Dynamic Coattention Networks for Reading Comprehension

Hayk Tepanyan*
Department of Computer Science
Stanford University
Stanford, CA 94305
tehayk@stanford.edu

Abstract

We investigate the reading comprehension task using SQaAD dataset. The reading comprehension task is formulated as finding the answer span in the given paragraph for the given question. We describe multiple approaches to solving this problem starting from very simple baseline that gradually evolves into a much more complicated architecture. For each model we describe the underlying architecture and analyze its performance on the training and validation data. Our final model is a modification of the dynamic Coattention network described in the work of Socher et al. [1] and it achieves $F1 = 58.6\%$ and $EM = 44.6\%$ scores on the test set.

INTRODUCTION

We investigate 3 different models. First is the simplest baseline, that has the smallest architecture required to bind the input to the output. The second model has a encoder described in the [1] and the same decoder as before, i.e. a linear layer. As expected, the results for this model are better then the former one, since it conveys more knowledge and does a much better job encoding the relationship between the question and the paragraph. In the third model, we substitute the linear decoder used in the former model with a modified version of the decoder described in [1]. Again, as expected we observe a performance boost over the the former model as it had limitations due to using linear decoder. We use this final version with Common Crawl 840B glove vector embeddings from [2] to achieve the final scores of $F1 = 58.2\%$ and $EM = 44.5\%$ scores on the dev set.

1 Model 1. Simple Baseline

Below we describe the architecture of the simple baseline model, its performance and analyses on the SQuAD dataset.

1.1 Architecture

The purpose of the basic model was to setup a working encoder-decoder-loss architecture as a foundation for future more complicated models. Here are the steps used to build this model:

- Encoding
  - pad question and paragraph to have fixed lengths max_q and max_p, build masks.
  - encode question using dynamic RNN with BasicLSTM cell, store the last hidden state h_q

*Codalab username: tehayk

Table 1: Performance of all models

	Train Loss(Max → Min)	**Train F1**	**Train EM**	**Val. F1**	**Val. EM**
Model 1.	10.0 → 7.1	14%	8%	4%	2%
Model 2.	9.7 → 3.9	27%	23%	9.5%	5.9%
Model 3.	9.7 → 1.2	84%	65%	61.4%	42.8%

  - encode paragraph using dynamic RNN with BasicLSTM, feed in the the last hidden state h_q as the initial state to build a conditional representation of the paragraph
  - store the last hidden state as the knowledge vector
- Decoding
  - predict the start of the span using a linear layer on top of the encoded knowledge vector
  - predict the end of the span using a linear layer on top of the encoded knowledge vector
- Loss
  - mask the start and end logits using the paragraph mask, not to fit padded values
  - $l_1 =$ cross-entropy-loss $(start_{predicted}, start_{real})$
  - $l_2 =$ cross-entropy-loss $(end_{predicted}, end_{real})$
  - $l = l_1 + l_2$

This working model performs very badly but makes good ground for future improvements since now all we have to do is to substitute the decoder and the encoder with better models.

1.2 Performance and analyses

The performance of this model is very poor. After 3 epochs of learning the training $F1$ and EM do not exceed 14% and 8%, whereas validation $F1$ and EM do not exceed 4% and 2% respectively. More details about the performance of all 3 models can be seen in Table 1.

So the results are slightly better than the random guessing. The reasons for poor performance are the poor encoding and decoding. Only using the last hidden state of question encoding does not capture well the relationship between the question and the paragraph and only using the last hidden state of the paragraph encoding does not convey much information about the question and context. Also the linear decoder is not capable of capturing complex relationships.

2 Model 2. DCN Encoder with linear Decoder

2.1 Architecture

In this model we substitute the simple encoder from Model 1 with the encoder described in [1]. The high level architecture now looks like this:

- Encoding
  - pad question and paragraph to have fixed lengths max_q and max_p, build masks.
  - encode question using dynamic RNN with LSTM cell, store all outputs as question encoding Q
  - encode paragraph using dynamic RNN with LSTM, store all outputs as paragraph encoding D
  - calculate coattention matrices:
$$L = D^T Q, A_Q = softmax(L), A_D = softmax(L^T)$$
$$C^Q = DA^Q, C^D = [Q, C^Q]A^D$$
  - calculate final knowledge matrix U (question-paragraph encoding) using bidirectional RNN with LSTM cells, and taking D and C^D as inputs.

- Decoding
  - predict the start of the span using a linear layer on top of the encoded knowledge vector
  - predict the end of the span using a linear layer on top of the encoded knowledge vector
- Loss
  - mask the start and end logits using the paragraph mask, not to fit padded values
  - $l_1 =$ cross-entropy-loss $(start_{predicted}, start_{real})$
  - $l_2 =$ cross-entropy-loss $(end_{predicted}, end_{real})$
  - $l = l_1 + l_2$

As expected we gain a huge performance boost, as now the encoder captures well the relationship between the question and the paragraph. This is achieved by using intermediate projections and mixes of question encoding and document encoding $Q, D, L, A_Q, A_D, C_D, C_Q$ to capture the coattention (details in [1]). Also, now instead of dynamic rnn we are using a bidirectional dynamic rnn to get the final knowledge matrix.

2.2 Performance and analyses

The dev $F1$ and EM are at %9.5 and %5.9 which is a boost from the previous model, but still very poor performance. This is expected as we still have a poor linear decoding layer. After several epochs of learning we saw that as time goes we overfit the training data but the model as a whole has a very strict limit on the validation performance.

3 Model 3. DCN Encoder with HMN Decoder

3.1 Architecture

Here we substitute the linear decoder with a modified version of the decoder described at [1]. Instead of running it multiple iterations to let start and end converge to best values, we run it once but make the initial state of the LSTM cell a trainable variable. Details about this modification are in the next section. The final architecture looks like this:

- Encoding
  - pad question and paragraph to have fixed lengths max_q and max_p, build masks.
  - encode question using dynamic RNN with LSTM cell, store all outputs as question encoding Q
  - encode paragraph using dynamic RNN with LSTM, store all outputs as paragraph encoding D
  - calculate coattention matrices:
$$L = D^T Q, A_Q = softmax(L), A_D = softmax(L^T)$$
$$C^Q = DA^Q, C^D = [Q, C^Q]A^D$$
  - calculate final knowledge matrix U (question-paragraph encoding) using bidirectional RNN with LSTM cells, and taking D and C^D as inputs.
- Decoding
  - use trainable variable $state_0$ as initial state for $LSTM$ cell
  - use knowledge matrix U as input to LSTM cell
  - use output state h of the LSTM cell and U as inputs to HMN,
  - calculate α, β using different variables but same HMN architecture
  - start = $argmax(\alpha)$, end = $argmax(\beta)$
- Loss
  - mask the start and end logits using the paragraph mask, not to fit padded values
  - $l_1 =$ cross-entropy-loss $(start_{predicted}, start_{real})$
  - $l_2 =$ cross-entropy-loss $(end_{predicted}, end_{real})$

– $l = l_1 + l_2$

The Highway Maxout Networks(HMN) described in [3] are good at providing 'smooth' information flow through deep layers. This helps with vanishing gradient problem in deep neural networks. In Socher et al [1] authors provide their results for 4 iterations of LSTM -¿ HMN process, whereas here we substitute that pipeline with 1 iteration plus a trainable initial state for the LSTM to carry on information from one training epoch to another. The results are below.

3.2 Performance and analyses

By running only 1 iteration, we can observe that the LSTM cell becomes just a non-linear layer that takes as input the knowledge vector, a trainable variable $state_0$ and outputs a mix to be used in HMN for finding start and end position distributions (α and β in the paper). Since we only run 1 iteration, we get a huge speedup in training, and we were able to run 1 epoch of training (one iteration over 80K training data points, with embedding size = 300 glove vectors) under 40 minutes on Tesla M60, 8GB gpu Memory, 56GB RAM. After running for 10 epochs this model yields dev $F1 = 61.4\%$, $EM = 42.8\%$ scores. The training loss decreases from 9.7 to 1.0 over time as can be seen in Figure 1.

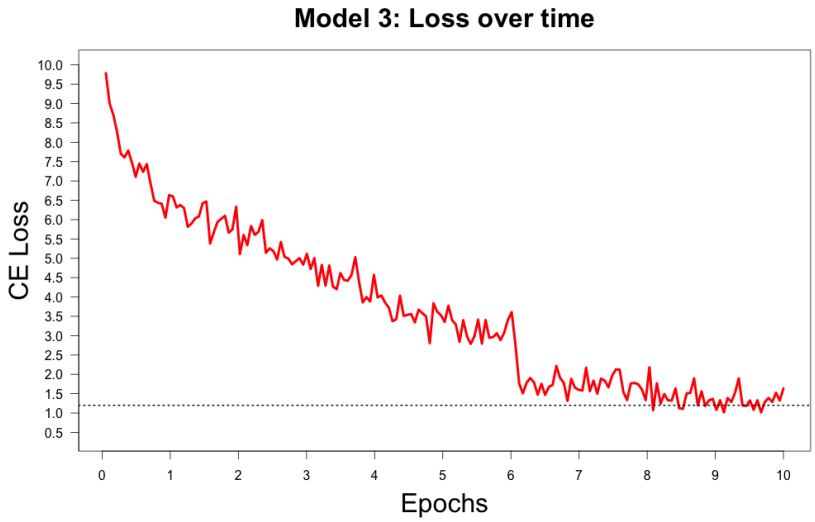

Figure 1: Training Cross-Entropy Loss over time for Model 3

We also see that over the course of 10 epochs, at some point we start overfitting the training data while not improving the validation performance anymore.

Same is true for both the F1 and EM scores as can be seen from Figures 2. and 3.

4 Implementation details

In all of the models described above we used Adam Optimizer with learning rate between 0.01 and 0.0001 depending on the loss (as loss gets smaller we decrease the learning rate not to 'overshoot' min points). Also we used xavier initialization for all the variables.

5 Conclusion

In this paper we described the process of building our final model block by block. The final architecture we came up with is the Dynamic Coattention Network described in [1] with small modifications. Even though we could not replicate the results mentioned in [1], our modification let us gain a speedup in the training process as desribed in the previous section. It is worth mentioning, that aside

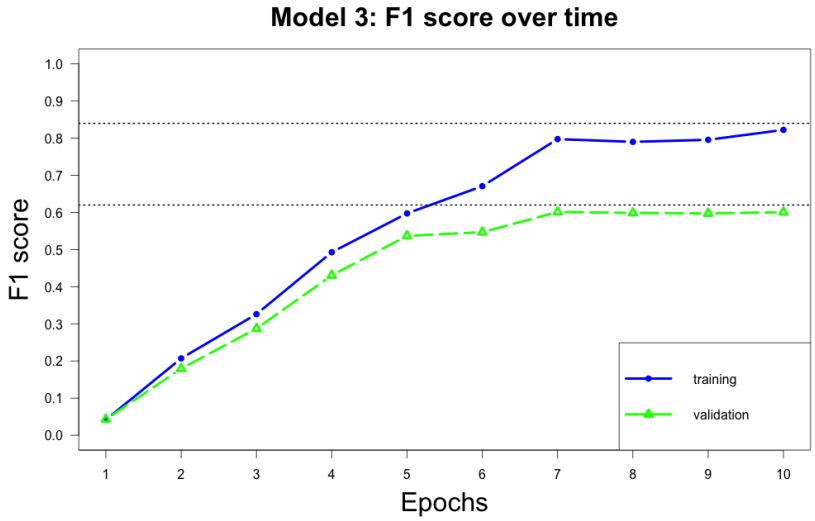

Figure 2: Training vs Validation F1 over time for Model 3

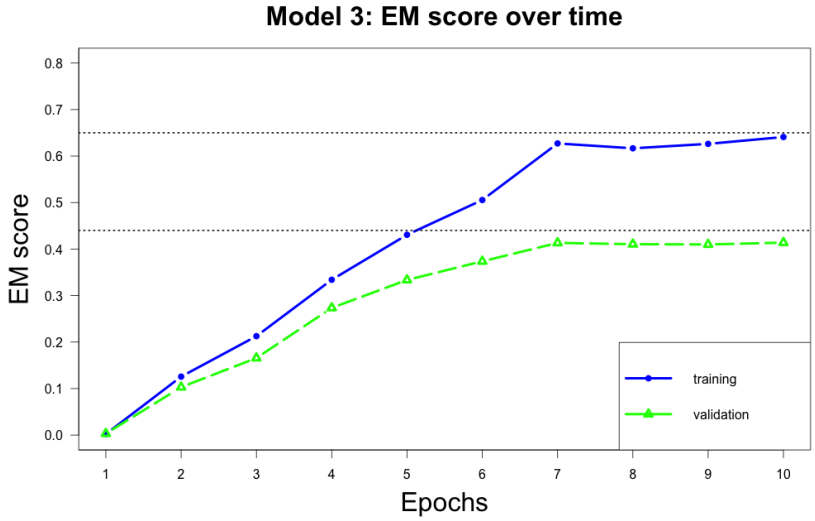

Figure 3: Training vs Validation EM over time for Model 3

from less accurate $F1$ and EM, there is another difference between our model and model in [1] in terms of performance. The gap between $F1$ and EM in our model is relatively bigger than in [1]. The reason is the presence of 4 iterations in [1] aimed at making start and end positions converge to ground-truth. Since we modified that part in our model, we lost some accuracy on EM score.

References

[1] Caiming Xiong & Victor Zhong & Richard Socher (2016) Dynamic Coattention Networks For Question Answering. Eprint arXiv:1611.01604

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[3] Srivastava, R.K. & Klaus Greff & Jrgen Schmidhuber (2015) Highway Networks. Eprint arXiv:1505.00387