Machine Comprehension

Group No. - 25 **Group Name - Team Rocket** **Mentor - Abhinav Garg**

Shanu Kumar
sshanu@iitk.ac.in
150659

Harshit Saini
harshits@iitk.ac.in
150279

Ribhu Nirek
ribhu@iitk.ac.in
150578

Aviraj Mishra
aviraj@iitk.ac.in
150170

Ria Sharma
riash@iitk.ac.in
150577

1 Problem Statement

Machine comprehension is a question answering problem in Natural Language Processing. Our aim is to implement the BiDAF [1] model on our own using SQuAD [2] dataset and improving the embedding layer by introducing Part-of-speech (POS) embedding.The model will take a passage and questions related to it and will output the answers to the questions or a sub-phrase of the paragraph as a answer.

Our Code is available at `https://github.com/Sshanu/Machine-Comprehension`

2 Motivation

People ask questions to improve their understanding and knowledge of the world.Asking questions is a way to access the knowledge of others or to direct ones own information-seeking behavior. Here we study the generation of natural-language questions by machines, based on text passages. This task is synergistic with machine comprehension, which pursues the understanding of written language by machines at a near-human level.

3 Existing Work

Some of the notable previous works in end-to-end machine comprehension have employed attention mechanisms in the following distinct ways.

1. Using dynamic attention mechanism [5] in which the attention weights are updated dynamically given the query and the context, as well as the previous attention.

2. The second group computes the attention weights once, which are then fed into an output layer for final prediction (e.g., Kadlec et al. (2016)). Attention-over-attention model [6] uses a 2D similarity matrix between the query and context words to compute the weighted average of query-to-context attention. In contrast to these models, BiDAF[3] does not summarize in the attention layer and instead use a bidirectional attention mechanism.

4 BiDAF Model

BiDAF is a hierarchical multi-stage architecture consisting of six layers. It includes character-level, word-level, and contextual embeddings; and uses bi-directional attention flow to obtain a query-aware context representation.

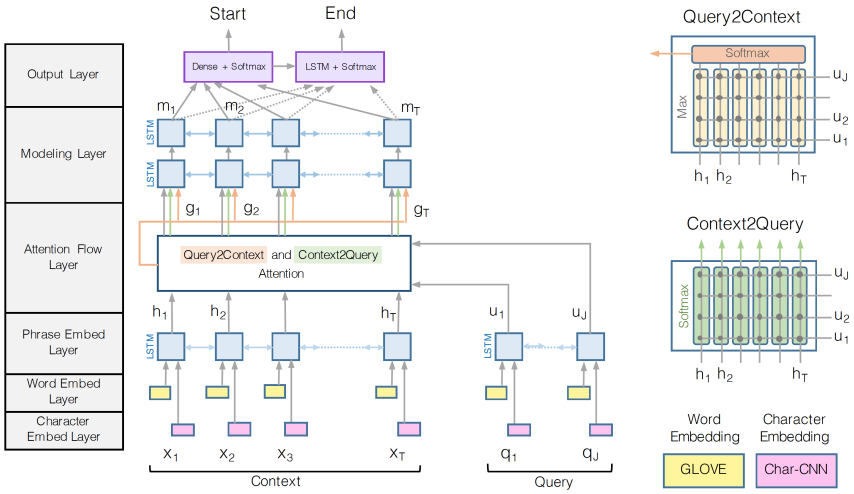

Figure 1: BiDAF Model[1]

4.1 Character Level Embedding

We obtain the character level embedding of each word using Convolutional Neural Networks (CNN), CNN is best in deriving the local features. Character level embedding enables us to deal with common miss-spellings, different permutations of the word such as run , runs , running etc.

Let $(\boldsymbol{x_1}, ...\boldsymbol{x_T})$ and $(\boldsymbol{q_1}, ...\boldsymbol{q_J})$ represent the words in the input context paragraph and query, respectively. These characters are embedded into vectors as $\boldsymbol{A_{cx}} \in [N, M, JX, W, dc]$ and $\boldsymbol{A_{cq}} \in [N, JQ, W, dc]$, where N: batch size, M: max number of sentences, JX: max number of words in a sentence, JQ: max number of words in a question, W: max word size, dc: dimension of character level embedding, which can be considered as 1D inputs to the CNN. The outputs of the CNN are max-pooled over the entire width for each word too obtain a fixed-size vector for each word.

4.2 Word Embedding Layer

We are using GloVE [3] a pre-trained model to map each word to a vector space. GloVe is an unsupervised learning algorithm for obtaining vector representations for words by Stanford. It covers more than 400k words in its vocab; and it clusters words with similar semantics. Ex Frog, toad, rana, lizard. It has a linear substructure which quantifies the relatedness of the two words. We use pre-trained word vectors with 100 dimension to obtain the fixed word embedding of each word.

4.3 Contextual LSTM

This Layer utilizes contextual information from surrounding words to refine the embedding of the words.

For this Bi-directional Long Short-Term Memory Networks (LSTMs) are used on top of the embeddings, to model the temporal interactions between words.

Bi-directional LSTMs are used to preserve the information from both past and future of the sequence. It then concatenates the outputs of the two LSTMs. From the two LSTMs we get $\boldsymbol{H} \in R^{2dxT}$ from the context word vectors X, and $\boldsymbol{U} \in R^{2dxJ}$ from query word vectors Q.

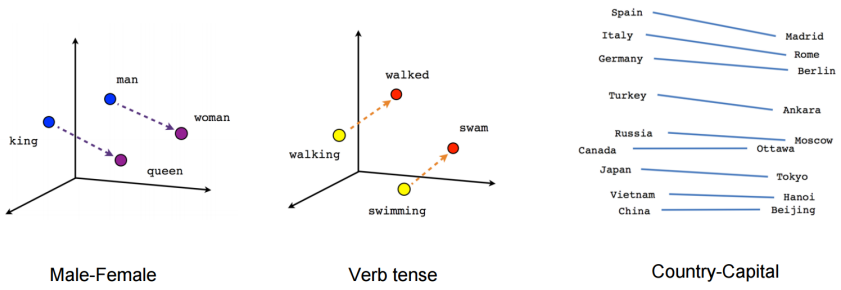

Figure 2: Word Embeddings using GloVE[3]

4.4 Attention Layer

Attention flow layer is responsible for linking and fusing information from the context and the query words. Previous works in Attention flows use only query to context attention, which helps summarizing the context conditioned on the question; but a lot of relevant information is lost using only this. So, we compute attention in two directions: from context to query and from query to context.

To calculate the weights for both the attention layers, we compute a Similarity Matrix $\boldsymbol{S} \in R^{TxJ}$ between the contextual embeddings of the context **(H)** and the query **(U)**, where $\boldsymbol{S_{tj}}$ indicates the similarity between t-th context word and j-th query word. The similarity matrix is:

$$\boldsymbol{S_{tj}} = \boldsymbol{w}[\boldsymbol{H_{:t}}; \boldsymbol{U_{:j}}; \boldsymbol{H_{:t}} \circ \boldsymbol{U_{:j}}]$$

where $\boldsymbol{w} \in R^{6d}$ is trainable vector, [;] represents concatenation across row and $\circ$ represents element wise multiplication of matrices.

4.4.1 Context-to-query Attention

Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. The attention weights $\boldsymbol{a_t}$ on the query words by the t-th context word is computed by taking Softmax across t-th row of the Similarity Matrix $\boldsymbol{S}$. Then the attended query vector $\tilde{\boldsymbol{U}}_{\boldsymbol{:t}} = \sum_j \boldsymbol{a_{tj}} \boldsymbol{U_{:j}}$ is a 2d-by-T matrix containing the attended query vectors for the entire context.

4.4.2 Query-to-context Attention

Query-to-context (Q2C) attention signifies which context words have the closest similarity to one of the query words. We obtain the attention weights on the context words by $\boldsymbol{b} \in R^T$ by taking Softmax over the max(column), where max(column) represents maxiumum across a column. Then the attended context vector is $\tilde{\boldsymbol{h}} = \sum_t \boldsymbol{b_t} \boldsymbol{H_{:t}} \in R^{2d}$. $\tilde{\boldsymbol{h}}$ is tiled T times across the column, thus giving $\tilde{\boldsymbol{H}} \in R^{2dxT}$.

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

$$\boldsymbol{G_{:t}} = \boldsymbol{\beta}[\boldsymbol{H_{:t}}; \tilde{\boldsymbol{U}}_{\boldsymbol{:t}}; \boldsymbol{H_{:t}} \circ \tilde{\boldsymbol{H}}_{\boldsymbol{:t}}; \boldsymbol{H_{:t}} \circ \tilde{\boldsymbol{U}}_{\boldsymbol{:t}}]$$

where $\boldsymbol{G} \in R^{8dxT}$ and β is a trainable vector.

4.5 Modelling Layer

This layer employs a Recurrent Neural Network to scan the context.The output of the modeling layer captures the interaction among the context words conditioned on the query.

This is different from the contextual embedding layer, which captures the interaction among context words independent of the query.

For this, two layers of bi-directional LSTMs, with the output size of $\boldsymbol{d}$ for each direction is used.

We obtain matrix $\boldsymbol{M} \in R^{2dxT}$ where each column vector of $\boldsymbol{M}$ contain contextual information about the word with respect to the entire context paragraph and the query.

4.6 Output Layer

QA tasks require the model to find a sub-phrase of the paragraph to answer the query. Output layer predicts the starting and the ending indices of the phrase in the paragraph to answer the query.

We obtain the probability distribution of the start index over the entire paragraph by

$$p^1 = softmax(w_{p^1}^T[G; M])$$

where $w_{p^1} \in R^{10d}$ is a trainable weight vector. For the end index of the answer phrase,we pass $\boldsymbol{M}$ to another bidirectional LSTM layer and obtain $\boldsymbol{M^2} \in R^{2RxT}$ .Then we use $\boldsymbol{M^2}$ to obtain the probability distribution of the end index in a similar manner:

$$p^2 = softmax(w_{p^2}^T[G; M^2])$$

5 Training

Training loss is defines as the sum of the negative log probabilities of the start and end indices by the predicted distributions, averaged over all examples:

$$\boldsymbol{L(\theta)} = \frac{1}{N}\sum_{i=1}^{N} log(p_{y_i^1}^1) + log(p_{y_i^2}^2)$$

where θ is the set of all trainable weights in the model (the weights and biases of CNN filters and LSTM cells, w_s , w_p^1 and w_p^2) , N is the number of examples in the dataset, y_i^1 and y_i^2 are the true start and end indices of the i-th example, respectively, and p_k indicates the k-th value of the vector p.

6 Testing

The answer span (i, j) where i$\leq$j with the maximum value of p_i^1 p_j^2 is chosen, which can be computed in linear time using dynamic programming.

7 Our Contribution

7.1 Part Of Speech Tags

Part-of-speech tagging is the process of marking up a word in a text, corresponding to a particular part of speech; based on both its definition and its context. We aim to capture the relationship with adjacent, related words in a phrase, sentence, or a paragraph. Part of speech helps machine distinguish the context in which a word may be used, consider the example where the word "park" is used in two different contexts in the ensuing two sentences:

'Play in the park.' and *'Park the car.'*

In the first sentence 'park' is a noun, whereas in the second sentence 'park' is a verb.

We used Stanford Part-of-Speech Parser[4] on the SQuAD[2] dataset for both the context and the query to get the tags.

7.2 Skip-Gram Model

The skip-gram neural network model is a simple neural network with a single hidden layer with no activation function on the hidden layer neurons, but the output neurons use softmax.

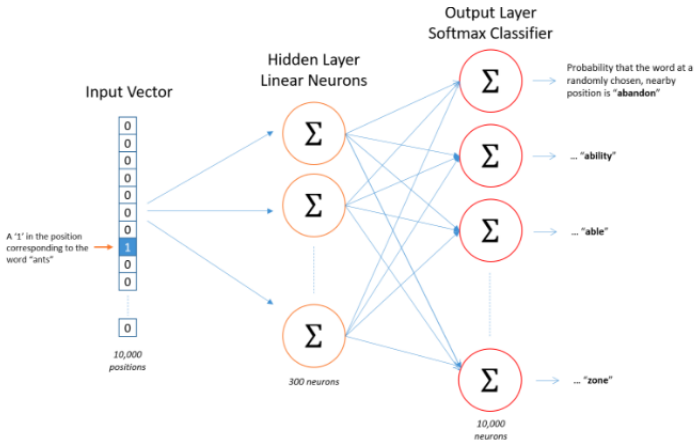

Figure 3: Skip-gram model

Tag Name	POS tags	Count
Noun, singular or mass	NN'	391092
Preposition or subordinating conjunction	'IN'	313428
Proper noun, singular	'NNP'	286327
Determiner	'DT'	246818
Adjective	'JJ'	194522
Noun, plural	'NNS'	150057
Comma	','	138811
Full Stop	'.'	92716
Coordinating conjunction	'CC'	86433
Verb, past tense	'VBD'	83807
Cardinal number	'CD'	77651
Adverb	'RB'	71899
Verb, past participle	'VBN'	69596
to	'TO'	48800
Verb, 3rd person singular present	'VBZ'	41862
Verb, base form	'VB'	40118
Verb, gerund or present participle	'VBG'	35283
Colon	':'	34343
Verb, non3rd person singular present	'VBP'	28324
Personal pronoun	'PRP'	24380
Foreign word	'FW'	22213
Possessive pronoun	'PRP$'	18771
Possessive ending	'POS'	14923
Whdeterminer	'WDT'	13756
Double quotes	''''	13344
Quote	'``'	12561
Modal	'MD'	11441
Proper noun, plural	'NNPS'	10637
Adjective, superlative	'JJS'	6400
Adjective, comparative	'JJR'	6373
Whadverb	'WRB'	5119
Whpronoun	'WP'	4378
Particle	'RP'	3189
Adverb, comparative	'RBR'	2910
Other	'OTHER'	2645
Existential there	'EX'	2623
Adverb, superlative	'RBS'	2274

Table 1: Most frequent POS tags in SQuAD Training dataset

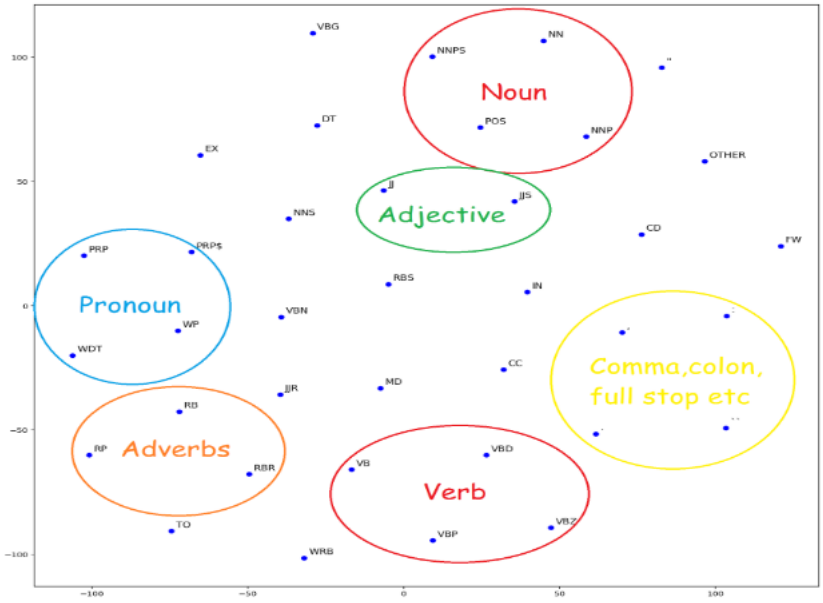

Figure 4: TSNE Projection of POS Tags embedding

Skip Gram Model gives the probability for every word in our vocabulary of being the nearby word to the word that we choose. The nearby word is decided by a window size, 5 in our case.
We train the neural network to do this by feeding it word pairs found in our training documents. However, the goal is actually just to learn the weights of the hidden layer, which are actually the word vectors that were trying to learn. In our model, we learn the word vectors for the top 36 POS tags from our data.

7.3 POS embeddings

Our data was very imbalanced as seen from the Table 1. For instance, the count of 'NN' tags was 391092, but that of 'JJS' was only 6400. So we used weighted loss function by adding more weight to the rare tags, where weight $\boldsymbol{w}$ is calculated by

$$\boldsymbol{w_{:t} = a\left[1 - b\frac{f_t}{F}\right]^n}$$

where $\boldsymbol{f_t}$ is count of t-th tag and $\boldsymbol{F}$ is total count of tags, a, b, n are parameters to be optimized. From experiments we found $a = 3, b = 4, c = 3$.

To avoid overfitting, we used dropout i.e. we added noise by dropping some tags that were too frequent using subsampling method by Mikolov. These tags are discarded with the probability $1 - \sqrt{t/f(w)}$, where t is the threshold parameter.

7.4 Experimental Results

After training the Skip gram model on POS tags we got a 50 dimensional POS embedding. Projecting the POS embedding using TSNE we got beautiful clusters of same types of POS tags, which is shown in the Figure 4.

After adding the extra POS embedding layer in the BiDAF model, the results obtained are illustrated in Table 2.

	Steps	EM	F1
BiDAF	20k	68.0	77.3
BiDAF+POS	8K	31.0	39

Table 2: Results on the SQuAD test set

7.5 Skills & Concepts Honed en route

- Understood the nuances of Deep Learning
- Learning about the significance of word embedding.
- Necessity of LSTMs for processing sequential interactions in the text.
- Using CNN to derive local features in the text.
- Significance of Attention mechanism in machine comprehension related tasks.
- We also learnt how to implement these techniques in **Tensorflow** framework.

7.6 Future Work

BiDAF model with POS embedding is now trained for 8k steps with the hyper parameters of BiDAF model. BiDAF is originally trained with 20k steps to achieve an F1 score of 77.3. Our model's performance can be optimized by tweaking the POS embedding: by increasing the number of steps and optimizing the hyper-parameters.
We shall employ ensemble methods to further increase the model accuracy.

References

[1] Minjoon Seo1, Aniruddha Kembhavi, Ali Farhadi, Hananneh Hajishirzi Bi-directional Attention Flow for machine comprehension

[2] Pranav Rajpurkar, Jian Zhang and Konstantin Lopyrev, Percy Liang SQuAD: 100,000+ Questions for Machine Comprehension of Text

[3] Jeffrey Pennington, Richard Socher, Christopher D. Manning GloVe: Global Vectors for Word Representation

[4] Part of Speech Tagging Daniel Jurafsky & James H. Martin.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. ICLR, 2015

[6] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. In ACL, 2016.

[Figure 2] https://www.quora.com/What-is-word-embedding-in-deep-learning

[Figure 3] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/