SQuAD2.0 with Conditional End Prediction and Character Embeddings

Stanford CS224N Default Project

Alaskar Alizada
Department of Computer Science
Stanford University
ala5kar@stanford.edu

Daniel Valner
Department of Computer Science
Stanford University
dvalner@stanford.edu

David Malinak
Department of Computer Science
Stanford University
dmalinak@stanford.edu

Abstract

In this project, we developed a series of models for the question-answering task on the Stanford Question Answering 2.0 (SQuAD2.0) dataset. We worked with all aspects of a successful deep learning model in mind, from a proper encoding skeleton, to experimenting with different predictive layers, to augmenting data. Specifically, we focused on improving the baseline BiDAF model through character embeddings and a conditional pointer layer. Our results were mixed: while the dependent relationship of a start- and end-tokens showed to be of little importance; character embeddings revealed the importance of rich sub-word relationships; and a foray into data augmentation revealed some potential pitfalls in that area.

1 Key Information to include

- Mentor: Kaili Huang
- External Collaborators (if you have any): None
- Sharing project: No

2 Introduction

A robust question answering system is an important and ubiquitous application of NLP in today's world, from search engines to chat bots. These QA systems use existing text to answer questions posed in natural language. To do so, they select a contiguous fragment of the given text. Systems like these are a great way to test the capacity of machine learning models to "comprehend" a piece of text. In this context, "comprehension" applies to both understanding the question being asked as well as understanding how the context passage relates to the question. Furthermore, the system needs to encode the structure of the passage and identify how facts relate to each other, which clauses depend on each other, and which words contain the most important information. Hence, identifying the proper fragment of the context is a complex problem.

The Stanford Question Answering Dataset version 2 (SQuAD2.0) has been a seminal dataset in the question-answering problem. Using online sources like Wikipedia, the architects of the dataset constructed contexts with rich semantic structure that require a significant level of "understanding" to answer. See the following example:

Question: When were the French wars of religion?
Context: The French Wars of Religion in the 16th century and French Revolution in the 18th

successively destroyed much of what existed in the way of the architectural and artistic remnant of this Norman creativity.
Answer: 16th century.

This is the structure for every example in SQuAD (a 3-tuple with question, context, and answer). To find the right words from the passage to include in the answer, we need to capture word meanings and functions which generate relationships within the text. (The word 'in' could be key here, for example.) Many attempts have been made to capture this complexity, including the BiDAF model, provided for the baseline, and the Match-LSTM with Bi-directional Ans-Ptr model, both described below.

We approached this problem differently in that we modified three different components of the existing baseline model. Chronologically, we started at the output layer and worked backwards. First, we replaced the output layer with a layer inspired heavily by Pointer Net (described in the Match-LSTM paper) and multiplicative attention. Seeing that results were weak, we moved back to the embedding layer to capture subword relationships with character embeddings. Encouraged by strong results but concerned about overfitting, we briefly experimented with a custom form of data augmentation which was inspired by the 'EDA' (Easy Data Augmentation) paper, described below. Unfortunately, we did not reach a highly effective implementation of data augmentation for reasons we will discuss in the analysis.

3 Related Work

3.1 BiDAF

The given baseline for our model and one of the first models used for SQUAD is Bi-Directional Attention Flow or BiDAF [1]. It uses the concept of having attention flow both from the question to the context and from the context to the question, thus the "bi-directional" attention. This allows the model to have some question-encoded information linked to the context. BiDAF then feeds this information to into an LSTM to predict the context spans. This has been shown to have 65/68 EM/F1 scores, which are standard evaluation metrics for SQuAD2.0 and QA models in general.

3.2 Match-LSTM with Bi-directonal Ans-Ptr

Another model that has seen improvement on EM/F1 evaluation metrics on the SQuAD dataset is the Match-LSTM model with Bi-directional Ans-Ptr [3], which combines methods from Match-LSTM and PointerNet [2] to create a question answering model. Match-LSTM was originally designed for predicting textual entailment (i.e. plausibility of coincidence) between a premise and a hypothesis but those methods were tranfered over to QA. At each position of the context, attention mechanism is used to obtain a weighted vector representation of the question. This weighted question is then to be combined with a vector representation of the current token of the context and fed into an LSTM. The match-LSTM essentially sequentially aggregates the matching of the attention-weighted question to each token of the context and uses the aggregated matching result to make a final prediction.

Described in the same paper, Ptr-Net introduces the method of using the attention mechanism as a pointer to select a position from the context as an output symbol, instead of simply picking an output token from a fixed vocabulary. This strategy happens to fit well with the SQuAD QA task; in their Ptr-net inspired efforts, Wang and Jiang add an Ans-Ptr layer to the Match-LSTM which conditions the distribution of the answer's end token on the answer's start token. This is done by feeding the attention output, produced when generating the start token distribution, into an RNN, which then produces the attention distribution on the end token. By mixing these two methods, Wang and Jian's model was able to produce predictions with 67.6/77 EM/F1 ratio. Note that these scores are on the SQuAD dataset and not SQuAD2.0, which includes unanswerable questions. These scores are similar (very marginally worse) to the BiDAF scores on SQuAD but had not been tested out on SQuAD2.0 prior to our explorations.

3.3 EDA

Other strategies that have helped improve performance on many NLP tasks have involved data augmentation to reduce overfitting. Outlined in EDA [4], methods like replacing words with synonyms,

randomly inserting words, randomly swapping words within a sequence, and randomly deleting words were proven to show improved performance in 5 NLP tasks, especially when using a small dataset. This is a result of preventing the model from fitting too closely to near-meaningless words like 'is' in a piece of text.

4 Approach

4.1 Baseline

We start by implementing the BiDAF model without character embedding as outlined in the hand-out/starter code. This includes an embedding layer, encoding layer, attention layer, modeling layer, and output layer.

4.2 Character Embeddings

The baseline approach to the SQuAD QA problem uses pre-trained, GLoVE-style word embeddings to train and test the model. More specifically, the baseline approach passes pre-trained word vectors through an embedding layer, consisting of dropout and a linear layer, to adjust the sizes of the embeddings to that of the model's hidden layer size. As we learned towards the beginning of the quarter, GLoVE-style word embeddings are able to capture inter-word relationships, but they miss semantic relationships that exist between characters from the same and different words. For instance, verb tense ('-ed') may be relevant to the meaning of the context and the position of the answer. Hence, we altered the embedding layer to process character embeddings for concatenation into final word embeddings.

Similarly to word embeddings, we use each character to look up a pre-trained character vector, and we use dropout to prevent overfitting (probability 0.2). Our first variation from the word embeddings process is a 2D convolution to relate characters within words and from different words. We follow the convolution with a max pooling step to reduce the resultant matrix to the proper dimensions for concatenation. Finally, we concatenate our word embeddings and character embeddings to create our final word embeddings of dimension, packaged in a matrix of size (batch_len, seq_len, hidden_size * 2).

4.3 Conditional Pointer Layer

Another approach we introduced to our model came from rethinking the loss that was used in the BiDAF approach. Specifically, the negative log likelihood loss they were trying to minimize in the baseline model was $loss_{NLL} = -\log p_{start}(i) - \log p_{end}(j)$. In other words, they were trying to maximize the likelihood of $P(a|\theta) = P(a_{start} = i, a_{end} = j|\theta) = P(a_{start} = i|\theta)P(a_{end} = j|\theta)$, where a is the answer and a_{start} and a_{end} variables indicate the start and end positions of the answer in the context. The assumptions made in the two equations are that answers to questions are contiguous in the context (such that answers exist including and between two boundary words), and that the a_{start} and a_{end} variables are independent of each other. Our approach focused on challenging the second assumption by building a more expressive model, where we would instead maximize the likelihood of $P(a|\theta) = P(a_{start} = i, a_{end} = j|\theta) = P(a_{start} = i|\theta)P(a_{end} = j|a_{start} = i\theta)$.

The implementation of our idea was inspired by the work done by Wang and Jiang through their implementation of Match-LSTM in machine comprehension [3]. We were impressed by the improvements the model had on its predecessors and its conditional layer seemed like an intuitive addition to BiDAF. (Intuitively, the selection of the end token should depend to some degree on the selection of the start token.) As explained in section 3.2, the goal of Match-LSTM in a textual entailment is to combine some representation of the premise with every token of the hypothesis and to run that representation through an LSTM to encode sequential information. In this setting, we use the output from the Model layer of the BiDAF M, which is a matrix that encodes representation of the context conditioned on the question with integrated temporal information. From here, we generate the probability distribution of the start token. Then, we use the start token distribution and the Match-LSTM technique to generate the probability distribution of the end token.
Specifically, we will run the output of the Model layer (matrix M) through the following functions:

$$F_k = \tanh(VM + (Wh_{k-1} + b))$$

$$\beta_k = (v^T F_k + c)$$

where $h_k = LSTM(M\beta_k^T, h_{k-1})$ (one-layer, unidirectional LSTM) and $k \in \{0, 1\}$.

Thus, β_0 becomes the probability distribution of the start token, and β_1 becomes the probability distribution of the end token conditioned on the start.

To achieve this, we first create the Start Token Layer (layers.StartTokenLayer) that calculates $k = 0$, with h_{-1} being 0. Then, the output β_0 from the Start Token layer is fed into the Output layer (layers.Output) where the same process is repeated, with the exception that we include a non zero h_0 vector that encodes information from both matrix M and the probability distribution of the start token. We also integrate their temporal information via the LSTM. The output of the Output layer is thus β_s and β_e (s is substituted for 0 and e is substituted for 1 for clarity of notation), which are used to calculate the altered negative loss likelihood through $loss_{ANLL} = -\log \beta_s[i] - \log \beta_e[j]$.

4.4 Data Augmentation

Seeing that we had not yet attempted to reduce overfitting, we considered data augmentation. Inspired by Wei and Zou's efforts from their 2019 EDA paper [4], we sought out the point in our architecture at which we could alter the context and the question. Our search brought us to the piece of the train.py file which surfaces the context/question word/character indices, just before passing the sequences to the model.

Here, we use an approach which we call Shuffling that varies slightly from Wei/Zou's Random Swapping (RS) technique, which involves swapping words in pairs until some fraction of the words or characters in the sequence has been altered (call this fraction α). Instead, we simply take a random sample of size $\alpha * len(sequence)$ columns from each batch (where rows are input sequences), shuffle them, and replace them into the batch matrix. This mostly has the same effect as RS, with the pitfalls that a) words/characters have a chance of residing in the same position, and b) some words switch positions with padding.

5 Experiments

5.1 Data

The dataset we will be using is the Stanford Question Answering Dataset (SQuAD 2.0). SQuAD 2.0 is made up of paragraphs from Wikipedia with questions and answers crowdsourced using Amazon Mechanical Turk. The dataset contains roughly 150k questions and split into three parts: train, test and dev sets. For roughly half of these questions there exists an answer somewhere in the provided context paragraph. The rest of the questions are unanswerable using the provided context. Our model has been designed to take in a context and an associated question, and return an answer from the context, if it exists.

5.2 Evaluation method

Our model performance was evaluated using two important metrics: Exact Match (EM) score and F_1 score. EM score is a metric that assesses whether our model output answer is an exact match of the ground truth answer. The dev and test sets both contain three answers per answerable question, which should add flexibility to this metric's criteria. Nevertheless, this is a very strict measurement of our model performance, so we will also be using the F_1 score. The F_1 score is a harmonic mean of precision and recall, mathematically defined as $F_1 = 2 * \frac{precision * recall}{precision + recall}$, where $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$ (TP: True Positive; FP: False Positive; FN: False Negative).

5.3 Experimental details

The first model we ran as our baseline was the standard BiDAF model. We ran full experiments on four different models and kept all of the hyperparameters constant. That is, for all four experiments we ran the models for 30 epochs with a batch size of 64, a learning rate of 0.5, and an Adadelta optimizer. We had some partial experiments where we toyed with some parameters the learning rates

(0.4, .7) and decreasing epochs (20-25) and got worse in both F1/EM results across the board on the baseline model. As a result, we decided to stick with the default hyperparameters for the following models.

Second, we altered baseline and ran the true BiDAF model in which we added character embeddings (as described in 4.2), with the same hyperparameters.

Third, we altered the baseline BiDAF model to add more expressiveness by rephrasing the loss function as a joint distribution of start and end tokens without making independence assumptions (described in 4.3). We made several attempts at trying to obtain this joint relationship. The first attempt, inspired by the multiplicative attention technique, involved trying to learn a relationship between all possible start-end token pairs of embeddings through $u^T W v$ where W was the learnable matrix of parameters and u,v were vector embeddings of individual start and end tokens. Seeing poor results from this technique, we decided to base our approach more closely to the approach used in match-LSTM (described in 4.3). We investigated using different number of layers for the unidirectional LSTM that integrates start token distribution into end token distribution in the Output layer and found that using a > 1-layered LSTM increased overfitting (discussed in analysis section), even with dropout, decreasing the performance of the model. We ran this method with the baseline model to be able to investigate its efficacy in isolation.

For the fourth experiment, we combined the two approaches from second and third models introducing our final model which had character embeddings as well as the coniditional pointer layer (from 4.2 and 4.3).

For our data augmentation experiments, we only ran partial training sessions. In our first trial, we implemented our Shuffling technique on 5% of batches per epoch and used $\alpha = 0.1$ (recall that α is the percentage of each given sequence that is altered). We altered all four (context-word, question-word, context-character, question-character) relevant sequences independently. Seeing several problems with this, we changed our strategy during a second trial to alter only the question (qw and qc), and swapping the same words in each sequence. This was to prevent questions and contexts from losing important meaning. Meanwhile, we kept our 5% and $\alpha = 1$ numbers the same.

5.4 Results

We see the results of our experiments in Table 1 above showing the F1/EM scores. Note that we only have the test scores for some of our models due to our limited attempts accepted by the leaderboard. We also show a chart of the AVNA (Answer vs No Answer) to better understand the model's classification accuracy considering only the answer vs no-answer predictions.

Configuration	**color**	**F1 (dev)**	**EM (dev)**	**F1 (test)**	**EM (test)**
Baseline (BiDAF)	orange	60.32	56.94	-	-
Baseline w/ Char Embeds	blue	61.98	58.62	-	-
Baseline w/ Conditional	fuchsia	60.84	57.39	-	-
Conditional w/ Char Embeds	gray	62.19	58.91	62.69	59.71
Baseline w/ Char Embeds, Data Aug	-	61.99	59.08	-	-

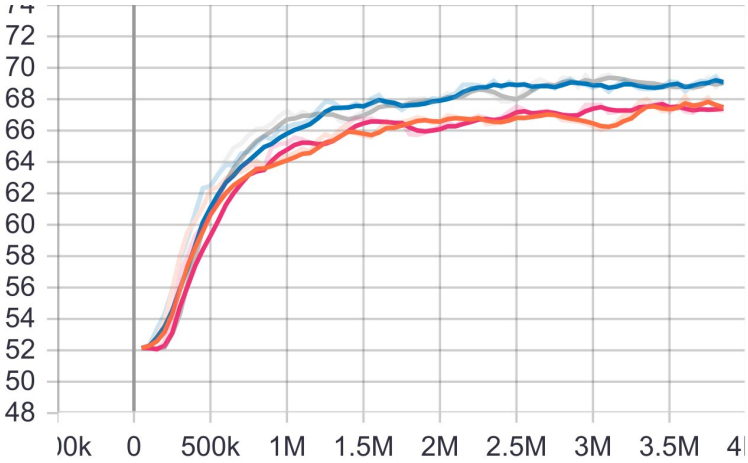

Figure 1. AvNA

6 Analysis

To understand how our models work, we looked over some of the examples from the dev set that were passed into our models and the associated predictions and ground truth answers. We found that all of our models had a common pattern in answering questions: Incorrectly answered questions were not so different from the actual answer so as to be incomprehensible. When the model had been marked incorrect in Exact Match, its prediction would often include too few or too many words, or words that were close to (spatially within the paragraph) the correct set of words.
Another pattern we noticed was that a lot of the incorrect answers had either incorrectly answered answerable questions with "No Answer", or incorrectly answered unanswerable questions with some set of words. We think that the former may be a result of the model having an option to predict "No Answer"; this option provides the model with a "safe" answer that it can pick when it is unsure of the exact set of words that need to be used in the answer. The latter, we think, is due to the fact that unanswerable questions can inherently be more convoluted than their answerable counterparts.

Crucially, in a lot of cases incorrectly labeled as "No Answer", we noticed that the mislabel was due to the fact that the context paragraph would answer the question with words that were synonymous to, but not exactly the same as, the words used in the question. As a result, unsure of whether two synonyms are correctly related for the answer, the model would answer with "No Answer". Perhaps if that option were taken away, the option with next highest probability would have been the correct answer.

Similarly, a lot of times when the ground truth answer was "No Answer", the model would overemphasize words in the context that exactly matched words in the question. Naturally, a similar type of behaviour was observed in the correctly predicted answers. Questions that used exact words that appear in the context paragraph, and even more so if they are in the same sequence, were guessed correctly. These behaviours can likely be attributed to the fact that we were working with a boundary model that was looking for an answer which included two boundary words, as opposed to selecting individual words that would answer the question.

In addition, after looking deeper into the testing examples, we can see specific cases where our models perform well and where they still struggle. One example where our conditional models outperforms the baseline is when the question is in tricky, unanswerable questions as seen below.

- **Question:** What was the high pressure engine an important component of?
- **Context:** In 1781 James Watt patented a steam engine that produced continuous rotary motion. Watt's ten-horsepower engines enabled a wide range of manufacturing machinery to be powered. The engines could be sited anywhere that water and coal or wood fuel could be obtained. By 1883, engines that could provide 10,000 hp had become feasible. The stationary steam engine was a key component of the Industrial Revolution, allowing factories to locate where water power was unavailable. The atmospheric engines of Newcomen and Watt were large compared to the amount of power they produced, but high pressure steam engines were light enough to be applied to vehicles such as traction engines and the railway locomotives.
- **Answer:** N/A
- **Prediction:** Industrial Revolution

In this example, the baseline fails while our conditional model correctly concludes that there is no answer. The question is asking about the "high pressure engine" while the context talks about "stationary steam engine" being a component of the Industrial revolution. We hypothesize that the added layer of attention on how long the answer should be allowed the model to realize that if we extend the considered object "engine" to its modifiers, the context is actually referring to a different object than the query. BiDAF likely simply had a high enough attention score with "engine" and "component" enough to trigger an incorrect answer. While these examples might have improved they were only a small part of the dataset and thus there wasn't that much of an improvement shown. This marginal improvement is not too surprising as its implementation on top of Match-LSTM was not better than BiDAF but it was worth the effort to test it on a new base encoder model.

Even with conditioning our model still struggles to consider context with strange sentence structure or when modifiers are very separated from some of the key words in the context. Consider our conditional model's output in the examples below.

- **Question:** NSF was engineered and operated by who?
- **Context:** The Very high-speed Backbone Network Service (vBNS) came on line in April 1995 as part of a National Science Foundation (NSF) sponsored project to provide high-speed interconnection between NSF-sponsored supercomputing centers and select access points in the United States. The network was engineered and operated by MCI Telecommunications under a cooperative agreement with the NSF. By 1998, the vBNS had grown to connect more than 100 universities and research and engineering institutions via 12 national points of presence with DS-3 (45 Mbit/s), OC-3c (155 Mbit/s), and OC-12c (622 Mbit/s) links on an all OC-12c backbone, a substantial engineering feat for that time. The vBNS installed one of the first ever production OC-48c (2.5 Gbit/s) IP links in February 1999 and went on to upgrade the entire backbone to OC-48c.
- **Answer:** N/A
- **Prediction:** MCI Telecommunications

In this example NSF is mentioned early in the passage while "engineered and operated" is more in the middle of the passage and was likely a hint that this was a "hot spot" of where the answer could be found. It is possible that because of a vanishing-gradient-like mechanism the earlier information isn't encoded correctly and thought to be closer to these words or even forgot. Because of this misplacement the model ends up answering the wrong question (outputting the networks and engineers and operator instead of recognizing that there was no answer to the question). A self-attention mechanism would likely help with these types of mistakes to keep track of where in the context words are.

Finally, with respect to data augmentation: our failure in our first run (5% of batches, modifying 10% of context-word, question-word, context-char, and question-char embeddings independently) was likely due to scrambling that was too severe. First of all, this strategy created memory issues at runtime which prevented us from running this version through a full training session. Secondly, shuffling the sequences independently created instances where one embedding would hold information for two totally different words. Moreover, shuffling all four sequences likely resulted in a loss of meaning between the context and question, which prevented the model from calculating a reasonable loss with which to respond to its own prediction.

On the other hand, we did find an example where data augmentation improved the prediction. The following is the given answer for the first data augmentation model, given the context:

- **Question:** Who was on Celeron's expedition?
- **Context:** Céloron's expedition force consisted of about 200 Troupes de la marine and 30 Indians. The expedition covered about 3,000 miles (4,800 km) between June and November 1749. It went up the St. Lawrence, continued along the northern shore of Lake Ontario, crossed the portage at Niagara, and followed the southern shore of Lake Erie. At the Chautauqua Portage (near present-day Barcelona, New York), the expedition moved inland to the Allegheny River, which it followed to the site of present-day Pittsburgh. There Céloron buried lead plates engraved with the French claim to the Ohio Country. Whenever he encountered British merchants or fur-traders, Céloron informed them of the French claims on the territory and told them to leave.
- **Answer:** 200 Troupes de la marine and 30 Indians
- **Prediction:** Troupes de la marine and 30 Indians

Meanwhile, our 'Baseline w/ Char Embeds' model answered 'St. Lawrence.' It's possible that we prevented the model from overfitting the association between 'Who' in the question, and 'St.' in the context. Of course, the association would be that saints are usually who's.

Our second trial (5% of batches, shuffling 10% of question-word and question-char embeddings together, i.e. same columns shuffled in each sequence, with a minimum of 2-column swap) acted on

par with the Baseline w/ Char Embeds for as long as we ran the training session. Hence we infer that the changes made to the dataset were not significant enough to force a change.

7 Conclusion

In this paper, we have implemented and explored different layers on top of the baseline BiDAF and measured their performance on the SQuAD 2.0 question-answering task. We achieved performance of F1/EM 62.69/59.71 on Test Leaderboard on our final model that used both character embeddings and a conditional layer. In our experimentation efforts, we found that the conditional layer was not as effective in improving our model as we hypothesized, while the character embeddings made a distinct improvement. We predict that there was enough information encoded in the baseline join distribution for our condition approach to make much, although we showed it improved performance for certain scenarios. We attributed the character embedding improvement to the additional out-of-vocabulary flexibility it brings which reduces overfitting to whole words.

We hypothesize that with more data or time for iterations on the architecture we could have optimized our model and improved its performance. We also believe that further implementation of data augmentation would add to the performance of our model.

References

[1] Seo, Minjoon, et al. "Bidirectional attention flow for machine comprehension." arXiv preprint arXiv:1611.01603 (2016)

[2] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Proceedings of the Conference on Advances in Neural Information Processing Systems, 2015

[3] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[4] Wei, Jason, and Kai Zou. "Eda: Easy data augmentation techniques for boosting performance on text classification tasks." arXiv preprint arXiv:1901.11196 (2019).

A Appendix (optional)

Contributions: In short we all contributed equally to the project. Alaskar focused on developing the conditional layers although David and Daniel assisted with in group coding efforts. David and Daniel worked on the character embeddings, tuning and data augmentation layers also with the support of Alaskar. We all contributed to writing our respective sections on the write-up and poster.