Learning Reading Comprehension with Neural Nets

Li Cai
Stanford University
licai0@stanford.edu

Jason Huang
Stanford University
jhuang99@stanford.edu

Charles Huyi
Stanford University
chuyi@stanford.edu

Abstract

We explored use of Neural Network architectures to learn question-answering over the **Stanford Question Answering Dataset (SQuAD)**. Question-answering is relevant to many NLP applications ranging from coreference resolution to virtual assistants. SQuAD is among the largest reading-comprehension datasets of its kind and is well-suited for the training of machine comprehension models. Our model is based on a BiLSTM architecture that draws from recent papers and incorporates a wide variety of optimizations at each layer of the network to predict an answer span within a context paragraph.

1 Introduction

SQuAD is one of the largest reading comprehension datasets of its kind. With *Contexts* paragraphs sourced from Wikipedia, it contains over 100,000 question-answer pairs on over 500 articles. Given the power of Neural Networks in tackling a wide variety of tasks with minimal feature engineering, we set out to design and build a Neural Network architecture that predicts answer spans within *Context* paragraphs for a given question about that paragraph.

1.1 Preprocessing

We stood on the shoulders of past work. To encode the over 2 million words that appear in *Contexts*, *Questions*, and *Answers*, we used Pennington, Socher, and Manning's Global Vectors for Word Representation (or GloVE vectors). Our *Context*, *Question*, and *Answer* data was fed into the model in the form of lists of GloVE vectors. These GloVE vectors were not trained as part of our network because we believed that this was a good standard and did not want word representations to be affected by the particulars of our training data.

To build our model, we relied heavily on Google's Tensorflow library. Our Neural Network architecture leveraged many of Tensorflow's built-in cell types and helpers to handle dropout or compute loss.

1.2 Problem Statement

Our objective was to predict the correct span of words within a *Context* paragraph given a *Question* about that paragraph. For training, we fed our network a corresponding set of human-labeled *Answer* spans. We evaluated our performance using standard measures of F1 and EM scores that account for both precision and recall.

2 Model

2.1 High-Level Architecture

Our architecture was inspired by a variety of recent work, though we modeled our design most closely off of *Wang, Mi, et al. (2016)*. Wang et al. (2016) At a high level, we run a BiLSTM over the *Context* and the *Question*, use their outputs to compute an attention vector, which we then use to weight the *Context* outputs. The resulting object is the pre-knowledge representation that incorporates both the *Context* and *Question*, as well as their interaction. We feed this object into a third BiLSTM, from which we take the end hidden states to get our knowledge presentation. This concludes our Encoding process. The knowledge representation is then passed to a Decoder, which applies two separate linear layers (both with softmax transformations) to predict probability distributions over the *Context* for the start and end of the *Answer* span.

The flow of data in our model can be roughly broken up into three parts: Encoding the question, Decoding the knowledge representing, and interpreting the decoded result.

2.2 Encoding

2.2.1 Using BiLSTMs

We chose to use bidirectional recurrent networks because these networks are effective at capturing a large amount of input data. Since information flows in both directions, we get, in addition, the advantage that information presented later in a *Context* paragraph is combined with earlier information. In this sense, we relied on our BiLSTMs to summarize information intelligently.

In our implementation, we decided first to run a BiLSTM over the *Question*. Ignoring the outputs from this network, we took the end hidden states, in the forward and backward directions, and used them to initialize the forward and backward hidden states of a second BiLSTM we ran over the *Context*. The intent was that we wanted to seed the representation of the *Context* paragraph with information from the *Question*.

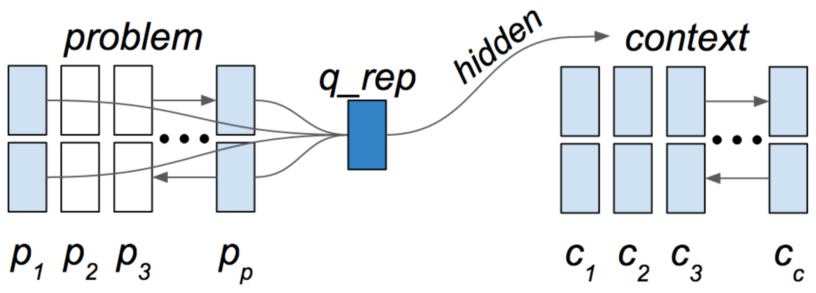

Figure 1: Interaction between Question and Context

2.2.2 Attention

While seeding the second BiLSTM (over the *Context*) does allow for some interaction effects, the type of interaction we wanted was for the *Question* to inform which words in the *Context* mattered more and which mattered less. To this end, we concatenated the end hidden states of the *Question*, our ”question representation,” and took an element-wise product between this representation and each output state of our BiLSTM over the *Context*. We further transformed this product via a sum to produce a single scalar for each word in the *Context*. This scalar can be thought of as an attention weight.

We used the attention weight to scale each output vector in the BiLSTM over the *Context*. Finally, we fed the resulting set of vectors into a third BiLSTM to allow for additional synthesis, and formed our knowledge representation by concatenating the end hidden states of this BiLSTM. This concluded our Encoding process.

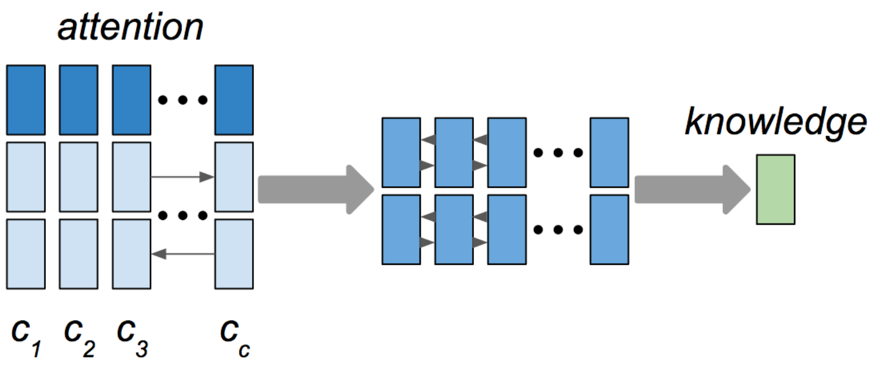

Figure 2: Attention and Derivation of Knowledge Representation

2.3 Decoding

The knowledge representation, which combines the *Context*, *Question*, and the interaction between the two, served as the input to our Decoder.

In the Decoder, we run two separate linear layers, each with its own weight matrix and a bias term, and apply a softmax at the end, to product two probability distributions. The length of the probability distributions is the length of the maximum *Context* length we wanted to account for. (Based on some quick analysis of the *Context* dataset, we found that *Contexts* of 400 were quite rare – less than 0.1 percent of the data – and so we chose to ignore *Contexts* longer than 400 by simply capping our probability distributions at this length.

These probability distributions are returned by the Decoder to our model.

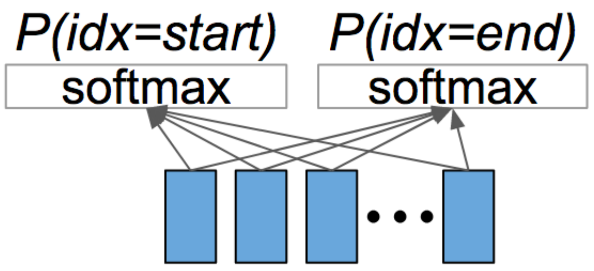

Figure 3: Decoding the Knowledge Representation

2.4 Interpreting the Decoder's Output

The first probability distribution represents the likelihood that a particular word in the *Context* is the start index. The second distribution does the same for the the end index. Our model then simply takes the indices of the largest element in the first and second probability distributions and uses those as the predicted start and end span.

One extension we would have liked to look into was to follow the work of *Wang, Mi, et al. (2016)* and instead teach our model to predict a joint probability over both the start and end indices. Intuitively, it is reasonable to assume that the probability distribution over the span's start would depend in part on the prediction for the span's end. However, in the linear layers in our Decoder, we compute the probability distributions with the assumptions that the probabilities are independent of one another.

2.5 Loss

As our formulation of loss, we use sparse softmax cross-entropy loss, which conveniently is provided as a helper function in Tensorflow. In our case, the correct label is a one-hot vector (with a one label at the correct start / end index), and our predictions are probability distributions. By penalizing the probabilities that we didn't assign to the correct index, cross-entropy loss fit well with what we wanted to penalize.

2.6 Dropout and Regularization

While our model easily overfit training data, it was important that we guard against overfitting when trying to improve our F1 and EM scores on unseen test data. Two techniques we used to this end were: applying dropout and adding regularization loss to our previously computed loss. Dropout allows us to disable neurons during learning, which helps what is learned generalize better. We also added a regularization loss to our calculated loss in order to penalize parameters that are large. Both dropout and adding regularization loss are forms of regularization in general.

2.7 Hyperparameter Tuning

Finally, we played around with various combinations of hyperparameters. It was often diffcult to attribute particular increases to certain values, since there were often multiple hyperparameters to tune, and simply finding an optimal value for one hyperparameter while holding everything else constant did not necessarily mean that was *always* the best value for that parameter. In the addition, the long training times for the model, combined with the fact that (toward the end) we did not have as much time to tune, meant that we often made calls about hyperparameter values after only a handful of epochs.

3 Experiment

3.1 Performance and Loss

At the time of this writing, we're continuing to train our latest model. We've observed in our penultimate model that validation F1 and EM had consistently gone up, although a slowdown started around epoch 50. This resulted in our tweaking a few hyperparameters further and modifying the size of our Decoder's predicted probability distributions. This new model is currently running on Azure, with Codalab results to be posted tomorrow morning.

Below, we share the performance of the current state of our latest model, even as it will still undergo additional hours of training (overnight).

First, we observed performance according to our key metrics of F1 and EM.

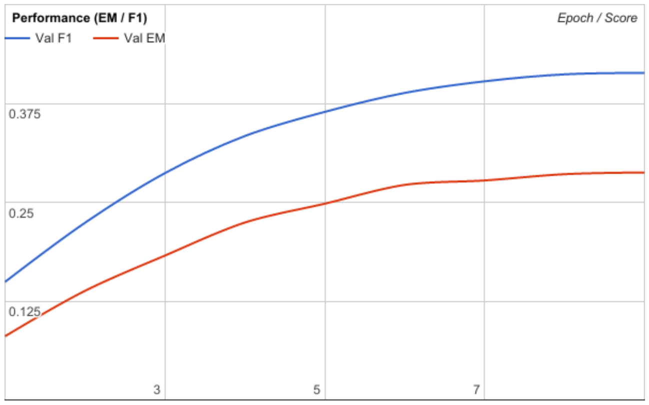

Figure 4: Model Validation Performance

We saw that the validation F1 has reached 0.4 and is showing some signs of slowing down. The training F1, however, has reached over 0.7, suggesting that we might be overfitting. With additional time to tune hyperparameters, we could try further increasing our regularization loss constants.

We also examined the loss over time.

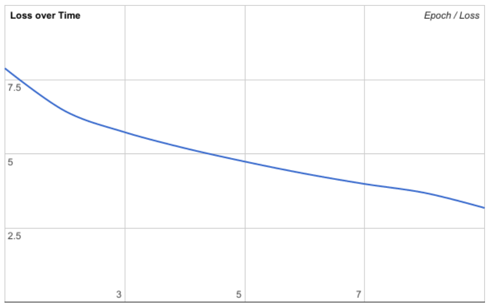

Figure 5: Model Loss

Here we see that loss is steadily decreasing, and as of the time of this writing, the loss has dipped below 3.

Looking more closely at the predicted spans that our model gave, we notice that our model tended to predict answer spans that closely matched the correct answer span but did not align perfectly. The close matching may be something that was made possible by attention. On the other hand, the misalignment was often due to relatively high probabilities appearing around the correct answer span, with the maximum index not always being at the start. This may be more of a model-design issue, where we could have perhaps made our predictions less fragile by designing our model to predict which words are contained in the span and which words are not, as opposed to simply selecting a maximum from a probability distribution.

Here is one example of our predicting an answer that covers more than the true answer

- ground truth: friction
- prediction: friction include other contact forces , tension , compression , and drag

This hurts not only our EM score but also F1 since we have low precision.

In other instance, our prediction did not cover enough of the true answer.

- ground truth: forces as being due to gradient of potentials
- prediction: gradient of potentials

For short answers, however, our model seems to perform well consistently.

1. ground truth: gravitational force
2. prediction: gravitational force

This is likely due to the fact that there are fewer candidates from which to pick a max, so the design of our model more closely resembles the alternate design in which the Decoder predicts which words are part of the span and which are not.

4 Related Literature

The initial attempts at tackling the question answering tasks involved rule-based approaches and prediction models on engineered features. The first paper that presents the SQuAD data, Rajpurkar et al. (2016), create chunks of potential answers and rank them. They they ranked them using Logistic regressions on some hand crafted features. But this ranking method resulted in 20% of the chunks that did not contain the correct answer. Yu et al. (2016) used part-of-speech pattern to build the candidate list, which increased recall. They then ranked the answers using a attention-based RNN. Lee et al. (2016) first listed all possible answers up to length 30. They then used a multi-layered BiLSTM to learn the a fixed length representation for all those possible answers. They then ranked the answers based on those representations. They use this fixed width presentation to overcome the complexity of enumerating all possible continuous sub-strings, a neural architecture they call RaSoR.

Instead of extracting candidate answers, the literature moved onto a predicting the answer span. Wang and Jiang (2016) put forth a match-LSTM model to match the passage with the question. They used Pointer Network developed in Oriol Vinyals and Jaitly (2015) to select the start and end of the answer span. Caiming Xiong and Socher (2016) proposed the Dynamic Coattention Network. The model first used a co-attentive encoder to model the interaction between questions and context. It then used a dynamic pointing decoder to predict the start and the end of the answer. Seo et al. (2016) build on the Caiming Xiong and Socher (2016) by using a bi-directional attention network to create a question-aware context representations. They then used this hidden state representation to predict the start and end of the answer. Yang et al. (2016) proposed a fine-grained gating mechanism to dynamically create a hybrid word representation that combined word-level and character-level representations based on the word properties.

5 Conclusion

We are impressed by the ability of neural network architectures to learn by backpropagation alone — without being fed handcrafted features. In building our model, a primary challenge was how to combine various structures (RNNs) and optimizations (attention, dropout, et al.) in an effective way. While our model was inspired by existing work, we wanted to contribute original structures and designs. Toward this end, we often went back and forth on decisions such as how to mix and match various components, or to tune hyperparameters. Often these decisions were made on the basis of observed performance over a small amount of data, but, frequently, performance was difficult to attribute to a particular decision given the large number of interaction effects. If given additional time to extend this work, we would focus our efforts on being more methodical and rigorous in testing: selectively removing parts and carefully measuring its impact on model performance.

6 Contributions

We have submitted descriptions of each of our contributions via the provided Google Form. In general, each of us contributed significantly and equally to the final product, and enjoyed working together as a team!

References

Caiming Xiong, V. Z. and Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.

Lee, K., Salant, S., Kwiatkowksi, T., Parikh, A., Das, D., and Berant, J. (2016). Learning recurent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*.

Oriol Vinyals, M. F. and Jaitly, N. (2015). Pointer networks. *Advances in Neural Information Processing Systems*, page 2692–2700.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2016). Bi-directional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Wang, S. and Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*.

Wang, Z., Haitao Mi, W. H., and Florian, R. (2016). Multi-Perspective Context Matching for Machine Comprehension. *Quarterly Journal of Economics*, page 488–500.

Yang, Z., Dhingra, B., Yuan, Y., Hu, J., Cohen, W. W., and Salakhutdinov, R. (2016). Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*.

Yu, Y., Zhang, W., Hasan, K., Yu, M., Xiang, B., and Zhou, B. (2016). End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*.

Acknowledgments

Thank you to the countless hours that the CS224N staff has put into this class. We appreciate your work!