Machine Reading Comprehension with Deep Learning

Prasanna Kumar Challa, Prathwish Shekar Shetty, Sriram Jayaraman

ABSTRACT

Machine reading comprehension is the task of answering a natural language question about a passage. We tried to reimplement the Bi-Directional Attention Flow (BiDAF) network model developed by Minjoon Seo et.al. This implementation of attention model uses a six layer method to answer the questions. Our implementation of this model on PyTorch is able to have similar performance to the original implementation by the authors. We were able to get an EM score of 51.135 and an F1 score of 64.106, which is slightly less than the original implementation of which got EM scores of 67.7 and F1 scores of 77.3. Our slightly lower scores can be attributed to our focus to obtain computationally efficient models.

1. PROBLEM DESCRIPTION

Machine reading comprehension is the task of answering a natural language question about a passage. The tasks of machine comprehension and question answering have gained significant popularity over the past few years within the natural language processing community. 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. The task is to enable the computer to understand the documents and questions syntactically and semantically and also understand the types of reasoning required to answer the questions. A typical context for machine comprehension looks like the following example:

Context Paragraph: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other raindrops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called “showers”.

Question: Where do water droplets collide with ice crystals to form precipitation?

Answer: within a cloud

Here, the relevant information required to answer a question is distributed across multiple sentences. Understanding the relations between these sentences is key to finding the correct answer.

In this project, we reimplement Bi-Directional Attention Flow (BiDAF) network, a multi-stage hierarchical process that was proposed by Minjoon Seo et.al(2016) for machine comprehension.

2. RELATED WORK

The more traditional models developed until 2016 were the ones where in the query always attended to the context. The SQuAD dataset was a bit more challenging and the traditional methods didn’t seem to work effectively.

Rajpurkar et al. (2016) started off with an implementation of a logistic regression model with a range of features(~180 million). They found that lexicalized and dependency tree path features are important to the performance of the model. And the model performance worsens with increasing complexity of (i) answer types and (ii) syntactic divergence between the question and the sentence containing the answer; interestingly, there is no such degradation for humans. Their best model achieves an F1 score of 51.0%.

Xiong et.al mentions the use of dynamic coattention networks (context-aware-query and query-aware-context representation) for machine comprehension. This approach integrates mutual awareness between the query and the context. It generates first order context to query and query to context attention outputs, and then using one on the other to generate a second order output, and ultimately representation of the context which encodes the relationships between the two.

Our model is based on Minjoon Seo et.al. who utilizes bidirectional attention flow encoding and attention schemes, Combining question and context influence via a bidirectional LSTM and has Multistage encoding decoding.

3. DATA

Rajpurkar et al. (2016) released the Stanford Question Answering (SQuAD) dataset consisting of a set of Wikipedia articles with over 100,000 questions and has gained a huge attention over the last couple of years. Even now, this dataset is used as a benchmark dataset in machine comprehension with highly competitive leaderboard. It is one of the largest available MC datasets with human-written questions and serves as a great test bed for our model. The dataset is a closed dataset meaning the answer to every question is in the paragraph itself. So the task of answering the questions narrows down to finding the span from the corresponding reading passage. One other interesting fact about the dataset is that ~75% of the answers are less than or equal to four words.

4. EXPERIMENTS

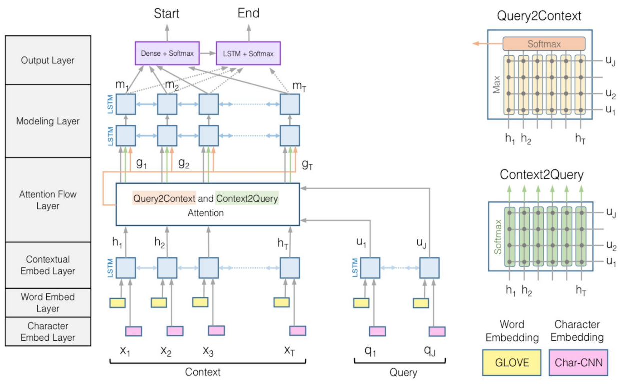
4.1. DATA PREPROCESSING

The preprocessing steps performed can be listed as follows

1. Tokenized paragraphs, questions and answers by splitting on spaces and separating punctuation from text.
2. Clipped any paragraph longer than 400 words in length, and conversely padded shorter paragraphs with the buffer token.
3. For question with multiple answers, have multiple instances of the same question with different answers assigned to each instance.

4.2. MODEL USED

BiDAF hierarchical multi-stage process and consists of six layers.



4.2.1. Character Embedding Layer

Input: List of characters from both query and context.
Output: Fixed-size vectors for each word max-pooled over entire width. The output is 16 dimensional vector per every single character.

Character level encoding helps in increasing the robustness against unknown tokens like misspelled words, emoticons, non alphabetic characters or the out-of-vocabulary words that could be encountered while testing. It also handles infrequent words better.

To do this, we first encode the characters in the passage as a list of character IDs which are then used to look up trainable character embedding vectors to obtain the character embedding representation of each word in the context. We then pass each word into a 1d convolutional neural network with a set window size to filter the character embeddings. We then max pool across the convolutional outputs for each character of the word to obtain character level representation of the word.

4.2.2. Word Embedding Layer

Input: List of words from context and query.
Output: Pretrained 100-dimensional Glove vectors to obtain the fixed word embedding for each word.

This layer separately maps the tokens in the context and the question into two sequences of pre trained word vectors. The mapping of each word to a high dimensional vector space is done using a pre trained GloVe word embedding model.

We then concatenate the character and word embedding vectors and is passed to a two-layer Highway network. The output of this Highway network are two sequences of d-dimensional vectors.

The concatenation of the character and word embedding vectors is passed to two-layer Highway Network.

4.2.3. Phrase Embedding Layer

Input: Char-word embeddings of query and context.
Output: Word representations aware of their neighbors.

We use a Bidirectional Long Short-Term Memory Network (LSTM) 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. The higher vector space representations for document and query.

Till now, everything we have done is to compute features from the context and the query at different levels of granularity.

4.2.4. Attention Flow Layer

Input: Phrase-aware context and query words.
Output: Query-aware representations of context words.

This layer is responsible for linking and fusing information from the context and query words. In this layer, we compute attentions in two directions: from Context to Query as well as from Query to Context.

Context to query attention models which query words are most relevant to each context word. Query to Context models which context words have the most similarity to one of the query words. Then combine the output the above two layers by concatenating them.

Query to context attention models signifies which context words have closest similarity to the query words and hence very important in answering the query.

The attention flow layer is not used to summarize the query and context into single feature vectors. Instead, the attention vector at each time step, along with the embeddings from previous layers, are allowed to flow through to the subsequent modeling layer. This reduces the information loss caused by early summarization.

4.2.5. Modeling Layer

Input: Query-aware representations of context words.
Output: Interaction among the context words conditioned on the query.

We use two bidirectional LSTM in both directions i.e. forward and backward to scan the context of the words in both directions i.e. how a word is related to the word to its front and back. The output of this layer captures the interactions among the context words conditioned on the query. This is different from the contextual embedding layer in the sense that the contextual embedding layer captures the interactions among the context words independent of the query words.

4.2.6. Output Layer

Input: Interaction among the context words conditioned on the query.
Output: Probability distribution of the start index and end index over the entire paragraph.

The loss function that we tried to minimise is the sum of negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples

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

y_i^1	True start index of example i
y_i^2	True end index of example i
$\mathbf{p}^1$	Probability distribution of start index
$\mathbf{p}^2$	Probability distribution of stop index

The output layer is responsible for finding the start and end indices of the answer in a context.

5. EXPERIMENTS

5.1. Evaluation Metrics

Exact Match (EM) and F1 score are the evaluation metrics used to tune the model.

EM Score: EM score is the proportion of the test questions that the model answers exactly correctly. EM will be 1 if predicted and ground truth answer is exactly same and 0 otherwise

F1 score: F1 score is harmonic mean between precision and recall. It is a measure of the overlap between the model's guesses and the true answers

$$Precision = \frac{Number\ of\ common\ tokens}{Number\ of\ predicted\ tokens}$$

$$Recall = \frac{Number\ of\ common\ tokens}{Number\ of\ ground\ truth\ tokens}$$

In case of multiple answers, we take the maximum of the scores

5.2. Experimental Settings

The model was trained using Google Colab, which offered us a NVIDIA Tesla K80 GPU with 12 GB of RAM. The training process took around 14 hours.

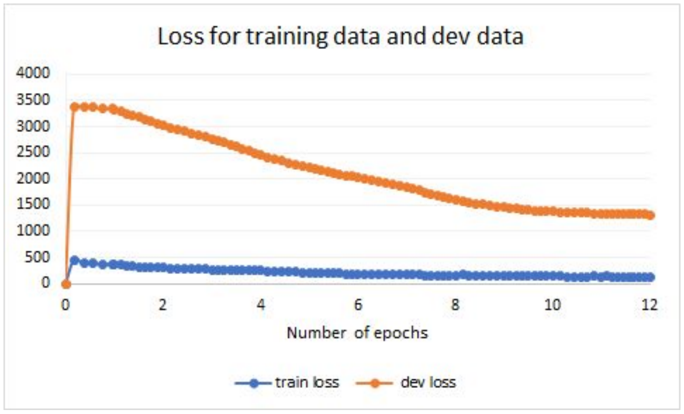
Since deep learning is computationally intensive, we set our hyperparameters in a decent range that gave us. Here are the list of hyperparameters used for the final model.

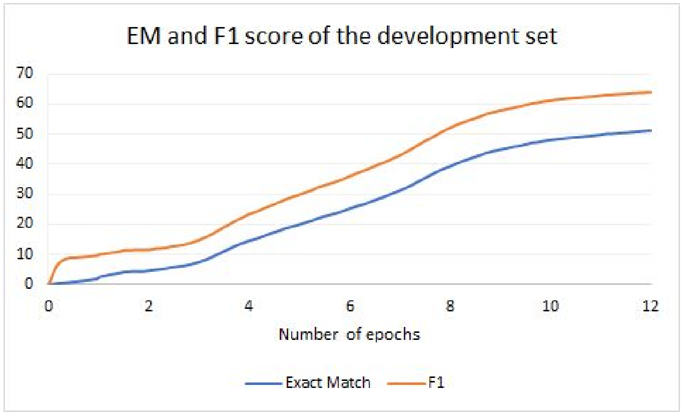
1. Paragraph Context Limit was set at 400 words as RNN needs all the context paragraphs to be of the same size for processing. In the original paper the context threshold was set at 750 words.
2. Character embeddings of size 8, we used 8 as our limit after evaluating the words by other authors and to reducing the computational overhead
3. We used 100 dimensional GLoVE embeddings for the word embedding layer
4. Dropout for LSTM were set at 0.2
5. Learning Rate for LSTM: 0.5
6. Epochs: 12
7. Exponential Decay for the LSTM was set at 0.999
8. Hidden Units for the bi-LSTM was 100
9. Train Batch Size was 250
10. Dev Batch Size was 100
11. Loss function used was cross entropy loss function

5.3. Results

We got an exact match score of 51.135 and a F1 score of 64.016 on the test data. The original model developed by Minjoon Seo et. al. gave an EM of 67.7 and F1 score of 77.3.

Models	**EM**	**F1**
Original BiDAF	67.700	77.3
Our implementation	51.135	64.016





We can see from the above plot that shows training and dev loss that there is significant difference between the loss of training and development data which implies that we are overfitting on our training data. In the next steps, we would like to work on not overfitting our models.

5.4. Future Work

We had to make a lot of compromises due to computational restrictions and make the model train faster. Here are the list of things we had to do to increase the computational efficiency:

- *Higher batch size:* We increased the batch size of the reduce the training time and this is one of the reasons, our model did not perform on par with the original implementation.
- *300D GloVE word embedding*: We used 100d GloVE word embeddings to develop the model, in the future we plan to use 300D and check the model performance
- *Lower hidden layers:* We reduced the number of hidden layers in our LSTM's to facilitate faster training, which has lead to overfitting on our training data. To remedy this we aim to increase the number of hidden layers in our LSTM.
- *Hyperparameter Tuning:* As discussed above, we had to play safe with the hyperparameters to make the project computationally efficient, we would like to try multiple models by trying different hyperparameters to see if it yields better results.

6. References

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

[2]. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi. "Bidirectional Attention Flow for Machine Comprehension"

[3]. Caiming Xiong, Victor Zhong, and Richard Socher. "Dynamic coattention networks for question answering"

[4]. https://rajpurkar.github.io/SQuAD-explorer/