CHATBOT BASED QUESTION ANSWERING SYSTEM

By

YASH SINHA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2018

To my family, friends and teachers

ACKNOWLEDGMENTS

I thank Professor Andy Li, Professor Sanjay Ranka and Professor Daisy Wang for their continuous support and guidance. I would like to thank Purnendu Mukherjee and other members of Li Lab for all the help during my thesis. I am also thankful to my family and friends who gave me their love and support. Finally, I would like to thank all the people associated with the University of Florida, especially my advisor Adrienne Cook for all the support during my academic life here.

TABLE OF CONTENTS

page

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolution Neural Networks
DMN	Dynamic Memory Network
DMN	Dynamic Memory Networks
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MC	Machine Comprehension
NER	Named-Entity Recognition
NLM	Neural Language Model
NLP	Natural Language Processing
NMT	Neural Machine Translation
OOV	Out of Vocabulary
POS	Part-of-Speech
Ptr-Net	Pointer Networks
QA	Question Answering
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
SFU	Semantic Fusion Unit

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

CHATBOT BASED QUESTION ANSWERING SYSTEM

By

Yash Sinha

May 2018

Chair: Xiaolin Li
Major: Computer Science

Chatbots – also known as “conversational agents” – are software applications that mimic written or spoken human speech for the purposes of simulating a conversation or interaction with a real person. Traditional chatbots have made heavy use of NLP concepts like slot filling. In this work, we have used deep learning models to build a chatbot for flight booking. We first start with the task of Machine Comprehension, which is one of the most important problems in Natural Language Processing. There have been mainly three reasons for the recent advancement in this task. The attention mechanism, the concept of Pointer Networks to constrain the output tokens to be from the input sequences and character embedding to solve the problem of OOV words. Apart from these, Memory Networks have helped in solving multi-sentence reasoning problem and Reinforcement Learning has been used in the training phase as the optimization strategy when answer boundary is fuzzy or too long.

There also has been research to improve the encoding layer by using parts-of-speech tags and named-entity tags. We perform a literature review of some of the important existing approaches, we also explore their performance and present their results. We also introduce the concept of mixed attention to capture the query

information along with the context information. We see that the deep learning models can be effectively used in machine comprehension tasks as well as fully functional chatbots.

CHAPTER 1
INTRODUCTION

Question answering task has been one of the most important problems in NLP. Teaching machines to read, process and comprehend and then answer questions has proved to be a challenging task. The Machine Comprehension task [1] is as follows. Given a passage P and question Q, our task is to predict an answer A to question Q based on information found in P. Answer A often includes non-entities and can be much longer phrases. This setup challenges us to understand and reason about both the question and passage in order to infer the answer. An example of the task can be seen in Figure 1-1.

Passage: Tesla later approached Morgan to ask for more funds to build a more powerful transmitter. **When asked where all the money had gone, Tesla responded by saying that he was affected by the Panic of 1901**, which he (Morgan) had caused. Morgan was shocked by the reminder of his part in the stock market crash and by Tesla's breach of contract by asking for more funds. Tesla wrote another plea to Morgan, but it was also fruitless. Morgan still owed Tesla money on the original agreement, and Tesla had been facing foreclosure even before construction of the tower began.

Question: On what did Tesla blame for the loss of the initial money?

Answer: Panic of 1901

Figure 1-1. An example from the SQuAD [2] dataset

This proves to be challenging because of many reasons. Firstly, the traditional models couldn't capture OOV words. This is a problem when the word in the test set isn't present in the training set and thus, the network is not able to generate word embedding vectors for a new word. The next challenge is the multi-sentence reasoning. Humans increase their understanding by re-reading the context and the query. To help the machine achieve that kind of understanding, Memory Networks [3, 4] have proved to be quite useful to incorporate multi-hop reasoning. Another challenge is that we fail to fully capture the long-distance contextual interaction between parts of the context, by

only using LSTM [5] or GRU [6]. The problem here is that the answer candidate is often oblivious to important cues in the passage outside its surrounding window. Fusing the context into itself has proved to be useful to handle this problem. One of the challenges that has not yet been solved to a good extent is that we fail to capture the semantics of the passage. There have been various approaches in Neural Language Models to capture some of the semantics by using POS tagging, NER tagging, and using binary matching between context and passage. However, these too have been unable to completely capture the semantics of the passage.

The task of machine comprehension can be effectively translated into a chatbot system. We can think of chatbot as a question answering system where the paragraph P is the text entered by the user. To cater a specific domain, like that of flight booking task, we can generate a set of internal questions to related to the entities required for the search. These will serve as question Q. The answer A will be the values of those entities. We will delve into the chatbot systems in more detail in the later chapters.

Traditionally, the chatbots have used the concepts of NLP, like that of slot filling. Imagine we want to create an agent for a t-shirt shop [7]. In order to place an order, we need to know the following information:

- Color
- Size
- Number

Knowing this, we can expect a user's conversation to go something like this:

User: I'd like to buy a t-shirt.

Agent: Great, first tell me a couple of things.

Agent: How many do you want?

User: 3

Agent: What color would you like?

User: Black

Agent: What size?

User: Medium

Agent: Got it, that was three, black t-shirts in medium. Did I get that right?

User: Yes!

Agent: Alright thanks, would you like to keep shopping or checkout?

To use slot-filling, we set these three parameter values to be required. If users omit one or more of the parameters in their response, we prompt them to provide values for each missing parameter. The problem with these approaches is that we need to generate a lot of statements, wherein the answer from the user can be used to fill the slots. In many cases, it becomes virtually impossible to cover all the cases. In the cases where we want to generalize these statements, deep learning models prove to be very effective. In the next section, we will cover the basics of deep learning and other concepts which will provide a foundation to some of the state-of-the-art question answering models.

CHAPTER 2
DEEP LEARNING BASICS

2.1 Deep Learning

Deep learning [8] (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised. Neural networks [9] are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. We can think of them as a clustering and classification layer on top of data we store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. To be more precise, neural networks extract features that are fed to other algorithms for clustering and classification, so we can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.

Figure 2-1 gives an example of a basic neural network. A deep learning network is composed of several layers. The layers are made of nodes. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a

set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs for the task the algorithm is trying to learn These input-weight products are summed and the sum is passed through a node's activation function, to determine whether and to what extent that signal progresses further through the network to affect the ultimate outcome, say, an act of classification. The weights can be tuned based on the data.

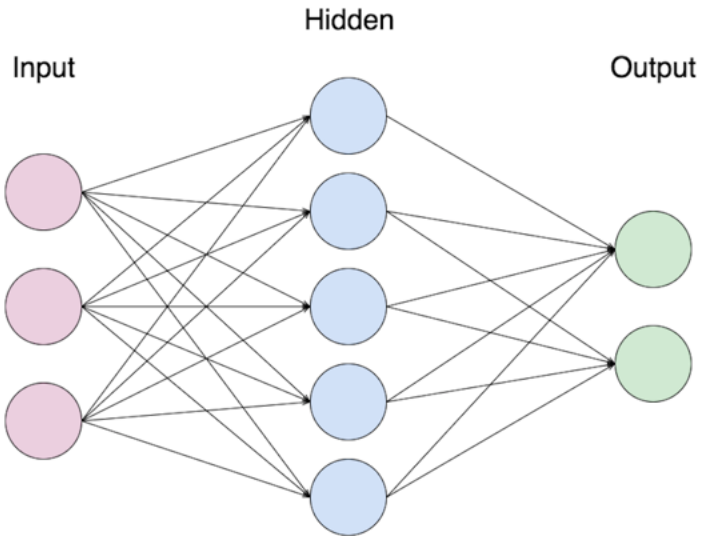

Figure 2-1. An artificial neural network

The connections have weights that can be tuned based on the data, making neural nets capable of learning. The tuning is done through an algorithm called backpropagation based on gradient descent. In the next few sections, we will cover some of the basics of deep learning.

2.2 Convolutional Neural Network

Convolutional networks [10, 11] are deep artificial neural networks that are very effective for image recognition and classification. They have been proved successful in identifying faces, individuals, objects, traffic signs and powering vision in robots and self-driving cars. Figure 2-2 [12] illustrates the architecture of a basic CNN.

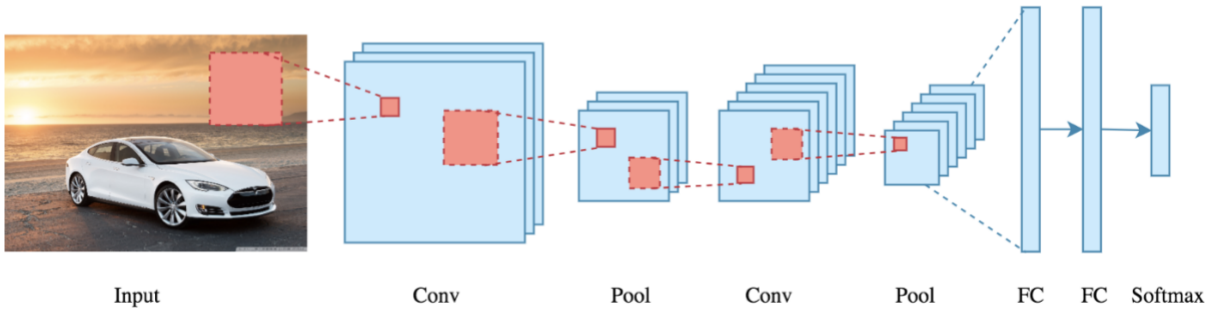

Figure 2-2. A convolutional neural network

There are primarily four operations in a standard CNN model:

- Convolution: The primary purpose of Convolution in the ConvNet is to extract features from the input image. The spatial relationship between pixels, i.e. the image features are preserved and learned by the convolution using small squares of input data.
- Non-Linearity (ReLU): Rectified Linear Unit (ReLU) [13] is a non-linear operation that carries out an element wise operation on each pixel. This operation replaces the negative pixel values in the feature map by zero.
- Pooling or Sub Sampling - Spatial Pooling reduces the dimensionality of each feature map but retains the most important information. For max pooling, the largest value in the square window is taken and rest are dropped. Other types of pooling are Average, Sum etc.
- Classification (Fully Connected Layer): The Fully Connected layer is a traditional Multi-Layer Perceptron as described before, that uses a softmax activation [14] function in the output layer. The high-level features of the image are encoded by the convolutional and pooling layers which is then fed to the fully connected layer which then uses these features for classifying the input image into various classes based on the training dataset.

When a new image is fed into the CNN model, all the above-mentioned steps are carried out (forward propagation) and a probability distribution is achieved on the set of output classes. With a large enough training dataset, the network will learn and generalize well enough to classify new images into their correct classes.

2.3 Recurrent Neural Network

RNNs [15] are one of the popular deep learning architectures. They have shown great promise in many NLP tasks. The idea behind RNNs is to make use of sequential

information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If we want to predict the next word in a sentence, we want to know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory, RNNs can make use of information in arbitrarily long sequences, but in practice, they are limited to looking back only a few steps. This is because of the vanishing gradients. Figure 2-3 [16] depicts a basic RNN architecture.

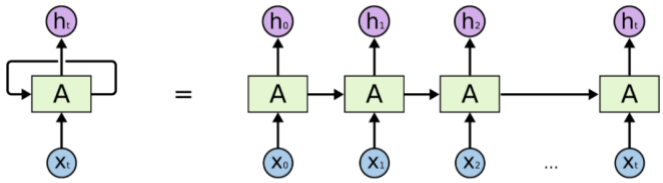

Figure 2-3. A recurrent neural network architecture

To counter the vanishing gradient problem, other special kinds of RNNs are generally used. Long Short-Term Memory networks are capable of learning long-term dependencies. They work tremendously well on a large variety of problems. The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. An LSTM has three kinds of gates:

- Forget Gate Layer
- Input Gate Layer
- Output Gate Layer

A forget gate layer decides what information to throw away from the cell state. The input gate layer decides which values to update. The output gate layer decides how much of the cell should be sent ahead. Figure 2-3 [17] depicts the architecture of a typical LSTM. Another variation of RNN is a Gated Recurrent Unit which has two gates, reset and update gates.

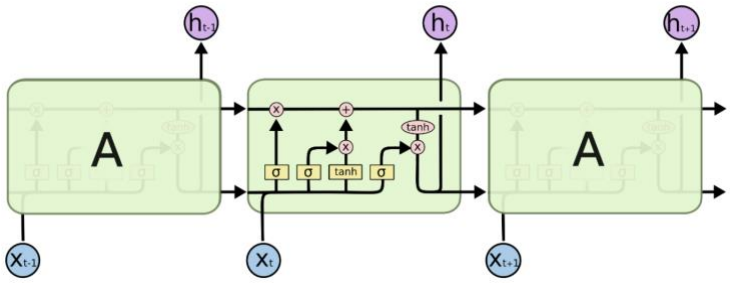

Figure 2-4. An LSTM architecture

2.4 Word Embedding

Word embeddings [19] are one of the most interesting research areas of deep learning. Word representations like word embedding, have been used across various tasks like part-of-speech tagging, named entity recognition [20], parsing and semantic role labeling. Any word of a dictionary (the set of words recognized for the specific task) is transformed into a numeric vector of a certain number of dimensions. A word embedding W **:** words $\rightarrow \mathbb{R}^n$ is a parameterized function mapping words in some language to high-dimensional vectors. Word embeddings are the texts converted into numbers and there may be different numerical representations of the same text.

One of the tasks of word embedding is predicting whether an n-gram is valid. This is particularly helpful in detecting grammatical errors in the text. Another common task is predicting the next word in the sentence. Word embeddings make a lot of intuitive sense in NLP tasks. Similar words are close together and are closest in the

embedding. These words tend to be quite similar. Figure 2-5 [19] shows t-SNE visualizations of word embedding.

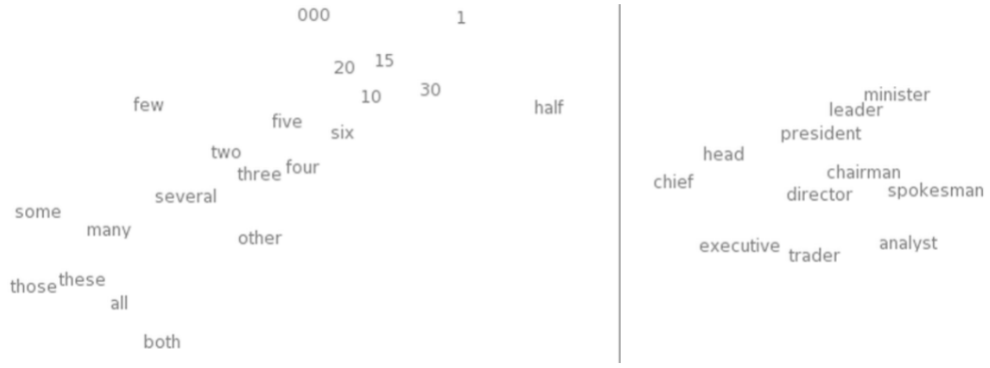

Figure 2-5. t-SNE visualizations of word embeddings

Some of the most popular word embedding models are Word2Vec [21], GloVe [22] and CoVe [23]. Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. It detects similarities mathematically and creates vectors that are distributed numerical representations of word features, features such as the context of individual words. GloVe seeks to make explicit what word2vec does implicitly. It encodes the meaning as vector offsets in an embedding space, seemingly only a serendipitous by-product of word2vec, is the specified goal of GloVe.

2.5 Attention Mechanism

Attention Mechanisms [26] in Neural Networks are very loosely based on the visual attention mechanism found in humans. With attention mechanism, we allow the decoder to “attend” to different parts of the source sentence at each step of the output generation. The model learns what to attend, based on the input sentence and what it

has produced so far. Figure 2-6 [27] depicts the attention mechanism. Each decoder output word y_t depends on a weighted combination of all the input states. The a's are the weights that define in how much of each input state should be considered for each output. So, if $a_{3,2}$ is a large number, the decoder pays a lot of attention to the second state in the source sentence while producing the third word of the target sentence. The a's are typically normalized to sum to 1.

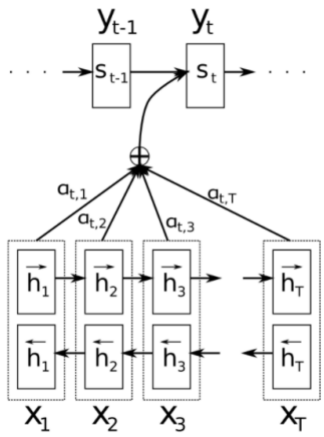

Figure 2-6. Attention mechanism

Attention mechanism have proved very useful in machine comprehension tasks to predict the answers from the input passage. We calculate the attention of the question over the passage to get a query-aware context. This helps us in finding out which query words are most relevant to each context word. The query-to-context attention signifies which of the context words have the closest similarity to one of the query words. We will explore this in more detail in later chapters.

2.6 Pointer Networks

Traditionally, the sequence-to-sequence models have been used for tasks like Neural Machine Translation. In NMT, we map the meaning of a sentence into a fixed-length vector representation and then generate a translation based on that vector.

These models use RNN to map an input sequence to an embedding and another RNN to map the embedding to an output sequence. However, these methods still require the size of the output dictionary to be fixed a priori. Thus, we cannot directly apply this to problems where the size of the output depends on the length of the input. Pointer networks [29] are a variation of the sequence-to-sequence model with attention. The output of pointer network are pointers to the elements of the input sequence. This helps us the problem of the output being fixed length vectors. The pointer networks also help in solving the out-of-vocabulary problem to some extent.

2.7 Memory Networks

Memory Networks [3] is an attention-based neural network architecture that operates an external symbolic memory component to perform reasoning. This can achieve interesting performances on various tasks of question answering and dialogue management and appears to be a promising avenue towards a better machine comprehension of language. A memory network combines learning strategies from the machine learning literature with a memory component that can be read and written to. The model is trained to learn how to operate effectively with the memory component. The high-level view of a memory network is as follows:

- Memory m**,** which is an array of objects like vectors or array of strings
- Input feature map I, which converts the incoming input to the internal feature representation
- Generalization component G, which updates old memories given the new input.
- Output feature map O, which produces a new output (in the feature representation space), given the new input and the current memory state.
- Response R, which converts the output into the response format desired.

Here, the components I, G, O and R can be potentially learned. In the case of a question answering system, we can use the components in the following way. **I** can

make use of standard pre-processing such as parsing, co-reference, and entity resolution. It could also encode the input into an internal feature representation by converting from text to a sparse or dense feature vector. G can be used by introducing a function *H* which maps the internal feature representation produced by I to an individual memory slot, and just updates the memory at *H* (I (*x*)). In this case, G updates the index *H (x)* of m, but all other parts of the memory remain untouched. O reads from memory and performs inference to deduce the set of relevant memories needed to perform a good response. R can be used to produce the actual wording of the question answer based on the memories found by O, for example, a textual response.

2.8 Reinforcement Learning

Reinforcement Learning [31] refers to a kind of Machine Learning method in which the agent receives a delayed reward in the next time step to evaluate its previous action. There's no answer key and the reinforcement learning agent has to decide how to act to perform its task. In the absence of existing training data, the agent learns from experience. It collects the training through trial-and-error as it attempts its task, with the goal of maximizing long-term reward. An RL setup is generally composed of two components, an agent and an environment. The environment refers to the object that the agent is acting, and the agent is the RL algorithm. The learning starts with the environment sending a state to the agent. The agent takes an action in response to that state. Now, the environment sends a pair of next state and reward to the agent. The agent then updates its knowledge based on the reward returned by the environment. The loop keeps going on until the environment sends a terminal state, which ends the episode. There are two popular type of RL algorithms, Q-learning and Policy Learning.

Q-learning [33] is a type of RL algorithm in which we evaluate the action to take based on an action-value function. This then determines the value of being in a certain state. The agent decides what action to take based on the state. The input to the Q function is one state and one action and it returns the expected reward of that action at that state. The following equation [31] shows how the value of Q is updated, based on the reward we get from the environment.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Figure 2-7. Q-learning equation

The learning rate alpha shows how aggressive we want to be when updating our value. The reward is obtained by taking action a_t at state s_t. This reward is added to the old estimate. The estimate of optimal future value is the maximum achievable reward Q for all available actions at x_{t+1}. The old value of Q is subtracted to make sure that only the difference in the estimate is incremented or decremented. The action is taken based on the action-selection strategy.

Another popular RL algorithm is the Policy Learning [33]. In policy learning, the state is mapped to the action. Unlike the Q-learning algorithm where the value function is learned to estimate the value of each state-action pair, a policy is a map from state to action. It implies that Policy Learning learns the Q-value based on the action performed by the current policy instead of the greedy policy. The Q-learning has no constraint over the next action, as long as it maximizes the Q-value for the next state. The equation [31] for policy learning is similar to Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Figure 2-8. Policy learning equation

The next chapter explores the existing state-of-art models for Question Answering task.

CHAPTER 3
LITERATURE REVIEW

3.1 Dynamic Memory Networks

The paper [35] is based on the concept that most tasks in natural language processing can be cast into question answering (QA) problems over language input. The authors introduce a neural network architecture called dynamic memory network, which processes input sequences and questions, forms episodic memories, and generates relevant answers. There is an iterative attention process, which is called hops, which allows the model to condition its attention on the inputs and the result of previous iterations. These results are then reasoned over in a hierarchical recurrent sequence model to generate answers. The training is done on raw input-question-answer triplets. The model can solve sequence tagging tasks, classification problems, sequence-to-sequence tasks and question answering tasks that require transitive reasoning.

The basic flow of the model is as follows. The DMN first encodes the inputs and questions using a Gated Recurrent Unit. In case the input sequence is a single sentence, the input module outputs the hidden states of the recurrent network. When the input sequence is a list of sentences, the sentences are concatenated into a list of word tokens, with an end-of-sentence token inserted after each sentence. This is given as an input to the episodic memory module. An iterative attention process is triggered which searches the inputs and retrieves relevant facts. The episodic memory module is comprised of an attention mechanism as well as a recurrent network with which it updates its memory. This module also reasons over retrieved facts and provides a

vector representation of all relevant information to an answer module which generates the answer. Figure 3-1 [35] shows the architecture of Dynamic Memory Network.

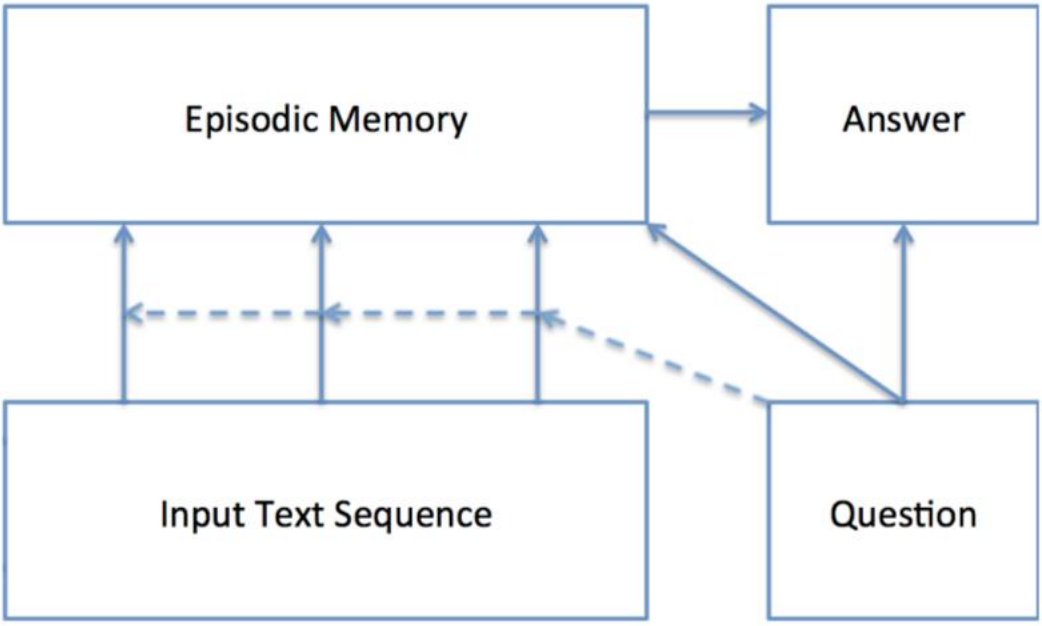

Figure 3-1. Overview of DMN modules.

The Input Module encodes raw text inputs into distributed vector representations. The paper focuses on natural language related problems in which the input may be a sentence, a long story, a movie review, a news article, or several Wikipedia articles. The Question Module encodes the question into a distributed vector representation. Here, the question may be a sentence such as 'Where did the author first fly?'. The representation forms the basis of the episodic memory module upon which the episodic memory module iterates. The episodic module chooses which parts of the inputs to focus on through the attention mechanism. It produces a 'memory' vector representation, given the question and the previous memory. The iterations provide the module with new relevant information about the input. Thus, the module retrieves new information, in the form of input representations, which might have been irrelevant in

previous iterations. Finally, the answer module generates an answer from the final memory vector of the memory module.

The most important takeaway from the model is the iterative nature which is also called hops. This iterative nature allows the episodic module hops to attend to different inputs during each pass. It also allows for a type of transitive inference, since the first pass may uncover the need to retrieve additional facts.

Story

```
mary got the milk there
john moved to the bedroom
sandra went back to the kitchen
mary travelled to the hallway
john got the football there
john went to the hallway
john put down the football
mary went to the garden
john went to the kitchen
sandra travelled to the hallway
daniel went to the hallway
mary discarded the milk
where is the milk ?
answer:  garden
```

Figure 3-2. Example from Facebook bAbI dataset. [36]

An example from the Facebook bAbI [36] dataset is shown in Figure 3-2. If the question is *'Where is the football?'*, in the first iteration, the model will attend the sentence 7 - *John put down the football*, since, *football* is mentioned in the question. However, in the second iteration reasons that *John* is relevant, and it then retrieves where *John* was.

The concept of multiple iterations has proved to be very useful in Question Answering datasets like Facebook bAbI, SQuAD, and TriviaQA [37] datasets.

3.2 Match-LSTM And Answer Pointer

Match-LSTM and Answer pointer [38] is an end-to-end neural architecture which is a combination of match-LSTM model and Pointer Networks. The match-LSTM [39] model is used for textual entailment. Pointer Networks, as described in chapter 1, is used to constrain the output tokens to be from the input sequence. 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 is called the match-LSTM. The match-LSTM 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.

The authors use the pointer networks in two different ways. It can treat an answer as a sequence of tokens from the input passage but ignore the fact that these tokens are consecutive in the original passage. So, the answer is represented as a sequence of integers $a = (a1 , a2 , . . .)$, where each a_i is an integer between 1 and P, where a_P is the last token of the passage. Another way in which pointer networks has been used is to ensure that the answer is a consecutive subsequence of the passage. Thus, pointer networks can be used to predict the start and end pointer of the answer, and all the tokens between the start and end are treated as the answer. The former has been called the *sequence* model and the latter is called the *boundary* model. Figure 3-3 [38] depicts *sequence* and *boundary* model.

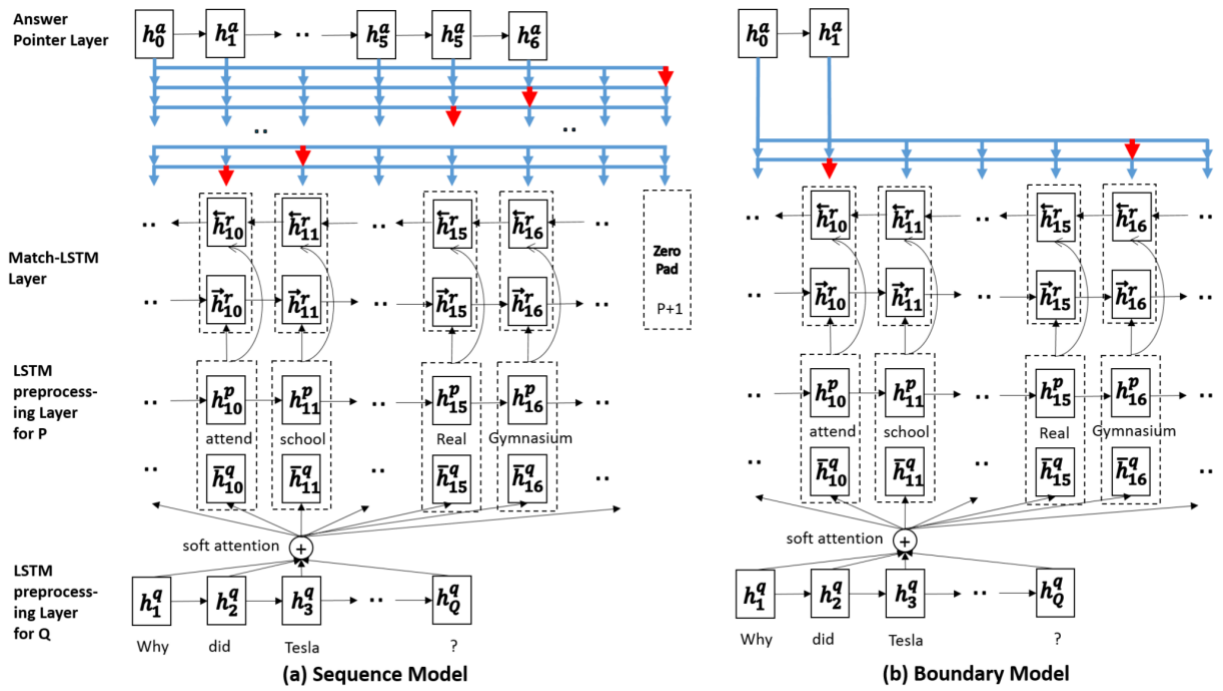

Figure 3-3. Match-LSTM and Answer Pointers boundary model.

The first layer is the LSTM preprocessing Layer which processes the passage and question using a one-directional LSTM to incorporate contextual information into the representation of each token in the passage and the question. This layer gives the matrices: H^p = *LSTM*(P) and H^q = *LSTM*(Q). Then the match-LSTM model is applied in which the question is treated as a premise and the passage is treated as the hypothesis. The match-LSTM sequentially goes through the passage. At position *i* of the passage, it first uses the standard word-by-word attention mechanism to obtain attention weight vector *α* as follows:

$$\begin{aligned}\overrightarrow{\mathbf{G}}_i &= \tanh(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + (\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}) \otimes \mathbf{e}_Q),\\ \overrightarrow{\alpha}_i &= \mathrm{softmax}(\mathbf{w}^{\intercal}\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q),\end{aligned}$$

Figure 3-4. Equations for calculating attention vector. [38]

$\alpha_{i,j}$ indicates the degree of matching between the i^{th} token in the passage with the j^{th} token in the question. The attention vector is used to obtain a weighted version of the

question and is combined with the current token of the passage to obtain vector z_i. This is fed to a one-directional LSTM to form the match-LSTM. A similar match-LSTM is built in the reverse direction. These two vectors are concatenated and fed to the pointer networks layer to get the answer pointers.

3.3 Character Aware

Char Aware [40] is a neural language model that relies only on character-level inputs. The model employs a convolutional neural network (CNN) and a highway network [41] over characters, whose output is given to a long short-term memory (LSTM) recurrent neural network language model (RNN-LM). The problem with conventional NLMs is that they don't consider the subword information. For example, they do not know, a priori, that *eventful*, *eventfully*, *un- eventful*, and *uneventfully* should have structurally related embeddings in the vector space. Embeddings of rare words can thus be poorly estimated, leading to high perplexities for rare words. Unlike a conventional NLM, that takes word embeddings as inputs, the char aware model takes the output from a single-layer character-level convolutional neural network with max-over-time pooling.

Figure 3-4 [40] illustrates the use of the Char Aware model to predict the next word. The first layer performs a lookup of character embeddings and stacks them to form the matrix C^k. The start-of-word and end-of-word characters are appended to each word to better represent prefixes and suffixes. Then convolution operations are applied between C^k and multiple filter matrices. Then, a max-over-time pooling operation is applied to obtain a fixed-dimensional representation of the word, which is given to the highway network. The highway network's output is used as the input to a multi-layer

LSTM. Finally, an affine transformation followed by a softmax [14] is applied to the hidden representation of the LSTM to obtain the distribution over the next word.

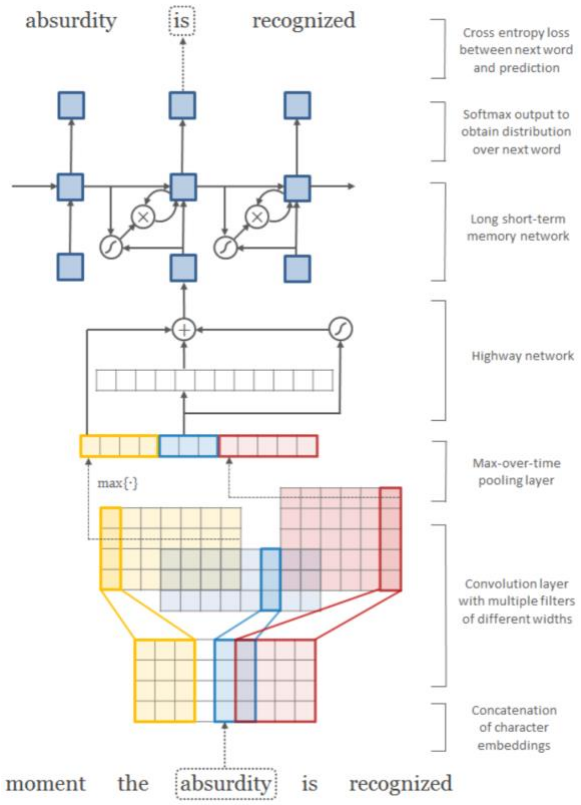

Figure 3-5. Char Aware language model applied to an example sentence

The Char Aware model helps in solving the out-of-vocabulary problem. There might be words in a question answering task with no word-embedding vectors. Traditionally, these words are mapped to *<UNK>.* With the help of character embedding, we can get better prediction in the question answering task. For example, the learned representations of OOV words (*computer-aided, misinformed*) are positioned near words with the same part-of-speech. In many of the recent models like BiDAF, R-Net and Reinforced Mnemonic Reader, the character embeddings are

concatenated with the word embedding vectors to get the better representation of the words.

3.4 R-NET

R-NET [42] is an end-to-end neural networks model for reading comprehension style question answering. First, the recurrent network encoder is used to build representation for questions and passages separately. Next, the model matches the question and passage with gated attention-based recurrent networks to obtain the question-aware passage representation. Then a self-matching attention mechanism is used to refine the representation by matching the passage against itself. This is done to encode the information from the whole passage. Finally, pointer networks model is used to locate the position of answers from the passages. Figure 3-6 [42] shows the R-NET architecture.

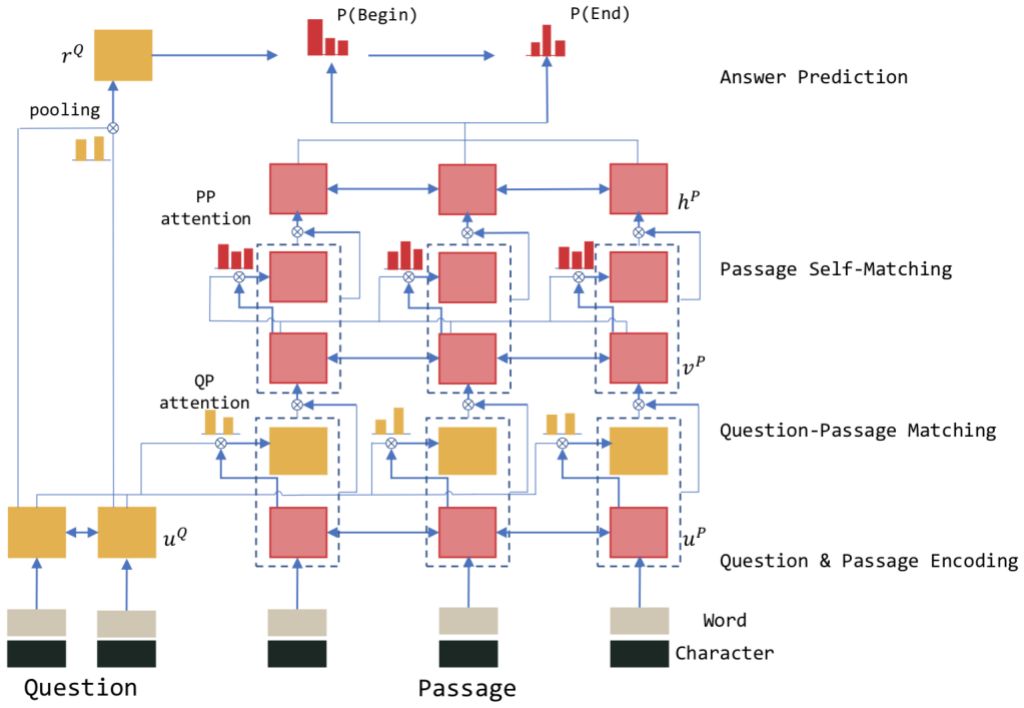

Figure 3-6. Architecture of R-NET.

The question and passage encoding layers use word-level embeddings and character-level embeddings to get the word representations. A BiRNN is used over the embeddings to get the representations of the words in question and passage. The question-passage matching layer uses a gated attention-based recurrent network and incorporates question information into passage representation. It uses an additional gate to focus on the relation between the question and current passage word. This is based on the theory that that only parts of the passage are relevant to the question in reading comprehension and question answering. The question-passage matching layer passage shows the important parts of the passage. However, the representation has very limited knowledge of context because one answer candidate is often oblivious to important cues in the passage outside its surrounding window. The passage self-matching layer collects evidence from the whole passage for words in the passage and encodes the evidence relevant to the current passage word and its matching question information into the passage representation. The self-matching extracts evidence from the whole passage according to the current passage word and question information. The output layer uses pointer networks to predict the start and end position of the answer.

3.5 Bi-Directional Attention Flow

The Bi-Directional Attention Flow [43] model uses a multi-stage hierarchical process that represents the context at different levels of granularity and uses bi-directional attention flow mechanism to obtain a query-aware context representation without early summarization. The model includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation. The attention mechanism in BiDAF is different than other

attention mechanisms. The attention layer is not used to summarize the context paragraph into a fixed-size vector. The attention is computed for every time step and does not summarize the context paragraph into a fixed-size vector. The attended vector at each time step, along with the representations from previous layers, is allowed to flow through to the subsequent modeling layer. The model also uses a memory-less attention mechanism. That is the attention at each time step is a function of only the query and the context paragraph at the current time step and does not directly depend on the attention at the previous time step. Thus, the attention layer focuses on learning the attention between the query and the context and enables the modeling layer to focus on learning the interaction within the query-aware context representation. The model also uses two attentions, query-to-context and context-to-query, which provide complementary information to each other.

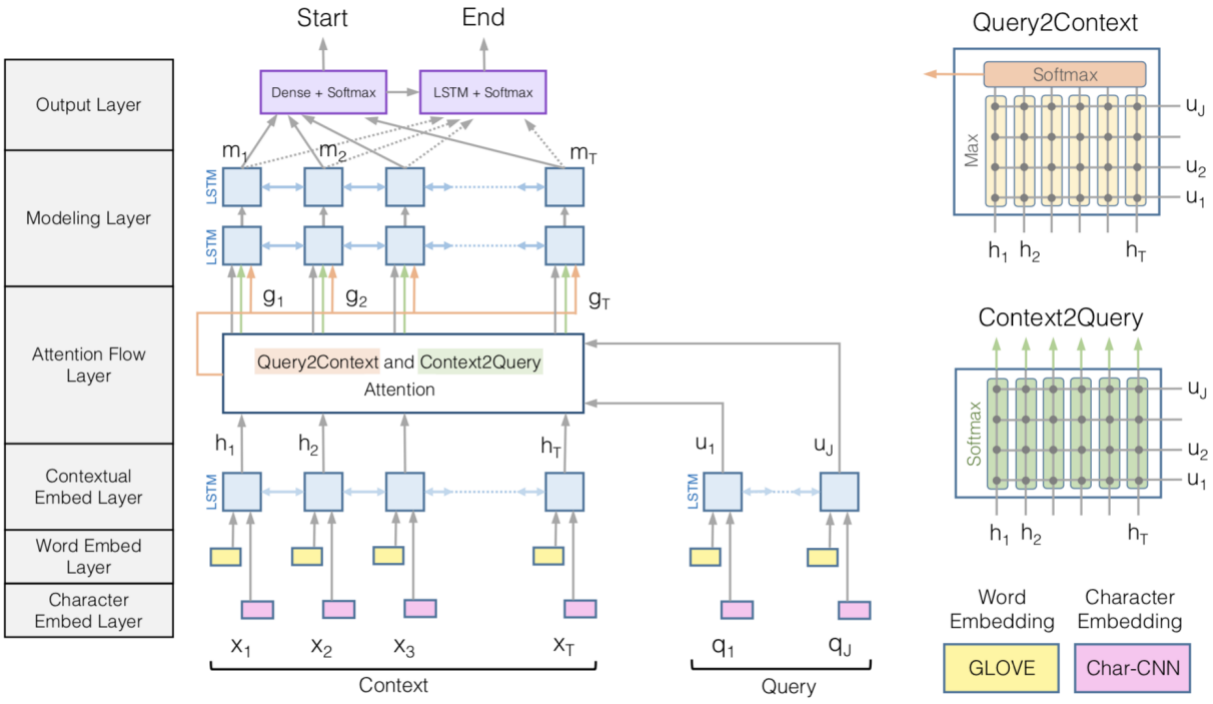

Figure 3-7. Architecture of BiDAF model. [43]

The model builds the representations of the context paragraph at different levels of granularity. The model consists of six layers:

- Character Embedding Layer maps each word to a vector space.
- Word Embedding Layer maps each word to the vector space using pre-trained word-embedding model.
- Contextual Embedding Layer utilizes contextual cues from surrounding words to refine the embedding of words.
- Attention Flow Layer produces a set of query-aware vectors for each word in the context
- Modelling Layer employs a RNN to scan the context.
- Output Layer provides an answer to the query.

The authors also introduced self-attention [44] between the passage and itself. The architecture of the updated model is shown in Figure 3-8.

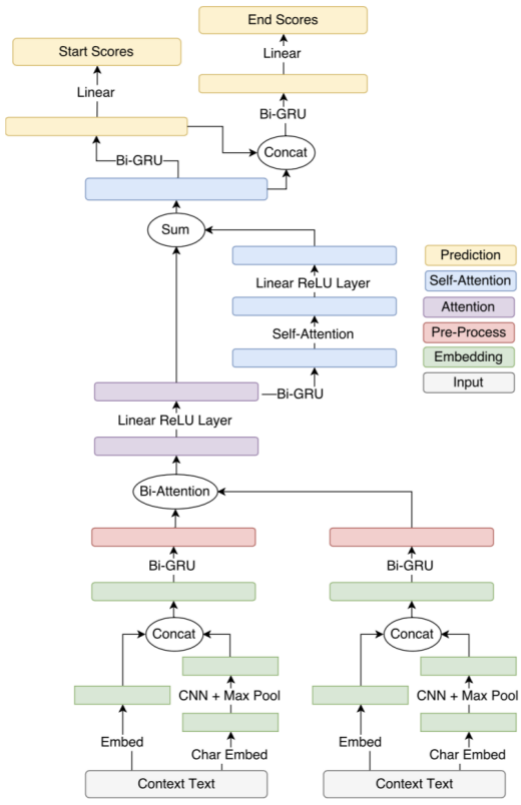

Figure 3-8. The Bi-Directional Attention Model with self-attention. [44]

The basic model is similar to the initial model except for the self-attention layer which matches the passage with itself. Since the context can be long and distant parts of text may rely on each other to fully understand the content, recent advances have proposed to fuse the context into itself. This is somewhat similar to the self-matching layer in R-NET. First, the character embedding and word embedding is computed and concatenated. This is passed to the pre-processing layer where a shared bi-directional GRU is used to map the question and passage embeddings to context-aware embeddings. The attention layer then calculates the attention matrix [44] using the equation:

$$a_{ij} = w1 \cdot h_i + w2 \cdot q_j + w3 \cdot (h_i \odot q_j) \quad (3\text{-}1)$$

where w1, w2 and w3 are learned vectors and $\odot$ is element-wise multiplication. This layer computes two attentions: context-to-query and query-to context. The context-to-query attention and query-to context attentions are computed as:

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{n_q} e^{a_{ij}}}$$

$$\mathbf{c_i} = \sum_{j=1}^{n_q} \mathbf{q_j} p_{ij}$$

Figure 3-9. Context-to-Query Attention vector. [44]

$$m_i = \max_{1 \le j \le n_q} a_{ij}$$

$$p_i = \frac{e^{m_i}}{\sum_{i=1}^{n_c} e^{m_i}}$$

$$\mathbf{q_c} = \sum_{i=1}^{n_c} \mathbf{h_i} p_i$$

Figure 3-10. Query-to-Context Attention vector. [44]

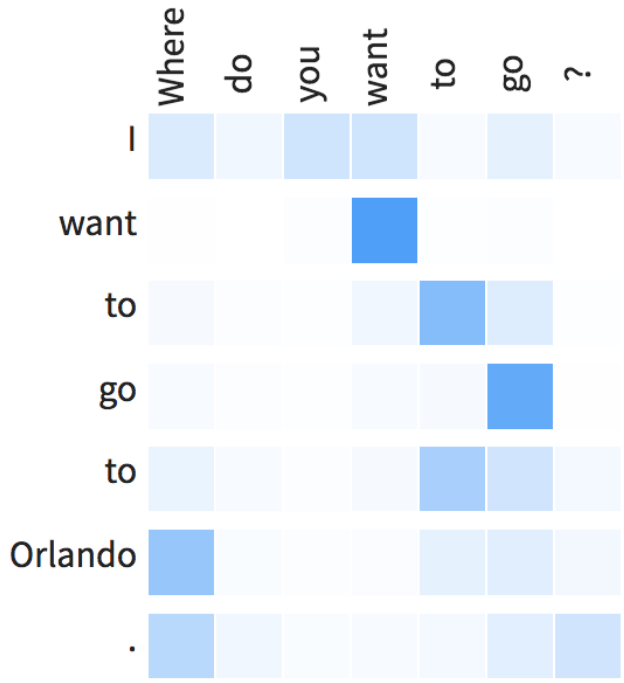

Figure 3-11. Attention Matrix for BiDAF

The Figure 3-11 illustrates the heatmap of a_{ij} matrix for a sample query and passage. To compute the context-to-query attention, we first take do a softmax over each row of the matrix to obtain p_{ij} matrix in Figure 3-9 [44]. The row c_i is query vector multiplied by these softmax values and summed. This gives us a vector with weighted query tokens. We can imagine that the query token with the highest matching will contribute the most to c_i vector. However, the weights of other query tokens are not lost. For example, if we see the third row of the Figure 3-11, we see that 'to' token in the context is matched with 'to' token in the query. This essentially signifies which query words are most relevant to each context word. To compute the query-to-context attention, we take the maximum of the values in a row. Then we apply softmax to these values to get the vector pi in Figure 3-10 [44]. Then, we multiply the context vector to these softmax values to get q_c. So, the query-to-context signifies which context words have the closest similarity to one of the query words. Finally, the vectors h_i, c_i, $h_i \odot c_i$,

and $q_c \odot c_i$ are concatenated, passed through a ReLU activation layer [13] and sent as input to the self-attention layer. The self-attention layer calculates the attention between the passage and itself. This is summed with Bi-Attention and sent to the prediction layer. The prediction layer uses two bi-directional GRU, each followed by a linear layer, to calculate start and end scores for each token. Then softmax is applied to get start and end probabilities to get the start and end tokens.

3.6 Reinforced Mnemonic Reader

The Reinforced Mnemonic Reader [45] model introduces a lot of novel mechanisms for the machine comprehension task. These include enhancing the capacity of the encoder, modeling long-term dependencies of contexts, refining the predicted answer span, and directly optimizing the evaluation metric. The model is shown in Figure 3-12 [45].

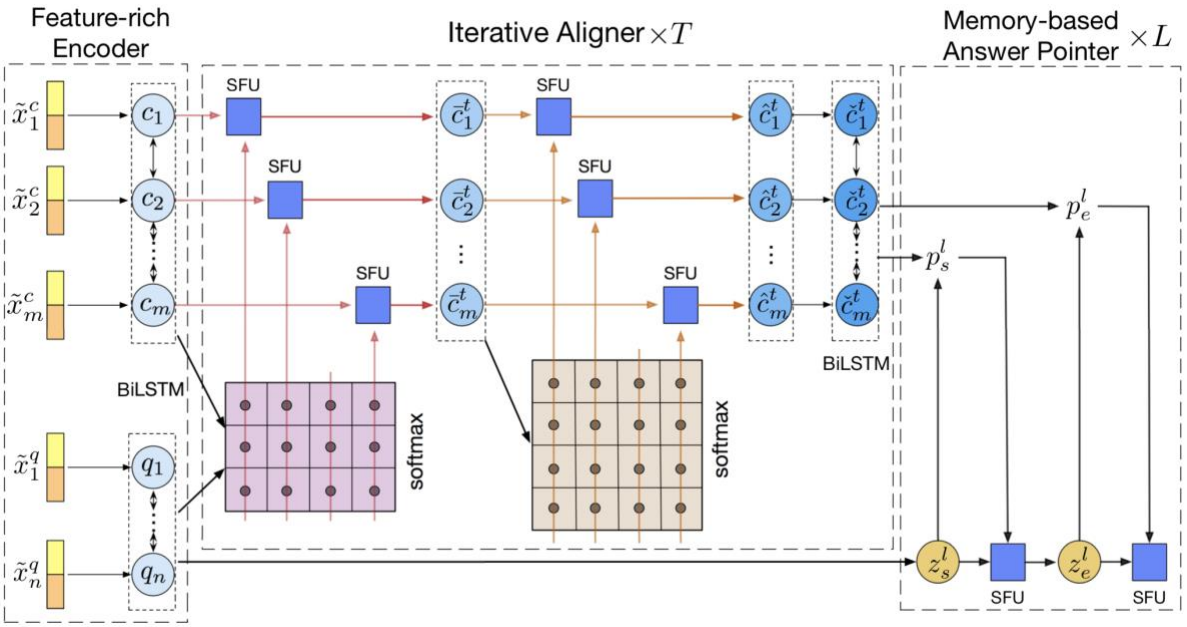

Figure 3-12. Architecture of Reinforced Mnemonic Reader.

In the encoding layer, the authors use parts-of-speech tags, named-entity tags, and the query category in addition to word-embedding and character embedding. The encoding layer also includes binary feature binary feature which indicates whether a word in context can be exactly matched to one query word and vice versa for a word in the query. The query category provides additional clues for searching the answer. For example, the “when” query focuses on temporal information whereas the “where” query focuses on spatial information. These embeddings are incorporated with the existing word-embeddings.

The authors introduce a Semantic Fusion Unit [45] where the output vector is expected to not only retrieve correlative information from fusion vectors but also retain partly unchanged as the input vector. This SFU is used in the Iterative Aligner and prediction layer. The iterative aligner uses multiple horizontal hops. The module updates context word representation in each hop. The iterative aligner has two sub-modules: the interactive aligning and self-aligning. The interactive aligner reads the query and context over *T* hops to capture the interaction between query and context and generates query-aware context representation. The coattention matrix is calculated between the *i-th* question word and *j-th* context word from the previous hop. For each context word, the intention is to find the most relevant query word by computing an attended query vector and fuse it back to the context word. The self-aligning module aligns the query-aware context representation with itself to synthesize contextual information among context words. This is similar to calculating the self-attention in BiDAF and self-matching in R-NET. This is done due to the limited capability of the recurrent neural network to model long-term dependencies of contexts.

The Memory-based Answer Pointer layer uses memory networks and pointer networks to predict the answer pointers. This module maintains a memory vector to record necessary reading knowledge for continuously refining the predicted answer span. This module runs for a total of *L* hops. This is similar to the iterative nature of the episodic memory of the Dynamic Memory Networks. The Semantic Fusion Unit is also used here for refinement of the memory vector.

Another important contribution of this paper is the use of Reinforcement Learning along with the standard maximum- likelihood estimation during training. The reason is that directly determining the exact boundary may be difficult in some situations where the answer boundary is fuzzy or too long. For example, it is quite hard to define the answer boundary of a “why” query, where the answer usually is not a distinguishable entity. The aim is to optimize the F1 score as Exact Match metric serves as a hard metric. The F1 score is taken as the reward in Reinforcement Learning algorithm and policy learning is used to maximize the model’s reward. The maximum-likelihood estimation is integrated with the reinforcement learning by using a linear interpolation. So, minimizing the loss optimizes both the F1 score and the Exact Match score.

3.7 FusionNet

The FusionNet [46] model makes three important contributions. It introduces a concept called “history of word” to characterize attention information from the lowest word-level embedding up to the highest semantic-level representation. It also identifies an attention scoring function to utilize the “history of word” concept. Finally, it proposes a fully-aware multi-level attention mechanism to capture the complete information in one text and exploit it in its counterpart layer by layer.

The authors make an important observation in the existing attention mechanisms. Taking image recognition as an example, information in various levels of representations can capture different aspects of details in an image: pixel, stroke and shape. This hypothesis also holds in language understanding and Machine Reading Comprehension. They propose an approach that utilizes all the information from the word embedding level up to the highest-level representation. The “History of Words” concept captures different levels of contextual information to fully understand the text.

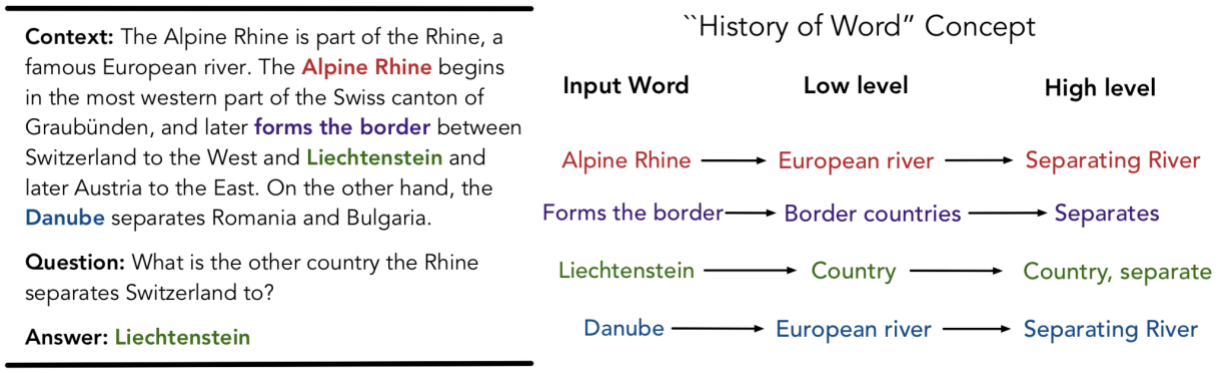

Figure 3-13. History of Word concept. [46]

The History of Word concept is shown in the above figure. If we consider the above example, both the high-level concept of *forms the border* and the word-level information of *Alpine Rhine* need to be considered. If only the high-level concept is considered, both *Alpine Rhine* and *Danube* will be answers. The History of Word of the *i-th* word, HoW_i, is the concatenation of all representations generated for this word, i.e., word embedding, multiple intermediate and output hidden vectors in RNN, and corresponding representation vectors in any further layers. A standard attention mechanism includes the following three steps:

- Computing an attention score
- Using softmax to form the attention weight
- Concatenating the word vectors with the summarized information

This is done in BiDAF, R-NET and Reinforced Mnemonic Reader, albeit in different ways. The FusionNet model uses History of Word, instead of the word vectors, to form the fully aware attention. The FusionNet model is shown in Figure 3-14. [46]

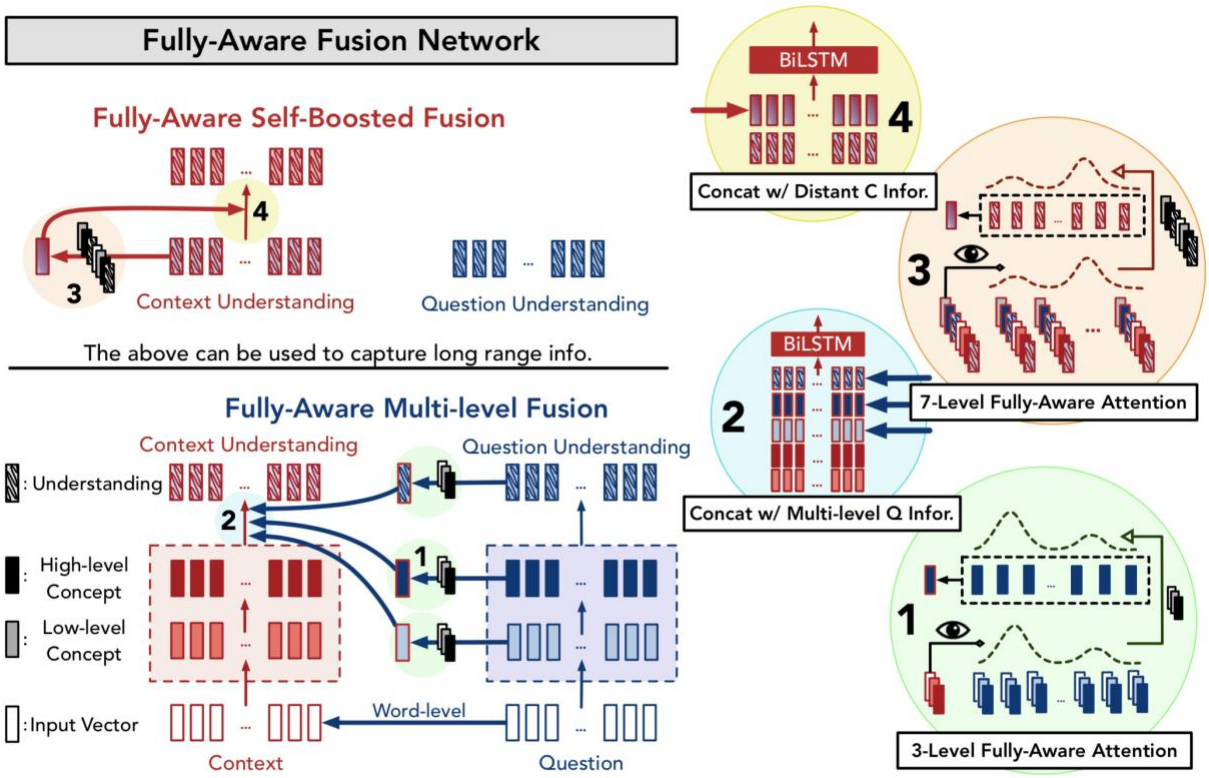

Figure 3-14. Architecture of FusionNet model. [46]

In the multi-level fusion, word-level and higher-level are fused separately. Word-level fusion informs *C* about what kind of words are in *Q*. The high-level fusion component fuses all higher-level information in the question *Q* to the context *C* through fully-aware attention on history-of-word. Circle 1 denotes the Fully-aware attention between context *C* and query *Q*. In circle 2, all the concepts in *C* are concatenated with multi-level *Q* information. Then, self-boosted fusion is used to consider distant parts in

the context. In circle 3, fully aware attention is applied to context *C* itself. Finally, in circle 4 the understanding of vector *C* is concatenated with self-attention information.

3.8 Summary

This chapter does a literature review on some of the existing state-of-the-art models for Machine Comprehension task. We start with Dynamic Memory Network which uses memory networks with multiple hops to refine the attention span. We then explore the Match-LSTM and Pointer Networks model which uses match-LSTM for textual entailment and pointer networks for answer prediction. The next model we explored was Char Aware which uses character embedding. This helps in solving the out-of-vocabulary problem. Then we looked at R-NET model which uses gated attention to assign different levels of importance to passage parts depending on their relevance to the question. This helps in masking out irrelevant passage parts and emphasizing the important ones. R-NET also uses self-matching to aggregate evidence from the whole passage to infer the answer. Bi-Directional Attention Flow was the next model we discussed, which uses three attentions: context-to-query, query-to-context and self-attention. The important thing BiDAF does is that it allows the attention to flow through the network instead of an early summarization. The next model, Reinforced Mnemonic Reader, introduces some novel mechanisms, like Feature rich encoder, which also does part-of-speech tagging, named entity recognition, query categories, and binary matching along with word-embedding and character-embedding. It also introduces Semantic Fusion Unit for iterative aligning where aligning is done in multiple hops. Finally, it uses Memory Networks and Pointer Networks for answer prediction and Reinforcement Learning along Maximum Likelihood Estimate to optimize F1 score and Exact Match. The FusionNet model introduces the “History of Word” concept to capture the word

representation at all levels which include low level and high-level representations. This forms the Fully-Aware attention.

If we look at these models, there are some common steps involved. The context and query are first encoded into vectors. Then attention is used over context and query in different ways. Some models also apply attention on the context with itself. Finally, most of the model use pointer networks to get the answer pointers. Figure 3-15 [46] shows the common steps involved in these models.

Architectures	(1)	(2)	(2')	(3)	(3')
Match-LSTM (Wang & Jiang, 2016)		✓			
DCN (Xiong et al., 2017)		✓			✓
FastQA (Weissenborn et al., 2017)	✓				
FastQAExt (Weissenborn et al., 2017)	✓	✓		✓	
BiDAF (Seo et al., 2017)		✓			✓
RaSoR (Lee et al., 2016)	✓		✓		
DrQA (Chen et al., 2017a)	✓				
MPCM (Wang et al., 2016)	✓	✓			
Mnemonic Reader (Hu et al., 2017)	✓	✓		✓	
R-net (Wang et al., 2017)		✓		✓	

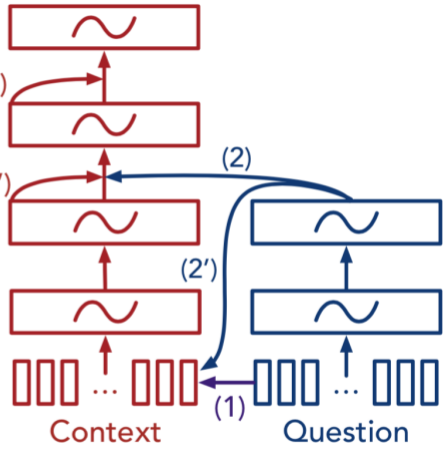

Figure 3-15. Summarized view of fusion process used. [46]

The rectangular box is the integration component and is usually implemented using an RNN such as an LSTM or GRU. The attention or fusion in done in multiple ways. (1) denotes the word-level fusion. (2) denotes the high-level fusion which informs the context about the semantic information in the question. (2') is also a kind of high-level fusion which fuses the high-level concept of the query to word-level of context. (3) is the self-boosted attention. This helps us refine passage representation with information from the whole passage. (3') is another kind of self-boosted attention where the self-boosted attention is conditioned on the query.

The following tables compare the performance of some of the state-of-the-art models on SQuAD dataset and TriviaQA dataset [45].

Model	Test Set EM/F1
QANet *Google Brain & CMU*	80.929/ 87.773
Reinforced Mnemonic Reader + A2D *Microsoft Research Asia & NUDT*	**80.919/ 87.492**
Reinforced Mnemonic Reader *NUDT and Fudan University*	79.545/ 86.654
BiDAF + Self Attention + ELMo *Allen Institute for Artificial Intelligence*	**78.580/85.833**
r-net *Microsoft Research Asia*	76.461/ 84.265
FusionNet *Microsoft Business AI Solutions team*	**75.968/ 83.900**
Match-LSTM with Bi-Ans-Ptr *Singapore Management University*	64.744/ 73.743

Figure 3-16. Performance comparison on SQuAD dataset

Model	Domain	Full		Verified	
		EM	F1	EM	F1
Classifier[1]	Wiki	22.5	26.5	27.2	31.4
BiDAF[2]		40.3	45.9	44.9	50.7
MEMEN[3]		43.2	46.9	49.3	55.8
M-Reader		**46.9**	**52.9**	**54.5**	**59.5**
Classifier[1]	Web	24.0	28.4	30.2	34.7
BiDAF[2]		40.7	47.1	49.5	55.8
MEMEN[3]		44.3	48.3	53.3	57.6
M-Reader		**46.7**	**52.9**	**57.0**	**61.5**

Figure 3-17. Performance comparison on TriviaQA dataset. [45]

In the next chapter, we will explore the Multi-Attention model which has been developed as part of the thesis.

CHAPTER 4
MULTI-ATTENTION

4.1 Multi-Attention For Machine Comprehension

In this section, we introduce the Multi-Attention for Machine Comprehension model. The model is built on top of the Bi-Directional Attention Flow model with self-attention. To understand the intuition behind the model, let us take the following example from SQuAD dataset:

P: The Panthers used the San Jose State practice facility and stayed at the San Jose Marriott. The Broncos practiced at Florida State Facility and stayed at the Santa Clara Marriott.

Q: At what university's facility did the Panthers practice?

In most of the existing state-of-the-art models for question answering, attention mechanisms have been used in one way or another to get the relevance of question words with respect to the context and vice-versa. For example, if we look at the above query *Q* and passage *P*, and focus on the attention phase of the models, the similarity matching will find words like ‘at’, ‘facility’, ‘practice’, and ‘Panthers’. One problem with such approaches is that they lose the individual question token information when creating a query-aware context. The question tokens are weighted to create the context-to-query vectors. In this process, some words with less relevance are given more weightage and important words are sometimes given less weight.

4.1.1 Intuition

In complex query questions, which generally happen in real life scenarios, many tokens in the query are required to create a meaningful question. We use Bi-RNN to map the question and passage embeddings to context-aware embeddings. However,

even though Bi-LSTMs and Bi-GRU should handle the long-term dependencies, in theory, tokens in complex queries require the context of the whole query and not just its neighbours. This is the intuition behind the multi-attention model. The multi-attention model is a hierarchical attention model which computes attention over query attended context vector and context attended query vector. The model adds a query-self-attention layer so that the query is given equal weightage as the context. The model then treats the output of the context-self-attention and query-self-attention layers as new context and query vectors and applies bi-attention and self-attention to get the final query-aware context vector.

4.1.2 Model

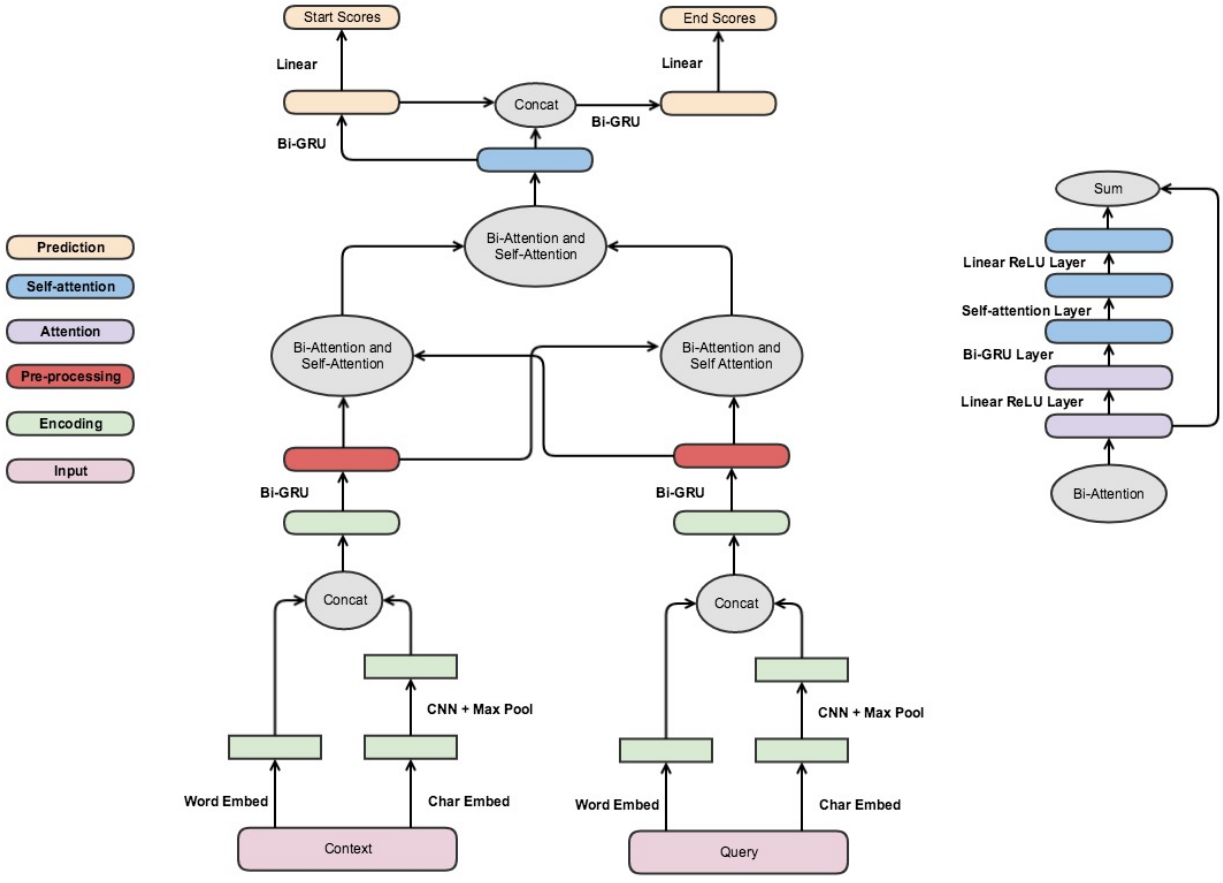

Figure 4-1. Multi-Attention for Machine Comprehension

Figure 4-1 illustrates the multi-attention model for machine comprehension. The model consists of following layers:

- Encoding Layer: The encoding layer converts the words from context *C* and query *Q* to their word-level embeddings and character-level embeddings. The character-level embeddings are generated by running a convolution neural network followed by max. The character-level and word-level embeddings are concatenated and passed to next layer.

- Pre-processing Layer: We use bi-directional GRU to map the context and query embeddings to context-aware embeddings.

- Context Attention Layer: The context attention layer uses the attention mechanism from the BiDAF (Bi-Directional Attention Flow) model to build a query-aware context representation. We first compute the attention matrix where the attention between context word *i* and query word *j* is defined as:

  $$a_{ij} = w1 \cdot h_i + w2 \cdot q_j + w3 \cdot (h_i \odot q_j) \quad (4\text{-}1)$$

  where w1, w2 and w3 are trainable weights, $\odot$ is the element-wise multiplication, h_i is the context word *i* and q_j is the query word *j*. Then we calculate the context-to-query attention vector c_i as:

  $$p_{ij} = softmax(a_{ij})_{j=1}^{n_q} \quad (4\text{-}2)$$

  $$c_i = \sum_{j=1}^{n_q} q_j p_{ij} \quad (4\text{-}3)$$

  We also calculate the query-to-context attention q_c as:

  $$m_i = \max_{j=1}^{n_q} a_{ij} \quad (4\text{-}4)$$

  $$p_i = softmax(m_i)_{i=1}^{n_c} \quad (4\text{-}5)$$

  $$q_c = \sum_{i=1}^{n_c} h_i p_i \quad (4\text{-}6)$$

  The vectors h_i, c_i, $h_i \odot c_i$ and $q_c \odot c_i$ are concatenated and are passed through a linear layer with ReLU activation and sent as input to the context-self attention layer.

- Context Self-Attention Layer: The context self-attention layer takes the output from context attention layer and passes it through a bi-directional GRU. Then we apply the context-to-query attention (like in the context attention layer) over this input with itself. a_{ij} is set to *-inf* if $i = j$. We again pass the concatenated output through the linear layer with ReLU activation and add the input of this layer with the output.

- Query Attention Layer: This is similar to the context attention layer, except that h_i is now the query word *i* and q_j is now the context word *j*. We calculate the context-to-query attention and query-to-context attention like in the context attention layer. We again pass the concatenated output through linear ReLU activation and pass it as input to the query self-attention layer.

- Query Self-Attention Layer: This is same as the context self-attention layer. The input is passed through bi-directional GRU, matched with itself and then passed through linear ReLU activation layer. The output is then summed up with the output.

- Context Query Bi-Attention Layer: The input to this layer are the outputs from context self-attention layer and the query self-attention layer. We treat the output from context self-attention output as new context vector *C'* and the query self-attention output as the new query vector *Q'*. We then calculate the bi-attention using the *C'* and *Q'*. This is similar to the context attention layer. The concatenated vectors are passed through the linear layer with ReLU activation and passed to the context query self-attention layer.

- Context Query Self-Attention Layer: This layer is similar to the context self-attention layer. We pass the input through a bi-directional GRU and then calculate the attention of the input with itself. We again pass the concatenated output through linear ReLU activation and add the input of this layer with the output. The final output is sent to the prediction layer.

- Prediction Layer: The prediction layer uses a bi-directional GRU followed by linear layer to compute the start scores of the answers for each context token. The hidden states of that layer are concatenated with the input and fed to another bi-directional GRU and linear layer to get the end scores of answers. Softmax is applied to start and end scores to get the start and end probabilities. Negative log-likelihood is optimized for selecting correct start and end tokens.

This concludes the architecture of the model.

4.1.3 Training

The training is done on the SQuAD dataset. SQuAD is a popular machine comprehension dataset consisting of 100,000+ questions created by crowd workers on 536 Wikipedia articles. Each context is a paragraph from an article and the answer to each question is guaranteed to be a span in the context. During training, Adadelta optimizer [47] is used and the batch size is set to 45. At test time, the most probable answer span of length 17 or less is selected. The GloVe 300-dimensional word vectors

are used for word embeddings. The dimensionality of size 100 for the GRUs and 200 for the linear layers are used after each attention mechanism. An exponential moving average of the weights with a decay rate of 0.999 is used.

4.1.4 Summary

We introduce the multi-attention model for machine comprehension. The model computes the context-to-query attention and query-to-context attention, with respect to the context as well as the query. This helps us solve complex query sentences up to some extent. The query-to-query attention extracts query information from all query words and not just the neighbors. The hierarchical nature of the model provides a deeper attention to the query attended context vectors and context attended query vectors. The model achieves 77.77 EM (Exact Match) and 85.44 F1 score on the SQuAD dev set.

4.2 Chatbot

Chatbots – also known as “conversational agents” – are software applications that mimic written or spoken human speech for the purposes of simulating a conversation or interaction with a real person. There has been a rapid increase in the use of chatbots in customer care automation, flight booking, e-commerce sites etc. Applications like Facebook Messenger and Google Assistant also use conversational agents for various tasks. A chatbot can be specific to a particular domain, like flight booking, or can be a generic question answering system. While there is still a long way for chatbots to replace real human interaction, they are doing a decent job to simulate smaller subtasks. For example, we can easily book tickets through google assistant or ask for the latest movies running nearby.

Chatbots are one of the popular problems in Natural Language Processing. The question answering models discussed in the previous section allow for an interesting use case of chatbots. A simple chatbot for flight booking was designed as a part of this thesis. It has been integrated with OneTask, a task manager for intra-project collaborations, which is currently under development in the NSF Center for Big Learning, University of Florida. The chatbot server is built using Python Flask. We have exposed REST APIs which are consumed by the OneTask Vue.js server to get and send responses. The basic flow is as follows:

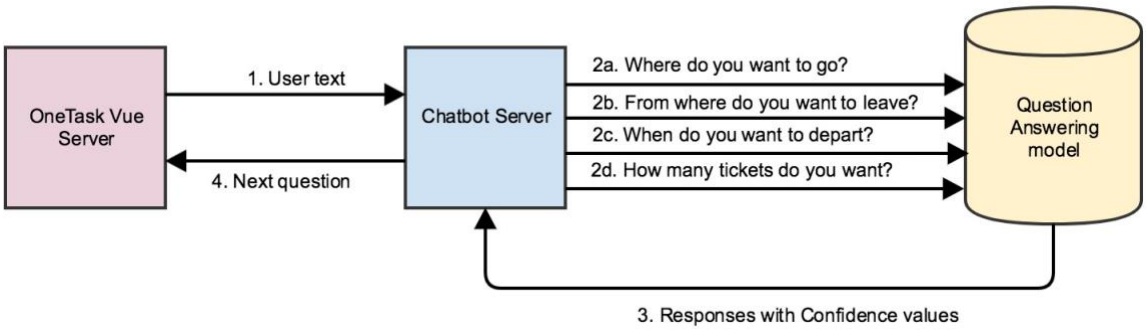

Figure 4-2. Chatbot module flow

The aim of the chatbot server is to get the values of four entities: source, destination, date and number of tickets for the travel.

1. The conversation starts with the system greeting the user. The user specifies an open-ended query to the system. For example, the query can be “I want two tickets from Orlando to Gainesville”.

2. The system then treats this text as the passage and runs four questions on the passage. These questions are:

    - Where do you want to go?
    - From where do you want to leave?
    - When do you want to depart?
    - How many tickets do you want?

3. The chatbot server gets the response to the questions along with their confidence values. The confidence threshold is set to 7. If the question response

gets a confidence value of greater than equal to 7, the question is marked as ‘satisfied’. The server then sorts the questions based on the confidence value and sends the next question which has the lowest confidence value and has not been ‘satisfied’. The server also updates the passage with a ‘prefix’ for the question. This is treated as the new passage.

4. The next question is sent to the OneTask server which displays the question to the user.

The process goes on until all the questions are marked as ‘satisfied’. In some cases, the chatbot might not get a good confidence value for the answer predicted by the model. In that case, the server asks the user to confirm whether the predicted answer is correct. If the user replies ‘yes’, the question is marked as ‘satisfied’. The user interface of the OneTask dashboard along with a sample conversation has been shown in Figure 4-3.

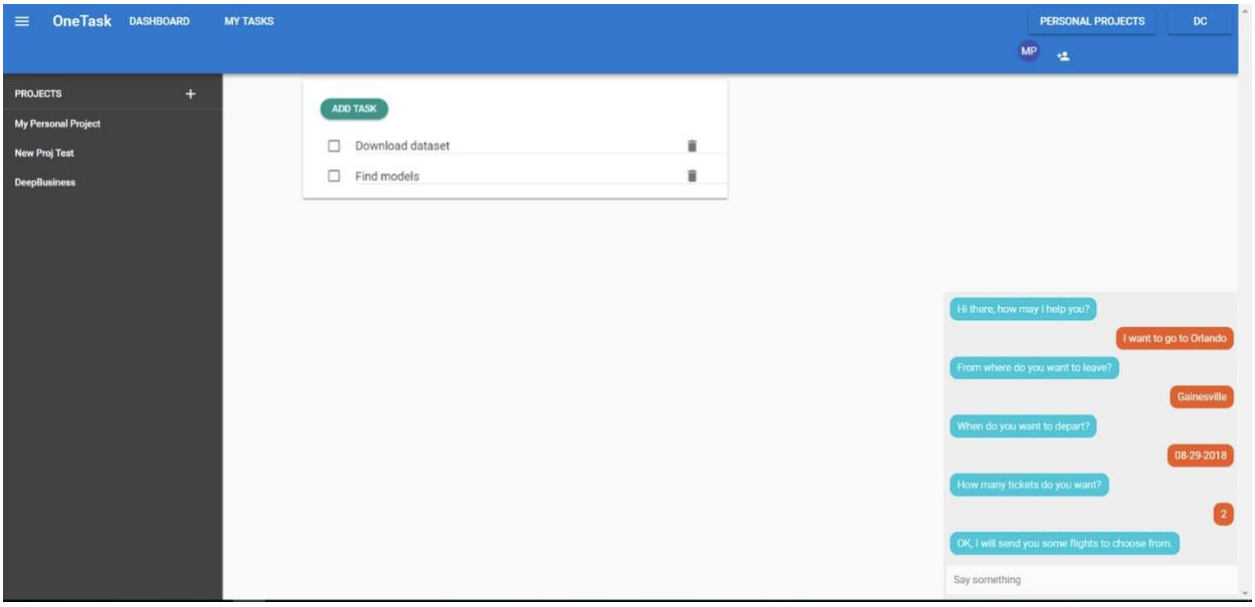

Figure 4-3. OneTask Chatbot

4.3 Attention Visualization

A user interface for attention visualization has also been built as a part of the thesis. The users can add their own passage and question or select from a pre-populated passage-question pair from the dropdown to visualize the attention. The

interface helps us in finding what the top ten answers predicted by the model are and what their confidence values are. As a part of future efforts, we want to make this website public and give the user the capability to select the correct answers for their own passage and question, if the answer predicted by the model is wrong. This will help us in augmenting the training data. Currently, some of the samples from the SQuAD datasets have been added, which are not predicted correctly by the Bi-Directional Attention Flow model. We want to also include other state-of-the-art models and the users would be able to see the performance of several models on their own passage and question. A sample passage-question pair has been shown in Figure 4-4. The user interface consists of two main frames. The left frame contains the dropdown, containing some passage-question pair, and two text boxes, one for the passage and one for the question. Clicking on the Run button will run the passage and question on the model. The right frame contains the whole passage, with the closest answers highlighted. The darker the color of the highlighted text, the higher is the confidence value of the prediction. For example, in the sample passage-question pair, the candidate answers are “San Jose State” and “Florida State university Facility”. The answer predicted by the model is “Florida State university Facility”. The visualization also shows the confidence values of the candidate answers.

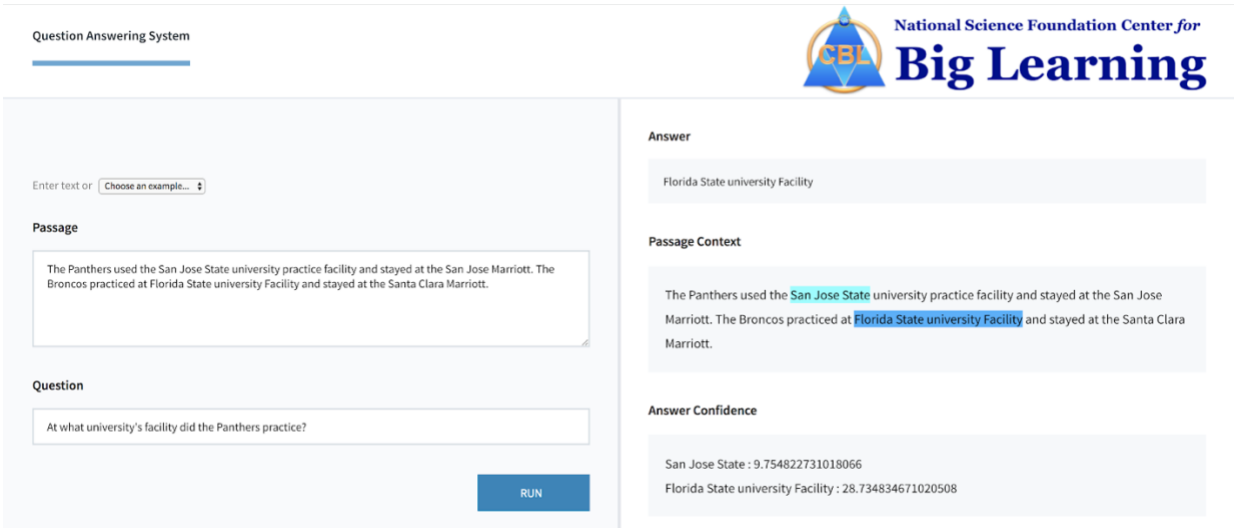

Figure 4-4. Attention Visualization on wrong samples

CHAPTER 5
CONCLUSION AND FUTURE DIRECTION

In this thesis, we first start with understanding the Machine Comprehension task. We went on to cover the deep learning basics which included CNN, RNN, Word Embeddings, Attention Mechanism, Pointer Networks, Memory Networks, and Reinforcement Learning. These form an essential base in understanding the state-of-the-art models for Question Answering task. We then covered some of the state-of-the-art models which included Dynamic Memory Network, Match-LSTM with Answer Pointer, Char Aware, R-NET, BiDAF, Reinforced Mnemonic Reader and FusionNet. These models are common in several ways and we looked over some of the important concepts introduced by them. Finally, we discussed the Multi-Attention for Machine Comprehension model, which was designed as a part of the thesis. We built the intuition behind query-to-query attention and hierarchical attention. We then present the flow of the flight booking chatbot. We also present the Attention Visualization user interface.

Question-Answering task is one of the rapidly growing research areas in NLP and Deep Learning. The sheer power of computers to read the passage and question, logic and reason and predict the answer, presents some very interesting research opportunities. Recently, the Reinforced Mnemonic Reader + A2D (ensemble model) by Microsoft & NUDT surpassed the human performance on the SQuAD dataset in Exact Match metric. The attention mechanisms have been researched for a long time and there are various improvements and variations that have been proposed over time. One area that still needs to be improved is capturing the semantics of passages and questions. The deep learning models have been able to compute attention but have not been able to capture the semantics properly. POS tagging, NER, Memory hops helps in

this regard up to some extent. Recently, Geoffrey Hinton, the prominent deep-learning researcher now at Google, popularized a term called “Thought Vector” [48]. In a speech to the Royal Society of London in 2015, he said:

"The implications of this for document processing are very important. If we convert a sentence into a vector that captures the meaning of the sentence, then Google can do much better searches; they can search based on what's being said in a document. Also, if you can convert each sentence in a document into a vector, then you can take that sequence of vectors and [try to model] natural reasoning. And that was something that old fashioned AI could never do. If we can read every English document on the web, and turn each sentence into a thought vector, you've got plenty of data for training a system that can reason like people do. Now, you might not want it to reason like people do, but at least we can see what they would think. What I think is going to happen over the next few years is this ability to turn sentences into thought vectors is going to rapidly change the level at which we can understand documents. To understand it at a human level, we're probably going to need human level resources and we have trillions of connections [in our brains], but the biggest networks we have built so far only have billions of connections. So, we're a few orders of magnitude off, but I'm sure the hardware people will fix that."

A thought vector is a vectorized thought, and the vector represents one thought’s relations to others. A thought vector is trained to generate a thought’s context. Like the words are linked by the grammar (a sentence is just a path drawn across words), the thoughts are linked by a chain of reasoning, a logical path. Traditional, rules-based AI, a pile of if-then statements locking brittle symbols into hard-coded relationships with others, is not flexible enough to represent the world without near infinite amounts of human intervention. Symbolic logic and knowledge graphs may establish strict relations between entities, but those relations are unlikely to adapt quickly to the new. There have been some approaches for thought vectorization like Doc2Vec [49], skip-thought vectors and Seq2seq bilingual translation.

One of the research directions is the use of thought vectors, if possible, along with the existing word and character level embeddings. This can help in capturing

semantics and can also help in a better reasoning. Another interesting research direction is exploring the effectiveness of Capsule Networks [50] in context embedding instead of Bi-RNNs that have been used in the presented models. Capsule Networks have been able to effectively capture deep features of an image and in theory, they will provide help in capturing the structure of sentences. For example, Capsule Networks can recognize the image effectively even if the images are tilted or rotated. This is analogous to two sentences being paraphrases of each other. Another concept that can prove useful in capturing the relation between the words of the sentence is Syntax trees. A syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. We can add the tagging like Noun Phrase and Verb Phrase along with the POS tagging. This can enhance the representation of a word. Figure 5-1 [51] shows a syntax tree.

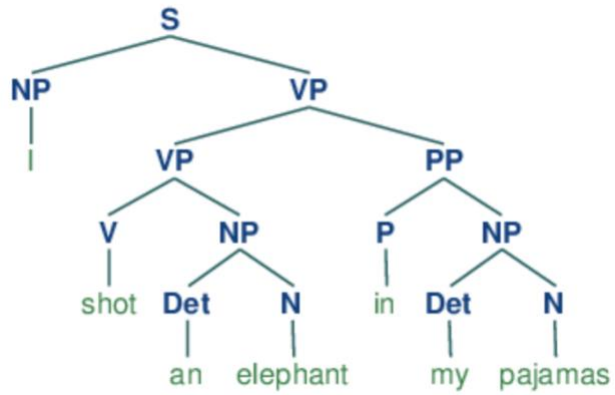

Figure 5-1. Syntax Tree

We hope that this thesis contributes to the ongoing research in Machine Comprehension task and helps in developing effective chatbot systems.

LIST OF REFERENCES

[1] Chen, Danqi, Adam Fisch, Jason Weston, and Antoine Bordes. "Reading wikipedia to answer open-domain questions." *arXiv preprint arXiv:1704.00051* (2017).

[2] Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang. "Squad: 100,000+ questions for machine comprehension of text." *arXiv preprint arXiv:1606.05250* (2016).

[3] Weston, J., Chopra, S., and Bordes, A. “Memory Networks.” *arXiv preprint arXiv:1410.3916v11* (2015).

[4] Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. "End-to-end memory networks." In *Advances in neural information processing systems*, pp. 2440-2448. 2015.

[5] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

[6] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).

[7] "Slot Filling | Dialogflow". 2018. *Dialogflow*. Accessed March 16 2018. https://dialogflow.com/docs/concepts/slot-filling.

[8] Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. Cambridge: MIT press, 2016.

[9] Chris V. Nicholson, Skymind team. 2018. " Introduction To Deep Neural Networks (Deep Learning) - Deeplearning4j: Open-Source, Distributed Deep Learning For The JVM ". *Deeplearning4j.Org*. Accessed March 16 2018. https://deeplearning4j.org/neuralnet-overview.

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[11] "Cs231n Convolutional Neural Networks For Visual Recognition". 2018. *Cs231n.Github.Io*. Accessed March 16 2018. http://cs231n.github.io/convolutional-networks/.

[12] "Applied Deep Learning - Part 4: Convolutional Neural Networks". 2017. *Towards Data Science*. Accessed March 16 2018. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[13] "Rectifier (Neural Networks)". 2018. *En.Wikipedia.Org*. Accessed March 16 2018. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[14] "Softmax Function". 2018. *En.Wikipedia.Org*. Accessed March 16 2018. https://en.wikipedia.org/wiki/Softmax_function.

[15] Elman, Jeffrey L. "Distributed representations, simple recurrent networks, and grammatical structure." *Machine learning* 7, no. 2-3 (1991): 195-225.

[16] "Introduction To Recurrent Neural Network – Towards Data Science". 2017. *Towards Data Science*. Accessed March 16 2018. https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3.

[17] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks.” In *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[18] "Understanding LSTM Networks -- Colah's Blog". 2018. *Colah.Github.Io*. Accessed March 16 2018. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[19] "Deep Learning, NLP, And Representations - Colah's Blog". 2018. *Colah.Github.Io*. Accessed March 16 2018. http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/.

[20] Passos, Alexandre, Vineet Kumar, and Andrew McCallum. "Lexicon infused phrase embeddings for named entity resolution." *arXiv preprint arXiv:1404.5367* (2014).

[21] Chris V. Nicholson, Skymind team. 2018. " Word2vec, Doc2vec & Glove: Neural Word Embeddings For Natural Language Processing - Deeplearning4j: Open-Source, Distributed Deep Learning For The JVM ". *Deeplearning4j.Org*. Accessed March 16 2018. https://deeplearning4j.org/word2vec.html.

[22] Pennington, Jeffrey. 2018. "Glove: Global Vectors For Word Representation". *Nlp.Stanford.Edu*. Accessed March 16 2018. https://nlp.stanford.edu/projects/glove/.

[23] "Salesforce Research". 2018. *Einstein.Ai*. Accessed March 16 2018. https://einstein.ai/research/learned-in-translation-contextualized-word-vectors.

[24] Brownlee, Jason. 2017. "Attention In Long Short-Term Memory Recurrent Neural Networks - Machine Learning Mastery". *Machine Learning Mastery*. Accessed March 16 2018. https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/.

[25] "Attention And Memory In Deep Learning And NLP". 2016. *Wildml*. Accessed March 16 2018. http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/.

[26] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473* (2014).

[27] "How To Visualize Your Recurrent Neural Network With Attention In Keras". 2017. *Medium*. Accessed March 16 2018. https://medium.com/datalogue/attention-in-keras-1892773a4f22.

[28] "Pointer Networks In Tensorflow (With Sample Code) – Dev Nag – Medium". 2016. *Medium*. Accessed March 16 2018. https://medium.com/@devnag/pointer-networks-in-tensorflow-with-sample-code-14645063f264.

[29] Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. "Pointer networks." In *Advances in Neural Information Processing Systems*, pp. 2692-2700. 2015.

[30] Z., Zygmunt. 2017. "Introduction To Pointer Networks - Fastml". *Fastml.Com*. Accessed March 16 2018. http://fastml.com/introduction-to-pointer-networks/.

[31] "Introduction To Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)". 2018. *Towards Data Science*. Accessed March 16 2018. https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287.

[32] "Deep Reinforcement Learning: Pong From Pixels". 2018. *Karpathy.Github.Io*. Accessed March 16 2018. http://karpathy.github.io/2016/05/31/rl/.

[33] "Machine Learning For Humans, Part 5: Reinforcement Learning". 2017. *Medium*. Accessed March 16 2018. https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265.

[34] "Deep Reinforcement Learning | Deepmind". 2016. *Deepmind*. Accessed March 16 2018. https://deepmind.com/blog/deep-reinforcement-learning/.

[35] Kumar, Ankit, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. "Ask me anything: Dynamic memory networks for natural language processing." In *International Conference on Machine Learning*, pp. 1378-1387. 2016.

[36] "Babi". 2018. *Facebook Research*. Accessed March 16 2018. https://research.fb.com/downloads/babi/.

[37] Joshi, Mandar, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension." *arXiv preprint arXiv:1705.03551* (2017).

[38] Wang, Shuohang, and Jing Jiang. "Machine comprehension using match-lstm and answer pointer." *arXiv preprint arXiv:1608.07905* (2016).

[39] Wang, Shuohang, and Jing Jiang. "Learning natural language inference with LSTM." *arXiv preprint arXiv:1512.08849* (2015).

[40] Kim, Yoon, Yacine Jernite, David Sontag, and Alexander M. Rush. "Character-Aware Neural Language Models." In *AAAI*, pp. 2741-2749. 2016.

[41] Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. "Highway networks." *arXiv preprint arXiv:1505.00387* (2015).

[42] Wang, Wenhui, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. "Gated self-matching networks for reading comprehension and question answering." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, pp. 189-198. 2017.

[43] Seo, Minjoon, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. "Bidirectional attention flow for machine comprehension." *arXiv preprint arXiv:1611.01603* (2016).

[44] Clark, Christopher, and Matt Gardner. "Simple and effective multi-paragraph reading comprehension." *arXiv preprint arXiv:1710.10723* (2017).

[45] Hu, Minghao, Yuxing Peng, and Xipeng Qiu. "Reinforced mnemonic reader for machine comprehension." *CoRR, abs/1705.02798* (2017).

[46] Quan, Tran Minh, David GC Hilderbrand, and Won-Ki Jeong. "FusionNet: A deep fully residual convolutional neural network for image segmentation in connectomics." *arXiv preprint arXiv:1612.05360* (2016).

[47] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

[48] Chris V. Nicholson, Skymind team. 2018. " Thought Vectors, Deep Learning & The Future Of AI - Deeplearning4j: Open-Source, Distributed Deep Learning For The JVM ". *Deeplearning4j.Org*. Accessed March 16 2018. https://deeplearning4j.org/thoughtvectors.

[49] Chris V. Nicholson, Skymind team. 2018. " Doc2vec, Or Paragraph Vectors, In Deeplearning4j - Deeplearning4j: Open-Source, Distributed Deep Learning For The JVM ". *Deeplearning4j.Org*. Accessed March 16 2018. https://deeplearning4j.org/doc2vec.html.

[50] Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." In *Advances in Neural Information Processing Systems*, pp. 3859-3869. 2017.

[51] "8. Analyzing Sentence Structure". 2018. *Nltk.Org*. Accessed March 16 2018. http://www.nltk.org/book/ch08.html.

BIOGRAPHICAL SKETCH

Yash Sinha was born in Gaya, India and did his schooling in Gaya. He earned his bachelor's degree in MSc. (Tech.) Information Systems at BITS-Pilani K.K. Birla Goa Campus, India. He then worked at CA Technologies, Hyderabad, India as a Software Engineer for two years. He has also worked as an intern for six months at VMware Software India Private Limited, India. He then pursued a Master of Science degree in computer science at the University of Florida, USA. He worked at the National Science Foundation, Center of Big Learning at the University of Florida as a Graduate Student Researcher. His academic interests include Deep Learning, Machine Learning, Reinforcement Learning, Question Answering systems and Data Science.