University of Information Technology

Faculty of Computer Science

Knowledge Engineering, Final Year Project

Question Answering System using Bi-Directional Attention Flow

5KE- 6 Ma Khine Myat Thwe

5KE- 10 Ma Su Myat Noe

ACKNOWLEDGEMENT

We thank our colleagues from University of Information Technology, Yangon who provided insight and expertise that greatly assisted the research, although they may not agree with all of the interpretations/conclusions of this paper. We thank our supervisors Dr. Thiri Hay Mar Kyaw, Head of Computer Science Department and Daw May Soe Htay.

We would also like to show our gratitude to the Dr. Saw Sandar Aye , rector of the University of Information Technology, and Dr. Thin Thin Wai and Dr. Khin Thandar Nwet, our respective teachers from Department of Computer Science, for sharing their pearls of wisdom with us during the course of this research and for comments that greatly improved our project.

ABSTRACT

Question answering is one of the major goals in natural language processing. While high-performance general question-answering remains beyond the reach of the machine learning field, computers are capable of strong performance in many domain-specific question-answering tasks, and they can perform quite well when substantial restrictions are placed on the problem domain. Question answering remains one of the most difficult challenges we face in Natural Language Processing. The idea of creating an agent capable of open-domain question answering - answering arbitrary questions with respect to arbitrary documents - has long captured our imagination. Our research involves the Machine comprehension (MC), the objective of the system is answering a question about a given context paragraph.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

In the past decade, most of the Question Answering System based on Information Retrieval. But nowadays the advancement of the machine learning and deep neural networks, QA has recently received attention from the information retrieval, information extraction, machine learning, and natural language processing communities. To overcome the burden of human crafted rules in building a QA system, the machine learning approach has been an alternative.

The tasks of machine comprehension (MC) and question answering (QA) have gained significant popularity over the past few years within the natural language processing and computer vision communities. Systems trained end-to-end now achieve promising results on a variety of tasks in the text and image domains. One of the key factors to the advancement has been the use of neural attention mechanism, which enables the system to focus on a targeted area within a context paragraph. Attention mechanisms in previous works typically have one or more of the following characteristics. First, the computed attention weights are often used to extract the most relevant information from the context for answering the question by summarizing the context into a fixed-size vector. Second, in the text domain, they are often temporally dynamic, whereby the attention weights at the current time step are a function of the attended vector at the previous time step. Third, they are usually uni-directional, wherein the query attends on the context paragraph or the image.

In this paper, we introduce the Bi-Directional Attention Flow (BI DAF) network, a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. BI DAF includes character-level, word-level, and contextual embeddings, and uses bi-directional attention flow to obtain a query-aware context representation. Our attention mechanism offers following improvements to the previously popular attention paradigms. First, our attention layer is not used to summarize the context paragraph into a fixed-size vector. Instead, the attention is computed for every time step, and the attended vector at each time step, along with the representations from previous layers, is allowed to flow through to the subsequent modeling layer. This reduces the information loss caused by early summarization. Second, we use a memory-less attention mechanism. That is, while we iteratively compute attention through time as in [Bahdanau et al. (2015)], 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. We hypothesize that this simplification leads to the division of labor between the attention layer and the modeling layer. It forces the attention layer to focus 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 output of the attention layer). It also allows the attention at each time step to be unaffected from incorrect attendances at previous time steps. Our experiments show that memory-less attention gives a clear advantage over dynamic attention. Third, we use attention mechanisms in both directions, query-to-context and context-to-query, which provide complimentary information to each other. Our BI DAF model1 outperforms all previous approaches on the highly-competitive Stanford Question Answering Dataset (SQuAD) test set leaderboard at the time of submission. We also provide an in-depth ablation study of our model on the SQuAD development set, visualize the intermediate feature spaces in our model, and analyze its performance as compared to a more traditional language model for machine comprehension [Rajpurkar et al., 2016)].

CHAPTER 2
BACKGROUND THEORY

2.1 Character-level CNN

Character-level encodings are becoming increasingly popular recently. Compared to word vectors, they offer the advantages of allowing us to condition on the internal structure of words (this is known as morphology), and better handle out-of-vocabulary words. Suppose we have a word w which consists of characters c1, . . . , cL. We usually start by representing the characters using trainable character embeddings e1, . . . , eL. You can think of character embeddings as analogous to word embeddings, though character embeddings typically have smaller dimension dc, and the 'vocabulary size' is much smaller (just the size of the alphabet, plus some digits and punctuation). In a character-level Convolutional Neural Network (CNN), we take our character embeddings e1, . . . , eL 2 Rdc, and compute another sequence of hidden representations h1, . . . , hL 2 Rf . The core idea is that each hi is computed based on a window of characters [ci−k, . . . , ci, . . . , ci+k] centered at position i (k is the window width). Finally, you apply max pooling to obtain the character-level encoding: embchar(w) = maxi hi. Typically, once we have obtained the character-level encoding embchar(w), we concatenate it with the usual pretrained word embedding embword(w) to get a hybrid representation for w. You could augment the baseline by using hybrid representations in place of just pretrained word embeddings. If you choose to implement this, you will need to start by creating a new tf.placeholder of shape (batch_size, context_len, word_len) to feed in the character IDs for e.g. the context. (word_len, similar to context_len, will be the maximum word length that the model can process). To get started with character CNNs, look at the TensorFlow function tf.layers.conv1d. The two important parameters are filters, which corresponds to f (dimension of the output states hi) here, and kernel_size, which corresponds to k (window width) here. We found that dc = 20, k = 5, f = 100 gives reasonable performance, though we did not perform an exhaustive hyper-parameter search. You could also construct a multi-layer character-CNN.

One way to improve the treatment of out-of-vocabulary words is to implement a character-level model. Our baseline represents input words using pre-trained fixed GloVe embeddings. Out-of-vocabulary words (i.e. those that don't have a GloVe embedding) are represented by the UNK token. This is a special token in the vocabulary that has its own fixed word vector (set to a small random value).

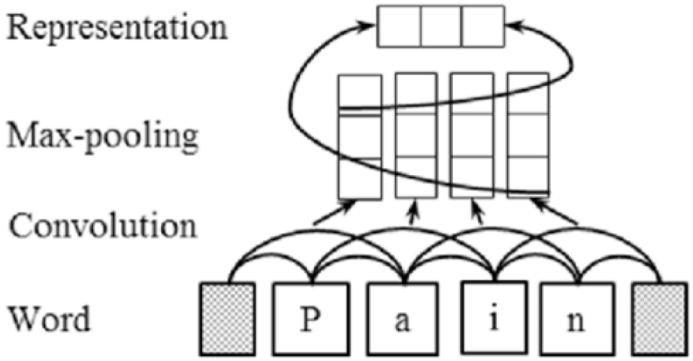

Figure 2.1 : Character Level CNN

Example: “ello” à”hello”

“pai”à”pain”

“pple”à “apple”

Input words will be fixed size vector for efficient calculation for next layers.

2.2 GloVe Word Embeddings

We used pertained GlovVE vectors to represent the words in the context and question. Empirically, we found the model performance did not vary appreciably with the size of the embeddings, so we defaulted to use size h = 100 embeddings. That is,

2.3 Additional Input Features

In addition to character CNNs, there are a number of input features that have the potential to dramatically increase the accuracy and reliability of the model. One particular input feature that we explored is the exact match feature.

This is how the exact match input feature works: given all the context tokens in the context, we can add an additional Boolean feature that has value 1 if the context token appears in the question and value 0 of if it doesn’t. The rationale behind this is that a context token is more likely to be included in the final answer if it also appears

2.4 Attention Mechanism

1.Attention layer is not used to summarize the context paragraph into a fixed-size vector which is allowed to flow through to the subsequent modeling layer.

2.Use a memory-less attention mechanism :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. (leads to the division of labor between the attention layer and the modeling layer)

3.Use attention mechanisms in both directions, query-to-context and context-to-query, which provide complimentary information to each other.

2.5 LSTM Networks

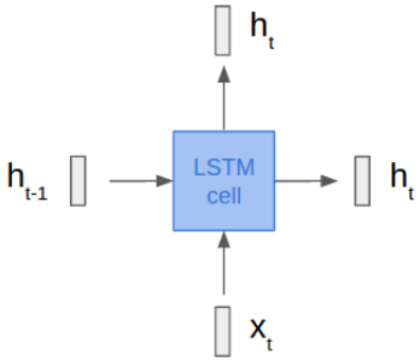

Figure 2.2 : LSTM Networks

Our outside view of the LSTM cell, which we will use as our fundamental unit for more complex recurrent structures. For the purpose of this report we will build up from the LSTM cell, considering only the equations such as ht = LSTM(xt, ht−1), where a more thorough treatment of LSTMs can be found on Christopher Olah's blog [2]. Further, for the entirety of this section we will us d × T as the size of the input to an LSTM network, and we will take d 0 to be the state size of the LSTM(s), thus the output will be of size d 0 × T.

To use LSTMs we create a rolled out network, where the (internal) weights are shared between each of the cells ,

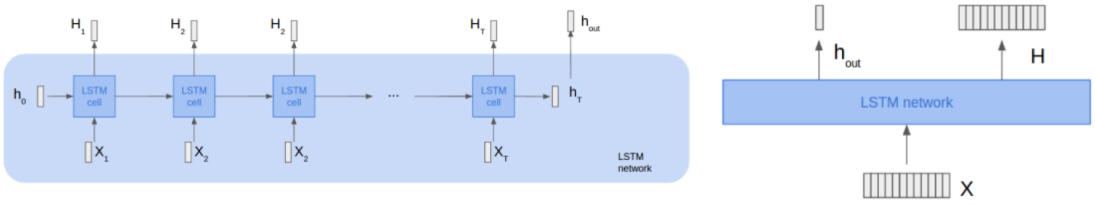

Figure 2.3 : Left, a rolled out LSTM network, one layer deep, and right, a block used to represent this network.

In the network from figure 4 we have one input and two outputs, which are X ∈ R d×T, H ∈ R d 0×T and hout ∈ R d 0 respectively. The equations for the network are H:0 = 0, H:t = LSTM (X:t, H:(t−1)), hout = H:T , noting that H:0 is separate from H. We could set H:0 to something non-zero, however in all of the models we consider in this report every LSTM has its state initially set to 0.

2.6 Bi-Directional LSTM Networks

In a similar fashion to LSTM networks, we can define a recurrent network that can at each step utilize information from the future and the past, not only the past.

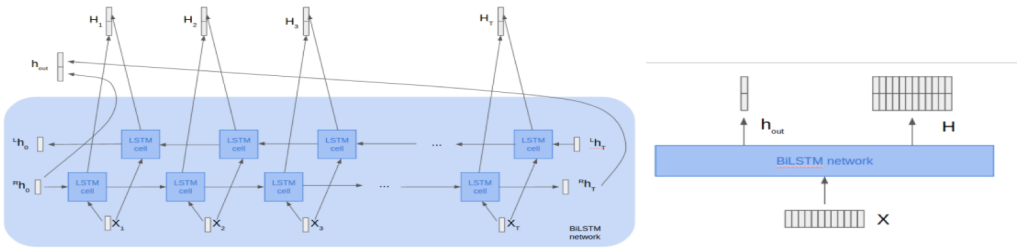

Figure 2.4 : Left, a rolled out BiLSTM network, one layer deep, and right, a block used to represent this network.

Again, we consider a rolled out network in figure 5, however, noticeably we now have that H ∈ R 2d 0×T and hout ∈ R 2d 0 , as we concatenate the outputs from the forward and backward networks. In figure 5 we again reduce it to a single block. The equations for this network are a similar to those in section.

$$h_0^R = 0, \qquad h_t^R = \mathrm{LSTM}(X_{:t}, h_{t-1}^R), \qquad h_T^L = 0,$$
$$h_t^L = \mathrm{LSTM}(X_{:t}, h_{t+1}^L), \qquad H_{:t} = [h_t^R; h_t^L], \qquad h_{out} = [h_T^R; h_1^L]$$

[Equation 2.1]

2.7 Deep (Bi)LSTM networks

Finally, we can chain the rolled out LSTM networks defined in sections 2.4 and 2.6 to create deep recurrent networks. Writing h, H = LSTM(X) as shorthand for the network in figure 4 and it's corresponding equations. We can then define a deep network of depth D with the following equations.

$$H_0 = X, \qquad h_i, H_i = \mathrm{LSTM}(H_{i-1}), \qquad H_{out} = H_D, \qquad h_{out} = [h_1; ...; h_d].$$ [Equation 2.2]

In exactly the same fashion we construct a deep BiLSTM network, simply replacing the 'LSTM' by a 'BiLSTM'.

CHAPTER 3
QUESTION ANSWERING SYSTEM

3.1 System Flow

The user enter the input paragraph and question to the system. The system preprocessing answer by using preprocessing inputs, vectorizing inputs and modeling answer. And the system will output the answer to the user.

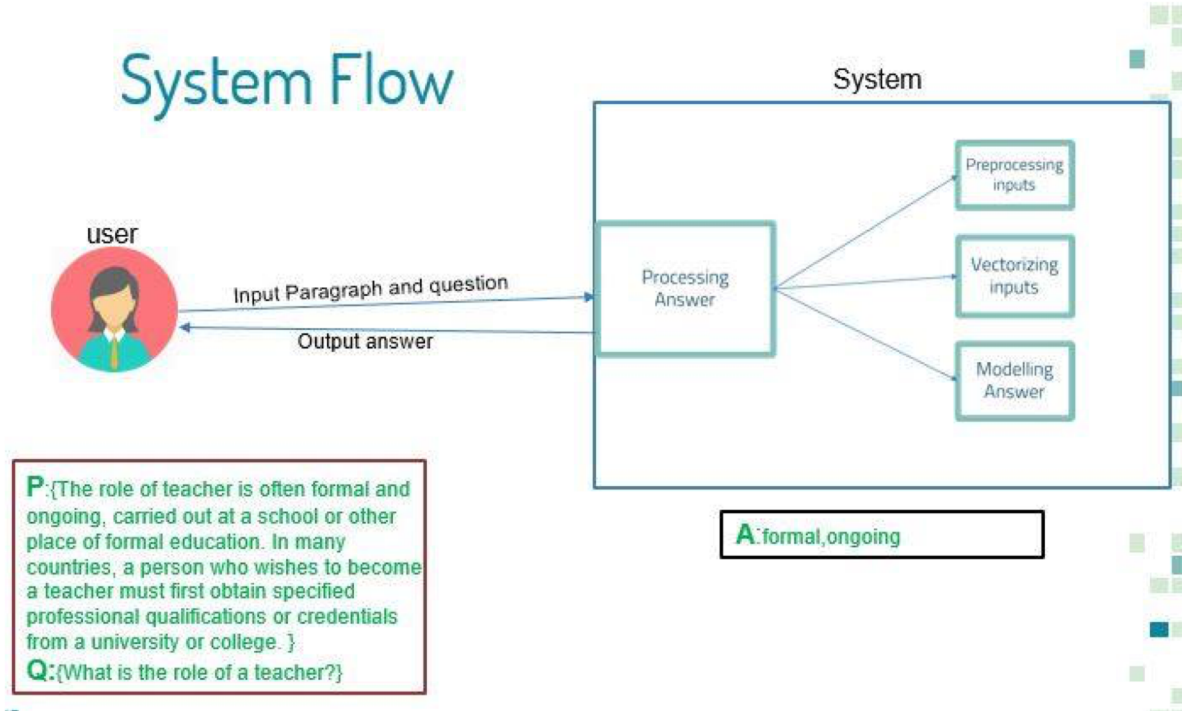

Figure 3.1 : Architecture of how system works

3.2 Question Answering Model Architecture

Our Question answering model is a hierarchical multi-stage process and consists of six layers :

1. Character Embedding Layer: maps each word to a vector space using character-level CNNs.

2. Word Embedding Layer: maps each word to a vector space using a pre-trained word embedding model.

3. Contextual Embedding Layer: utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context.

4. Attention Flow Layer: couples the query and context vectors and produces a set of query aware feature vectors for each word in the context.

5. Modeling Layer: employs a Recurrent Neural Network to scan the context.

6. Output Layer: provides an answer to the query.

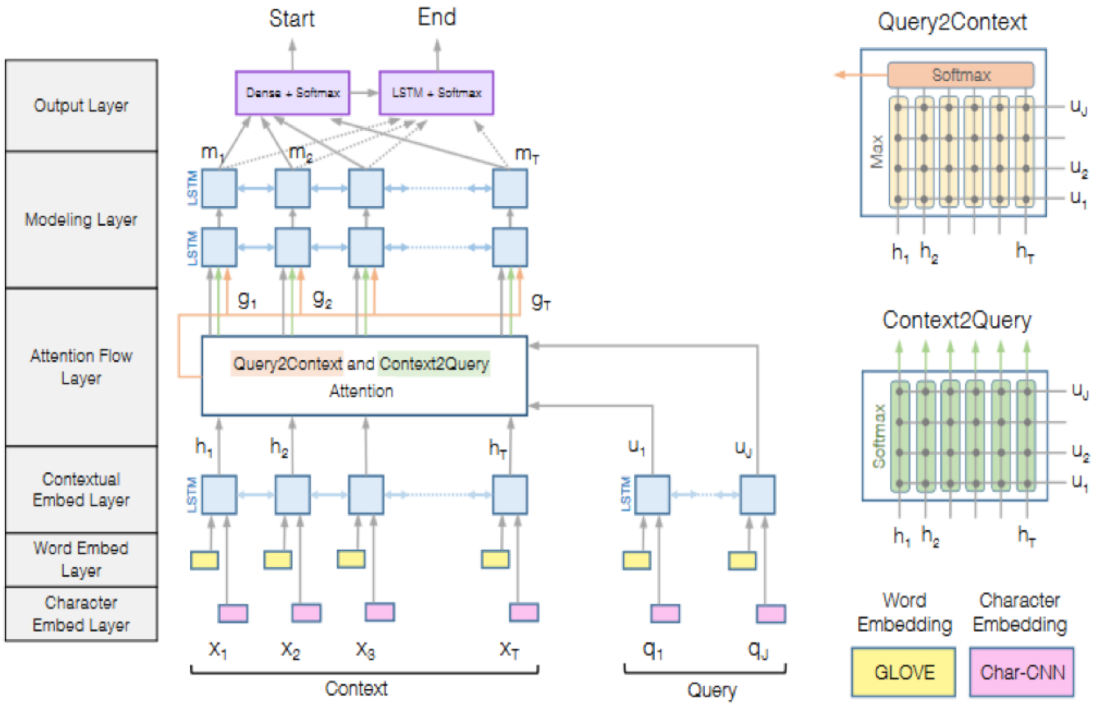

Fig 3.2 : Bi-Directional Attention Flow Model

1. Character embedding layer is responsible for mapping each word to a high-dimensional vector space. Let {x1, . . . xT } and {q1, . . . qJ} represent the words in the input context paragraph and query, respectively. Following Kim (2014), we obtain the character level embedding of each word using Convolutional Neural Networks (CNN). Characters are embedded into vectors, which can be considered as 1D inputs to the CNN, and whose size is the input channel size of the CNN. The outputs of the CNN are max-pooled over the entire width to obtain a fixed-size vector for each word.

In Character Level CNN, the steps include are -

- Encode the input characters
- Convolution
- Pooling
- Fully Connected layer
- Back Propagation

First, encoding the input character by using one hot vector,

DICTIONARY	INPUT							
	i	n	p	u	t	_	i	s
a	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
i	1	0	0	0	0	0	1	0
(	0	0	0	0	0	0	0	0
_	0	0	0	0	0	1	0	0

Figure 3.2 : Convolution

- Convolution means multiplication of weights in the filter and corresponding representation characters.
- The greater the number of filters, the greater the number of feature maps. Need filters to convolve.
- Filter Width is the dimensions of character representation
- Length of the filter and stride is considerable things in convolution.
- The output may depend on them.
- Must be tuned throughout the training.

2. Word Embedding Layer. Word embedding layer also maps each word to a high-dimensional vector space. We use pre-trained word vectors, GloVe (Pennington et al., 2014), to obtain the fixed word embedding of each word. The concatenation of the character and word embedding vectors is passed to a two-layer Highway Network (Srivastava et al., 2015). The outputs of the Highway Network are two sequences of d dimensional vectors, or more conveniently, two matrices: X 2 Rd→ıT for the context and Q 2 Rd→ıJ for the query.

Word vectors. By default, the bi-directional attention flow model uses 100-dimensional pre-trained GloVe embeddings to represent words, and these embeddings are held constant during training.

Two Layer Highway network,

- concat character and word embeddings
- pass onto a two-layer highway network.
- is used for selectively controlling information flow

Highway Networks:

$$y = \text{ReLU}(xW^{(x)} + b^{(x)}) \bullet \sigma(xW^{(T)} + b^{(T)}) + x(1 - \sigma(xW^{(T)} + b^{(T)}))$$

[Equation 3.1]

- The output of the highway network are two matrices

$$document : X \in (D, T)$$
$$query : Q \in (D, J)$$
$$\mathrm{T} = \text{num. words in document}$$
$$\mathrm{J} = \text{num. words in query}$$

[Equation 3.2]

3. Contextual Embedding Layer. We use a Long Short-Term Memory Network (LSTM) [Hochreiter & Schmidhuber, 1997] on top of the embeddings provided by the previous layers to model the temporal interactions between words. We place an LSTM in both directions, and concatenate the outputs of the two LSTMs. Hence we obtain H 2 R2d→T from the context word vectors X, and U 2 R2d→J from query word vectors Q. Note that each column vector of H and U is 2d-dimensional because of the concatenation of the outputs of the forward and backward LSTMs, each with d-dimensional output. It is worth noting that the first three layers of the model are computing features from the query and context at different levels of granularity, akin to the multi-stage feature computation of convolutional neural networks in the computer vision field.

4. Attention Flow Layer: Find the "resonance" between contexts and query, namely find the most relevant query words for each context words and the most relevant context words for each context words In our work, the attention vector will change in every step, base on the attention vector at last step and the current word. So the attention vector at each time step and the embeddings from previous layers flow during the reading process of the whole text. And this temporal related property of LSTM successfully prevents information losing. Two attentions, namely query-to-context attention and context-to-query attention, are computed from question encoding H and the query encoding U. The outputs of the layer are the query-aware vector representations of the context words, G, along with the contextual embeddings from the previous layer. We compute attentions in two directions: from context to query as well as from query to context. We create a matrix S to describe the similarities between the contextual embeddings of question(U) and the embeddings in context(H), $S \in R^{T \times J}$, where S_{tj} indicates the similarity between t-th context word and j-th query word and it I_s computed via $S_{tj}=\alpha(H:t,U:j \in R.)$

where, αis a function that computes the similarity between its two input vector with trainable weights, H:t is t-th column vector of H, and U:j is j-th column vector of U. There multiple ways of choosing αand w did experiment to find out the best function. We will come back to this in later part of this report. Now we use S to obtain the attentions and the attended vectors in both directions. In our work we use different kinds of inner product to describe the similarity of words in question and in context. For example, one way we used to calculate α(h, u)is α(h, u)=h⊤W_{bi}u, and the matrix W_{bi} can be optimized by training.

Context-to-query Attention:Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. In other words, it likes that we first read the context then find the match part in the question. $a_t \in R^J$ represents the attention weights on the query words by t-th context word, Sat j=1forallt.The attention weight is computed by a_t=softmax(S_t:)$\in R^J$., and each attended query vector is U˜:t=$S_j a_{tj}$ $U_{:j}$. Hence U˜is a $2^{d\times T}$ matrix containing the attended query vectors for the entire context.

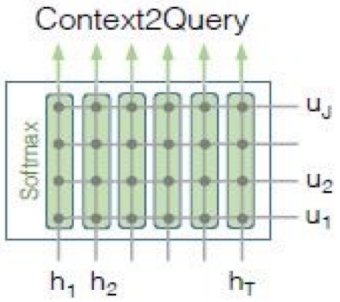

Figure 4.2 : Context-to-query Attention

Query-to-context Attention: Query-to-context (Q2C) attention signifies which context words have the closest similarity to one of the query words and are hence critical for answering the query. We get the attention weights on the context words by

b = softmax (maxcol(S))$\in R^T$, [Equation 3.3]

where the maximum function (maxcol) is performed across the column. Then the attended context vector is h˜ = $S_t b_t$H:t$\in R^{2d}$. This vector indicates the weighted sum of the most important words in the context with respect to the query. h˜is tiled T times across the column, thus giving H˜$\in R^{2d\times T}$After the two steps we get two sets of attentions U˜and H˜. Finally,

the contextual embeddings and the attention vectors are combined to yield G, where each column vector can104be considered as the query aware representation of each context word. We define G by

$$G{:}t = \beta(H{:}t, \tilde{U}{:}t, \tilde{H}{:}t) \in R^{dG} \qquad \text{[Equation 3.4]}$$

where G:t is the t-th column vector (corresponding to t-th context word), β is a trainable vector function that fuses its (three) input vectors, and ^{dG}is the output dimension of the β function. βcan be different, such as in the paper: $\beta(h, \tilde{u}, \tilde{h}) = [h;\tilde{u};h\circ\tilde{u};h\circ\tilde{h}] \in R^{8d\times T}$ (i.e.,^{dG}=8d).In our model, we use a modified version of $\beta(h, \tilde{u}, \tilde{h}) = \max(0, W_{mlp}[h;\tilde{u};h\circ\tilde{u};h\circ h] + b_{mlp} \in R^{8d\times T}$, which is a linear transformation of the original β, and it's like a ReLU. After this process, we combined the information in question and in context.

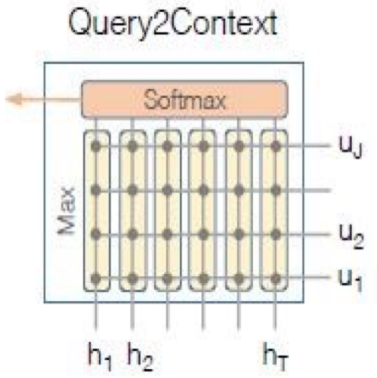

Figure 4.3 : Query-to-context Attention

5. Modeling Layer: We use again, LSTM, to encode query-aware representation for final output. The output of the modeling layer represents the interaction among the context words conditioned on the query. We use two layers of bi-directional LSTM, with the output size of d for each direction. So we get a matrix $M \in R^{2d\times T}$, which is passed onto the output layer to predict the answer. Each column vector of M contains contextual information about the word with respect to the entire context paragraph and the query, which comes from the information in G.

6. Output Layer: To get the start word and the end word in the answer, we try to transform the question to be that finding the probability of every word to be the start word. We obtain the probability distribution of the start index over the entire paragraph by

$$p_1 = \text{softmax}(w^T(p_1)[G;M]) \qquad \text{[Equation 3.5]}$$

where $w(p_1) \in R^{10d}$ is a trainable weight vector. For the end index of the answer phrase, we pass M to another bidirectional LSTM layer and obtain $M^2 \in R^{2d \times T}$. Then we use M^2 to obtain the probability distribution of the end index in a similar manner:

$$p_2 = \text{softmax}(w^T(p_2)[G;M^2]) \qquad \text{[Equation 3.6]}$$

7. Training: We define the training loss (to be m inim ized) as the sum of the negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples:

3.3 Training Details

Padding

As describe above, we use the same Bi –GRU to encode the GloVE embeddings for the context and question. Since the Bi-GRU expects inputs to be the same size, we included padding so that the number of question word embeddings matched the number of context word embeddings.

Hyper parameter Searching

To describe the best hyper parameter settings, we performed a greedy search on the hyper parameter space. Specifically, we tuned each hyper parameter with all of the hyper parameters held fixed and used the best hyper parameters found. To conserve time and resources, we only allowed runs to continue to about 15K iterations before cutting them off. We performed greedy searches on the learning rate, batch size, hidden size, regularization constant and dropout rate.

Choosing an Optimizer

We experimented with using Adam, AdaDelta, and SGD to optimize the learning models. Keskar and Socher showed that switching from Adam to SGD at later stages of learning can improve generalization error, though we did not observe this is our experiments. Seo et al. used AdaDelta to optimize their learning models with annealing learning schedule. We experimented with this idea for some time, but eventually backed off to using Adam with

a fixed learning rate after noticing that training took much longer with AdaDelta than with Adam.

Test. The answer span(k,l) where k≤l with the maximum value of $p^1_k\, p^2_l$is chosen, which can be computed in linear time with dynamic programming. We use the same metrics to evaluate our model. Exact match is a metric that measures the percentage of predictions that match one of the ground truth answers exactly. F1 score is a metric that loosely measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1. We take the maximum F1 over All the ground truth answers for a given question, and then average over all the questions.

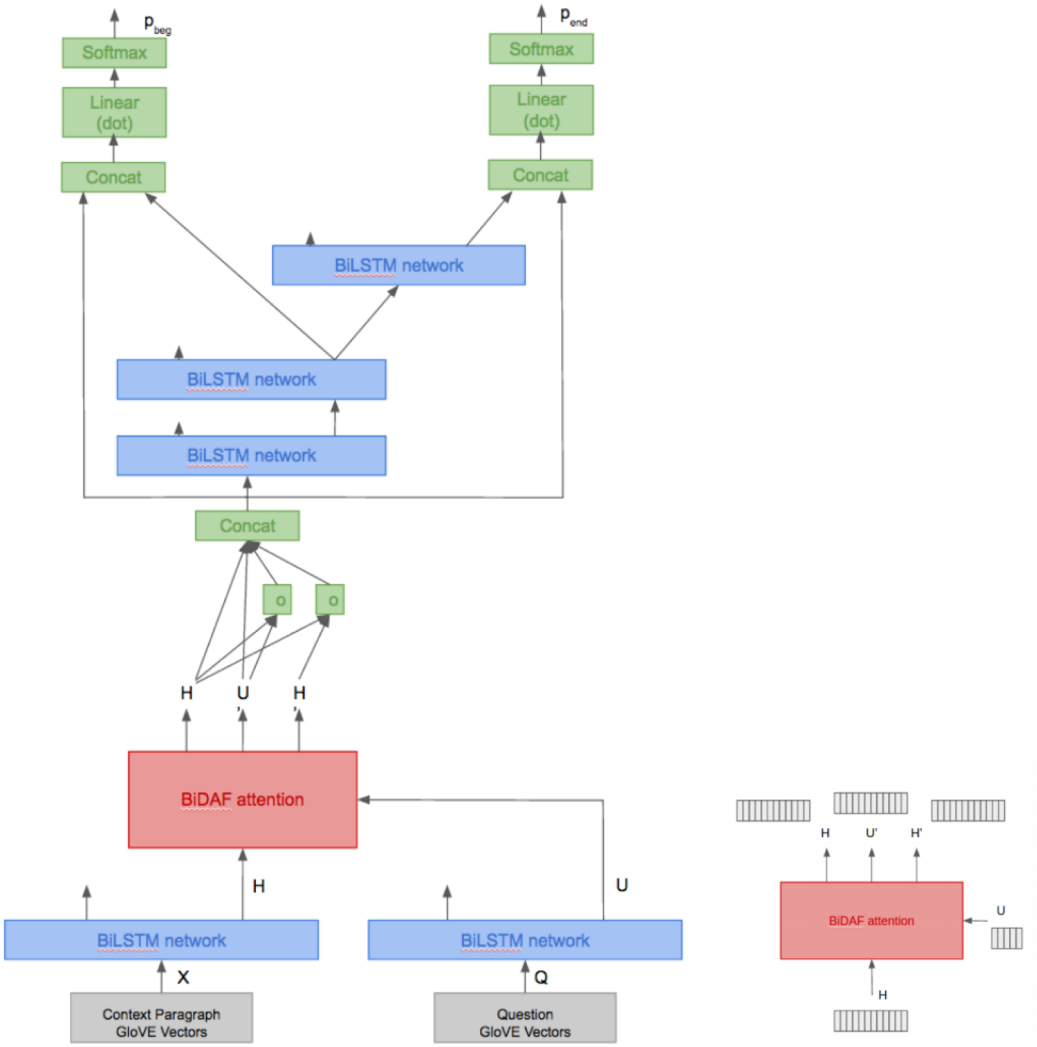

Figure 3.3 : The Bidirectional attention flow model

CHAPTER 4
DATASET

4.1 SQUAD DATASET

The Stanford Natural Language Processing Group has released yet another large-scale question answering dataset, SQuAD . However, unlike the existing cloze-form datasets, the answers to questions are spans within the document. This dataset is not only large enough to allow the development of expressive models, but natural in its task formulation. The SQuAD dataset is comprised of articles from English Wikipedia and annotated solely by workers on Amazon Mechanical Turk. In machine reading-style question answering datasets like SQuAD, the system has to locate the answer to a question in the given ground truth paragraph.

A SQuAD like question answering example:

In 2004, Obama received national attention during his campaign to represent Illinois in the United States Senate with his victory in the March Democratic Party primary, his keynote address at the Democratic National Convention in July, and his election to the Senate in November. He began his presidential campaign in 2007 and, after a close primary campaign against Hillary Clinton in 2008, he won sufficient delegates in the Democratic Party primaries to receive the presidential nomination. He then defeated Republican nominee John McCain in the general election, and was inaugurated as president on January 20, 2009. Nine months after his inauguration, Obama was controversially named the 2009 Nobel Peace Prize laureate.

When was Obama's keynote address?

July

Who did Obama campaign against in 2008?

Hillary Clinton

Where was the keynote address?

Democratic National Convention

In contrast to prior datasets, SQuAD does not provide a list of answer choices for each question. Rather, systems must select the answer from all possible spans in the passage, thus needing to cope with a fairly large number of candidates. While questions with span-based answers are more constrained than the more interpretative questions found in more advanced standardized tests, we still find a rich diversity of questions and answer types in SQuAD. We

develop automatic techniques based on distances in dependency trees to quantify this diversity and stratify the questions by difficulty. The span constraint also comes with the important benefit that span-based answers are easier to evaluate than freeform answers.

CHAPTER 5
SYSTEM IMPLEMENTATION

5.1 PROGRAM INSTALLATION REQUIREMENT

We also implement the BIDAF and Match-LSTM models based on Tensorflow 1.0. You can refer to the official guide for the installation of Tensorflow. The complete options for running our Tensorflow program can be accessed by using python run.py -h. General

- Python (verified on 3.5.2. Issues have been reported with Python 2!)
- unzip, wget (for running download.sh only)

Python Packages

- tensorflow (deep learning library, only works on r0.11)
- nltk (NLP tools, verified on 3.2.1)
- tqdm (progress bar, verified on 4.7.4)
- jinja2 (for visaulization; if you only train and test, not needed)

Pre-Processing

First, prepare data. Donwload SQuAD data and GloVe and nltk corpus (~850 MB, this will download files to $HOME/data):

```
chmod +x download.sh; ./download.sh
```

Second, Preprocess Stanford QA dataset (along with GloVe vectors) and save them in $PWD/data/squad (~5 minutes):

```
python -m squad.prepro
```

Training

The model has ~2.5M parameters. The model was trained with NVidia Titan X (Pascal Architecture, 2016). The model requires at least 12GB of GPU RAM. If your GPU RAM is smaller than 12GB, you can either decrease batch size (performance might degrade), or you can use multi GPU (see below). The training converges at ~18k steps, and it took ~4s per step (i.e. ~20 hours). Before training, it is recommended to first try the following code to verify everything is okay and memory is sufficient.

5.2 IMPLEMENTATION RESULTS AND CHALLENGES

To avoid overfitting in our model, we experimented with different dropout rates and gradient clipping was enabled and the model was trained using a non-zero dropout rate. In addition, all inputs (questions or context paragraphs) with lengths lower than their corresponding maximum were zero-padded and those with longer lengths were clipped at the maximum allowed length. In addition, in cases where the true answer resided in the clipped section of a paragraph, we would feed the model -1 to imply that an answer does not exist.

We faced numerous challenges on the path to finishing this project, which was a humbling experience. Beside the initial challenges of understanding the organization of the starter code, upon implementing the first baseline model, we had a major setback. The initial baseline model repeatedly crashed due to a memory error on Azure. After some investigation, it turned out that our model simply had a few variables which took up more than 2GB of memory space. The solution was to decrease the state size of the LSTM cells used as well as to use a smaller mini-batch size for training. Additionally, by looking at the histograms of the quantity of context paragraphs as well questions vs. their respective length, we realized that almost all context paragraphs in the training set have a maximum length of 300 tokens, and almost all questions in the training set have a maximum length of 60 tokens. This finding enabled us to save us a lot of space limiting the size of the LSTM layers processing the context paragraphs and questions. However, even after fixing that error, the baseline continued to perform poorly. Our effort to debug the problem was not effective until recently when we paid more attention to the variable scoping used in the project. The problem was that our variable scoping was not correctly used, in other words we would set up the Tensor flow nodes under a set of scope variables, but would not instantiate the model using the same variable scopes resulting in disjoint model training from evaluation. As a result, we took a careful look at the variable scopes and made sure we never left the scope while instantiating the model. Other minor setbacks included inputting the remove probability instead of keep probability in Tensor flow's dropout function, as well as problems with the dimensions of the model throughout implementation.

Figure 5.2.1 : Calculating F1/EM score in command line

```
                ^
IndentationError: unindent does not match any outer indentation level
Sus-Lappy:bi-att-flow sumyatnoe$ python -m squad.prepro
Traceback (most recent call last):
  File "/Users/sumyatnoe/anaconda3/lib/python3.6/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/Users/sumyatnoe/anaconda3/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 232, in <module>
    main()
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 16, in main
    prepro(args)
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 60, in prepro
    prepro_each(args, 'train', out_name='train')
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 125, in prepro_each
    source_data = json.load(open(source_path, 'r'))
FileNotFoundError: [Errno 2] No such file or directory: '/Users/sumyatnoe/data/squad/train-v1.1.json'
Sus-Lappy:bi-att-flow sumyatnoe$ python -m squad.prepro
100%|████████████████████████████████████| 442/442 [01:22<00:00,  5.33it/s]
Traceback (most recent call last):
  File "/Users/sumyatnoe/anaconda3/lib/python3.6/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "/Users/sumyatnoe/anaconda3/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 232, in <module>
    main()
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 16, in main
    prepro(args)
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 60, in prepro
    prepro_each(args, 'train', out_name='train')
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 216, in prepro_each
    word2vec_dict = get_word2vec(args, word_counter)
  File "/Users/sumyatnoe/Documents/GitHub/bi-att-flow/squad/prepro.py", line 89, in get_word2vec
    with open(glove_path, 'r', encoding='utf-8') as fh:
FileNotFoundError: [Errno 2] No such file or directory: '/Users/sumyatnoe/data/glove/glove.6B.100d.txt'
Sus-Lappy:bi-att-flow sumyatnoe$ python -m squad.prepro
100%|████████████████████████████████████| 442/442 [01:22<00:00,  5.37it/s]
100%|████████████████████████████████| 400000/400000 [00:11<00:00, 34628.75it/s]
69696/103324 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
100%|████████████████████████████████| 400000/400000 [00:11<00:00, 35664.28it/s]
70575/90219 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
saving ...
100%|████████████████████████████████████████| 48/48 [00:17<00:00,  2.68it/s]
100%|████████████████████████████████| 400000/400000 [00:11<00:00, 35158.82it/s]
22718/27989 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
100%|████████████████████████████████| 400000/400000 [00:11<00:00, 36052.31it/s]
22876/24887 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
saving ...
100%|████████████████████████████████████████| 48/48 [00:18<00:00,  2.61it/s]
100%|████████████████████████████████| 400000/400000 [00:10<00:00, 37285.69it/s]
22718/27989 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
100%|████████████████████████████████| 400000/400000 [00:10<00:00, 37189.54it/s]
22876/24887 of word vocab have corresponding vectors in /Users/sumyatnoe/data/glove/glove.6B.100d.txt
saving ...
Sus-Lappy:bi-att-flow sumyatnoe$
Sus-Lappy:bi-att-flow sumyatnoe$
```

Figure 5.2.2 : Pre-processing the data set

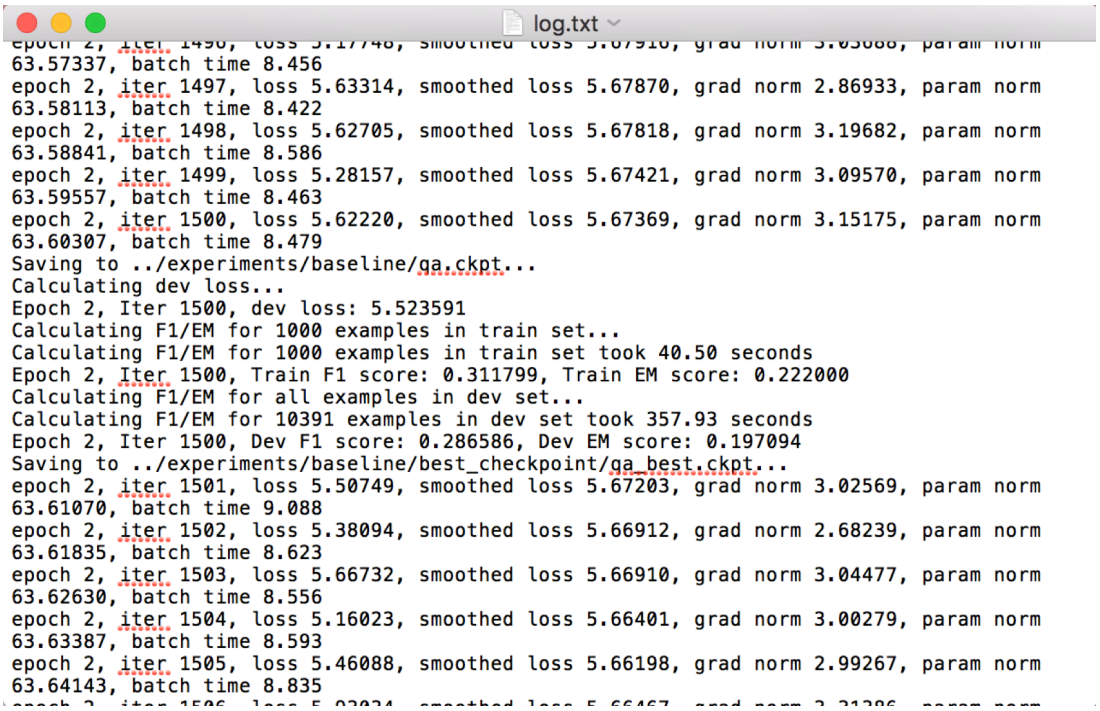

Figure 5.2.3 : Iterating Steps wile pre-processing the data set

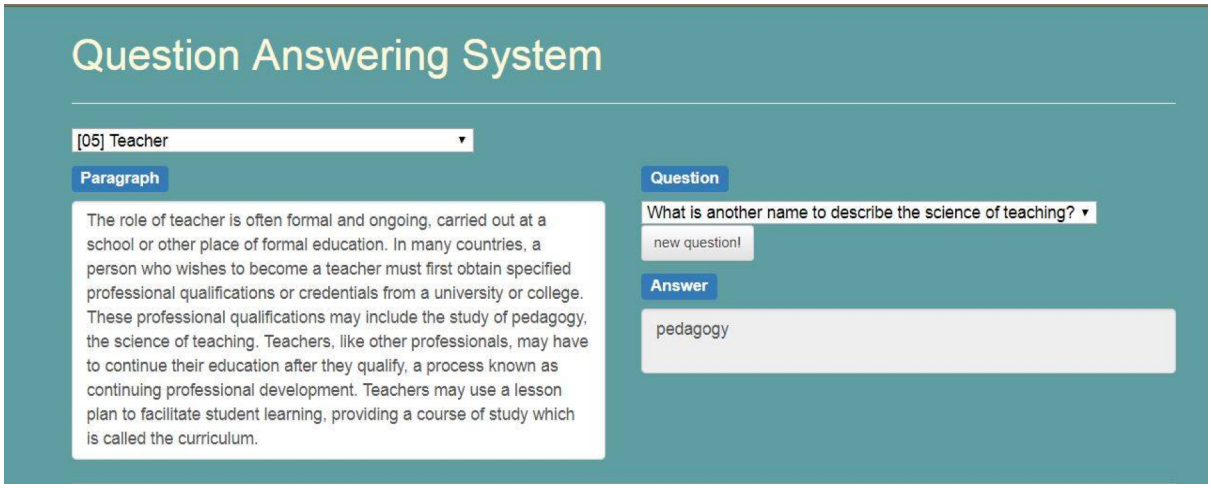

Figure 5.2.4 : Question Answering Program

CHAPTER 6
EVALUATION RESULT

6.1 Quantitative Analysis

Our model's performance to other state-of-the-art models. On the development set, out model achieved an F1 score of 74.952 and an EM score of 65.563, using a learning rate of 0.001 and dropout rate of 0.15. We optimized our model using Adam for 15K iterations.

For plots of the F1 and exact match scores for various hyper parameter settings.

6.2 Experiment and Analysis

Own Model (Dev Set)	F1 (%)	EM (%)
My BiDAF (with linked context embed layer)	74.952	65.563
Reimplementation of original BiDAF	74.04	63.916
BiDAF without char embedding	73.911	63.888
BiDAF with fixed Glove6B100d	72.904	62.242
BiDAF without modeling layer	69.863	58.6

Figure 6.1: Experiment Result on the dev set

6.3 Qualitative Analysis

Generating examples for random question-answer pairs in the development set, we noticed some shortcomings of our model.

6.3.1 Misplaced Attention

On some examples, we saw that out model paid attention to the wrong parts of the context. Consider the following example: Here, we see that the model likely placed a lot of attention on the words "district" and "cbd". The model likely fails for question-answer pairs like this one because no part of the ground truth appears in the question, so it has a difficult time to relevant parts of the context.

6.3.2 Model lacks background knowledge

Unsurprisingly, the model does not perform well on question-answer pairs for which the answer cannot be determined from the context paragraph alone. Consider the following example.

Context: Not only are all the major British architects of the last four hundred years represented, but many European (especially Italian) American architects' drawings are held in the collection. The Riba's holdings of over 330 drawings by Andrea Palladio are the largest in the world, other Europeans well represented are Jacques Gentilhatre and Antonio Visentini. British architects whose drawings, and in some cases models of their buildings, in the collection, include: Inigo JOnes, Sir Christopher Wren, Sir John Vanbrugh, Nicholas Hawksmoor, William Kent, James Gibbs, Robert Adam, Sir William Chambers, James Wynatt, Henry Holland, John Nash, Sir John Soane, Sir Charles Barry, Charles Robert Cockerell, Augustus Welby Northmore Pugin, Sir George Gilbert Scott, John Loughborough Pearson, George Edmund Street, Richard Norman SHaw, Alfred Warehouse, Sir Edwin Lutyens, Charles Rennie Mackintosh, Charles Holden, Frank Hoar, Lord Richard Rogers, Lord Norman Foster, sir Nicholas Foster, sir Nicho las Grimshaw, Zaha Hadid and Alick Horsnell.

Question: Which architect, famous for designing London's St. Paul Cathedral, is represented in the Riba collection?

Prediction: Andrea Palladio

Ground Truth: Sir Christopher Wren

Nowhere in the context paragraph does it describe Sir Christopher Wen designing London's St. Paul Cathedral, so the only information model is able to gleam from the question is the fact that the architect is represented in the Riba collection. The model pays attention to the word "Riba" and stumbles upon the name "Andrea Palaldio", which appears closet to "Riba".

This is a difficult for any model to solve, and even humans are likely to get this one wrong if they don't know the history of London's St. Paul Cathedral. Ways to address the issued would be to give the model access to a large textual database (perhaps an encyclopaedia) from which it can gather additional background information.

6.3.3 Attention is narrow

Sometimes, the model pays attention to only a part of the ground truth, such as in the following example:

Context: To remedy the causes of the fire, changes were made in the Block II spacecraft and operational procedures, the most important of which were use of a nitrogen/oxygen mixture instead of pure oxygen before and during launch, and removal of flammable cabin and space suit materials. The Block II design already called for replacement of the Block I plug-type hatch cover with a quick-release, outward opening door. NASA discontinued the manned Block I program, using the Block I spacecraft only for unmanned Saturn V flights. Crew members would also exclusively wear modified, fire- resistant Block II space suits, and would be designated by the Block II titles, regardless of whether a LM was present on the flight or not.

Question: What type of materials inside the cabin were removed to help prevent more fire hazards in the future?

Prediction: Space Suit

Ground Truth: flammable cabin and space suit materials

In this example, the model likely pays most of its attention to “space suit” due to its close proximity to the word “materials” which appears in the context.

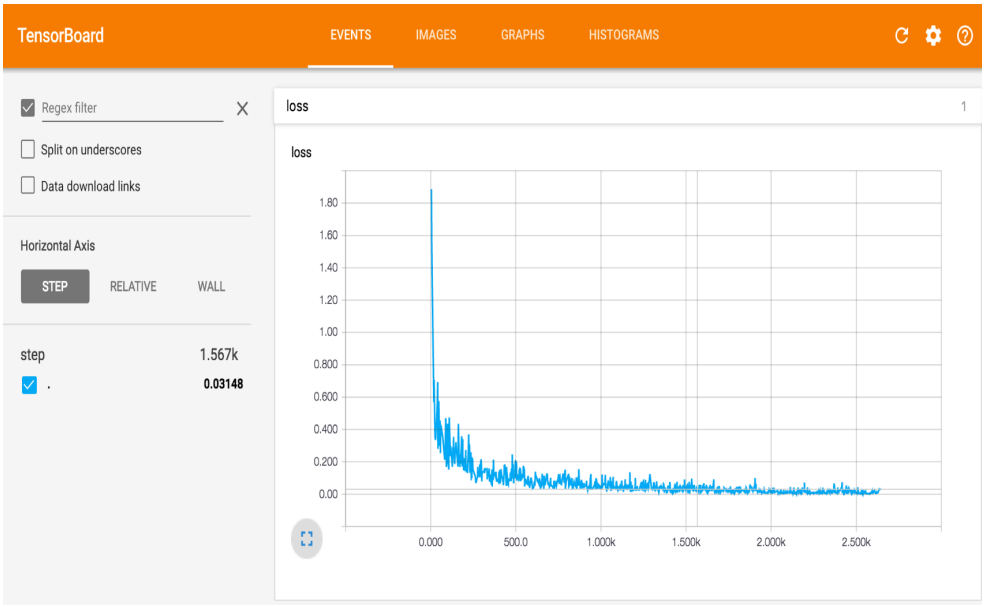

Figure 6.3.1 : TensorBoard show Loss

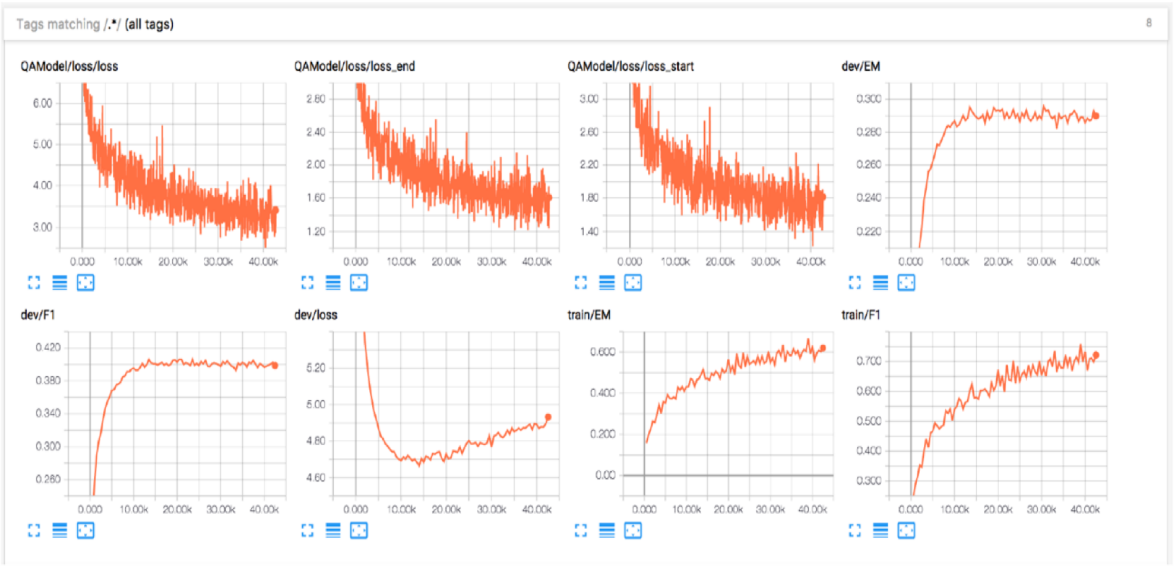

Figure 6.3.2 : TensorBoard show dev F1 and EM score , dev, train score

In particular, over 40k iterations we find that:

- the train loss, train F1 and train EM scores continue to improve throughout
- the dev loss begins to rise around 10-15k iterations (overfitting)
- the dev F1 and EM scores reach around 39 and 28 respectively, then plateau.

If we assume that the model begins overfitting and therefore reaches its peak performance at 15k iterations, then on an Azure NV6 machine, it will take you approximately 15k iterations = 5 hours to achieve peak performance. However, we stopped training earlier (i.e. around 15k iterations rather than 40k). This will give you something to compare your improved models against. In particular, TensorBoard will plot our new experiments and enable us to see how our improved models progress over time.

CHAPTER 7
CONCLUSION AND FUTURE WORK

In the future, we want to incorporate syntactic structure of the question and the paragraph into our model. One possibility is using dependency parsing to generate range of answers that makes sense grammatically. We could also incorporate character embedding, suggested by the original BiDAF paper. Finally, since the model performs better on quantitative questions than qualitative questions and shorter answers rather than longer answers, we could also try changing the output of the model. Rather than generating the probability of start and end indices, we could generate probabilities for each individual word in regards to whether they are in the answer. Using these probabilities, we could extract the longest positively classified contiguous sequence with some stop word exceptions to generate our answers. We believe this would allow the machine comprehension model to better answer more complex answers. With an F1 score of 74.952 and an EM score of 65.563, we were able to achieve reasonable performance using our bidirectional attention flow model. The performance is only 20% to 25% lower than current state of the art models and within a few percentage points of early models for SQuAD, in particular, the Match-LSTM model. Through our research, we have no only created a reasonably good model and explored potential extensions to the existing bi-directional attention flow model, we have also outlined weaknesses with possibility of improvements that we believe will help improve the model to be better comparable in performance with the current state of the art models.

REFERENCES

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

[2] CaimingXiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[3] Zhiguo Wang, Wael Hamza, and Radu Florian.Bilateral multiperspective matching for natural language sentences. arXiv preprint arXiv:1702.03814, 2017.

[4] Shuohang Wangand Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[5] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. arXivpreprint arXiv:1606.02858, 2016.

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

[7] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. arXiv preprint arXiv: 1611.01436, 2016.

[8] Yelong Shen, Po Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. arXiv preprint arXiv:1609.05284, 2016.

[9] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. arXiv preprint arXiv: 1610.09996, 2016.

[10] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhut-dinov. Words or characters? fine-grained gating for reading comprehension. arXiv preprint arXiv:1611.01724, 2016.

[11] MinJoon Seo, Aniruddha Kembhavi, Ali Farhadi, and, Hananneh Hajishirzi.Bi-Directional Attention Flow For Machine Comprehension. arXivpreprint arXiv:1611.01603v5, 2017.