Dynamic and Static Chunk Reader (DCR/SCR) on SQuAD 2.0 Task

Stanford CS224N Default Project, IID SQuAD
Mentor: Manan Rai

Sho Ko
Dept. of Electrical Engineering
Stanford University
`kosho@stanford.edu`

Rubens Lacouture
Dept. of Electrical Engineering
Stanford University
`rubensl@stanford.edu`

Mahammad Shirinov
Dept. of Management
Science and Engineering
Stanford University
`shirinov@stanford.edu`

Abstract

Question answering has been attempted by numerous models of many different architectures. The Dynamic Chunk Reader model proposed by Yu et al. [1], contrary to most, has attempted to frame the task as the prediction of chunks of text rather than independent prediction of start and end indices. We implement this model and put it to test under the updated SQuAD 2.0 dataset. We also discuss the shortcomings of the model, related with how it chooses to generate the answer chunks to be considered as candidates. To this end, we propose a variation, the Static Chunk Reader model, that aims to overcome the limitations of DCR by making use of the structure of the context paragraphs from which the questions will be answered, based on these paragraphs' dependency trees. We also refine the word embeddings fed into the model by including the character-level embeddings, following the CNN approach by Kim [2]. We find that this augmentation improves the performance of the BiDAF [3] model that we use as baseline. Our findings also show that DCR performs poorly on the SQuAD 2.0 dataset, and SCR follows a similar pattern, although leaving more room for improvement.

1 Introduction

In this project, we build off of the End-to-End Answer Chunk Extraction approach as covered in [1]. To summarize briefly, this paper explores the idea of modeling the question answering problem as a probability prediction over all possible *answer chunks* in the context paragraph, as opposed to prediction of start and end indices separately. Chunks are just continuous pieces of text that are potential answers; they are mapped to the embedding space and those embeddings, along with question embedding (representation) are used to "rank" and predict the correct chunk as the resulting answer.

The model described in [1] has two main drawbacks that we are aiming to address. First, the model is proposed for the original SQuAD 1.0 dataset, and does not take into account unanswerable questions introduced in SQuAD 2.0. Second, and more importantly, the model suffers from inefficiencies resulting from its choice of potential answer chunks: *all* chunks within the context paragraph up to a specified maximum length are considered, and this results in (1) expensive quadratic-time computations when constructing the representations of these chunks dynamically and (2) redundant consideration of many chunks that can never be answers, and/or intersect with each other to a large extent. In the work presented in this paper, we aim to reimplement the DCR model and examine its performance on the SQuAD 2.0 dataset. We also propose a novel variant of the DCR model called Static Chunk Reader (SCR) which generates the chunks statically as a preprocessing step using the dependency structure of the context.

2 Related Work

Several other methods using neural models have been proposed to attempt to tackle comprehension-based question answering (QA) tasks. Attention Reader is one such neural model that aims to extract a single token as an answer from a provided text [4]. Yin et al. proposed another method for selecting the answer from a list of human-provided candidates [5]. The work proposed in this paper builds on top of those previous models by allowing QA models to be able to predict answers with variable length and arbitrary position within a given text.

The DCR paper has three advantages compared to related works in this area. First, it proposes a new neural reading comprehension architecture for answer representation by incorporating the start and end indices in the probability prediction. Second, it designs a new word-by-word attention layer to improve passage word representation, instead of conventional attention layers. Third, it comes up with a few features to strengthen the attention mechanism and improve candidate ranking. However, there are several disadvantages of the paper. For example, it only evaluates the model on SQuAD 1.0 dataset. They could potentially use it across a variety of question-answering datasets and make the conclusion more general. In addition, the paper does have an error analysis and it shows the model performs well on factoid question-answering problems. But for those long non-factoid question-answering problems, it can be further improved.

3 Approach

We outline our main approaches in this section. Our contributions are three-fold: First, we augmented the baseline BiDAF [3] model to include character embeddings, as per original paper. We then used these character embeddings to improve our results both in the baseline BiDAF model, as well as the DCR model. Our second contribution is the implementation of the Dynamic Chunk Reader (DCR) model introduced by Yu et al. in [1]. Our final contribution is a novel idea based on DCR and dependency parsing that we call Static Chunk Reader (SCR).

3.1 Character-level embeddings

As per the original BiDAF model in [3], we included character embeddings into the representations of the words in context and question parts of the dataset. As proposed by Kim [2], we embed the characters into vectors, which are then passed through a Convolutional Neural Network to obtain the final representation. This is then concatenated with the GloVe embeddings to form the final embeddings of the words.

3.2 Dynamic Chunk Reader implementation

As discussed earlier, Yu et al. [1] have proposed the Dynamic Chunk Reader model as a solution to the SQuAD 1.0 challenge. The implementation of this model is not to be found in the standard software packages, nor have the paper authors shared their implementation. Thus, we implemented this model from scratch using PyTorch. The first two layers of the DCR model are pretty standard, with bi-directinal RNNs and attention mechanisms. Same is true for the final Ranker layer, which is a standard dot-product of questions and answers passed through a softmax. We refer the reader to the original paper for details of these layers.

What's more interesting is what's called the Chunk Representation Layer. This layer takes in the question-aware representations of the context words, generates chunks of text that are potential answers to the question at hand, and uses the word representations to construct a representation for the candidate answer chunk.

We follow the method proposed by Yu et al. for chunk generation (i.e. generation of candidate answer chunks): given a context paragraph, we create up to N chunks starting at any position j in this paragraph, where N is a pre-selected maximum chunk length. Some drawbacks of this method, later addressed by our Static Chunk Reader, are that the number of generated chunks is large ($N \times C$ where C is context length), and there are a lot of chunks that trivially cannot be answers (e.g. "and professional", "1995 and hosted" etc. [1]).

[1]Examples taken from the SQuAD dataset [6]

Once we have our answer chunks $c^{m,n}$ spanning from position m to n, and forward and backward representations for each word j, $\overrightarrow{\gamma_j}, \overleftarrow{\gamma_j} \in \mathbb{R}^d$ produced by the previous layers, we represent the chunk $c^{m,n}$ by

$$\bar{\gamma}_{m,n} = [\overrightarrow{\gamma_m}; \overleftarrow{\gamma_n}] \in \mathbb{R}^{2d}$$

where $[\cdot;\cdot]$ is the concatenation operation and d is the dimension of the hidden state, as defined in [1].

For the sake of completeness, let us mention that the final Ranker layer then takes these chunk representations, takes their dot product with the question representation and computes the softmax of the dot products, predicting the chunk with the highest dot product with the question. At train time, the model uses this softmax output to minimize the negative log-likelihood loss, and at test time, to predict the answer chunk that it thinks is the correct one.

To be able to generate "no-answer" predictions, we adapt an approach outlined in the project handout – we prepend an out-of-vocabulary (OOV) token to the list of chunks to be ranked. It is treated just like another chunk, and the model predicts "no-answer" if the probability assigned to this chunk is the highest.

3.3 Static Chunk Reader

We present our proposed model (inspired by Yu et al.), the Static Chunk Reader in this section. First, we motivate our approach and review dependency parsers, which lie at the heart of the model, and then we provide the details.

3.3.1 Motivation

Recall that in a standard task of dependency parsing, a given sentence is structurally analyzed and dependency relationships are extracted. These relationships are in the form $head \rightarrow dependent$. Each word in the sentence has a $head$, except for the special `ROOT` token. Not all words, however, have dependents, and such words are what interest us the most. Considering the dependency structure of a sentence as a tree, and following the standard tree terminology, we will call such words *leaves*.

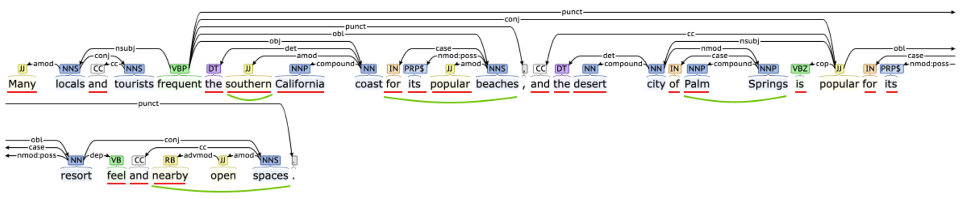

Many locals and tourists frequent the southern California coast for its popular beaches , and the desert city of Palm Springs is popular for its resort feel and nearby open spaces .
JJ NNS CC NNS VBP DT JJ NNP NN IN PRP$ JJ NNS CC DT NN NN IN NNP NNP VBZ JJ IN PRP$ NN VB CC RB JJ NNS
amod nsubj conj cc obj det amod compound obl punct conj case nmod:poss det compound nmod cop advmod dep

Figure 1: Dependency structure of an example sentence from the SQuAD dataset, generated by the Stanford CoreNLP dependency parser.

Figure 1 shows a sample sentence from the SQuAD 2.0 dataset, parsed using the Stanford CoreNLP tool. The leaves are underlined in red, and the green curves under word spans show all the answers to the (answerable) questions associated with this sentence. Notice that all the answer spans contain a leaf; furthermore, all answers are *either a leaf, or a span of text from a leaf to its head* (or something very close in terms of overlap, as in the case of the last answer, "nearby open spaces").

To generalize, we observe that the vast majority of answers fit this description – they are either leaves, leaf-head spans, or else very close. For example, our preliminary research shows that >95% of the answers in the SQuAD 2.0 dataset contain a leaf. This makes intuitive sense, because most questions can usually be answered using one word, possible with supporting words around it. In other words, most answers in a context are *local*, and the dependency tree is a natural way to reason about this locality. This predictable form of the answers motivates our SCR approach.

3.3.2 The model

Following the DCR framework, we are looking for the text span in the context that has the maximal probability of being the answer. However, one drawback of the DCR model is that it overgenerates invalid candidate spans in its Chunk Representation layer, so the search space is large and filled with noise. We propose an amendment by generating chunks not arbitrarily as in DCR, but using the dependency structure of the context.

Specifically, we generate chunks that are either leaves or leaf-head spans. For this, each context paragraph is parsed with a neural dependency parser, which gives a dependency tree. Our algorithm then iterates over the tree to find the words with no dependents (i.e. leaves), and adds them to the list of candidate chunks. Then, for each leaf, the algorithm finds its head, extracts the span of text ranging from the leaf to this head (or vice versa, depending on which comes first), and adds this span to the list of candidates as well. In the sentence in Figure 1, examples of such leaf-head chunks would be "many locals", "the southern California coast" and "Palm Springs".

This approach has the nice property of only considering the spans of text that semantically make sense, but it also, in a way, partitions the sentence into different local meaning groups, wherein answers to most questions lie. Notice that this generation algorithm can be changed and improved as one has more insight into the structure of the answers, but we chose to adopt this simple but powerful rule for the initial version.

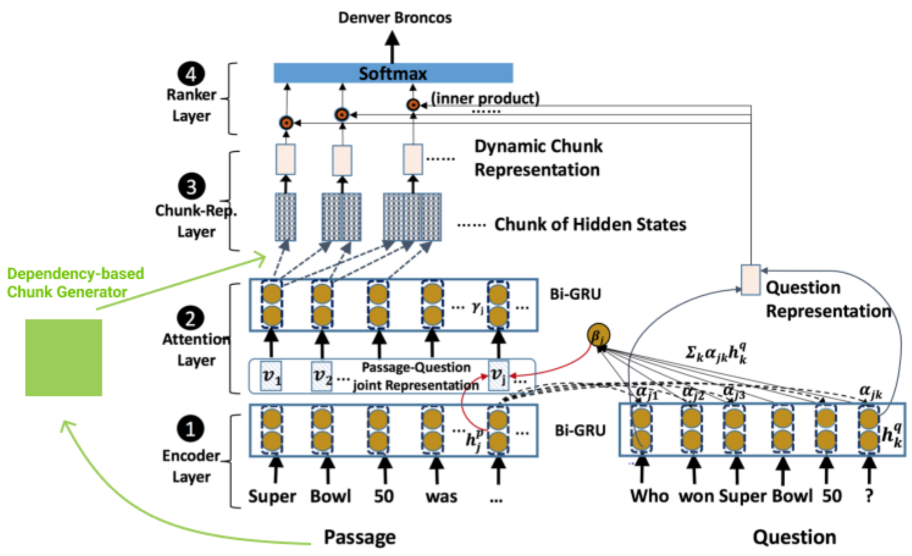


Figure 2: The Static Chunk Reader architecture. The right hand side of the figure (all parts without the green box and arrows) show the architecture of DCR and is taken from [1]

The rest of the model is the same as DCR – we generate representations for the chunks we have generated and pass them through a ranker layer. For efficiency we perform the dependency parsing and chunk generation beforehand as a preprocessing step. Thus, our model effectively takes the candidate answer chunks as input rather than dynamically generating them, thus the word "Static" in the name. Figure 2 shows the updated architecture of SCR.

4 Experiments

Next, we present technical details on our experiments and their results.

4.1 Data

As mentioned earlier, we use the SQuAD 2.0 reading comprehension for our experiments. This dataset contains questions by crowdworkers on a set of Wikipedia articles, where the answer to every question is either found as a span of text from the corresponding contextual reading passage, or is not answerable. SQuAD 2.0 dataset is split into 129,941 training examples, 6078 dev examples, and 5291 test examples.

4.2 Evaluation method

We evaluate our model using F1 and EM scores similar to the metrics used in the BiDAF baseline repository. The F1 score combines the precision and recall measurements into a single metric by taking their harmonic mean. The Exact Match (EM) score on the other hand which measures the percentage of predictions that match any one of the actual answers exactly.

4.3 Experimental details

To run our experiments, we configured our model using the hyperparameters used by Yu et al. in the original paper [1]. We used stochastic gradient descent with the ADAM optimizer utilizing an initial learning rate of 0.001. Unlike the BiDAF baseline, we set the hidden state size, d, to 300 for the encoder layers. We also applied dropout with a dropout rate of 0.2 to the embedding layer. For the embedding layer, we set the word embedding dimension to 300 and tested different dimensions for the character embedding dimension. We attempted with 100 and 200 dimension character embeddings and only used the one with the better performance for experiments.

Unlike the method used by Yu et al. where they applied early stopping after 10 epochs if the performance of the model did not improve, we fully trained for the maximum number of epochs which was set to 30. Whereas Yu et al. set a maximum passage length of 300 by pruning all the tokens after the 300-th token in the training set to save memory and boost training speed, we did not set any constraints on the passage tokens during training.

For our SCR variant, we performed the dependency parsing to gather the chunks offline prior to training. This information was then stored in a file. This was essentially added in the preprocessing step prior to running the model training where the stored information would be read from the file during training. This is different from the original DCR model as the chunks are generated statically rather than dynamically at run time like the DCR model, it also does not generate arbitrary length chunks from 1 to maximum chunk size as the original DCR model is doing.

4.4 Results

Here we present our experiment results:

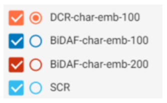
Model/Configuration	**F1**	**EM**
BiDAF (Baseline)	62.649	59.301
BiDAF w. char embed-100	64.894/ **62.115**	61.267/ **58.15**
BiDAF w. char embed-200	63.85	60.29
DCR w. char embed-100	52.2	52.2
SCR	52.1	52.1

Table 1: F1 and EM Scores for Different Models and configurations (plain: dev set, bold: test set).

We show the performance scores on the validation set for all of our models and configurations and also include the performance scores on the test set in bold for our best model. The BiDAF model with character embeddings shows the best results, both in F1 and EM metrics. Meanwhile, DCR and SCR weren't able to beat our baseline of BiDAF.

5 Analysis

The results show that the BiDAF implementation with character-level embeddings performs better than the baseline BiDAF model, and character-level embeddings size doesn't have too much of an impact on the model accuracy. We also observe that DCR and SCR models perform poorly on the



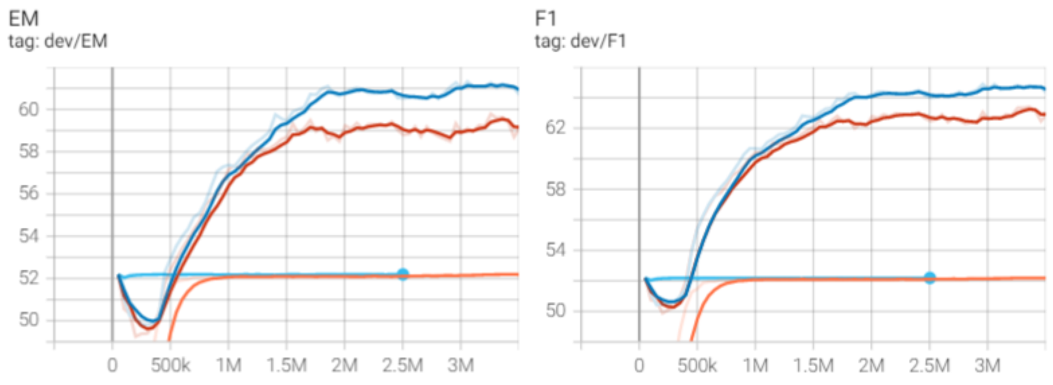


Figure 3: Tensorboard plot of model performance (F1/EM).

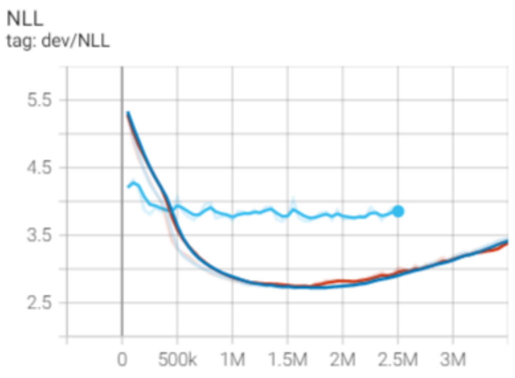


Figure 4: Tensorboard plot of model performance (loss).

SQuAD 2.0 dataset. The results of the DCR model are lower than what we expected, as it was shown in [1] to perform well on the SQuAD 1.0 dataset.

This poor performance could be because of a variety of reasons. The most prominent reason seems to be that this model (or the chunk representation approach) does not generalize well on the SQuAD 2.0 dataset, where a considerable portion of the questions are unanswerable. Recall that the power of the DCR model comes from combining word representations into the representation of the chunk containing them. This might be quite advantageous when the answer is certain to be among the chunks, but turns out to be not so good when a "no-answer" option is present.

Of course, there is always a possibility of an flawed implementation and insufficient hyperparameter tuning, but we have worked extensively to minimize the chances of this.

Similarly to DCR, the performance of SCR was less than expected. Because the architectures are so similar, this model likely suffers from the same problems as DCR. What's reassuring is that this model has lots of room for experimentation with different dependency-based chunk generators, which we expect will increase the performance.

6 Conclusion

Augmenting the word embeddings using character-level embedding proved to improve performance on the question answering task, signalling the power of character-level representations of word in high dimensions. As for the "chunk" approach to question answering, we conclude that it, at least in the form presented in [1], is not fit for the SQuAD 2.0 challenge. The Static Chunk Reader that we proposed did not perform better, but we are hopeful that it can be adapted or combined with a different architecture to deliver more promising results.

Future Work

In our proposed model, we used a simple chunk generation algorithm, which only considered leaves and leaf-head spans. One can definitely go into more detail analyzing the structure of answers in the SQuAD dataset (e.g. nouns connected by a “compund” relationship, words with “mod” relationship etc.), and use accordingly more powerful generation algorithms. We would be interested to see the performance of such approaches.

Another alley for improvement would be a different and more comprehensive usage of the dependency structure of contexts. It would be interesting to see applications that use this structure not only in preprocessing, but also in training and possible in attention mechanisms.

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. *arXiv preprint arXiv:1610.09996*, 2016.

[2] Yoon Kim. Convolutional neural networks for sentence classification. August 2014.

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

[4] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

[5] Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. Attention-based convolutional neural network for machine comprehension. *CoRR*, abs/1602.04341, 2016.

[6] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.