CS7.401 Introduction to NLP

Question Answering using SQuAD Dataset

Md Khaja Ghouse (2023201054)
Md Jawed Equbal (2023201051)
Abhinav Bahuguna (2023201040)

May 8, 2024

Abstract

We develop a deep learning framework for question answering on the Stanford Question Answering Dataset (SQuAD), combining ideas from advanced models to outperform the baseline. Our model using Bi-Directional Attention Flow (BiDAF) resulted in an F1 score of 68.1 and an EM score of 55.4 on the SQuAD dev set. This report presents the approach behind our question-answering model. We analyse the inadequacies and limits of our method.

1 Introduction

Question answering is an important task based on which the intelligence of NLP systems and AI, in general, can be judged. A QA system is given a short paragraph or context about some topic and is asked some questions based on the passage. The answers to these questions are spans of the context, that is they are directly available in the passage. To train such models, we use the SQuAD dataset. There are numerous ways in which QA model can classified. Here are some of them:

- **Open vs Closed Domain**: An open-domain model can access a knowledge repository, which it will use when responding to an incoming Query. The well-known IBM Watson is one example. A closed-domain model, on the other hand, does not rely on prior information; rather, it requires a Context to respond to a Query.
- **Abstractive vs extractive**: In extractive, the answer returned by the model can always be found verbatim within the Context. An abstractive model, on the other hand, goes a step further by paraphrasing this substring into a more human-readable format before returning it as the answer to the Query.
- **Ability to answer non-factoid queries**: Factoid Queries have answers that are short factual statements. Most Queries that begin with "who",

"where" and "when" are factoid because they expect concise facts as answers. Non-factoid Queries are all questions that are not factoids.

BiDAF is a closed-domain, extractive QA model. These qualities indicate that BiDAF needs a Context to respond to a Query. BiDAF always returns a substring of the given Context.

1.1 Problem Statement

Given a three-tuple $(Q, P, (a_s, a_e))$ consisting of a question Q, context paragraph C, and start and end indices as and ae, the goal of the problem is to predict the *answer span* $a_s, a_{s+1}, ..., a_{e-1}, a_e$ where each ai is an index of the context paragraph corresponding to the answer.

2 Dataset and Preprocessing

Before we get into the specifics of our neural network architecture, we'll go over the preprocessing steps we did to prepare our inputs for encoding. We analyzed the raw SQuAD dataset and assigned indices to each question and paragraph based on their word positions in an external dictionary. Next, we used pre-trained GloVe vectors to generate word vectors for each question and context. Initially, we used Glove6B100d vectors and while hyperparameter tuning we switched to Glove6B300d vectors as it produced better results in comparison. We have also built a char2index dictionary which will be used later to train char-level CNN in the BiDAF model.

Due to computation constraints, we have reduced the dataset and filtered out data points where context, query and answer tokens are above a particular threshold. To be exact, we have considered max context tokens to be 400, max query tokens to be 50 and max answer tokens to be 30. We have also removed some data points based on labels, which are start and end answer tokens, which are incorrect. The reason for the existence of such data points is that there are some spans that the tokenizer fails to capture or simply the data point is not cleaned.

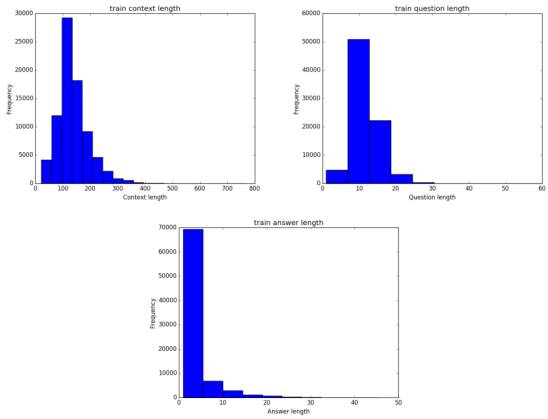

Figure 1: frequency vs context/query/answer lengths.

3 Implementation

3.1 Baseline Model

For baseline implementation, we used the model given in *Figure 2*

1. **Word Embedding Layer**: We used pre-trained word vectors, GloVe, to obtain the fixed word embedding of each word for both context and query
2. **Contextual Embedding Layer**: We used LSTM network on top of embeddings provided by previous layers to model contextual interactions between surrounding words. We used bi-directional LSTM.
3. **Attention Layer**: This layer is responsible for finding relevant information from the context and the query. Context-to-Query Attention is used in this layer.
4. **Modeling Layer**: The input to the modeling layer is the query-aware representation of context words. We used two layers of bi-directional LSTM in this layer
5. **Output Layer**: Run a softmax over the output of modeling layer concatenated with the output of the Attention layer to produce a probability distribution over the context words, each for the start and end word.

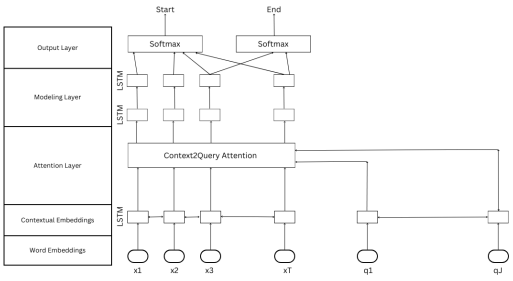

Figure 2: Baseline Model

3.2 BiDAF Implementation

Our BiDAF implementation consists of 6 layers as shown in *Figure 3*:

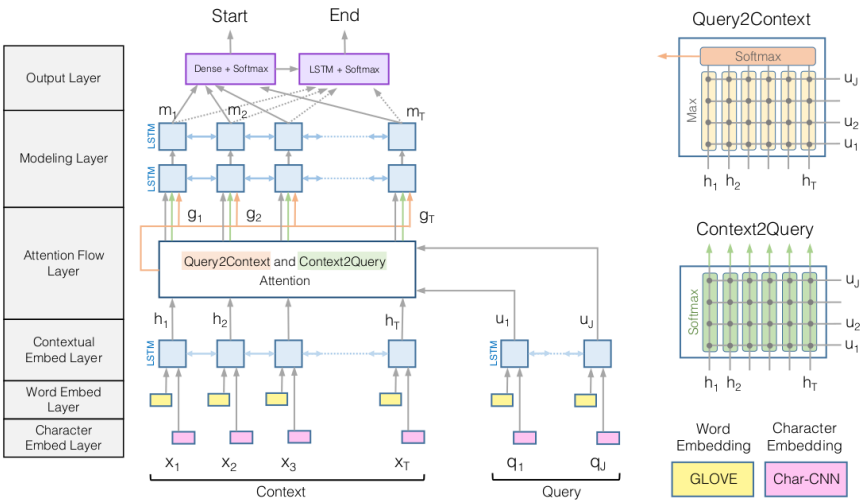

Figure 3: BiDAF Architecture

3.2.1 Character Embedding Layer

GloVe deals with these OOV words by simply assigning them some random vector values. We need another embedding mechanism that can handle OOV words. Character level embedding uses a one-dimensional convolutional neural network (1D-CNN) to find numeric representations of words by looking at their character-level compositions. We first pass each word through an embedding layer to get a fixed-size vector. Let the embedding dimension be d. Let $\mathbf{C}$ represent a matrix representation of a word of length l. Therefore, $\mathbf{C}$ is a matrix with dimensions $d \times l$.

Let $\mathbf{H}$ be a representation of a convolutional filter with dimensions $d \times w$, where d is the embedding dimension and w is the width or the window size of the filter. The weights of this filter are randomly initialized and learnt parallelly via backpropagation. We convolve this filter $\mathbf{H}$ over our word representation $\mathbf{C}$. The convolution operation is simply the inner product of the filter $\mathbf{H}$ and matrix $\mathbf{C}$.

The result of the above operation is a feature vector. A single filter is usually associated with a unique feature that it captures from the image/matrix. To get the most representative value related to the feature, we perform max pooling over the dimension of this vector. This same process is repeated with N number of filters. Each of these filters captures a different property of a word. N is also the size of the desired character embedding. We have trained the model with $N = 100$

3.2.2 Word Embedding Layer

We have used pre-trained GloVe embeddings to get the vector representation of words in the Query and the Context. The concatenation of the character and word embedding vectors is passed to a two-layer Highway Network. A highway network is very similar to a feed-forward neural network. In a highway network,

only a fraction of the input will be subjected similar to a feed-forward layer; the remaining fraction is permitted to pass through the network untransformed. The ratio of these fractions is managed by **t**, the *transform gate* and by **(1-t)**, *the carry gate*. The value of **t** is calculated using a sigmoid function and is always between 0 and 1.
A plain feed-forward layer is associated with a linear transform **H** parameterized by (W_H, b_H), such that for input x and output y is

$$y = g(W_H.x + b_H)$$

where g is non-linear function For highway networks, two additional linear transforms are defined viz $T(W_T, b_T)$ and $C(W_C, b_C)$. Then,

$$y = T(x).H(x) + x.C(x)$$

$$y = T(x).g(W_H.x + b_H) + x.(1 - T(x))$$

$$where\, T(x) = \sigma(W_T.x + b_T)\ \&\ g\ is\ relu\ activation$$

The outputs of the highway network are again two matrices, one for the Context $\mathbf{X} \in \mathbb{R}^{d \times T}$ and one for the Query $\mathbf{Q} \in \mathbb{R}^{d \times J}$, where d is the hidden size of LSTM, T is context length and J is query length. They represent the adjusted vector representations of words in the Query and the Context from word and character embedding steps.

3.2.3 Contextual Embedding Layer

The above word representations don't take into account the word's contextual meaning — the meaning derived from the word's surroundings. This is where the contextual embedding layer comes in. The contextual embedding layer consists of Long-Short-Term-Memory (LSTM) sequences. We employed a bidirectional-LSTM (bi-LSTM), which is composed of both forward- as well as backward-LSTM sequences. The combined output embeddings from the forward- and backward-LSTM simultaneously encode information from both past (backwards) and future (forward) states. Hence we obtain $\mathbf{H} \in \mathbb{R}^{2d \times T}$ for context word vectors $\mathbf{X}$ and $\mathbf{U} \in \mathbb{R}^{2d \times J}$ for query word vectors $\mathbf{Q}$

3.2.4 Attention Flow Layer

This layer is responsible for finding relevant information from the context and the query. The inputs to the layer are contextual vector representations of the context **H** and the query **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.

In this layer, we compute attention in two directions: from context to query and from query to context. Attention vectors for these calculations are derived from a common matrix which is called the similarity matrix and is denoted by $\mathbf{S} \in \mathbb{R}^{T \times J}$, between between the contextual embeddings of the context **(H)** and the query **(U)**. The similarity matrix is computed by,

$$\mathbf{S}_{tj} = \alpha(\mathbf{H}_{:t}, \mathbf{U}_{:j})\ where\ \mathbf{S}_{tj} \in \mathbb{R}$$

$\mathbf{S}_{tj}$ is a single float value that determines the similarity between the t-th context word and j-th query word. α is a trainable function that encodes the similarity between two input vectors $\mathbf{H}_{:t}$ and $\mathbf{U}_{:j}$. $\mathbf{H}_{:t}$ is is the contextual representation of the t-th context word and $\mathbf{U}_{:j}$ is is the contextual representation of the j-th query word. the trainable function is defined as $\alpha(\mathbf{h}, \mathbf{u}) = \mathbf{w}^T_{(\mathbf{S})}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$ where ; denotes concatenation and $\circ$ denotes an element wise product.

Context-to-query Attention. Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. Let $\mathbf{a}_t$ represent the vector that encodes the attention paid to each query word by the t-th context word.

$$\sum \mathbf{a}_{tj} = 1 \, \forall \, t$$

$$\mathbf{a}_t = softmax(\mathbf{S}_{:t}) \; where \; \mathbf{a}_t \in \mathbb{R}$$

Subsequently, each attended query vector is calculated by,

$$\bar{\mathbf{U}}_{:t} = \sum \mathbf{a}_{tj} \mathbf{U}_{:j}$$

This vector indicates the most important word in the query with respect to context. $\bar{\mathbf{U}}$ will be $2d \times T$ matrix.

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. The attention vector here is calculated as,

$$\mathbf{b} = softmax(max_{col}(\mathbf{S})) \in \mathbb{R}^T$$

where max_{col} performs the maximum function across the column. The attended context vector is then calculated as

$$\bar{\mathbf{h}} = \sum \mathbf{b}_t \mathbf{H}_{:t}$$

$\bar{\mathbf{H}}$ will be $2d \times T$ matrix.

Finally, the contextual embeddings and the attention vectors are combined together to yield $\mathbf{G}$, where each column vector can be considered as the query-aware representation of each context word. $\mathbf{G}$ is defined as

$$\mathbf{G}_{:t} = \beta(\mathbf{H}_{:t}; \bar{\mathbf{U}}_{:t}; \bar{\mathbf{H}}_{:t})$$

where β is a trainable function

$$\beta(\mathbf{h}, \bar{\mathbf{u}}, \bar{\mathbf{h}}) = [\mathbf{h}; \bar{\mathbf{u}}; \mathbf{h} \circ \bar{\mathbf{u}}; \mathbf{h} \circ \bar{\mathbf{h}}]$$

This concatenation gives a $8d$ vector. Hence the trainable weight here should have an input dimension of $8d$. However, here we don't involve a trainable weight because according to the authors, While the β function can be an arbitrary trainable neural network, such as a multi-layer perceptron, a simple concatenation as above still shows good performance in our experiments.

3.2.5 Modeling Layer

G is then passed on to this layer. This layer is responsible for capturing temporal feature interactions among the context words. This is done using a bidirectional LSTM. The difference between this layer and the contextual layer, both of which involve an LSTM layer is that here we have a query-aware representation of the context while in the contextual layer, encoding of the context and query was independent. Using an LSTM layer with a hidden size of d we get a $2d \times T$ output called **M**.

3.2.6 Output Layer

The start index $\mathbf{p}^1$ of the answer span is calculated by,

$$\mathbf{p}^1 = softmax(\mathbf{w}^T_{(\mathbf{p}^1)}[\mathbf{G}; \mathbf{M}])$$

where $\mathbf{w}_{(\mathbf{p}^1)}$ is a $10d$ trainable weight vector.

To predict the end of the answer span, **M** is once again passed to a bidirectional LSTM to get $\mathbf{M}^2$ which again is a $2d \times T$ matrix. The end is then computed as,

$$\mathbf{p}^2 = softmax(\mathbf{w}^T_{(\mathbf{p}^2)}[\mathbf{G}; \mathbf{M}^2])$$

where $\mathbf{w}_{(\mathbf{p}^2)}$ is a $10d$ trainable weight vector. In code, we don't explicitly calculate softmax since this is taken care of while calculating the losses during training.

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

$$L(\theta) = -\frac{1}{N}\sum_{i}^{N} log(\mathbf{p}^1_{y_i^1}) + log(\mathbf{p}^2_{y_i^2})$$

4 Training

We have used 100 1D filters for CNN char embedding, each with a width of 5. The hidden state size (d) of the model is 100. The model has about 2.6 million parameters. We have used the AdaDelta(Zeiler,2012) optimizer, with a minibatch size of 16 and an initial learning rate of 0.5, for 10 epochs. A dropout rate of 0.2 is used for the CNN, all LSTM layers, and the linear transformation before the softmax for the answers.

5 Results and Analysis

5.1 Result

We achieved F1 score of 68.1 and EM score of 55.4 on the dev set. Loss graphs and EM/F1 graphs have been showed

5.2 Error Analysis

We analyze the error cases in the dev set.

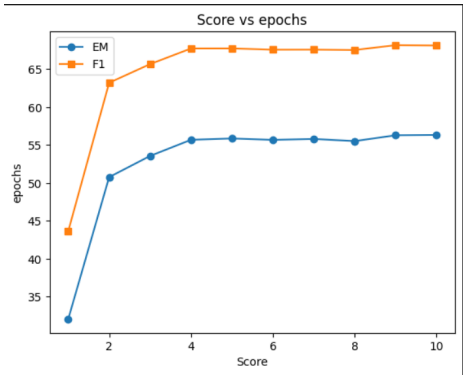

Figure 4: EM and F1 score vs epochs

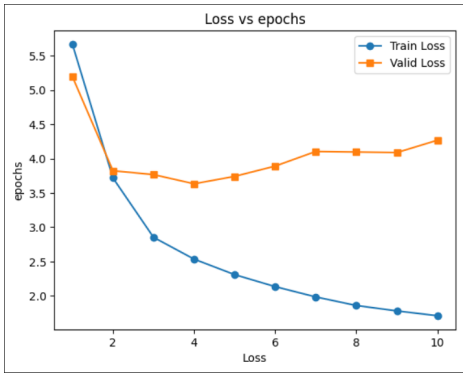

Figure 5: Train and Valid Loss vs epochs

5.2.1 Imprecise answer boundaries.

In most of the incorrect cases, our model gives an inaccurate position around the boundaries. 2-3 words around the answer span may be mistakenly omitted or included. It may be very hard to get rid of these mistakes completely.
Context: "The Free Movement of Workers Regulation articles 1 to 7 set out the main provisions on equal treatment of workers."
Question: "Which articles of the Free Movement of Workers Regulation set out the primary provisions on equal treatment of workers?"
Prediction: "1 to 7", **Answer:** "articles 1 to 7"

5.2.2 Wrong position of attention

Context: "The Panthers used the San Jose State practice facility and stayed at the San Jose Marriott. The Broncos practiced at Stanford University and stayed at the Santa Clara Marriott."
Question: "At what university's facility did the Panthers practice?"
Prediction: "Stanford University", **Answer:** "San Jose State"

In the above example, the model mistakenly focuses on 'Stanford University' because of the word 'practiced'. The real answer 'San Jose State' is also associated with 'practice' but the model failed to capture the word 'facility' in the context to distinguish it from the fake answer. The attention layer of our model is not strong enough to capture complex interactions between context and query. Since our model only used one layer of attention in two directions, we think that it may be helpful of adding deeper layers to recursively compute query-to-context and context-to-query attention. Intuitively, this will allow attention signals to flow through different regions with further comprehensive understandings so that the model can make multi-step complicated deductions.

References

[1] Rajpurkar P, Zhang J, Lopyrev K, et al. Squad: 100,000+ questions for machine comprehension of text[J]. *arXiv preprint* arXiv:1606.05250 (2016).

[2] Seo, Minjoon, et al. "Bidirectional Attention Flow for Machine Comprehension." *arXiv preprint* arXiv:1611.01603 (2016).

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