol{W}_{FC}\boldsymbol{b_i} + \boldsymbol{v}_{FC}) \in \mathbb{R}^h \ \forall i \in \{1, \ldots, N\}$$

where $\boldsymbol{W}_{FC} \in \mathbb{R}^{h \times 4h}$ and $\boldsymbol{v}_F C \in \mathbb{R}^h$ are a weight matrix and bias vector. Next, we assign a score (or logit) to each context location i by passing $\boldsymbol{b_i}\text{`}$ through a downprojecting linear layer:

$$logits_i^{start} = \boldsymbol{W}_{\boldsymbol{start}}^{\boldsymbol{T}}\boldsymbol{b}\text{`}_{\boldsymbol{i}} + u_{start} \ \in \mathbb{R} \ \forall i \in \{1, \ldots, N\}$$

where $\boldsymbol{w}_{start} \in \mathbb{R}^h$ is a weight vector and $u_{start} \in \mathbb{R} isabiasterm.$ Finally, we apply the softmax function to $logits_{start} \in \mathbb{R}^N$ to obtain a probability distribution $p_{start} \in \mathbb{R}^N$ over the context locations $\{1, \ldots, N\}$:

$$p^{start} = softmax(logits^{start}) \in \mathbb{R}^N$$

We compute a probability distribution p^{end} in the same way (though with separate weight $\boldsymbol{w}_{end}$ and u_{end}).

Loss : Our loss function is the sum of the cross-entropy loss for the start and end locations. That is, if the gold start and end locations are $i_{start} \in \{1, \ldots, N\}$ and $i_{end} \in \{1, \ldots, N\}$ respectively, then the loss for a single example is:

$$loss = -\log(p)^{start}(i_{start}) - \log(p)^{end}(i_{end})$$

During training, this loss is averaged across the batch and minimized with the Adam optimizer.

Prediction : At test time, given a context and a question, we simply take the argmax over p^{start} and p^{end} to obtain the prediction span (l^{start}, l^{end}):

$$l^{start} = \underset{i \in \{1,\ldots,N\}}{argmax} \ p_i^{start}$$
$$l^{end} = \underset{i \in \{1,\ldots,N\}}{argmax} \ p_i^{end}$$

7. Results

The model was trained for 20 epochs with around **20 min** training time of each epoch.The addition of Multihead attention resulted in improvement of F1 score by 5% which is significant.Other major improvement came from using forward and backward lstm, using the length of embedding found out from data analysis, using mask to prevent model from iterating on padding.

8. Analysis

The major difference in EM and F1 score is because of imprecise boundaries. This may be due to ambiguity in questions and answers provided in squad as the dataset is labeled by humans. Fig 19 shows one such case.Also the model is trained to answer to focus on "What" , "Who", "When" ,"How" phrases rather than understanding technicalities of questions like comparing with adverbial phrases like greatest, biggest, one among the following etc.

Model	Training Set		Test Set	
	EM	F1	EM	F1
Stanford Baseline	-	-	34.4	43.9
Stanford Baseline LSTM	-	-	35.3	44.9
MultiHead Attention	67.5	80	45.95	60.05

Table 2: Result Obtained on Multi Head Attention against baseline squad model by stanford *(Source: 6)*

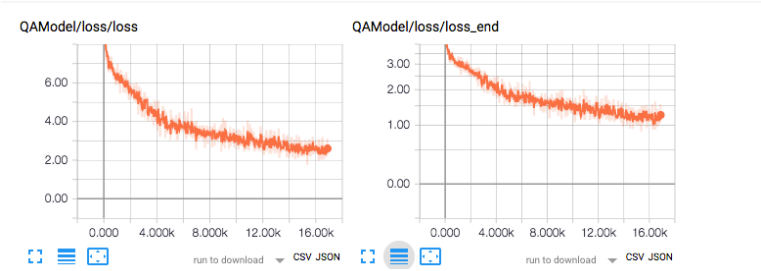


Figure 12: Model Loss

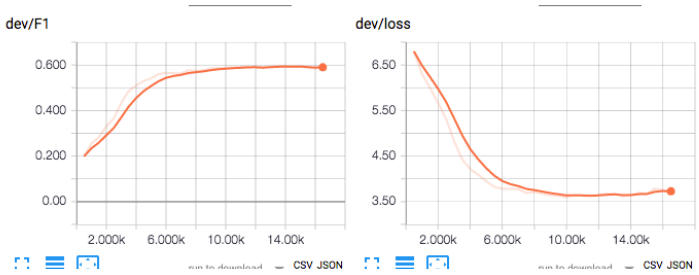


Figure 13: F1 score and loss

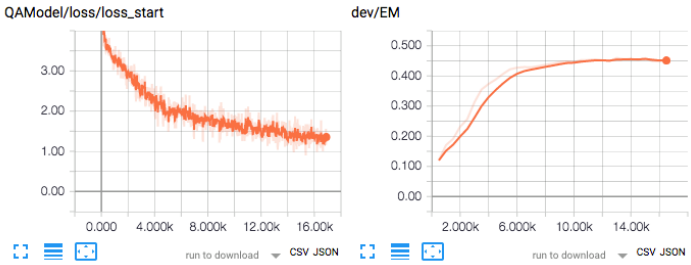


Figure 14: EM score and loss

9. Future extension

One idea for further improvement as proposed by Ryan Almodovar that we can use is to build questions summaries.It is conceivable that adding paragraph summaries to each question word would also be helpful in relating the paragraph to the question. One could compute the expected paragraph vector for each question word in the same way that we did the reverse, and augment the questions vectors with this information.

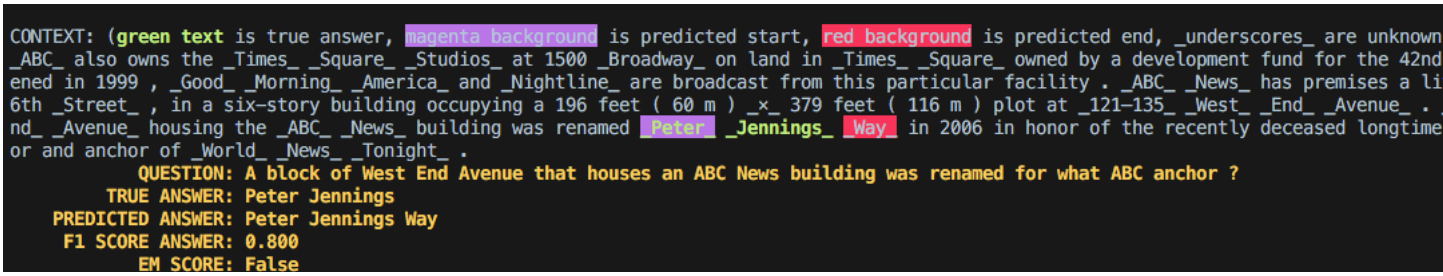


Figure 15: Controversial example showing difference in EM and F1

References

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

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

[3] 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`.

[4] 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`.

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

[6] 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`.

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

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv e-prints*, 2017. URL `https://arxiv.org/abs/1706.03762`.

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

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

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

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

10. Contribution

All group memebers did equal amount of work ie each 16.7%