A Dynamic Chunk Reader with Character Level Embeddings for Question Answering

Stanford CS224N Default Project

Tim Wu
Department of Computer Science
Stanford University
timwu0@stanford.edu

Abstract

In 2016, Yu et. al. proposed an end-to-end neural reading comprehension model, known as a Dynamic Chunk Reader (DCR), for question answering. In this model, Yu et. al. chose to input word embeddings as well as several other semantic and linguistic features such parts of speech and capitalization into their initial encoding layer. A natural follow-up to this is to experiment with different inputs to the encoding layer. One possibility is to input character embeddings in addition to the word embeddings. This paper describes a model that re-creates the DCR model from scratch and the creation of a character level embedding using CNNs to feed into the DCR model.

No external collaborators

1 Introduction

Over the past few years, several interesting NLP models have come out to tackle the task of question answering (QA). In particular, reading comprehension-based QA, the task of reading a text and then answering of a question using a chunk of the given text, has been of particular interest. With many applications from search engine QA to automated customer help services, this QA task has been popular for good reason. One model that arose in 2016 was the use of a *Dynamic Chunk Reader* (DCR). (1) In this model, Yu et. al.'s innovation was to include a DCR that would directly compare every possible answer chunk from the text (up to some reasonable maximum answer length) with the question encoding and determine the best possible chunk as the answer. This model is further elaborated on in Section 2. One interesting feature of Yu's model is that the Chunk and Output Layers were completely detachable from the rest of the layer. Therefore, through the research project we attach this DCR model to an encoder using BiDAF attention and character level embeddings.

The BiDAF-DCR model uses six layers. First, an *Embedding Layer* is performed on both the context takes a word embedding concatenated with a character embedding, fed through a Convolutional Neural Network (CNN), and puts the embeddings through a highway encoder. Then, the outputs of this layer are inputted into an *Encoding Layer*, where the input is put through an LSTM. Then, both the encoded representation of the question and context passage are fed into the *Attention Layer*, which implements a bi-directional attention flow (both context-to-question and question-to-context). The output is then put through another LSTM in the *Modeling Layer*. Then, this output is used to create all possible answer chunks in the *Chunk Representation Layer*, and then finally these answer chunks are compared with the question representation in the *Output Layer* in order to assign a final probability distribution. An in-depth description of the model is given in Section 3.2.

We approached this by task by first implementing a model with the baseline BiDAF models and character level embeddings. We then replaced the BiDAF model's output layer with a chunk extractor, this way ensuring that we could both measure the effectiveness of the character-level embeddings and the DCR. We also experimented with several output methods for the DCR output layer, described in

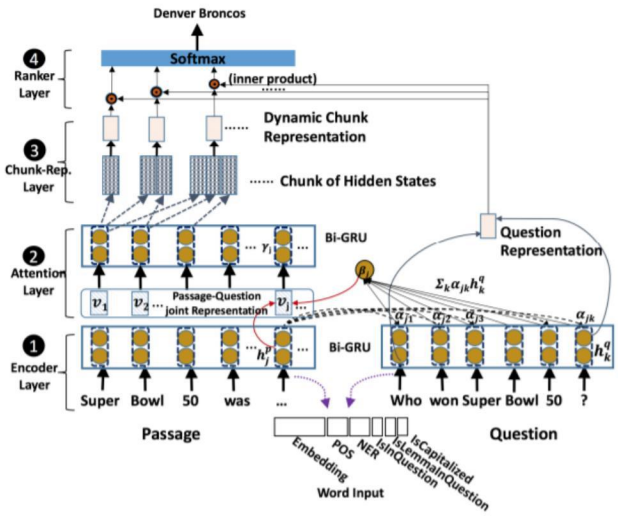

Figure 1: Yu et. al.'s DCR Model (1)

Section 4.3. Overall, we found that while the inclusion of character-level embeddings improved the model, the use of a DCR actually worsened QA performance relative to the baseline BiDAF model with character-level embeddings. By simply incorporating character-level embeddings to the BiDAF models with no chunk extraction, this model achieved an F1 score of 62.9 (baseline +1.3) and EM score of 59.7 (baseline + 1.6). However, with both the character-level embeddings and the DCR, this model only acheived an F1 score of just 52.2.

2 Related Work

As stated in the introduction, this model is largely inspired by the DCR model created by Yu et. al. as well as the BiDAF model provided as the baseline. A diagram of Yu's DCR model is provided above. All the layers of the the DCR are very similar to those in our model; Yu's encodes their word embedding with a bi-directional GRU (comparable to our bi-directional LSTM), applies attention, encodes the result with an the same bi-directional GRU, and the sends the result off to the Chunk Representation, from which point on our models are very similar.

As mentioned earlier, another important component of my model was the use of character level embeddings as input into the highway encoder in addition to the word level embeddings. We decided to base this off of a QA-net implementation by Kim et. al. (2015). (2) Following the logic from their implementation, this model uses a character-level CNN for the embedding.

3 Approach

3.1 Problem Statement

First, we formally define the problem statement: The goal of the task is to answer a given Q based on a supporting context passage C. The model must predict some element of the set of correct answers $\{A^1, A^2, ...\}$ where each answer $A^k \subset C$ is a subset of the passage. Each Q, C, and A^k is a sequence of words from some vocabulary V.

3.2 Model

Now, we describe the model. A diagram of the model is given on the following page. The input to the model was a question and context passage, and the output in training was a probability distribution over chunks, while the output in testing and evaluating was a single chunk with the highest probability. Note that this with the exception of the character level embedding, all of this code was either derived directly from the baseline or written from scratch myself, following along from the description

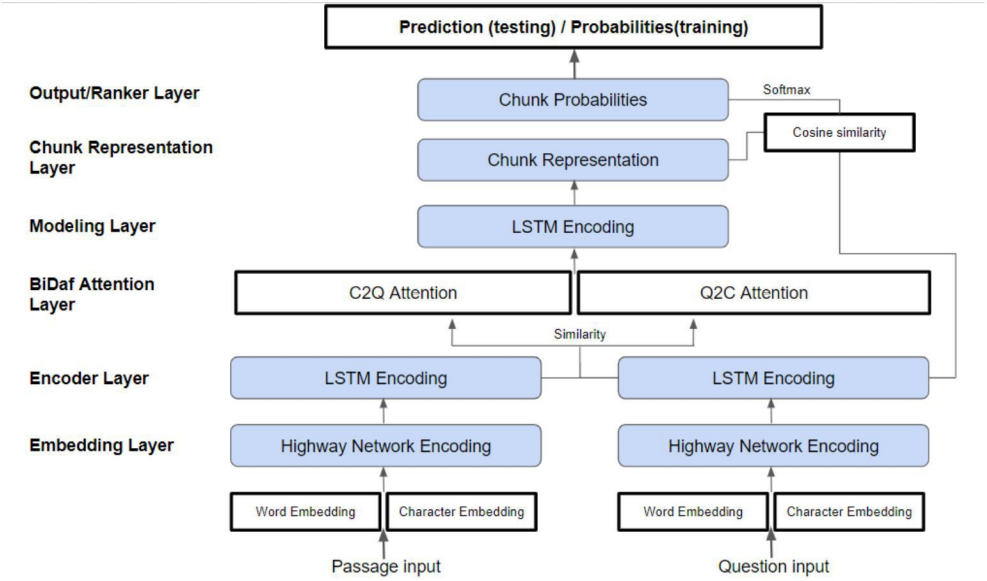

Figure 2: Model Diagram

of the DCR in Yu et. al.'s paper. Although the code for the character level embedding was also completely written from scratch by myself, it was largely inspired by Min Kim's 2018 implementation of QANet. (3) This code drew heavily from the character level embeddings portion Kim's open source implementation of QANet, with adjustments made after theoretical consultation with some peers with similar projects involving character level embeddings.

In this model, the question and context are given a word embedding and preliminary character embedding from its pretrained word and character vectors. The process for refining the character level embeddings through the use of CNNs is described below.

Character Level Embeddings

Let e_w be the word embeddings, e_c be the character embeddings, and v_c be the pretrained character vectors. Once we have obtained character level embeddings from our pretrained character vectors, they were fed into a CNN with one layer of 1-dimensional convolution, with the size of $|v_c|$. We then relu the outputted representation and then max-pool the results to down-sample the embedding. Then, to create the refined character embeddings, we perform:

$$e_c = ReLU(\text{1D-Conv}(e_c))$$
$$e_c = \text{Max-Pool}(e_c)$$

The resulting output is then fed into the embedding layer along with the raw word embedding.

Embedding Layer

Next, this character embedding is concatenated with the word embedding to create the overall embedding e and projected to the correct hidden size h. Then dropout is applied and it is sent into the highway encoder:

$$e = Dropout(\text{Proj}(e_w||e_c))$$
$$\text{Highway}(e) \rightarrow output$$

The details of the highway projection are further described by Srivastava et. al. (4). Essentially, it refines the inputted embedding using learnable parameters. The final output is a vector of hidden size h.

Encoding Layer

The encoding layer takes the output from the embedding layer h and puts it through a bi-directional LSTM. This trains the model to learn the temporal dependencies between timesteps. The LSTM

inductively computes both the forward hidden states $h'_{i,fwd}$ and backwards hidden states $h'_{i,rev}$ at word position i and concatenates these and applies dropout before returning the output of size $2h$:

$$h'_{i,fwd} = LSTM(h'_{i-1,fwd}, h_i)$$
$$h'_{i,rev} = LSTM(h'_{i+1,rev}, h_i)$$
$$Dropout(h'_{fwd}||h'_{rev}) \rightarrow output$$

BiDAF Attention Layer

This attention layer combines the outputted encoding layers of the context and question representation to a single attention output. First, the similarity between the encoded context representation c and encoded question representation q are computed with a similarity matrix $S \in^{2h \times 2h}$ defined with weight vector w such that

$$S_{ij} = Softmax(w^T(c_i||q_j||(c_i \circ q_j)))$$

Then, for each word i, we apply both context-to-question (C2Q) and question-to-context (Q2C) attention, which are the weighted sums of the hidden states of q (for C2Q) and c (for Q2C). The weights for these sums are determined by the rows of S:

$$C2Q_i = \sum S_{i,j} q_j$$
$$Q2C_i = \sum (S \cdot Softmax(S)^T)_{i,j} c_j$$

Then the final output is the size-$8h$ concatenation

$$c_i||ai||(c_i \circ C2Q_i)||(c_i \circ Q2C_i) \rightarrow output$$

for every word i.

Modeling Layer

This layers implements the same LSTM encoder as the Encoding Layer. However, it runs the LSTM encoder twice, while the Encoding Layer only does it once. The input to this layer is just a size-$8h$ attention vector att, and twice computes the LSTM encoding:

$$h'_{i,fwd} = LSTM(h'_{i-1,fwd}, att_i)$$
$$h'_{i,rev} = LSTM(h'_{i+1,rev}, att_i)$$
$$Dropout(h'_{fwd}||h'_{rev}) \rightarrow output$$

The output is once again of size $2h$ for each word.

Chunk Representation Layer

Now, we get to the two important layers that set this model apart from the standard BiDAF model. The chunk representation layer takes the output from the modeling layer mod and creates all possible answer chunks up to some maximal answer length n. After some experimentation, we found a trade-off between n and runtime; after experimenting with different maximum answer lengths of $n = 3, 6, 10, 15$, it was decided that using $n = 10$ provided the best balance. Looking through the answer lengths in the dataset, it seemed that very few had length > 10, so this was a reasonable result. Each word i was given up to n candidate answer chunks that began at index i (Note that every word was given exactly n possible candidate answer chunks unless exceeding a certain chunk length $k \leq n$ meant that word $i + k$ went beyond the entire passage length, in which case such chunks were ignored).

For word i, the candidate chunk of length $k \leq n$ was given by concatenating the forward RNN of mod_i with the backwards RNN of mod_{i+k}. Notice that this method only looks at the start and end position of the candidate chunk; through experimentation, Yu et. al. discovered that the embeddings of the starting and ending words could already predict chunk similarity to the question, largely accredited to the attention layer these embeddings had been put through. Formally, in this layer for each word i and chunk length $k \leq n$ such that $i + k$ did not exceed the length of the passage, we compute:

$$\overrightarrow{mod}_i = mod[: h]$$
$$\overleftarrow{mod}_{i+k} = mod[h :]$$
$$\overrightarrow{mod}_i||\overleftarrow{mod}_{i+k} \rightarrow output$$

Output/Ranker Layer

This final layer either outputs a probability distribution for training or a final answer. This, in conjunction with the loss representation was the most complicated layer, and one that we experimented. There were many hurdles with this layer, from runtime to compatibility with the BiDAF training algorithm. The output layer described is a chunk output that is very similar to that of Yu et. al.'s, and the final one of three that were tried. Although this was not the one that ultimately had the highest scores, this is the most theoretically natural output layer; an explanation of the resulting scores is given below in Section 4.4.

This layer first turned the tensor of all chunks into a tensor of all probabilities. It did so by first computing the cosine similarity between each chunk $a_{i,k}$, where i is the word position and k is the answer length, and a question representation q', obtained from the encoded question q outputted by the encoding layer. This representation is obtained by concatenating the last hidden state in the q's forwards RNN with the first hidden state in q's backwards RNN. Then, this cosine similarity is softmaxed and returned as an element of the probability distribution p. Formally,

$$q' = \overrightarrow{q}_{|Q|} || \overleftarrow{q}_1$$
$$p_{i,k} = Softmax(a_{i,k} \cdot q')$$

where $\cdot$ represents cosine similarity.

This p is then fed into a model trainer. Note that the implementation of the model trainer is very dependent on the structure of the output from the output layer, so the implementation described below is only compatible with the above output layer. For all other models, the standard training algorithm from the BiDAF baseline model was used. For training, we computed the negative log loss as

$$-\sum_{k=1}^{n}\sum_{i=1}^{|C|}\log(|p_{i,k} - x_{i,k}|)$$

where $x_{i,k} = 1$ if the correct answer did indeed start at word i and was of length k and $x_{i,k} = 0$ otherwise. For testing, we simply found the $p_{i,k}$ that had the highest probability across all words and chunk lengths and outputted $(i, i + k)$ as the start and end positions.

4 Experiments

4.1 Data

All of the data came from SQuAD, using the data/ directory from the baseline's provided squad repository. The train set had 129,941 passage-question pairs, the dev set had 6078 pairs, and the test set had 5915 pairs. The reported results were either the result of validation submissions to Gradescope or the highest reported EM/F1 scores at checkpoints during the training.

4.2 Evaluation method

The evaluation metric used was the standard EM and F1 scores for the SQuAD. The EM score was exactly the proportion of questions answered correctly, while the F1 score was a harmonic mean $2(\frac{1}{p} + \frac{1}{r})$ of precision $p = \frac{\#\text{true positives}}{\#\text{predicted positives}}$ and recall $r = \frac{\#\text{true positives}}{\#\text{actual positives}}$. A true positive is defined as a word that was predicted to be in the answer chunk and really was in the answer chunk; a predictied positive is any word that the model predicted was in the answer chunk; an actual positive is any word that is genuinely in the answer chunk

4.3 Experimental details

To begin, we downloaded the open source BiDAF model from `https://github.com/minggg/squad` as the baseline model. After running this to get preliminary baseline F1/EM scores, the character level embedding was added to this. Here, we experimented with a few different implementations and operations on the embedding, such as including a sigmoid function vs. no sigmoid function at the very end and using the product of the word and character embeddings rather than the concatenation, but found that ultimately removing the sigmoid and concatenating the word and character embeddings, as described in Section 3.2, ultimately produced the highest scores.

We then implemented the DCR model with several different output options. First, we outputted two separate score probability distributions, a p_{start} and p_{end} distribution. Here, we defined the probability of the i^{th} word in the context being the starting word of the answer p_{start_i} as the maximum probability across all chunks that begin with i. Then, p_{end_i} was defined as the maximum probability of all end location that ended up at word i across all chunks that were selected for p_{start}. While we believe that this would in theory produce rather high scores, unfortunately the runtime for computing such a p_{end} was unreasonable; the resulting model would have taken about 6 days to finish training all 30 epochs of 129900 questions each. Next, we tried another output method with the purpose of speeding up the runtime. The starting word probability distribution p_{start} was calculated in the same way, but we then fed these start probabilities through another encoding layer to obtain the end location probabilities. This produced decent F1/EM scores with very fast runtime, although the scores still lagged behind the baseline model's. Finally, we implemented the output layer described above in Section 3.2, outputting one probability distribution in terms of all possible chunks rather than start and end position. This required significant changes to the training algorithm, but ultimately it was the closest to the output layer in the DCR model by Yu. While we expected this model to perform very well, unfortunately this was not the case. Although the runtime for this model was very fast, allowing the entire model to train in only 8 hours, the FM/E1 scores were significantly lower than the baseline.

Due to time constraints, we tested the hyper-parameters by running the model with different parameters for a few epochs, comparing these intermediate F1/EM scores, and then stopping the model that produced worse scores. As a result, we could not obtain objective F1/EM scores from for some of these worse-performing trials. After such implementation, it was found that a learning rate of 0.5, dropout rate of 0.1, and 32 out channels with a kernel size of 10 on the CNN were optimal.

4.4 Results

Model Name	EM	F1
BiDAF Baseline Model	58.1	61.6
DCR, Yu et. al. 2016	62.5	71.0
CNN char. embedding, Wu 2021	**59.7**	**62.9**
DCR with start + end outputs, Wu 2021	**50.2**	**52.2**
DCR with chunk outputs, Wu 2021	**10.6**	**11.6**

Table 1: Comparison of Results

Here, perhaps the most surprising result is the low scores of the DCR model with chunk outputs. Some hypothesis for this include implementation error and incompatibility between the baseline code's training mechanisms and the nature of training the chunk output. Other than this, the scores for the DCR model with start and end outputs is unsurprising. While it is lower than the baseline model, this may be a result of the BiDAF model's output algorithm simply being theoretically better than that of the DCR model.

5 Analysis

Overall, we discovered that the DCR had difficulty of balancing runtime with performance. In addition, it had some compatibility issues with the training mechanisms of the baseline model. We believe this was largely responsible for the low scores with chunk outputs. All implemented models had the same code up through the modeling layer, and all DCR models had the same chunk representation layer, so from this it can be reasonably inferred that the low scores were the result of the output layer and training algorithm design.

Through these experiments, we discovered the surprising result that the BiDAF model with character embeddings actually performed the best. While this suggests that character embeddings may have also helped the DCR model, they might not have made a significant enough difference to offset the issues with the model.

6 Conclusion

Through this project, two main findings were discovered. First it was discovered that the BiDAF model can be improved by adding character level embeddings. Second, it was discovered that this specific variation of the Dynamic Chunk Reader actually performs worse than the standard BiDAF model. Although the DCR did score lower, this project still discovered many useful differences in the performances of DCR models with different output formats and also successfully compared the use of character-level embeddings to their non-use.

After an extensive search for open source code online, we could not find any implementations of the DCR. This suggests that 1) inherently, the model may not be as good as some other existing models and 2) there has not been much research into the use of DCRs. The model given by Yu et. al. acheived EM/F1 scores of 62.5/71.0, and at the time of the writing of this paper, it is ranked $\#60$ on the official SQuAD leaderboard. Whether for the purpose of simply marginally improving the DCR model for theoretical and research purposes or for the purpose of significantly improving the DCR model, there is still a lot of potential research to do with regards to the DCR model. One could experiment with including data on parts of speech, capitalization, etc. in addition to the character and word level embeddings to input into the embedding layer, experiment with other chunk probability output mechanisms, or even look at different ways to represent candidate chunks.

References

[1] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end Answer Chunk Extraction and Ranking for Reading Comprehension. In *IBM Watson*, 2016.

[2] Yoon Kim. Convolutional Neural Networks for Sentence Classification. *New York University*, 2014.

[3] Min Kim. Implementing QANet (Question Answering Network) with CNNs and self attentions. *Amazon*, 2018.

[4] Rupesh Srivasta, Klauss Geuff, and Jurgen Schmidhuber. Highway Networks. *The Swiss AI Lab IDSIA*, 2015.