A Question Answering System Using Encoder-decoder Sequence-to-sequence Recurrent Neural Networks

Bo Li

San José State University

May 11, 2018

Outline

Topic of This Project

- Question answering is the study of writing computer programs that can answer natural language questions.
- In this project, we focused on a scenario where a specific passage is already assigned to a question and the answer is a segment of the passage.
- Stanford Question Answering Dataset (SQuAD) is suitable for such scenario. It is used in this project.
    - It includes questions asked by human beings on Wikipedia articles.
    - The answer to each question is a segment of the corresponding Wikipedia passage
    - It contains more than 100,000 question-answer pairs on more than 500 articles.

An example

Examples

- Passage: The city had a population of 1,307,402 according to the 2010 census , distributed over a land area of 372.1 square miles (963.7 km2) . The urban area of San Diego extends beyond the administrative city limits and had a total population of 2,956,746 , making it the third-largest urban area in the state , after that of the Los Angeles metropolitan area and San Francisco metropolitan area . They , along with the RiversideSan Bernardino , form those metropolitan areas in California larger than the San Diego metropolitan area , with a total population of 3,095,313 at the 2010 census .
- Question: How many square miles does San Diego cover ?
- Answer: 372.1

Technique Used in This Project

The encoder-decoder sequence-to-sequence recurrent neural networks were used in this project.

- Encoder-decoder: encode an input to some vectors and then decode those vectors to an output.
- Sequence-to-sequence: Input is a sequence; output is also a sequence
  - For question answering, the input sequence includes a passage and a question and the output sequence is the answer
- Recurrent Neural Networks: Networks used for modeling sequential data

Contribution of This Project

- We successfully built a question answering system using an existing model and four models that were designed by making changes to the existing model.
- By comparing the results of five different models, we got two interesting observations. We will give details in Experiments part.

Word Feature Vector

Examples

An example from the GloVe word feature vectors.

- Word: the
- Word feature vector: [0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 -0.17862 -0.00066023 -0.6566 0.27843 -0.14767 -0.55677 0.14658 -0.0095095 0.011658 0.10204 -0.12792 -0.8443 -0.12181 -0.016801 -0.33279 -0.1552 -0.23131 -0.19181 -1.8823 -0.76746 0.099051 -0.42125 -0.19526 4.0071 -0.18594 -0.52287 -0.31681 0.00059213 0.0074449 0.17778 -0.15897 0.012041 -0.054223 -0.29871 -0.15749 -0.34758 -0.045637 -0.44251 0.18785 0.0027849 -0.18411 -0.11514 -0.78581]

Word Feature Vector, cont.

- A word feature vector represents a word according to its relationship with other words in a vocabulary.
- The word feature vectors for the vocabulary of a given text are learned by training a neural probabilistic language model on the text.
- In practice, in neural network models for natural language processing, word feature vectors are used to represent words. This is how we use word feature vector in this project.

Recurrent Neural Networks

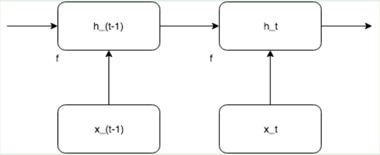
- Recurrent Neural Networks (RNNs) are used for modeling sequential data.
- In practice, due to vanishing problem, more complex learning unit such as Long Short Term Memory (LSTM) cell or Gated Recurrent Unit (GRU) are used. In this project, we used LSTM and GRU equally as learning unit.

Recurrent Neural Networks, cont.

Examples

A recurrent network with no outputs for encoding process.

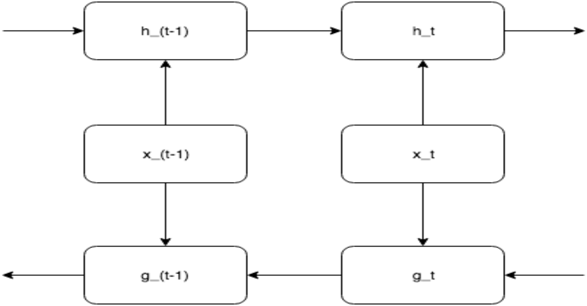
- x is the input. h is the state. θ is the hyperparameter.
- The relation between h and x is $h_t = f(h_{t-1}, x_t; \theta)$.



- An example of f is $h_t = sigmoid(W_h h_{t-1} + W_x x_t + b)$.
- An example of input sequence $x_1, ..., x_n$ is the word feature vector sequence which corresponds to the word sequence "How many square miles does San Diego cover ?".
- An example of what we want after operating this RNN is $h_1, ..., h_n$.

Bidirectional Recurrent Neural Networks

- Problem of RNNs: h_t only contains context information from x_1 to x_t
- Solution given by Bidirectional RNNs:
  - one cell operates from left to right, and another cell operates from right to left
  - using both h_t and g_t can get context information of the whole sequence



Bidirectional Recurrent Neural Networks

- In this project, we used bidirectional RNNs in encoding part. But for simplicity in presentation, I might talk about RNNs instead of Bidirectional RNNs in some parts.

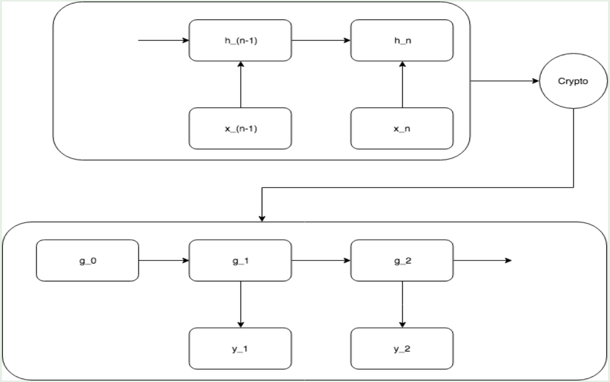
Encoder-decoder Sequence-to-sequence Architecture

- The process of understanding the input sequence is considered as encoding the input sequence to some vectors *Crypto*.
- The process of generating output is considered as decoding the *Crypto*.

Encoder-decoder Sequence-to-sequence Architecture, Cont.

Examples

x is the input, h is the state in encoding process, y is the output, and g is the state of decoding process.



Encoder-decoder Sequence-to-sequence Architecture, Cont.

- In this project, each input sequence actually includes two sequences - a question and a passage.
  - Attention mechanism is required to make each passage aware of the corresponding question.
- In this project, each output sequence is an answer which is represented by two indices for the input passage sequence.
  - The pointer network is used to make sure the output sequence comes from input sequence.

Attention Mechanism

- Intuitively, attention mechanism is one way to pay attention to a sequence of vectors.
- The key to understand attention mechanism is to understand how to get attention weight vector α.
- Using the attention weight vector α, we can get a weighted average of the sequence of vectors. This weight average is called attention vector.

Examples

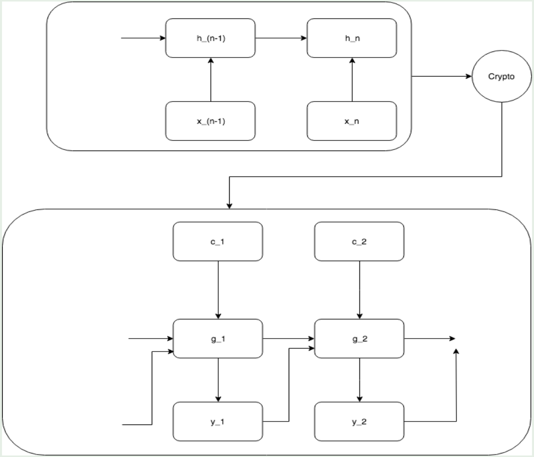
How attention mechanism was used in neural machine translation. In this scenario, the sequence of vectors to pay attention to is the encoding states.

- y is the output, g is the state, and c is the attention vector.

$$g_i = f(g_{i-1}, y_{i-1}, c_i).$$

Attention Mechanism, cont.

Examples



Attention Mechanism, cont.

Examples

- The attention vector c_i is produced by using g_{i-1} to "query" the encoding states $h_1, ...h_n$ through

$$c_i = \sum_j \alpha_{i,j} h_j$$

$$\alpha_{i,j} = \exp e_{i,j} / \sum_j \exp e_{i,j}$$

$$e_{i,j} = tanh(W_h h_j + W_g g_{i-1} + b).$$

Attention Mechanism, cont.

- In this project, the attention mechanism is used in both encoding and decoding part.
- In the encoding part, each word feature vector in passage pays attention to the corresponding question word feature vector sequence.
- In the decoding part, each decoding state pays attention to the sequence encoding states.

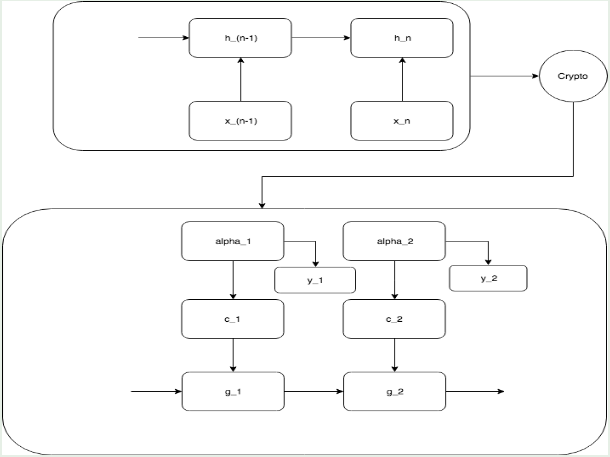
Pointer Network

- Using the pointer network enables the decoder to output tokens from input sequence.
- In this project, the pointer network was used in the decoding part which generates the answer.
- The attention weight vector α is considered as a probability distribution which indicates how likely each token in the input sequence is the current output token.

Pointer Network, cont.

Examples

$y_i = x_k$ where $k = argmax_j(\alpha_{i,j})$.



Relationship Between Five Different Models

- Model 1 is the Match LSTM & Answer Pointer model designed by Wang and Jiang.
- Model 2, 3, 4 and 5 are designed by us through making changes to Model 1.

Model 1
Overview

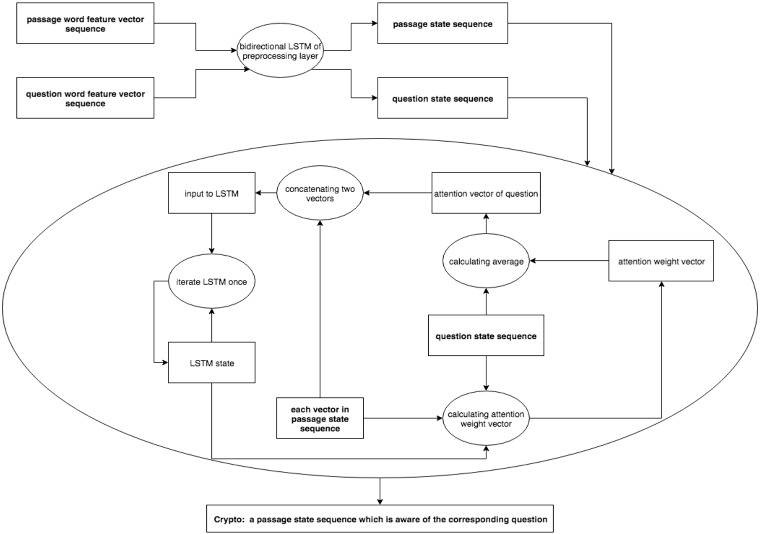
- Model 1 has an encoder-decoder sequence-to-sequence architecture.
- Model 1 is trained on SQuAD dataset.
  - Each instance of training data includes one passage, one question and one answer.
  - The passage is a sequence of tokens.
  - The question is a sequence of tokens.
  - The answer is a sequence of two indices indicating the start and end positions in passage.
- Before feeding training data into the model, tokens are converted to word feature vectors.

Model 1, cont.
Structure

- Encoder
  - the preprocessing layer
  - the bidirectional match LSTM layer
- Decoder
  - the answer pointer layer

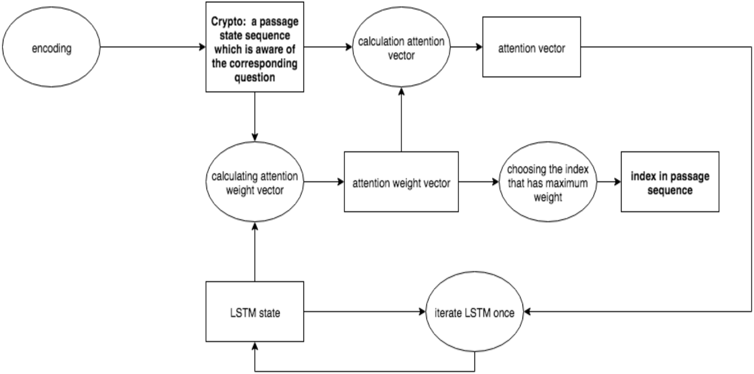
Model 1, cont.

Structure



Model 1, cont.

Structure



Model 1, cont.

loss function

- Let a_s denote the ground truth start index of the answer and a_e denote the ground truth end index, we have

$$p(a|H^r) = p(a_s|H_r)p(a_e|H_r) = \beta_{0,a_s} \times \beta_{1,a_e}$$

where

$$\beta_{k,j} = jth\ token\ of\ \beta_k$$

- To train the model, the loss function

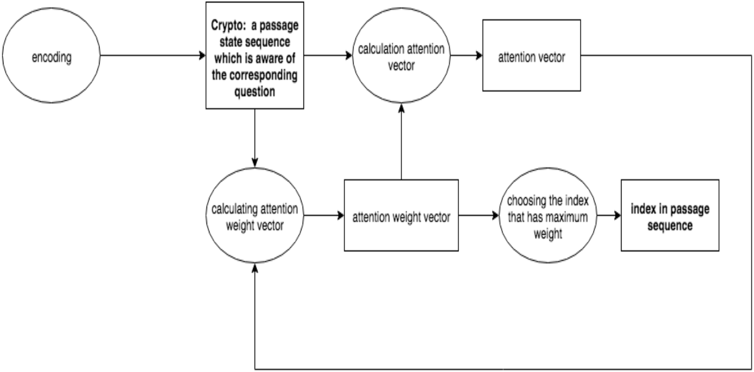
$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log p(a^n|H^r)$$

is minimized.

Model 2

The difference from Model 2 and Model 1 is in the decoding process.

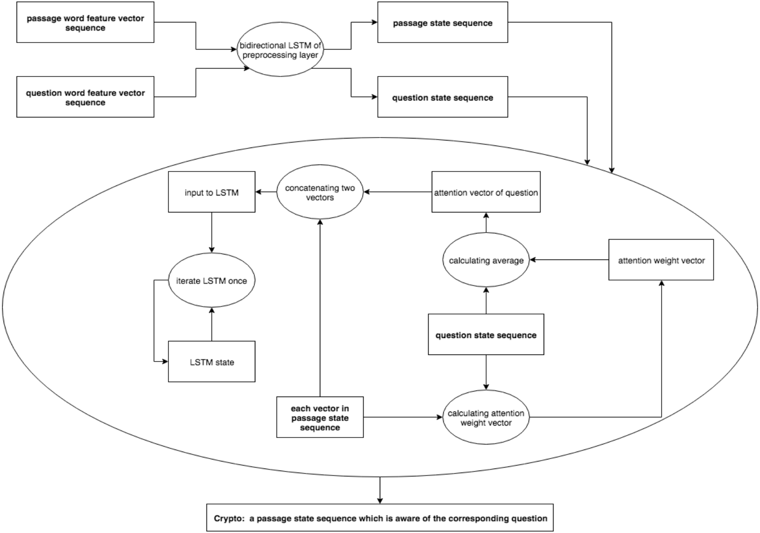
Model 2, cont.


encoding
Crypto: a passage state sequence which is aware of the corresponding question
calculation attention vector
attention vector
calculating attention weight vector
attention weight vector
choosing the index that has maximum weight
index in passage sequence

Model 3

The difference between Model 3 and Model 2 is in the bidirectional match LSTM layer.

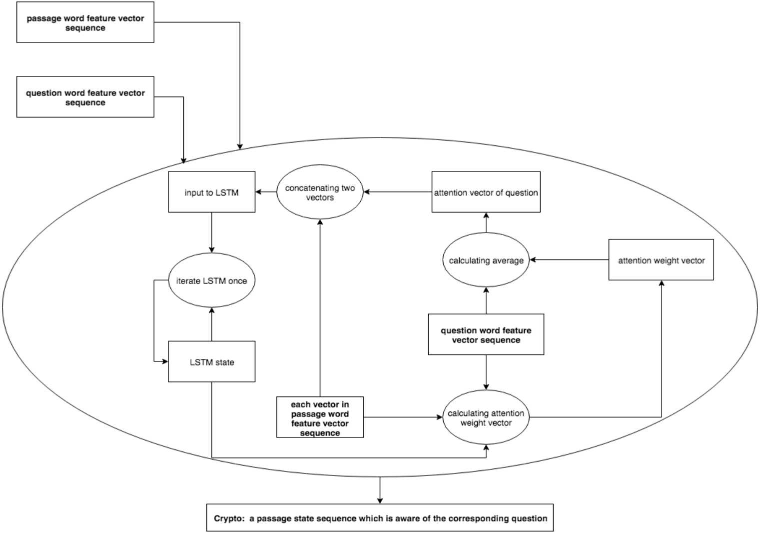
Model 3, cont.


passage word feature vector sequence
question word feature vector sequence
bidirectional LSTM of preprocessing layer
passage state sequence
question state sequence
input to LSTM
concatenating two vectors
attention vector of question
iterate LSTM once
calculating average
attention weight vector
LSTM state
question state sequence
each vector in passage state sequence
calculating attention weight vector
Crypto: a passage state sequence which is aware of the corresponding question

Model 4

The difference between Model 4 and Model 2 is that in Model 4 the the preprocessing layer is removed. This modification aims at checking whether the preprocessing layer carries some redundant context information.

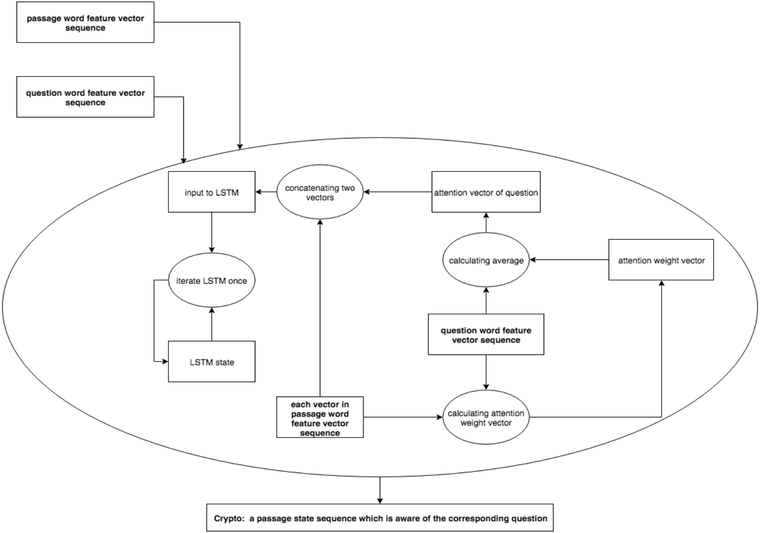
Model 4, cont.


passage word feature vector sequence
question word feature vector sequence
input to LSTM
concatenating two vectors
attention vector of question
calculating average
attention weight vector
iterate LSTM once
question word feature vector sequence
LSTM state
each vector in passage word feature vector sequence
calculating attention weight vector
Crypto: a passage state sequence which is aware of the corresponding question

Model 5

The difference between Model 5 and Model 2 is that Model 5 combines the changes of Model 3 and Model 4.

Model 5, cont.


passage word feature vector sequence
question word feature vector sequence
input to LSTM
concatenating two vectors
attention vector of question
iterate LSTM once
calculating average
attention weight vector
question word feature vector sequence
LSTM state
each vector in passage word feature vector sequence
calculating attention weight vector
Crypto: a passage state sequence which is aware of the corresponding question

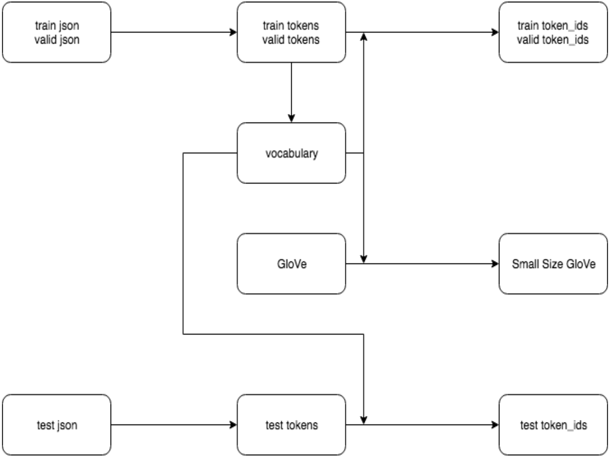
Data

- The Stanford Question Answering Dataset (SQuAD) is used to do experiments.

Set Name	Number of Instances
Train	78,839
Validation	8,760
Test	10,570

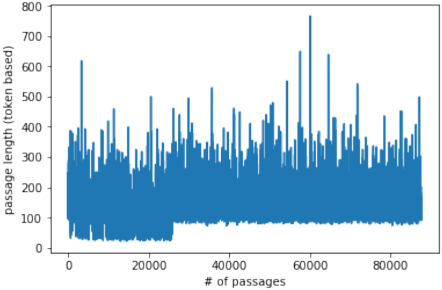
- The pre-trained GloVe word feature vectors are used to initialize words.

Data


train json
valid json
train tokens
valid tokens
train token_ids
valid token_ids
vocabulary
GloVe
Small Size GloVe
test json
test tokens
test token_ids

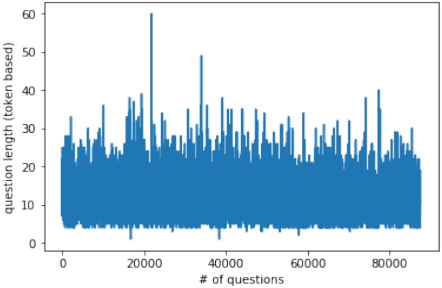
Settings

- 400 is set as *passage_padding_length*



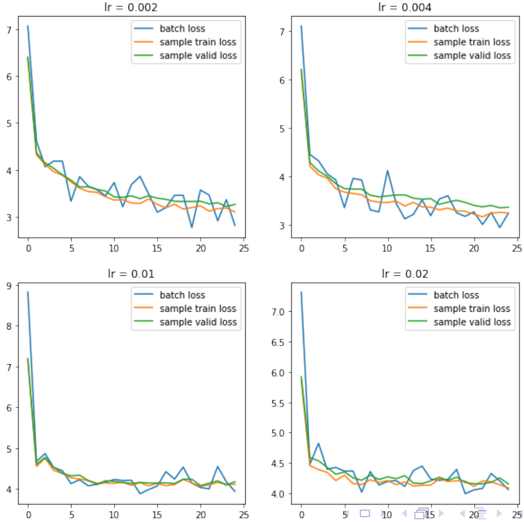
Settings

- 30 is set as *question_padding_length*



Settings

- The learning rate is set at 0.002 through experiments.



Settings

Hyperparameter Name	Value
Word Feature Vector Dimension (d)	100
Hidden State Size (l)	64
L2_regularization Scale	0.001
Hidden State Size (l)	64
Batch Size	64
Passage Length	400
Question Length	30
Clip Norm	5
Learning Rate	0.002

Settings

- The F1 score and the exact match score are used to evaluate the performance of each model.
  - F1 treats a predicted answer and a ground truth as bag of words and calculate a harmonic average of precision and recall;
  - exact match measures the percentage of predictions and ground truths that are exactly the same.
- The testing data contains several ground truth answers for one passage-question pair. The best score is chosen as the final score.
- A machine that has Tesla K80 12 GB Memory, 61 GB RAM and 100 GB SSD is used to train the models.

Training Process

- One epoch contains roughly 25 * 100 mini batches.
- The training loss and training scores are calculated every 100 mini batches using the 200 sample instances from training set. We do the same for validation loss and validation scores.
- Training one epoch takes roughly 100 minutes. A thorough training of each model requires around 10 epochs and takes around 17 hours.

Testing Results

Model	Test Exact Match	Test F1
Reference Paper	64.7	73.7
Model 1	23.4	33.6
Model 2	33.0	45.8
Model 3	33.0	46.2
Model 4	33.0	45.6
Model 5	24.3	33.9

Analysis

- Model 1 didn't reproduce the results of the original paper.
- Model 2 has better score than Model 1. This indicates using attention vector to query attention weight might be better than using the LSTM state.

Analysis

- Model 2, 3 and 4 have similar scores. This indicates either removing the preprocessing layer or not using LSTM state to query attention vector in the bidirectional match LSTM layer does not decrease test results.
- Model 5 performs worse than Model 2, 3 and 4. This means the change in Model 3 encoder and Model 4 encoder cannot be made together. A reasonable guess is the context information provided by these two parts is not provided in other parts of Model 2.

Contribution of This Project

- This project presented a thorough implementation of a question answering system.
- Five different models were tried and several interesting observations were found.

Future Work

- Further work is required to find out why Model 1 failed to reproduce the testing results of the reference paper.
- At the same time, more parameter tuning work is required to make the experiments more precise.
- Last but not the least, making novel architectures to bypass the state-of-art results is always a good way to move the question answering research forward.

Thank you! Questions?