

Reimplementing Dynamic Chunk Reader

Stanford CS224N Default (IID SQuAD Track) Project

Nathan Gugel

Department of Computer Science
Stanford University
ngugel@stanford.edu

Elijah Spiegel

Department of Philosophy
Stanford University
spiegele@stanford.edu

Abstract

Some SQuAD models calculate the probability of a candidate answer by assuming that the probability distributions for the answer’s start and end indices are independent. Since the two do depend on each other, it should be possible to improve performance by relaxing this assumption and instead calculating the probability of each candidate answer span’s start and end indices jointly. We do so by reimplementing the Dynamic Chunk Reader (DCR) architecture proposed in Yu et al.[1], which dynamically chunks and ranks the passage into candidate answer spans using a novel Chunk Representation Layer and Chunk Ranker Layer. We implemented this model on the SQuAD 2.0 dataset instead of Yu et al.’s SQuAD 1 implementation. Our results performed more poorly than the baseline, which may indicate that the DCR architecture may not apply well to the SQuAD 2.0 task, or that we may have misinterpreted certain implementation details from the original paper.

1 Introduction

In the SQuAD 2.0 task, a model is tasked with answering a question about a piece of text (the “context”) by either selecting a contiguous string from the context or by indicating that the answer is not present in the context. This is a natural language understanding task; it provides insight into how well our models are able to process and effectively utilize the information contained in samples of natural language. This task is a high level one, as it requires both understanding what information is being asked for in the question and understanding what information is present in the context. As anyone who has taken a standardized test can attest, the correct answer to such a question is often better understood as the *most* correct answer, and it is often difficult to make judgements about how to rank candidate answers. It is thus very impressive that models have been able to achieve such high performance on this task, with modern transformer models doing approximately as well as humans. Early successes, prior to the invention of transformer architectures, were able to succeed on this task often by making simplifying assumptions about the nature of the task. For example, the Bidirectional attention flow (BiDAF) architecture treats the task of selecting the span of text as one of independently selecting the beginning and the end of the span, without considering their dependence on each other.[2] As models become stronger, it is possible to relax these simplifying assumptions and approach the problem in a truer fashion. In this way, Yu et al.’s Dynamic Chunk Reader architecture (DCR) relaxes BiDAF’s conditional independence assumption and produces its answers by selecting from a probability space defined jointly over the beginning and end of candidate answer spans (which Yu et al. call “chunks”). Yu et al.’s original DCR architecture was built for the SQuAD 1.0 task, in which an answer is always present in the context and models never indicate there is no answer in the context. In this paper, we reimplemented DCR on the SQuAD 2.0 task in order to see whether DCR’s architecture generalizes well or, perhaps, is restricted to the specific challenge of SQuAD 1.0.

2 Approach

Our approach is drawn from Yu et al.’s *End-to-end Answer Chunk Extraction and Ranking for Reading Comprehension*. [1] The Dynamic Chunk Reader (DCR) architecture proposed by Yu et al. is composed of four layers: an encoder layer, an attention layer with a novel attention mechanism, a novel chunk representation layer, and a ranker layer.

DCR’s encoder layer is composed of two bidirectional gated recurrent units (bi-GRU). One operates over the input from the context input and one operates over the question input. In Yu et al.’s implementation and in ours, both bi-GRU’s had the same weights and so were effectively a single GRU called over both inputs. In Yu et al.’s implementation, the encoder layer operates over concatenations of word embeddings drawn from pre-trained 300-dimensional GloVe embeddings and five feature representations: (1) a one-hot encoding for a part-of-speech (POS) tag, (2) a one-hot encoding named entity (NE) tag, (3) a binary value indicating whether the word’s surface form is the same as any word in the question, (4) a binary value indicating whether the lemma form of the word is the same as any word in the question, and (5) a binary value indicating whether the word is capitalized. The surface form of a word is the way in which it appears in the text, whereas the lemma form of a word is its uninflected form. So, for example, if “sing” appears in both the context and the question then features (3) and (4) would encode 1 for it, but if “sing” appears in only the context and “sang” appears only in the question then feature (3) would encode 0 and feature (4) would encode 1 for “sing”.

The attention mechanism proposed by Yu et al. was a novel one based on word-by-word style attention methods. The similarity score α_{jk} between the context word j and the question word k is computed as the dot product of their respective encodings $h_j^p \cdot h_k^q$. A passage word’s weighted sum of question encodings is then calculated as $\beta_j = \sum_{k=1}^{|Q|} \alpha_{jk} h_k^q$. The concatenation of the passage word with its weighted sum of question encodings, $v_j = [h_j^p; \beta_j]$, is then fed into another bi-GRU to get the output of the attention layer $\gamma_j = [\vec{\gamma}_j; \overleftarrow{\gamma}_j]$.

The chunk representation layer builds chunk representations of every candidate span up to a maximum window size. We treat selection of any span that includes a token prepended to the original context input, as a no-answer-present response. Yu et al. proposed two ways of determining candidate spans, the first being to compare the spans POS trie to the question’s POS trie and the second being to enumerate all spans up to a maximum window-size. We adopted the latter approach since it did not require POS data to implement. Since the total number of spans in a sentence of n tokens is $\frac{n(n+1)}{2}$, storing all spans quickly overruns spatial constraints. By setting a maximum window-size, it is possible to make the problem spatially feasible with only a small sacrifice in performance. The function $\bar{\gamma}_{m,n}$ Yu et al. used to represent a candidate chunk from m to n , $c^{m,n}$, was to concatenate the forward state of γ_m with the backward state of γ_n . While at first counterintuitive, since neither state has seen the indices in the interior of the chunk, Yu et al. propose that this representation function best represents the chunk’s context, which is especially important for this task.

The ranker layer’s probability function is a softmax over the dot products of chunk representations with the concatenation of the question’s first backwards hidden state embedding and its last forward hidden state embedding. The chunk ranker layer implements the probability function $\mathbb{P}(c_i^{m,n} | P_i, Q_i) = \text{softmax}(\bar{\gamma}_{m,n}^i \cdot [h_{|Q_i|}^{\overleftarrow{Q_i}}; h_1^{\overrightarrow{Q_i}}])$ to represent for a particular example i the probability that a particular chunk $c_i^{m,n}$ in it is the correct answer given the example’s passage P_i and question Q_i . Both we and Yu et al. train the model by minimizing the negative log likelihood $\mathbb{L} = - \sum_{i=1}^N \log \mathbb{P}(A_i | P_i, Q_i)$

for examples where the answer A_i is included as a candidate chunk, i.e. when the length of A_i does not exceed the maximum window size. Both we and Yu et al. train the model by minimizing the negative log likelihood of this output for examples where the true is included as a candidate chunk. We compare our results to a modified BiDAF model that does not incorporate character-level embeddings but is otherwise equivalent to that implemented by Seo et al.

2016.[2]

3 Experiments

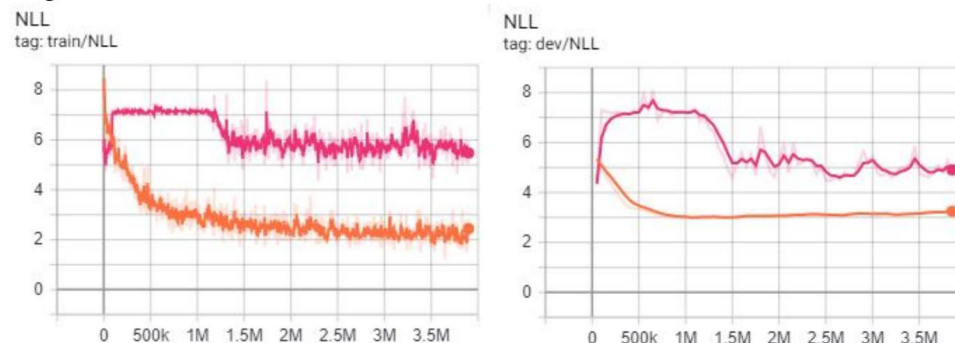
Data: We used the modified SQuAD 2.0 dataset provided by the CS224N course to train and validate our model. Notably, the training segment is identical to the original SQuAD 2.0 dataset, but the validation set is a partition of the original validation set (the other portion of the original validation set being used as a test set by the course).

Implementation Details: Whereas Yu et al. extracted features using the Stanford CoreNLP tool, we opted to use spaCy, as the sample for preprocessing the data was already using spaCy for tokenization. We loaded spaCy on its small English language model and ran it on each example sentence. We preprocessed all the example data this way to make training more efficient. SpaCy's tokens have an attribute labeled "pos" but we opted to use its "tag" attribute instead as this is based on a version of the standard POS set known as Penn Treebank. It is important to note that while Yu et al used a 46-dimensional one-hot vector to represent the POS feature, we used a 51-dimensional one-hot vector to be consistent with this the tag attribute. Similarly, while Yu et al. used a 14-dimensional one-hot vector to represent the NER feature, we used a 19-dimensional vector to be consistent with the number of possible values in the spaCy tokens' "ent_{type}" attribute.

Evaluation method: We measure results using standard EM and F1 scores. Our evaluation scores were calculated by making slight modifications to the BiDAF evaluation code provided by the course (the main modification being a conversion of target outputs into a format compatible with the chunk representations).

Experimental details: In accordance with the hyperparameters specified in Yu et al. we first used an initial learning rate of 0.001 with the ADAM optimizer with 30 epochs of training, a dropout of 0.2 after the embedding layer, and a hidden size was 300. Though Yu et al. terminates training early if the accuracy does not improve after 10 epochs, we continued training for all 30 epochs. We specified a maximum answer span length of 10 as this was found to generate candidate answers spans that contain the target answer 92% of the time by Yu et al. Our training configuration differs from Yu et al.'s primarily in that we do not use curriculum learning which provides a gentler introduction to learning by sorting sets of batches by length (such that easier examples are learning before harder example). We also do not prune tokens during training time. The original DCR uses a batch size of 180, but we changed the batch size to 64 due to GPU constraints. The initial weights of GRUs were initialized using Xavier uniform random variables in the range -0.057 to 0.057 while the original DCR used a uniform random range of -0.01 to 0.01. This difference was necessary due to the default behavior of pytorch which uses the hidden size to calculate the range.

In an early version of our model, we implemented DCR as described but without the features (i.e. word vector embeddings only) and with the aforementioned 0.001 initial learning rate. We found, surprisingly, that the loss was highly unstable. Due to this instability, we decided to diverge from Yu et al.'s approach and reduce the initial learning rate. Retraining the model with a learning rate of 0.0001 yielded a much more stable and steadily decreasing loss. The following plot shows the training and dev NLL progress of the 0.001 LR model in magenta against the same of the baseline BiDAF model in orange. We do not record loss on examples where the correct answer length is longer than our maximum chunk size.



We originally hypothesized that the instability, despite using the same learning rate as Yu et al., may have been due to not having implemented the features. However, after implementing the features as described and training with Yu et al.'s 0.001 learning rate for approximately one epoch, we found that

the loss was beginning to behave in the same way as the featureless model. Due to this, we opted to retain our modified learning rate of 0.0001.

Results:

On the test dataset, we achieved an EM score of 51.834 and an F1 score of 54.435. We display our results against the scores of the BiDAF baseline and Yu et al.’s scores in the following table, though it should be noted that Yu et al.’s scores are recorded against the SQuAD 1.0 task and therefore are not directly comparable to the other models in the table.

Table 1:

Models	EM	F1
BiDAF Baseline	56.6	60.1
Yu et al. DCR	62.0	71.2
Our DCR w/o features	52.2	52.2
Our DCR	51.8	54.4

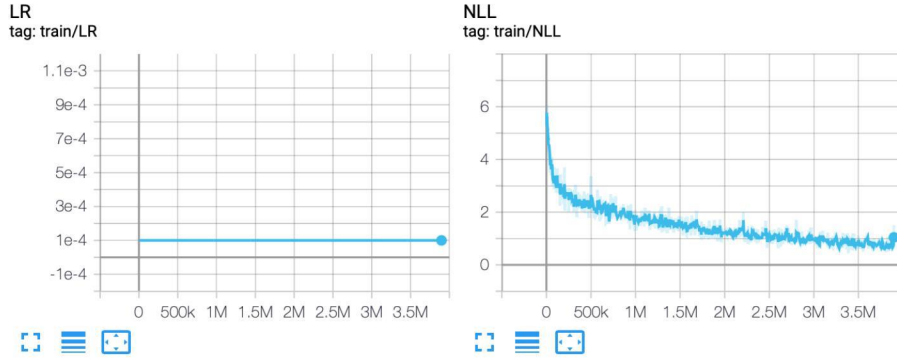
These results fall far short of what we expected the model to achieve. A discussion of possible explanations follows in our analysis section.

4 Analysis

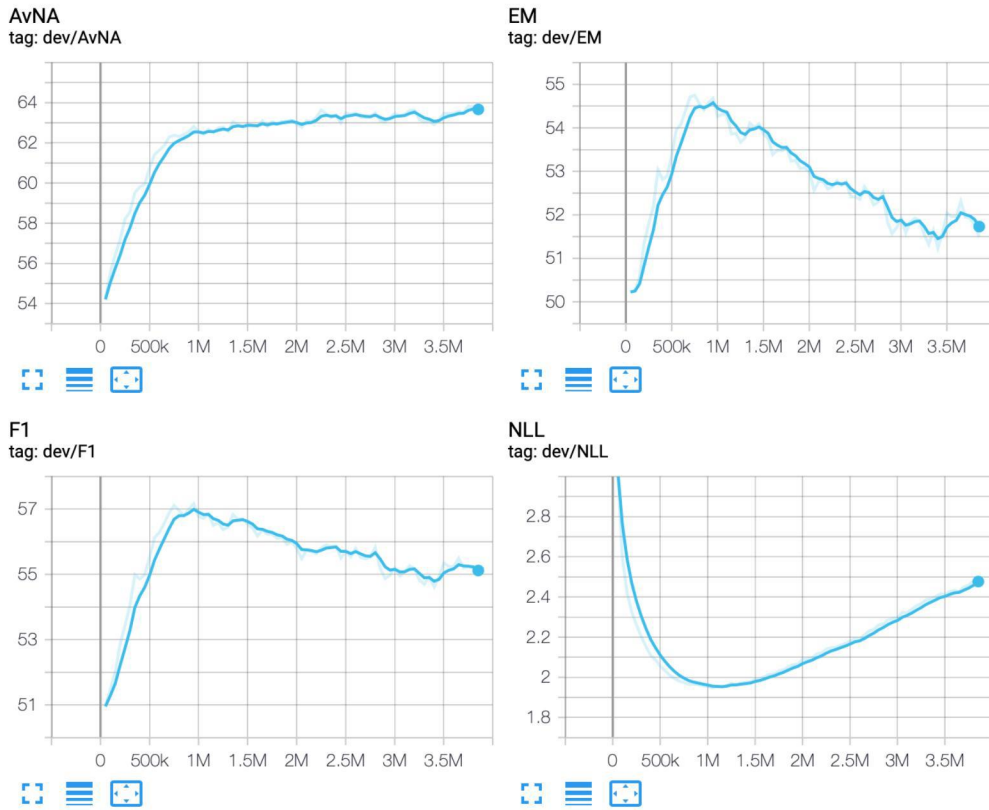
Our implementation of the DCR paper did not perform as well as we hoped, failing to surpass the BiDAF baseline and to approach the success of Yu et al.’s EM and F1 scores. We are uncertain what may have caused this disparity.

It is important to remember that Yu et al. trained and evaluated on the original SQuAD dataset whereas we trained and evaluated on SQuAD 2.0. We imagine that is within the realm of possibility that the DCR model is better suited to this first dataset than it is to the second. In particular, training on SQuAD 2.0 involves appending a “no-answer” token to the beginning of every example. A span is considered to be a “No answer” response in both the BiDAF baseline and in DCR if it contains the “no-answer” token. Since BiDAF chooses the beginning and end of its answer span independently, the ratio between the probability attributed to the no-answer token and the probability attributed to the rest of the probability space decreases linearly as sequence length grows. This is easy to see in the case of selecting start and end tokens on the basis of a uniform distribution, and the same principle holds in the case of BiDAF’s learned distribution. However, in the case of DCR and ignoring the maximum chunk size, the number of candidate chunks that include the no answer token grows linearly with sequence length, whereas the number of candidate chunks that indicate an answer grows quadratically. Even when considering a maximum chunk size, the ratio between probability attribution to “No answer” and probability attribution to answer responses decreases much faster with sequence length than in the case of BiDAF. This leads us to expect our model to predict answers when it shouldn’t much more often than BiDAF. In fact, we found that this is consistent with many of the examples in which our model failed. It is often the case that our model chooses some span from the passage even when the answer is “no answer” and the spans it chooses are “reasonable” in that they answer the question in a grammatical way and with an answer that fits the nature of the question. For example, with a question of “What did Gerooge Lenczowski do to the price of oil on October 16, 1973?” and a passage of “In response to American aid to Israel, on October 16, 1973, OPEC raised the posted price of oil by 70%, to \$5.11 a barrel.” (abbreviated here), it produces a prediction of “5.11 a barrel.” This is a reasonable answer. It simply misses the context of the actor in this question.

Furthermore, the Even with a learning rate of 0.0001, our model learned the training set very well, as can be seen in the following plot.



However, our model appears to have begun overfitting to the training set at around 1.25 million steps judging by the following plot showing the progress of our model’s dev NLL and dev scores. Such overfitting might be rectified through more dropout layers or other techniques, but any such measures are not mentioned in Yu et al.’s original paper.



5 Conclusion

Our reimplementaion of Yu et al.’s Dynamic Chunk Reader did not perform as well as hoped; we believe that this is primarily due to differences between the SQuAD and SQuAD 2.0 tasks and due to our model overfitting. Interestingly, our model’s loss was highly unstable with Yu et al.’s learning rate. There seemed to be no issue with cutting the learning rate, and training the model in this way appears to have actually resulted in overfitting on the data. Future work to attempt to combat this overfitting could shine a better light on whether DCR generalizes well to the SQuAD 2.0 context. As it stands, our two competing hypotheses for why our model did not perform well do not converge on an answer to this question.

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] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.