SQuAD reading comprehension deep learning model

M. Zaidi, N. Alnaji
Department of Computer Science
Stanford University

Abstract

We introduce a neural network model for reading comprehension using the SQuAD dataset. Our model is composed of a Dynamic Coattention Network encoder (Xiong et al. [2016]) and a novel decoder designed for runtime minimization. Our model obtained an F1 score of 52.283 when tested on the SQuAD dev set, and an exact-match score of 38.723.

1 Introduction

The Stanford Question answering Dataset (Rajpurkar et al. [2016]) is a reading comprehension dataset developed semi-automatically. The dataset is significantly large, compared to previous hand-annotated datasets, and is characterized by having each answer being a span in a given reference document (which we will call 'context paragraphs'). The large scale of the dataset allows the use of data-intensive models such as neural network models. Each example is a triplet composed of a question, a context paragraph, and an answer. The read comprehension task is to predict the answer given the question and context paragraph
We describe in this paper our implementation for a neural network model for this task. Our model is based on the dynamic coattention networks model (Xiong et al. [2016]) for encoding word representations but we devise a simpler, more time-efficient decoding model for generating answer span probabilities.

2 Model

Our model is primarily based on dynamic coattention networks (Xiong et al. [2016]), modified for time efficiency. The model uses a dynamic coattention encoder to capture the information in the question and paragraph and their interaction. Our decoder delivers vectors of predicted probabilities for the starting and end index of answer spans, with the latter dependent on the first.

2.1 Word Representations

We use a distributed rerpresentation for the words in the questions and context paragraphs. Specifically, we represent words using pretrained GloVe word vectors trimmed to 100 dimmensions for computational efficiency.

2.2 Encoder

We implemented a modified version encoder of the Dynamic Coattention Network encoder implemented by (Xiong et al. [2016]). The encoder receives as input a sequence of word vectors for the questions, $({x_1}^Q, ..., {x_n}^Q)$, and a sequence of word vectors for the context paragrpahs $({x_1}^D, ..., {x_n}^D)$. The encoder feeds the paragraph word vectors into a BiLSTM, to generate a paragraph encoding maxtrix $P = [p_1, ..., p_m]$, and also feeds the question word vectors into the

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

same BiLSTM to generate an intermediate question encoding matrix $Q' = [{q_1}', ..., {q_n}']$. The intermediate question encoding matrix is mapped to a final question matrix as $Q = tanh(W^{(Q)}Q' + b^{(Q)})$

Next the encoder computes an affinity martix, L, which computes the dot product between each column of Q and each column of P.

$$L = D^T Q \tag{1}$$

This matrix is normalized row-wise to estimate the relative attention weight of each of the words in the questions for a given word in the paragraph, and normalize column-wise to estimate the relative attention weight of each of the words in the paragraph for a given word in the question.

$$A^Q = softmax(L), A^D = softmax(L^T) \tag{2}$$

Next, the encoder computes an attention context of the document for each word in the question.

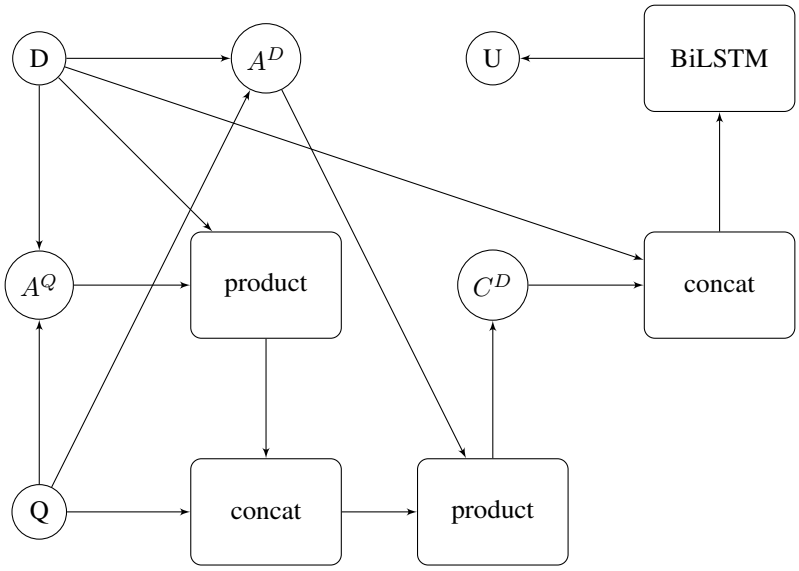
$$C^Q = DA^Q \tag{3}$$

It then computes

$$C^D = [Q; C^Q]A^D \tag{4}$$

where $[Q; C^Q]$ denotes the horizontal concatenation of these two matrices.

The final encoding matrix, $U = [u_1, .., u_m]$ is computed as the ouput of a separate BiLSTM with the input at each timestep $[d_t; {c_t}^D]$. This final matrix is the input to the decoder.



2.3 Decoder

Since answers in SQuAD are always substrings of the context paragraph, we can efficiently predict answers by predicting the beginning and end index of the answer span (Wang and Jiang [2016]). Our initial approach to the decoder was one that utilized Highway Maxout Gates (Xiong et al. [2016]) to make predictions on the beginning and end indices. This system attempts to resolve the issue of multiple likely answer spans. The HMN model was modified in order to be utilized without the use of an RNN. The final model was thus as follows:

$$HM(u_i) = \max(W^{(3)}[m_i^{(1)}; m_i^{(2)}] + b^{(3)}) \tag{5}$$

$$m_i^{(1)} = \max(W^{(1)}u_i + b^{(1)}) \tag{6}$$

$$m_i^{(2)} = \max(W^{(2)}m_i^{(1)} + b^{(2)}) \tag{7}$$

This model eventually proved too time-inefficient for our purposes. We eventually chose to use a

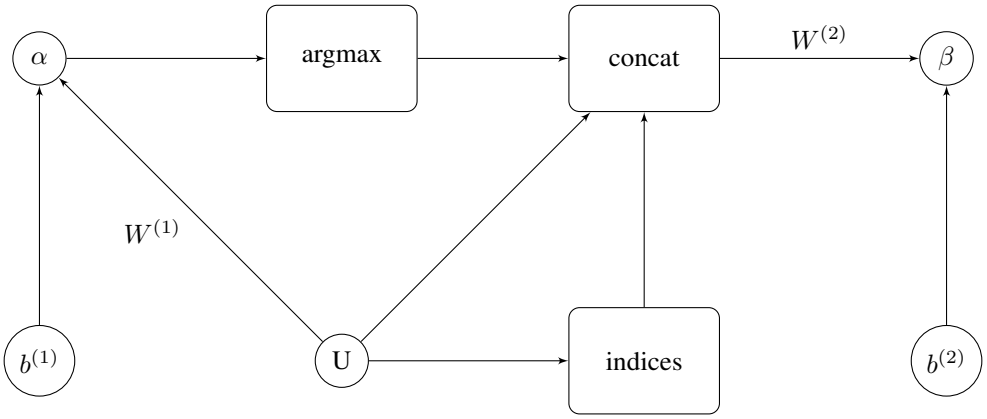
simpler decoder instead. This was done to reduce the number of parameters in the model in order to improve time efficiency. Our decoder outputs two prediction vectors: one for beginning indices and another for end indices. We form the two prediction vectors α, β based on the following:

$$\alpha_i = w_i^{(1)} u_i + b_i^{(1)} \tag{8}$$

$$s = \text{argmax}(\alpha_1, \alpha_2 \dots \alpha_p) \tag{9}$$

$$\beta_i = w_i^{(2)} [u_i; s; i] + b_i^{(2)} \tag{10}$$

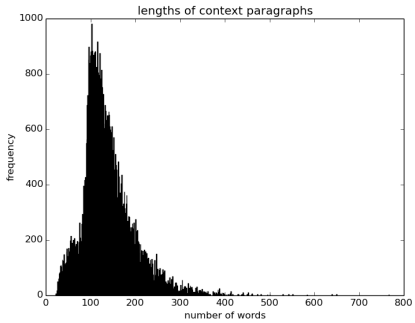
Where $w^{(1)}, w^{(2)}$ are weight vectors, $b^{(1)}, b^{(2)}$ are bias vectors, and u_i for $i \in [1, p]$ is the encoded word representation retrieved from the encoder. The end predictions use the indices of the start word (based on the previous prediction α) and the current word. We believe this is useful since the length of the answer spans are usually within a small range. Ideally, the model would be optimized to use the indices to make use of the possible answer's length.

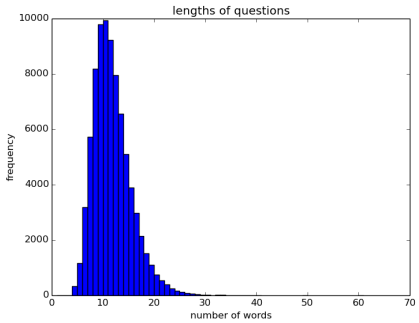


3 Implementation Details

Model was implemented with tensorflow on a NVIDIA multi-core GPU. The program utilized multi-core processing to improve time efficiency. The model is trained and evaluated on the SQuAD dataset, with a training set of around 80,000 points. For preprocessing the data, we've used the Natural Language Toolkit tokenizer.

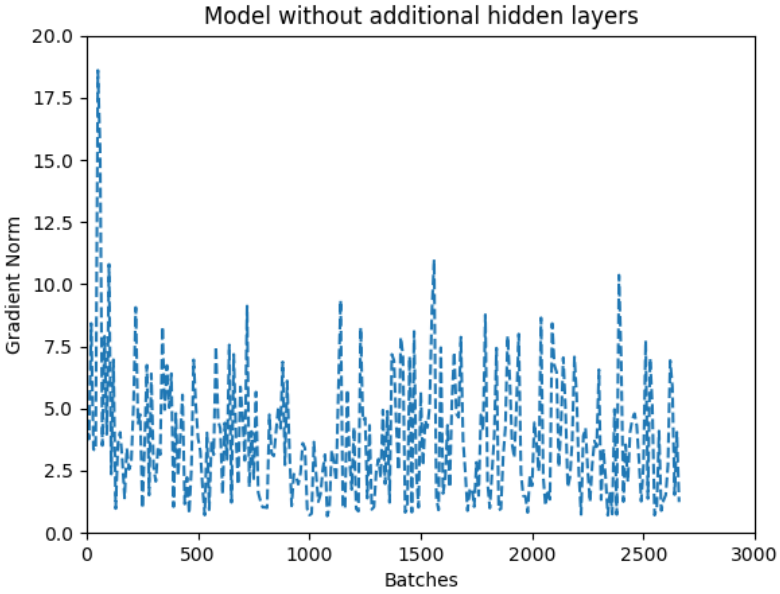
Our learning rate was set to 0.001 for the first five epochs, and then we manually lowered it to 0.0001 in an attempt to stabilize loss (it stabilized it somewhat but not as much as we'd hoped). We optimized our model using the ADAM optimizer (Kingma and Ba [2014]). We've also set a maximum length for the paragraphs (400 tokens) and questions (35 tokens); any tokens beyond the maximum length are not considered. This created an issue with answers in the training dataset spanning tokens past the maximum paragraph length. We simply ignore examples that include such answers. These maximum lengths were based on our interpretation of the common lengths of the data as we see below.

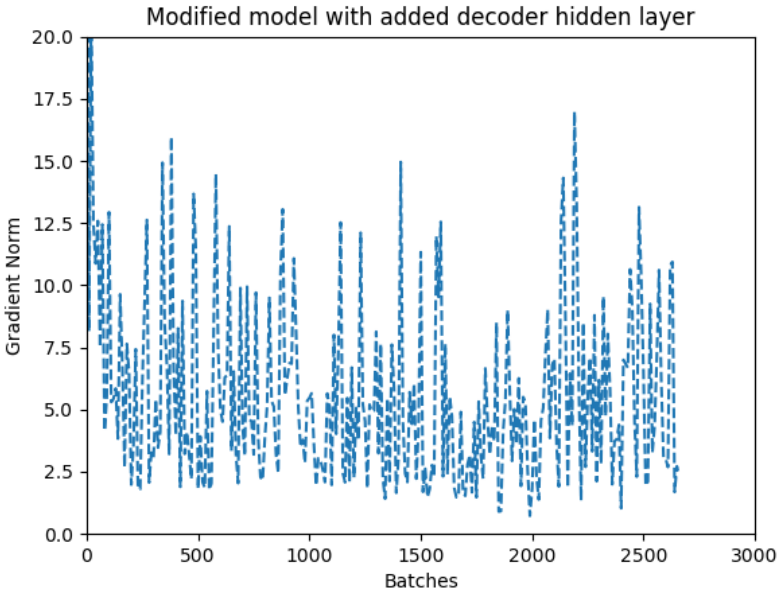


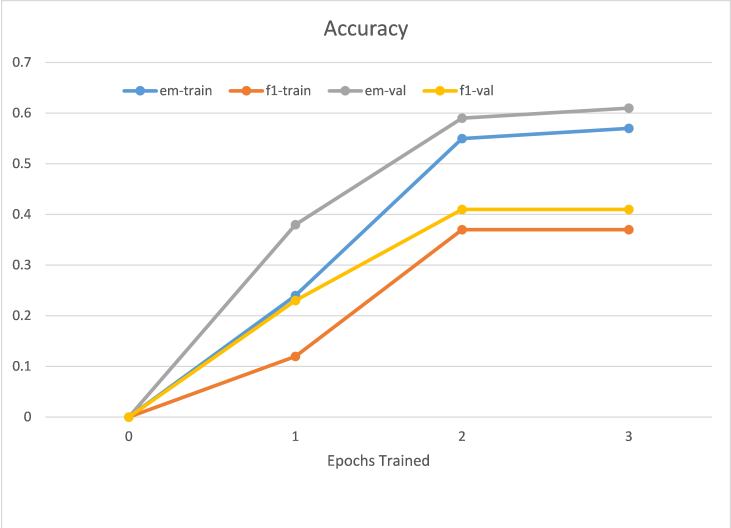


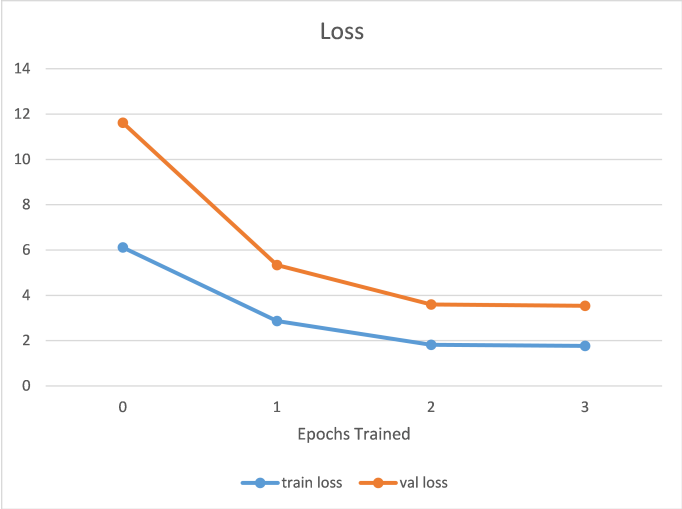
4 Results

Our model obtained an F1 score of 52.283 when tested on the SQuAD dev set, and an exact-match score of 38.723. This was achieved after training over only three epochs, but considering that the loss stabilized and stopped decreasing, it is likely that more epochs would not have significantly improved performance, but rather would only lead to overfitting.
We've also kept track of the model's gradients over several iterations of the model. The most comprehensive data we've collected were over the model we've described earlier, and for a modified version of the model which adds an additional hidden layer to the decoder infrastructure using relu gates. Below are the gradient plots of each, as well as the EM, F1, and Loss across number of epochs trained. It appears that performance of this model may be saturating and that a more powerful model is needed but we would need to train for more epochs to be sure.









5 Other Approaches and Ideas Explored

We first focused on attempting to implement a simplification of the baseline model described in the assignment4 handout. As suggested, we created an encoder that first generated the question representation by concatenating the initial and final outputs of a BiLSTM with question word vectors as inputs. The encoder next generated paragraphs representation as the outputs of another BiLSTM with inputs the paragraph word representations, each concatenated with the just computed question representation. Finally we computed an attention score for each paragraph word, using the outputs of second BiLSTM, relative to the question representation. We computed attention scores using billinear attention [Chen et al. [2016]]. We did not further compute attention vectors, reasoning that the attention score alone should be sufficient to estimate the importance of the words at a baseline level of accuracy.

Initially, rather than predicting start and end positions we attempted to predict for each word in a paragraph whether or not it was in the answer. To decode these predictions from the attention score, We tried directly using our attention scores as unnormalized predictions, adding a single layer hidden layer neural network, and adding a multiple hidden layers neural network. We also experimented with different combinations of activations functions.

Initially, no matter what we tried we obtain an F1 score of $10^-5\%$ and and EM score of 0 on the training data. Examining the actual predictions we found that our network was always predicting that a word was not in the answer. We realized this was because almost all the words in a paragraph were not in the answer, and the softmax crossentropy loss function we were using did not differentiate

between precision and recall in the same way that an F1 score does. We attempted to remedy this by calculating separate softmax cross entropy losses for word in the answer and words not in the answer and averaging them. After this change we were able to achieve an F1 score of 0.7% on the training data.

We found that both the number of hidden layers in our decoder and choice of activation functions had little effect on performance. Decoding the start and end tokens rather than predicting whether a word was in the answer also did not improve performance.

The fact that our model performed so poorly even on the training data led us to believe that our model was either not sufficiently powerful or there was an error in our implementation. We thus eventually moved to implementing the Dynamic Coattention model.

Our initial implementation of the Dynamic Coattention model was also unabe able to fit even the training set data, achieving and F1 of 1.4% and EM of 1%. After spending much time attempting to debug our model and attempting to tune hyperparameters such as paragraph size, hidden state size, and learning rate we found that our model suddenly was able to learn when we switched from a GradientDescent optimizer of an AdamOptimizer. Our experience highlights the importance of choice of optimizer in learning a model.

Finally, In one version of our coattention model we added a log mask that ensure that the computed probability of a start or end token being outside the paragraph words was zero. We achieved and EM and F1 of 0.42 and 0.6 on training samples and 0.56 and 0.32 on validation samples with this model. However, were unable to integrate this feature into our final model. In the future we would do so.

6 Future Work

Currently to compute the affinity matrix, L, through matrix multiplication, we effectively compute the compute the dot product between each question word encoded representation and each paragraph word encoded representation, as a measure of the their similarity or relevance to each other. In the future, we may consider replacing this dot product with billinear attention so that the affinity matrix will now be computed as $L = D^T W Q$ (Chen et al. [2016]). We may also consider computing these attention scores over multiple perspectives(Wang et al. [2016]). Indeed our current approach can be considered to be a special case consider just the single perspective given by a vector of all ones.
Our current implementation has also shown to overfit the training data, thus modifying our model with dropout might improve performance on the test set significantly, as dropout has shown to significantly reduce overfitting (Srivastava et al. [2014]).
We also believe our question and paragraph representations can be improved. Our model currently uses the NLTk Tokenizer, but it is possible that other tokenizers such as the Stanford Core-NLP tokenizer may deliver better results. We've also considered incorporating other information besides word-to-vector GLoVe representations, such as Part-of-Speech word tags, window-based representations, and logic-based knowledge representation (specifically, we've considered using NaturalLI (Angeli and Manning [2014]) to incorporate and relate information from multiple sentences).

We could also further tune our hyperparameters, including the dimmensions to which the GloVe vectors are clipped, as well as the hyperparameter such as hidden state size and max paragraph length.

References

Gabor Angeli and Christopher D Manning. Naturalli: Natural logic inference for common sense reasoning. 2014.

Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858, 2016. URL `http://arxiv.org/abs/1606.02858`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL `http://arxiv.org/abs/1606.05250`.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016. URL http://arxiv.org/abs/1608.07905.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *CoRR*, abs/1612.04211, 2016. URL http://arxiv.org/abs/1612.04211.

Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016. URL http://arxiv.org/abs/1611.01604.