Exploration and Analysis of Three Neural Network Models for Question Answering

Ziran Jiang
CodaLab username: willerutan
CS224n Assignment 4
Stanford University
Stanford, CA 94305
ziranj@stanford.edu

Abstract

Working by myself in the short time span of the assignment 4, three relatively simple models for question answering are implemented and their performances are compared. The models explored are: 1) Baseline model in the PA4Presentation.pdf [1], 2) The encoder from Xiong et al. [2] plus a simple decoder, and 3) The baseline model suggested on the Piazza forum [3]. These models are referred to as model 1, 2, 3 respectively in this paper. Some hyper-parameters such as dropout rate and learning rate are also explored to optimize the training. Model 3 with dropout rate of 0.15 achieves the best performance compared to the other two models, with F1/EM scores of 22.6%/14.5% on the test set of the Stanford question answering dataset.

1 Introduction

For this question answering task, the Stanford Question Answering dataset (SQuAD) was used to train the model. SQuAD contains approximately 80,000 context/question/answer pairs for training, and the answers are in terms of a span in the context, which is described by two integers: a_s for the start index of the answer in the context, and a_e for the end index of the answer.

I worked on the assignment 4 by myself, partly because I'm an SCPD student and it's hard to go on campus, and also because the previous three assignments had a reasonable amount of work for one person, and I hoped to learn all aspects of the assignment by doing it myself. I approached the task by first studying the starter code and implementing the model 1 which is the very basic model. However I soon realized that the task was non-trivial. Although the starter code had many helper functions, a significant amount of basic infrastructure code for training, model and answering had to be implemented first in order for the entire flow to work. Being new to TensorFlow, it took a long time just to study the starter code and understand what each part was doing. The Piazza forum was very helpful and I had many of my questions answered, but the number of questions/bugs I had was just too many and I made very slow progress. While I've put my maximum effort in this assignment and I sure learned a great deal of things, by the time when I had the basic infrastructure code and the model 1 completed and finished CodaLab submission issue debug, I only had three days left till the deadline. So please read the rest of the paper with this in mind.

2 Related Work

The model 1 presented by [1] is a simple baseline model that encodes the context and question using BiLSTM and one option is to take only the end states from the BiLSTM and

assume that has all the information from the context and question. The end states are then fed into separate linear functions for decoding the a_s and a_e.

In the model from Xiong et al. [2], it uses mainly LSTM to encode the context and question, and then calculates the coattention context by getting the interactions between the two. For decoder it uses a dynamic pointing decoder which alternates the estimation of the start and end positions in order to get out of local maxima.

The model from [3] is a somewhat simplified version of [2]. It uses LSTM to encode the question and context, and then calculates the interaction between the two which is represented by the affinity matrix. It then predicts a_s and a_e by multiplying with a single weight vector.

3 Approach and Experiments

Models 1, 2, and 3 were implemented in that order. The following describes the implementation details as well as the testing, debug and optimization experiments that I ran during the implementation.

3.1 Model 1

Model 1 is a very basic model implemented based on [1]. The implementation steps are described below:

1) Context -> BiLSTM -> concatenate the two end hidden states to make h_c
2) Question -> BiLSTM -> concatenate the two end hidden states to make h_q
3) a_s = W_s * h_c + W_s * h_q + b_s
4) a_e = W_e * h_c + W_e * h_q + b_e
5) Use softmax cross entropy loss for training

The model was trained for about 100 batches (batch size = 100), and as expected for a basic model, it got stuck at the validation F1 score of about 7% and did not learn any further. So the training was terminated. The purpose of this was to build a basic model to make sure all the training code worked, so that goal was achieved.

3.2 Model2

The model 2 was based on Xiong et al. [2]. This model was chosen because of its recent high score on SQuAD, as well as the interesting coattention encoder. The encoder was implemented as follows:

1) Context -> LSTM -> take the output from all the words as D
2) Question -> same LSTM as 1) -> take the output from all the words as Q'
3) Q = tanh(W_q * Q' + b_q)
4) L = D_transpose * Q
5) A_q = softmax(L)
6) A_d = softmax(L_transpose)

7) C_q = D * A_q

8) C_d = [Q; C_q] * A_d

9) C_d -> BiLSTM -> take the output, which is the final encoded representation

Next I attempted to implement the decoder described in [2], however at this point the remaining time for the assignment was running short and it was not likely I could implement the same encoder in [2]. So I decided to implement a simple feed forward de-coder instead, described below:

10) h = relu(encoded * W + b1)

11) h_drop = dropout(h, dropout rate)

12) a_s = h_drop * U + b2

13) repeat steps 10 to12 with different weights to get a_e

This model was trained for 5 epochs, however the average validation F1 score remained about 15% and did not grow anymore. So the training was terminated.

3.2.1 Overfitting test with model 2

In order to understand why the model 2 had limited training improvement, the overfitting test was done. In this test, a batch if 100 training samples is taken, and the model was trained with the same batch repeatedly. After each iteration, the F1/EM scores were taken with the same training batch. This is to test the basic capability of a model to overfit a small sample of training data.

When this test was run on model 2, it gradually learned and there was a steady increase of F1/EM scores over each iteration. However the learning eventually flattened and after about 60 iterations the F1/EM scores reached the peak of 70%/57% and did not reach any higher. This result has revealed a fundamental flaw in model 2, that it cannot overfit a small training sample very well.

3.3 Model 3

Since model 2 did not work well and I was running out of time for assignment 4, I decided to try another baseline model released on Piazza [3] and this is the model 3. The implementation is:

1) Context -> LSTM -> take the output at all words as C

2) Question -> LSTM -> take the output at all words as Q

3) A = softmax(C * Q_transpose)

4) C_p = A * Q

5) C = concat(C_p, C) * W + b

6) C_drop = dropout(C, dropout rate), this is the final encoded notation

7) a_s = C_drop * W1

8) a_e = C_drop * W2

3.3.1 Overfitting test with model 3

Before training the model, the same overfitting test was done on the model to make sure it is capable of overfitting a small training sample. The model learned rapidly and at iteration 18 it achieved the training F1/EM score of 99%/100%.

3.3.2 Learning rate optimization

In order to compare the effects of different learning rates, the learning rate was varied and the model was trained for several tens of batches to observe the change in loss in each batch. Figure 1 shows the change in loss for learning rate of 0.001, 0.05, 0.01 and 0.03. From this result, the learning rate of 0.005 was picked. Due to the time constraint of the assignment, I did not get a chance to look into exponentially decaying learning rate, however at least I tried to pick the best constant learning rate through this comparison.

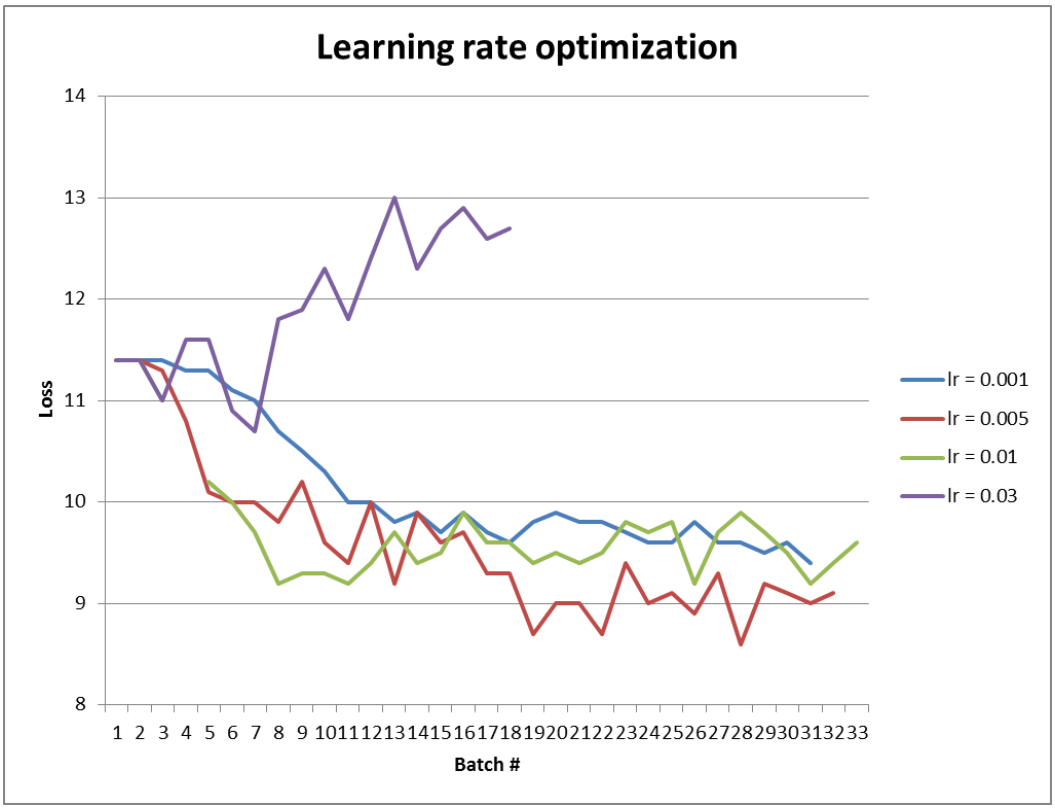

Figure 1: Sample Figure Caption

3.3.3 Training model 3 with different dropout rates

Another major hyperparameter to consider was the dropout rate. To compare the performance of the model with different dropout rate, two versions of the model 3 were implemented: model 3a) using the dropout rate of 0.15, and model 3b) using the dropout rate of 0.5. The two models were trained for a full 10 epochs, and Table 2 shows the result. Model 3a performed slightly better, meaning the dropout rate of 0.15 was better for this

particular model.

Model	Test set F1	Test set EM
3a: dropout = 0.15	22.6	14.5
3b: dropout = 0.5	21.2	13.1

Table 2: SQuAD test score for models 3a and 3b

There are other hyperparameters and functions that are important for improving the model's performance, including max gradient norm clipping, varying hidden state size, embedding size and optimizer type. Unfortunately due to the time constraint of this assignment I was not able to explore or implement these, but for future works these are interesting areas to look into. In this model the embedding size of 300 was used because generally higher embedding size seems to get better results.

3.3.4 Comparing the training vs validation score

During the training, training and validation scores were obtained regularly. Table 3 shows the comparison for the training and validation f1 scores for one random batch near the end of epoch 10. For both model 3a and 3b, there is massive difference in training vs validation score. The models were able to fit the training samples very well as result of training, but they performed poorly on samples they saw for the first time. This indicates that overfitting was still happening in spite of the increased dropout rate in model 3b.

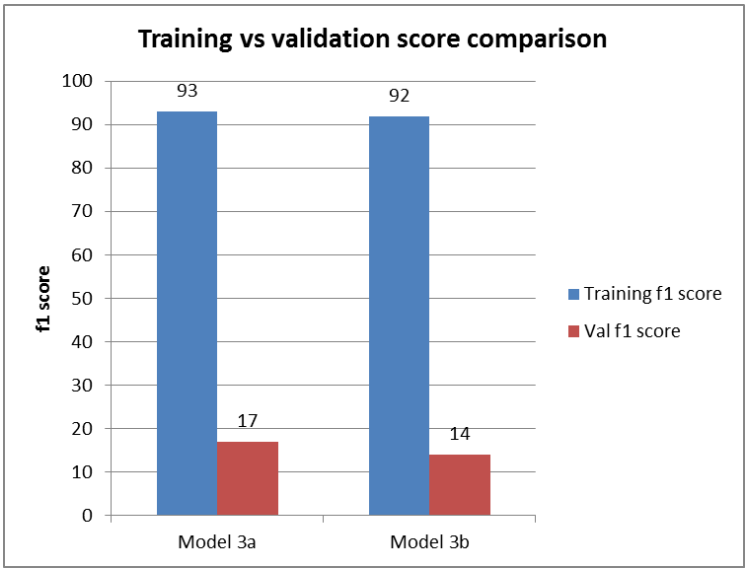

Table 3: Comparing training score and validation score

3.3.4 Analyzing the correct and incorrect predictions

For model 3a, a sample of validation samples were taken, and the question type, mode's prediction and the true answer were analyzed. Table 4 summarizes the type of question and the total F1 score for each question type. Overwhelmingly the "what" and "when" question types had the most correct predictions. This is likely due to the fact that these questions types

require a specific object or time as an answer and it's easy for the model to look for these contents in the context paragraph. On the other hand, the "why" and "how" question types received the least score. This is likely because these question types have more vague open answers, and they largely depend on the context sentence structure as well, so it is harder for the model to correctly predict for these question types.

Question type	F1 score total
What	12
When	6
Who	2
Where	1
Why	1
How	0

Table 4: F1 score per question type

Sample question/prediction/true answer pairs:

When:

Q: When was the Catered Affair released?

Pred: 1956

True: 1956

Who:

Q: Who designed the pyramid of Djoser?

Pred: Imhotep

True: Imhotep

What:

Q: What was Greece's jobless rate in 2015?

Pred: 24 per cent

True: 24 per cent

4 Conclusion and future works

As shown in Table 5, model 3 with dropout rate of 0.15 achieved the highest score among the three models I was able to try. Despite my very best effort, the resulting test score was somewhat low. However during the course of the assignment I really gained a solid hands-on experience of TensorFlow and also learned that it is not easy to implement a good performing model from the beginning.

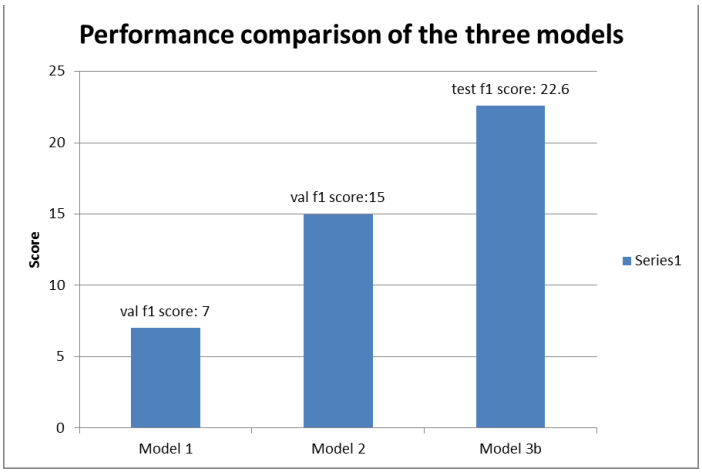

Table 5: Performance comparison of the three models

For future works I would like to explore these aspects more:

- Look into why model 2 was not able to overfit a small training sample
- Try solving the overfitting problem for model 3
- Explore with more hyperparameters and implement max gradient norm and learning rate annealing
- Consider what type of models can better predict qualitative questions such as "why" and "how"

4 References

[1] Ignacio Cases and Allen Nie. Programming Assignment 4: Reading Comprehension. PA4Presentation.pdf, available from CS224n website.

[2] Caiming Xiong, Victor Zhong, Richard Socher (2017) Dynamic Coattention Networks for Question Answering. *ICLR2017*.

[3] The model version 2 from Piazza forum for assignment 4: https://piazza.com/class/iw9g8b9yxp46s8?cid=2844