Question Answering

CS 22N Default Final Project

Omar Sow
Department of Computer Science
Stanford University
osow@stanford.edu

Abstract

For the CS224N Default Project, I used the provided starter code, but made three model changes: implementing a coattention layer, character-level encodings with a convolutional neural network, and restricted span selection. After refining hyperparameters, with 40k iterations, the model reached performance of F1: 72.9 and EM: 61.52 on the test set.

1 Problem Definition

As stated in the instructions, "the goal of this project is to produce a reading comprehension system that works well on SQuAD." [1]. The problem of question-answering, in this framework, refers to taking a 'context' $c = \{c_1, c_2, \ldots c_n\}$ and a query, $q = \{q_1, q_2, \ldots q_m\}$, which can be answered with some span of text $[c_i, \ldots c_j]$ from the context. Given training data comprised of contexts, corresponding queries, and the gold-standard span, the model must learn how best to map from an input of a query and a context, to a span pointing to a subset of the context.

The data provided in training had the following distribution of query/context length, and word length, all of which fed into hyperparameter decisions for this model.

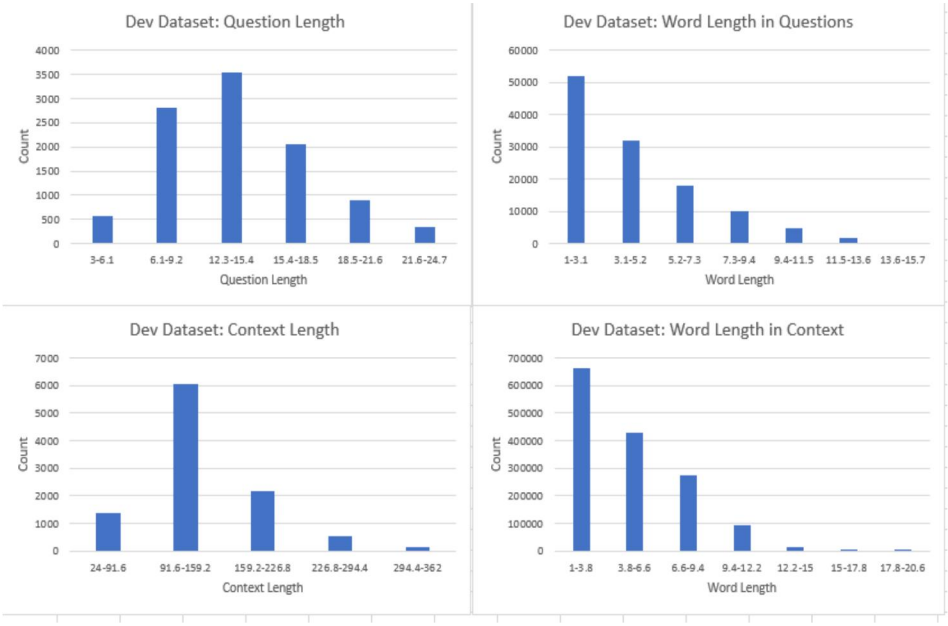

Figure 1: Histogram of characteristics of dev set.

2 Model

2.1 Architecture

The baseline provided as starter code consisted of three modules: an encoder layer, an attention layer, and an output layer. My architectural modifications were made in the encoding and attention layers, with slight change to the predictions made at test time, given the results of the output layer.

2.1.1 Encoding Layer

Baseline: The model, in its encoding layer, takes embeddings for the words in the context and query, and passes them through a shared bi-directional GRU, whose hidden states provide the context hidden states and the query hidden states for the next step. The word embeddings of the words in the context and query, each of which was in R^d, come from the set of pretrained GloVe embeddings.

Implementation: In order to improve the model's capacity to generalize and handle words outside of its vocabulary, I expanded this layer to include creation of character-level embeddings.

This was accomplished through creating a trainable character embedding matrix. With extensions to the data-processing parts of the program, this layer could take character ID's (their indices in an alphabet & special characters list) and retrieve their embeddings $e_1, \dots, e_L$. Then, these embeddings could be passed into a one-dimensional convolutional neural network, whose hidden states are max pooled, bringing them to the same dimension as word embeddings, R^d[2]. Like a CNN passing over an image, this CNN takes a pseudo window size, referring to number of words that filter through at each step.

The two embeddings could then be concatenated, generating for each word a new embedding, $w \in R^{2d}$ accounting for both word and character-level properties [2].

2.1.2 Attention Layer

Baseline: The attention layer took word embeddings from the previous layer and used basic unidirectional attention to make the context hidden states attend to the question hidden states.

Implementation: To improve the baseline, I implemented a coattention layer. This involves first applying a nonlinearity to the question hidden states, generating $q'_j = \tanh(Wq_j + b) \in R^{2d}$ where W and b are trainable parameters and where $W \in R^{num-qs * num-qs}$. We then add sentinel vectors to the hidden states of questions and contexts, as trainable parameters that allow the model to choose to ignore the input completely. With an affinity matrix, where entry (i,j) compares context i and question j, we calculate a context-to-question and question-to-context attention distributions, as done in source [3].

2.1.3 Output Layer

Since the baseline model simply took the argmax over the distributions of the start of the predicted span, and the end, and did so independently, it created situations where the predicted end preceded the predicted end. To avoid this, I modified the output layer to maximize the product of the start and end probabilities, constraining the end to come at a word that is equal to or after the start [4]. This was implemented as follows.

Take $p_{start}p_{end}^T$ to obtain a matrix S such that $S_{ij} = p_{start}(i) * p_{end}(j)$. Then, considering only the upper triangle of S, meant considering only the options where j > i, so I took the argmax of this upper triangle.

2.2 Training

2.2.1 Initial Analysis

The different models initially reached the following preliminary results after 2500 steps (a smaller number, in order to allow more time for hyperparameter testing, but enough to demonstrate the effective nature of the model changes).

The scores listed here are from the development set used throughout the training process (unofficial_eval). When I reach the hyperparameter tuning step, I employ the official_eval metric.

Model Version	Unofficial Eval Dev F1 / EM
Baseline	0.43/0.34
+Coattention	0.58 / 0.40
+Character-Level Encoding	0.61 / 0.42
+Smart spanning	**0.63 / 0.44**

In further examining these outputs, I looked at the attention distributions that resulted from the coattention layer, and the example sentences.

Take, for example, this query-context pair, which we'll consider throughout this paper:

Question: what one word did the nfl commissioner use to describe what super bowl 50 was intended to be ?

Context: in early 2012 , nfl commissioner roger goodell stated that the league planned to make the 50th super bowl "spectacular" and that it would be "an important game for us as a league "

Correct & Predicted Answer: spectacular

This, and examples taken at this step in training indicated that the problem of illogical spanning had been solved. Furthermore, over no runs did I find any larger issue in the span choice (e.g. it was too long, or spanned across sentences), so I did no further explore ways of limiting the spanning.

Furthermore, analysis of the attention distribution shows a reasonably intepretable distribution. In the context-to-question distribution, we see that, for example, the beginning of the commisioner's

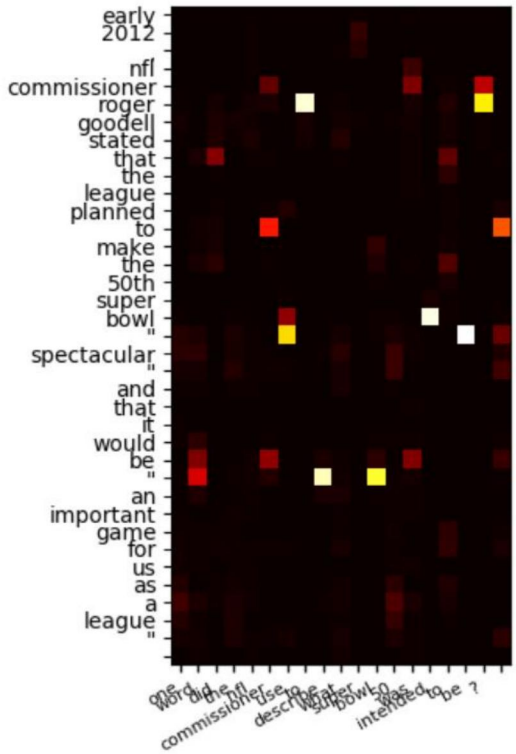

Figure 2: Question-to-context attention distribution.

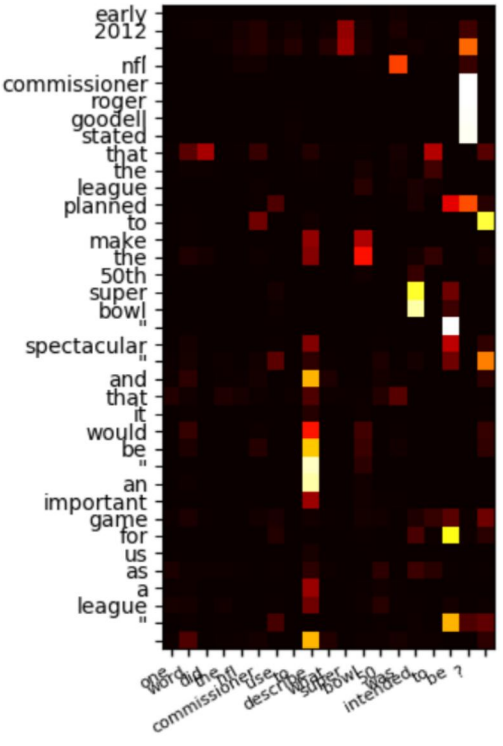

Figure 3: Context-to-question attention distribution.

quotations pay specific attention to the part of the question mentioning "describe". Likewise, we see that in the question-to-context distribution, "describe" and "superbowl 50" pay attention to the quotation mark before "spectacular". However, figure 3 shows a concerning amount of attention placed on the second quotation "an important game…", which shows the model is still having trouble distinguishing the two. It's attention is also still fairly spread out, and seems almost indecisive.

However, analysis of the model at this point indicates overfitting to the training data as well, which is sure to mean an even lower final test score.

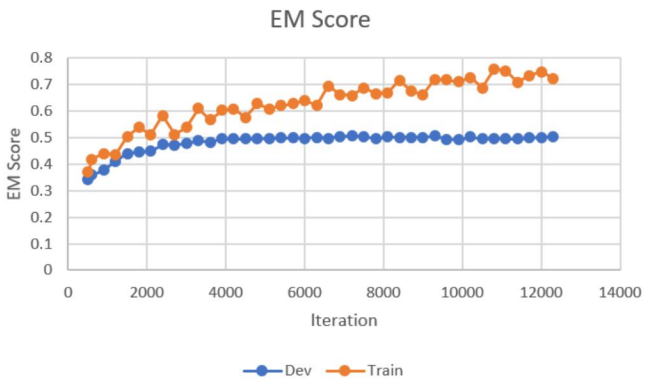

Figure 4: EM score of new architecture.

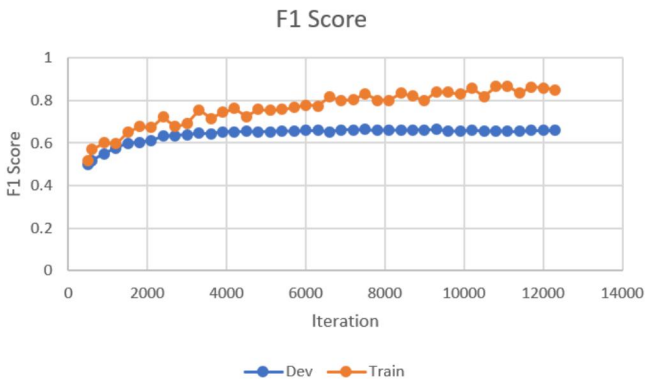

Figure 5: F1 Score of new architecture.

Two facts could be assumed from these charts. First, the fact that the training scores approach high values indicates the model is at least competent enough to represent the data. However, it seems to be overfitting the training set. Taking these pieces of information, I proceeded to focus on tuning the model.

2.2.2 Hyperparameter Experimentation

I will make a note here that these steps are presented in chronological order in which I performed them. In retrospect, there are certainly ways I would shuffle this around to a more systemic search of relevant parameters (see Limitations).

Word Length

A hyperparameter of the character encoding is word length. Due to time constraints, I did not seriously explore this parameter. I used information surmised from the Figure 1, and set the word length parameter to 15.

Model Size

The baseline model included 200 hidden layers. I experimented initially with a hidden layer size of 400, but this was too big, given computational capacity, so experimented briefly with increasing depts to 300. However, both since the results after 5000 iterations was relatively small, SOURCE WHAT SOURCE used hidden layer size 200, and principally to maintain computational speed, in order to explore hyperparameters more fully, I opted to keep hidden size at 200.

Hidden Layer Size	Unofficial Eval Dev F1 / EM after 5k iterations
200	0.6534 / 0.5016
300	0.6588 / 0.5021

Reducing Overfit

The baseline model had a dropout rate of 0.15 and had no regularization, I employed a grid search approach to find optimal parameters for dropout and L2 parametrization. Each combination ran for 14k iterations and was tested on official evaluation mode.

Official Eval F1 Scores:

Dropout	L2 Reg. Coefficient	0	0.0001	0.0003
0.15		66.1	71.6	72.1
0.2		69.9	69.9	71.9
0.25		71.0	**73.1**	71.7

Official Eval EM Scores:

Dropout	L2 Reg. Coefficient	0	0.0001	0.0003
0.15		57.6	61.2	60.5
0.2		57.9	57.9	59.4
0.25		59.97	61.5	60.2

A visual examination of the F1 and EM score across training iterations (with scores calculated throughout training, instead of on official_eval mode) reinforces that these values largely approach the same scores, oscillating around, though they provide an increase over the non-regularized scores.

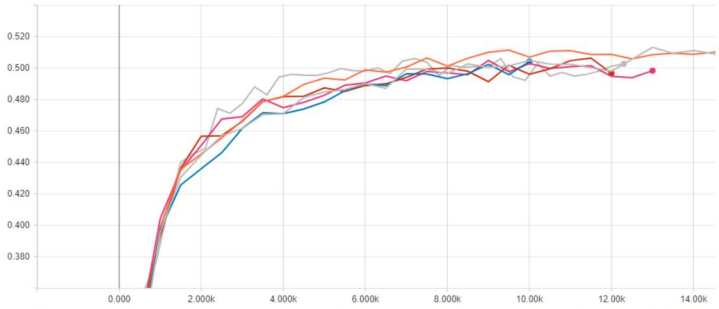

Figure 6: EM scores on unofficial_eval for all regularization modes over 14k iterations.

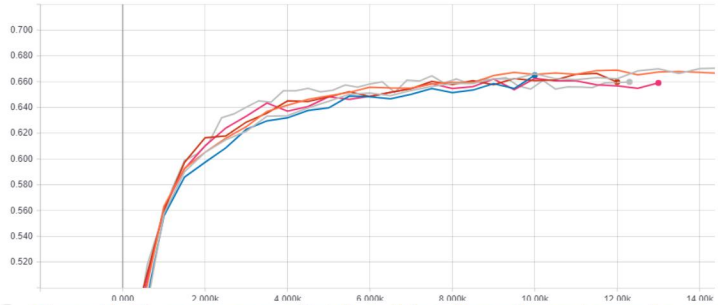

Figure 7: F1 scores on unofficial_eval for all regularization modes over 14k iterations

CNN Kernel Size

The handout stated that a value of k = 5 worked adequately well. In order to confirm, ran experiments to look for any noticeable improvement from changing this setting using the unofficial_eval dev set, and the best performing hyperparameters. Again, a judgement call given the computational and time limits of the project was to move forward with a value of k = 5.

Kernel Size	Unofficial Eval Dev F1 / EM after 15k iterations
4	0.66/0.51
5	0.67/0.517
6	0.66/0.514

Again, these scores relate to performance on the development set, while the final scores on the test set obviously varied. However, it highlights the importance of increasing the attempts to prevent overfitting, with final results more visible in the test set scores.

Unmodified Values

The following hyperparameters were not edited from the baseline:

Learning rate: 0.001 **Question max length: 30**

Embedding size: 100 **Gradient max: 5**

From analysis of figure 1, I reduced **context max length** to **350** words to speed up training, while not sacrificing performance.

Conclusions

It seemed that the strongest improvements to baseline hyperparameters came from attempts to control for overfitting. This makes sense since the architectural changes create a much more complex model, which performed well on the training data, but initially had trouble generalizing that knowledge to the dev set.

3 Results

These are the parameters I was inclined to submit as my final model based off these experiments. However, I first performed the same analysis on examples (here we have the same NFL query-context pair from earlier), to confirm this was improving correctly.

3.1 Parameters of final model

Word Length = 15 **Regularization**: L2 with 0.0001 coefficient

Dropout: 0.25% **Number of Hidden Layers** = 200

3.2 Qualitative

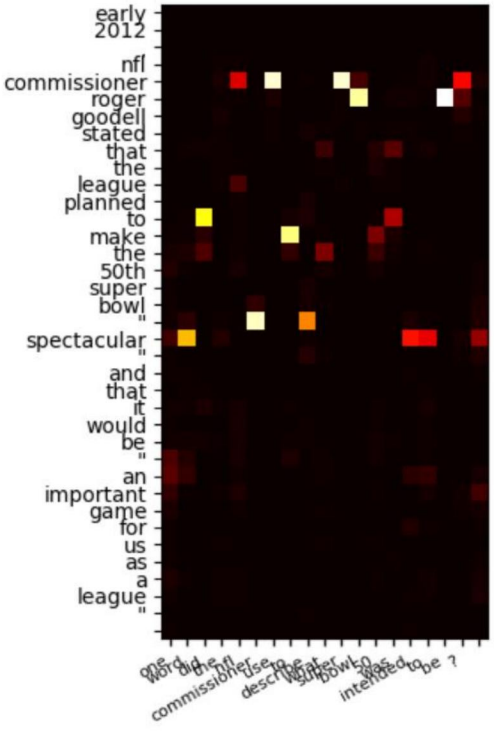

Figure 8: Question-to-context attention distribution.

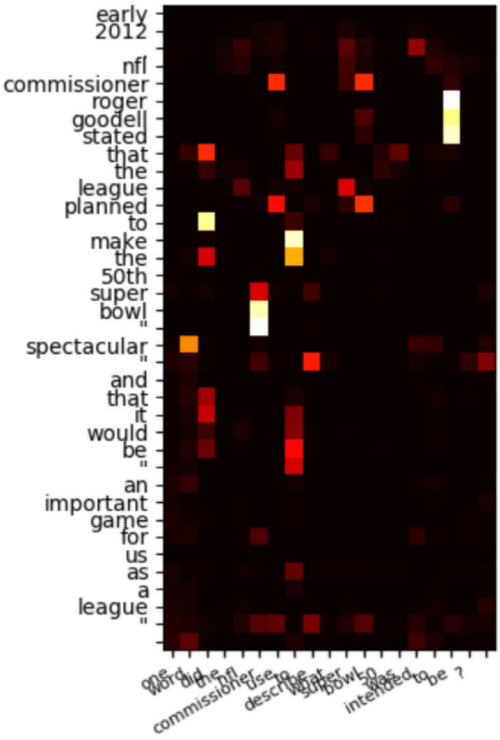

Figure 9: Context-to-question attention distribution

As we see in the attention distributions generated in figures 8 and 9, we notice that the query-to-context attention is much more focused around “nfl commissioner” and “spectacular”, which makes intuitive sense, the key parts of the question are paying attention to the answer part of the context.

Likewise, the context-to-question attention distribution is far less spread out, showing a similar pattern as the question-to-context distribution, which is perhaps more impressive given the potential for spread seen in the attention distributions earlier on. The part of the context “commissioner roger” pays attention to the end of the sentence, but the words related to the “50th superbowl” payed attention to the portion of the query with the key structural information (i.e. that it is a “what” question”).

3.3 Quantitative

The final model, trained for 40k iterations, had a test set score of **F1: 72.9 and EM: 61.52**.

Quantitatively, this indicates significant improvement on the baseline. It also highlights the relative weakness of the program remains in the exact match score.

This would potentially respond to further improvements in the output layer beyond the bound I introduced in my modifications.

4 Limitations

The first limitation of this examination of the question answering problem certainly remains the computational time of hyperparameter searching. With more time, I would want to further explore the following hyper-parameters:

- Size of character embeddings
- Parameter sharing across the model
- Further fine-grained experimenting with dropout/regularization combinations
- Further analysis of benefits of increasing model depth beyond cursory results here
- The organization of various hyperparameter tests, in order to test more combinations of different values, beyond just dropout/regularization.

References

[1] Abi See, Richard Socer, et al. CS224N Default Final Project: Question Answering. http://web.stanford.edu/class/cs224n/default_project/default_project_v2.pdf

[2] D. Britz, "Understanding Convolutional Neural Networks for NLP", WildML, 2018. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

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

[4] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016

[5] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051,2017.