Co-Dependent Attention on SQuAD

Codalab Team: ArtificialPaper

Test Set Leaderboard: F1:62.534, EM: 50.981

Siyue Wu
Department of Statistics
Stanford University
Palo Alto, CA
siyuewu@stanford.edu

Fabian Chan
Department of Computer Science
Stanford University
Palo Alto, CA
fabianc@stanford.edu

Xueyuan Mei
Department of Computer Science
Stanford University
Palo Alto, CA
xmei9@stanford.edu

1 Overview

In the realm of natural language processing, machine comprehension of textual documents is an incredibly important problem that presents various challenges and difficulties. A benchmark dataset for question answering named SQuAD is comprised of around a hundred thousand question-answer pairs, along with context paragraphs for each. The answer to each question is a span within the context, and it is the objective for the answering machine to predict this answer span.

In this report we outline our process of building a neural network model and the exploration we have done in order to tackle this dataset, which includes building a co-attention layer to represent the question-context encoder pair and a loss function that quadratically penalizes reversed answer span predictions.

2 Model Description

2.1 Encoder

GloVe vectors of 100 dimensions are used as the embedding matrix. We decided to train the embeddings as this leads to faster convergence. We encode both the question and context paragraph with bidirectional LSTM layers as each word in the inputs has significant relationships with words that appear before and after it. The output hidden vectors associated with the question and context are concatenated into separate matrices Q and D, as denoted below, where n is the question length, m is the context length, and l is the number of hidden units.

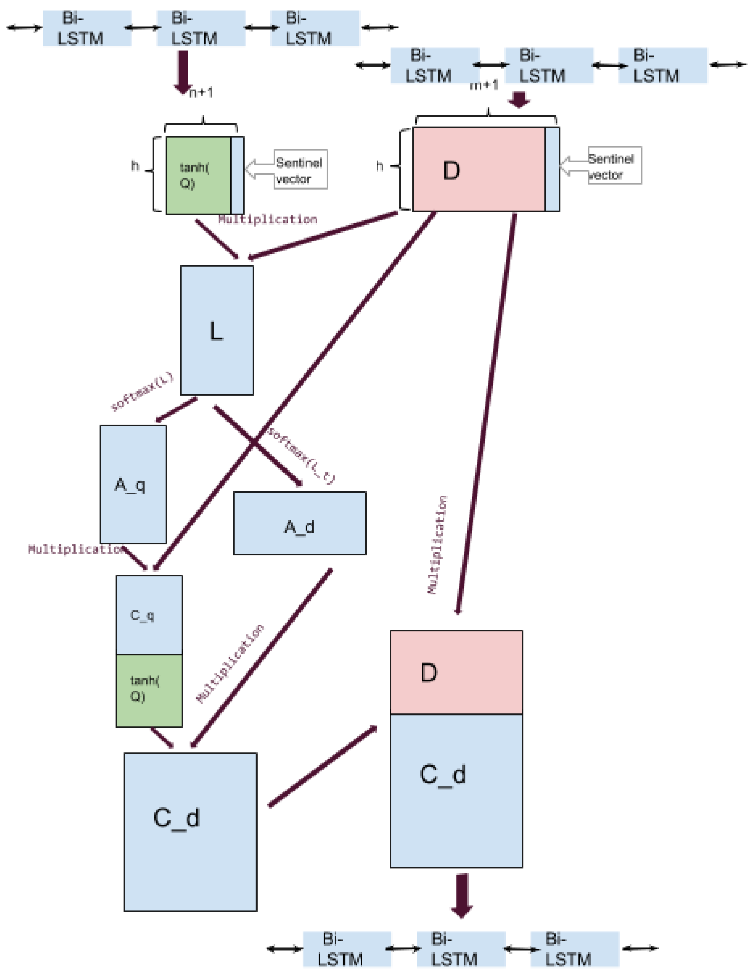
$$Q = (q_1, q_2, ..., q_n) \in \mathbb{R}^{l \times n}$$

$$D = (d_1, d_2, ..., d_n) \in \mathbb{R}^{l \times m}$$

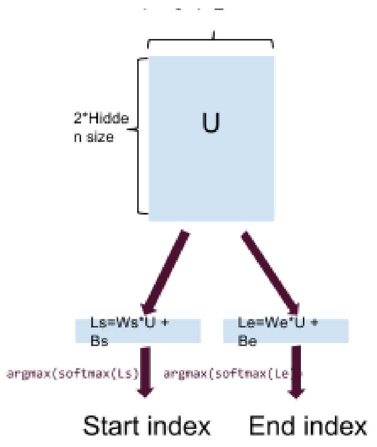
We also concatenated one more sentinel vector at the end of both D and Q, which allows the model to not attend to any particular word in the input. The new Q and D are below:

$$Q = (q_1, q_2, ..., q_n, q_{\oslash}) \in \mathbb{R}^{l \times (n+1)}$$

$$D = (d_1, d_2, ..., d_n, d_{\oslash}) \in \mathbb{R}^{l \times (m+1)}$$



(a) Encoder



(b) Decoder

Figure 1: QA Model

To allow for variations between questions and paragraph, we add a tanh nonlinear layer on top of the question representation matrix.

$$Q = tanh(W^Q Q + b^Q) \in \mathbb{R}^{l\times(n+1)}$$

2.2 Co-Attention Layer

We want to focus on the relevant parts of the context that allows the model to extract the answer span according to the particular question by calculating a one dimensional attention vector from the input representation computed from the encoder.

However, there are co-dependencies between the question and context that cannot be modelled by one dimentional attention vectors. A coattention representation matrix has been successfully used and presented in [1], and we incorporated it to our model. We compute C_q, which is the context matrix for question, using the dot product of the paragraph representation matrix and affinity matrix:

$$L = D^T Q \in \mathbb{R}^{m\times n}$$

$$A^Q = softmax(L) \in \mathbb{R}^{m\times n}$$

$$C_q = DA^Q$$

After that, we compute C_d, the co-dependent context matrix, which depends on C_q:

$$A^D = softmax(L^T) \in \mathbb{R}^{n \times m}$$

$$C_d = [Q; C^q]A^D$$

The last step of the co-attention layer is to use one more bidirectional LSTM and receive C_q and D as inputs:

$$u_t = BiLSTM(u_{t-1}, u_{t+1}, [d_t; C_q t]) \in \mathbb{R}^{2l}$$

$$U = [u_1, u_2, ..., u_m] \in \mathbb{R}^{2l \times m}$$

2.3 Decoder

We implemented a simple decoder to test our encoder. Our decoder consists of two classifiers, one on the start index and the other on the end index. Each classifier is used to predict which index should be the output for the answer start/end index among all probabilities of the size of the context paragraph.

Later in the exploration section we mention about implementing a more complex modelling layer for the decoder and there we outline the reasons why we ended up not using it.

2.4 Enriched Loss Function

It might be sufficient to simply take the sum of the cross-entropy loss for the start index and the cross-entropy loss for the end index as the loss function but we saw an opportunity to improve it: what is missing is that the loss function can be modified to penalize cases where the predicted start index is larger than the predicted end index, as follows:

$$\sum_i CE_{start} + CE_{end} + \lambda \|(E_{truth} - S_{truth} + 1) - (E_{predicted} - S_{predicted} + 1)\|^2$$

Our loss function takes 3 terms into account. The first two are the cross-entropy loss. The third term is the l2-norm of the difference between the true answer length and the predicted answer length. This term includes a regularization constant lambda multiplied by a quantity that quadratically penalizes the difference between the lengths of the true answer and predicted answer. The penalty is further increased, also quadratically, if the predicted answer span is negative in length.

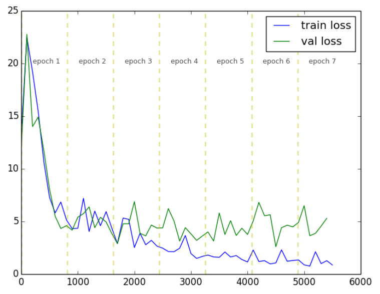
3 Experiments

3.1 Approach

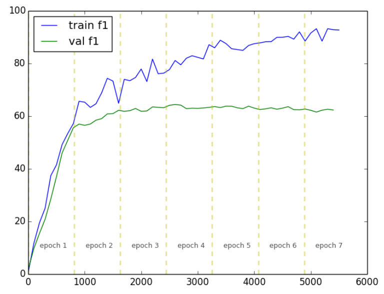
Given the time limits of this assignment, and the importance of being able to iterate quickly, we sought to find a balance between optimizing the model's performance and keeping the training time short. As a result, some of our data processing steps aim to reduce training time at some minor cost to performance.

Truncating inputs. One of the things we discovered during our data exploration steps is that most of the context paragraphs are far shorter than their maximum length of 766 words. Plotting the distribution of paragraph lengths shows that more than 98% of the paragraphs in the training and validation set have lengths shorter than 400 words. Since Tensorflow expects matrix inputs of uniform dimensions every batch, truncating each context paragraph to 400 words allows a much larger batch size during training while affecting performance in an almost negligible manner.

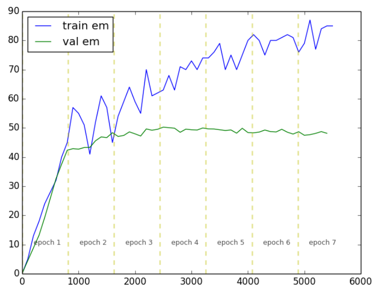
Shuffled batches. We had a choice on whether to feed the data in given order or in shuffled order. One may hypothesize that It would make sense that the model performance should not depend on the order in which the data is presented to the model for training. Although at the same time there is the opposite idea of curriculum learning, where training examples are presented to certain models to achieve a lower loss than what would be achieved by random ordering. When we tested both out, there was not much difference in peak validation performance, but in the case of randomized batches, learning converged noticeably faster and the loss curve during training declines in a more stable manner.



(a) Loss vs Batch Number



(b) F1 vs Batch number



(c) EM vs Batch number

Figure 2: Validation loss,f1,em vs Batch number

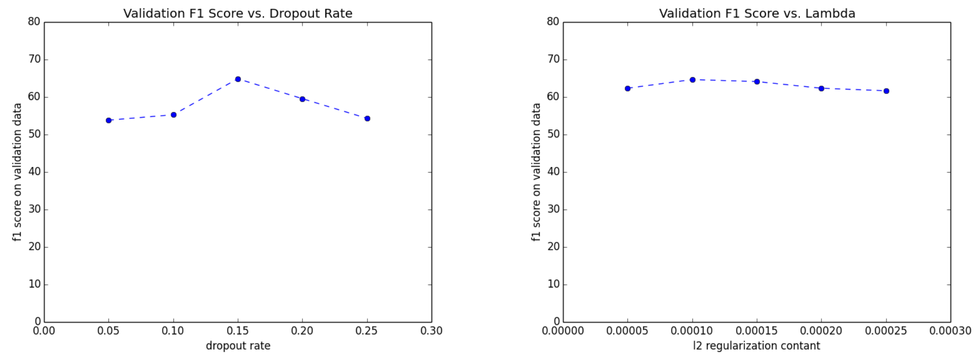
Batch size. The GPU has a limited amount of memory, and unlike jobs assigned to the CPU, cannot spill to secondary storage and immediately crashes if all memory is consumed. In general, the batch size is the main parameter outside the model itself that can affect how much of the GPU's memory is occupied. Too small of a batch size incurs large overhead, and thus slows down training, while too large of a batch size carries a risk of running OOM. We were able to increase batch size from 60 up to a value of 100 to optimize for training time after implementing the input truncating method described above.

Hyperparameter tuning. Standard machine learning techniques are available at our disposal to optimize for various hyperparameters, such as cross-validation and grid search. Although we agree wholeheartedly that diligent hyperparameter tuning is essential to a good model, we simply could not afford to spend valuable GPU time needed to optimize our model via repeated training of slightly different datasets (k-fold cross validation) or on numerous combinations of hyperparameters (exhaustive grid search). Instead we opt to approach it in a manner similar to coordinate descent, where we vary one hyperparameter at a time while keeping all others fixed. This may not direct us to the best optimum, however this is vastly more time efficient and from our perspective this is a worthy tradeoff. The next section shows our tuning efforts.

3.2 Training

From the plots in Figure 2, we see that our model converges very rapidly: in 1 epoch, F1 performance reached 51% and EM reached 42%, while loss dropped by 53.4% on the validation set. However, the validation performance plateaus after 3-4 epochs.Overfitting occurs from epoch 4 onwards as validation loss increases while train loss continues to drop, as shown in the figure.

As a consequence, we extracted the model weights at the moment before overfitting occurs, in order to ensure that the model generalizes its performance to the dev and test datasets, essentially performing early stopping.



(a) F1 vs dropout rate

(b) F1 vs λ

Figure 3: Validation F1 score vs. hyperparameter

3.3 Hyperparameter Tuning

Learning rate. We found that tuning the learning rate did not have a noticeable effect in the peak validation performance, however it did affect convergence. When the learning rate was set to a value too small, convergence was slow and took many hours to reach peak, and when it was too large, the model did not converge, and the F1 score did not improve beyond 11%. We ended up choosing a value of 0.0001, somewhere between too small and too large.

Lambda. The lambda parameter is used in the loss function designed to regularize the difference between the true answer's length and that of the predicted answer. A large lambda penalizes the length difference more heavily than a small lambda. If the predicted start index is larger than the predicted end index, this is also quadratically penalized by the loss function proportional to lambda. From the results of validation F1 score performance over different values this parameter, we can see that the overall impact of lambda is not as big as the other parameters, but is still significant. While the influence is not as obvious, we managed to find the optimal value of lambda for our model. The performance is increased by approximately 3% with the addition of this penalization

Dropout rate. Deep recurrent neural networks with millions of parameters are very powerful and overfitting is a serious problem. Dropout is a common technique to address this problem. We experimented with different dropout rates, and we can see from Figure 3 that the performance on validation peaks at 0.10. The performance is suboptimal when dropout rate is low as the model overfits, and is also the case when the dropout rate is high as the model underfits.

4 Exploration

4.1 Multi-perspective Matching

The first model we implemented, then threw away, was the Multiple Perspective Matching model. However, training time was high and it was difficult to iterate on, so we focused our efforts to the model we described in Section 2. Then we attempted to combine the two models by multiplying the output scores of these two models in an element-wise manner before applying softmax to compute the final output probabilities. However, it turns out that the performance of this combined model did not increase significantly. This combined model improved the F1 on validation by only 0.5% while the training time increased to 2.5x longer. After discussion, we decided the benefit of the extra 0.5% was not worth the valuable GPU time we could've used for other iterations.

4.2 Dynamic Decoder

As an improvement of our current model of decoder, we implemented dynamic decoder. Here, we applied iterative procedures so that the model eventually settles on an answer span by updating its start index and end index in an alternating manner. By alternating between start point and end point, the model can easily recover from the initial local optima trap caused by the incorrect answer span. However, the training time increased dramatically after we added dynamic decoder procedure into our model. The increase made it impossible for the training to complete. Therefore, we decided not to add this feature.

4.3 Double BiLSTM Modelling Layer

Such a layer was used successfully in [2], and we attempted to apply it to our model. The input to the modeling layer is U, which is the co-attention matrix between the question and context paragraph. The output of the modeling layer is expected to model the interaction among the context words, conditioned on the question words. This is passed through two BiLSTM layers and into the output layer to predict the answer span. We did not find any improvement in the validation performance with the use of this layer, nor any noticeable changes in training time.

5 Error Analysis

As an exercise to analyze errors, we selected two context-question-answer-prediction tuples representative of the type of errors made by our trained model. We have found the model frequently makes mistakes on certain types of inputs, such as context paragraphs with substantial ambiguity, or those with multiple local optima. The following examples illustrate these two types with more details.

Error Analysis Example 1:

Context: Unarmed fox hunting on horseback with hounds is the type of hunting most closely associated with the United Kingdom ; in fact , ” hunting ” without qualification implies fox hunting . What in other countries is called ” hunting ” is called ” shooting ” (birds) or ” stalking ” (deer) in Britain . Originally a form of vermin control to protect livestock , fox hunting became a popular social activity for newly wealthy upper classes in Victorian times and a traditional rural activity for riders and foot followers alike . Similar to fox hunting in many ways is the chasing of hares with hounds .

Question: In England , what is hunted when ” shooting ” is called for ?

Ground Truth: birds

Prediction: fox.

Analysis: Although in the context it is true that fox hunting is done in the UK, the model failed to predict the correct answer. What the model predicted is just one of the animal animal types being hunted. The cause of this might be that the occurrence of “fox hunting” (4 times) in the context is much higher than the equivalent for “birds” which occurs only once. Since our model does not specifically handle multiple local optima, it is very likely that it was stuck in one of the “fox” optima. This indicates that further improvements need to be made so that the model will be able to escape them. Although the answer is incorrect, the model succeeded to predict the correct type of answer: an animal. This type of error occurs commonly among the total errors made by our model. We estimate that perhaps up to 20% of errors fall under this category.

Error Analysis Example 2:

Context: Some of the best examples of later Islamic mosaics were produced in Moorish Spain . The golden mosaics in the mihrab and the central dome of the Great Mosque in Corduba have a decidedly Byzantine character . They were made between 965 and 970 by local craftsmen , supervised by a master mosaicist from Constantinople , who was sent by the Byzantine Emperor to the Umayyad Caliph of Spain . The decoration is composed of colorful floral arabesques and wide bands of Arab calligraphy . The mosaics were purported to evoke the glamour of the Great Mosque in Damascus , which was lost for the Umayyad family .

Question: Who created the mosaics in the Great Mosque in Corduba?

Ground truth: local craftsmen

Prediction: Byzantine Emperor

Analysis: Although the prediction gives an answer of the correct type (person), it failed to give the correct answer. The answer given by the prediction is actually the person who sent the supervisor to the craftsmen who created the mosaics. This type of mistake made by our model may be caused by the ambiguity of the question. Our model failed to learn the actual meaning of create. Furthermore, when a question is asking for a person (by being with the question with "Who"), our model tends to output proper nouns as an answer.

6 Conclusion and Future Work

Conclusion. Co-attention network worked well with our dataset and objective function. It offers a more complicated and mature attention mechanism. Adding the co-attention network manages to increase F1 performance by 5%. Regularization worked well when added to our loss function so that it penalizes quadratically in situations where predicted start indices are larger than the end indices (i.e. negative length). This improves the model performance on F1 by 3%.

Dealing with Suboptimal Optima. Our current model has some problems being trapped in local optima that produce the wrong answer spans. During training, we observed examples for which our model was unable to converge to the correct answer, even over multiple epochs. Perhaps a good next step to take this project is to apply techniques to deal with these optima. Although the dynamic decoder helps solving the local optima issues, it costs large amount of training time. Our future work may also involve improving our current implementation of dynamic decoder.

More Complex Decoder. Moreover, our current decoder can be improved as well. Generally, the questions and context can have different types, which may require different models to estimate answer spans, according to the particular questions. However, we can incorporate the dynamic decoder in [1], which iterates through three maxout layers to pool across multiple model variations.

7 References

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

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