Exploring Combinations of Character Embeddings and Coattention

Stanford CS224N Default Project: IID Squad Track

Derek Chung
Department of Computer Science
Stanford University
dchung22@stanford.edu

Abstract

In this project, I attempt to build a model for the Stanford Question Answering Dataset (SQuAD) v. 2.0 [1]. I consider 3 different models, the baseline model, or Bi-directional Attention Flow (BiDAF) without character level embedding [2], BiDAF with character level embedding, and a Dynamic Co-attention Network [3] with character level embedding. Some conclusions drawn from my experiment was that implementing character level embedding in the BiDAF model significantly improved EM and F1 scores over the baseline. However, even though the Dynamic Co-Attention Network with character level embedding was an improvement over the baseline, it scored lower on both F1 and EM scores than BiDAF with character level embedding. On the development set, the BiDAF with character embedding has an F1 score of 63.030 and EM score of 59.839. The Dynamic Co-attention Network with character embedding has an F1 score of 61.54 and an EM of 57.81. My best result on the SQuAD testing set was the BiDAF with character embeddings, achieving an F1 score of 62.266 and an EM score of 58.952.

1 Introduction

A Question-Answering model takes a question and some context as input. The goal of the model is to find the answer to the question within the context. Many solutions, such as the Bidirectional Encoder Representations from Transformers (BERT) [4], rely on deep model representations to achieve higher scores. However, the computing power needed to process such models are immense, due to the large size of the dataset and vocabulary.

The Bidirectional Attention Flow model (BiDAF) [2] is a model that doesn't achieve the highest accuracy, but uses less processing power per training step. One reason for this may be the reduced number of layers; in the baseline model, a piece of input passes through 3 Long Short Term Memory (LSTM) networks [5], which is a kind of of Recurrent Neural Network. The LSTMs contain 4 layers total, which I suspect contributes to high efficiency.

For this project, I implement 2 very simple models. The first is the original BiDAF model with character embeddings, and a Dynamic Coattention Network with character embeddings. The baseline model has word embeddings without character ones. To get character embeddings, I project both word and character embeddings to 1/2 of the original size and concatenate the resulting outputs together. For the Dynamic Coattention Network, I replace the BiDAF layer with the Dynamic Coattention Network described in a paper titled "Dynamic Coattention for Question Answering" [3]. I train the baseline and my two models for 30 epochs, validating on the dev set every 50,000 iterations. Each model is evaluated by 3 metrics: the EM (Exact Match) score, the F1 score, and AvNA score.

2 Related Work

End to end answer chunk extraction calculates a probability distribution for the start and end positions of the true answer, given a context and question. Most models predict the probability as 2 independent probabilities, namely:

$$P(S, E|C, Q) = P(S|C, Q) * P(E|C, Q) \tag{1}$$

where S and E are random variables representing the start and end positions of the answer given context C and question Q. A paper titled "End-to-End Reading Comprehension with Dynamic Answer Chunk Ranking" [6] takes all possible chunks up to a certain length, gets forward and backward states for each attention label, and extracts probability distributions for all chunks. To read more about my thoughts on implementing end to end chunk extraction, see Section 6: Conclusion.

Since I do not implement the baseline model, I will describe its properties here. The baseline model takes a word vector W as input. It is then fed through 4 different layers. The first layer,

$$H_1 = a(W) \tag{2}$$

gets the embedding of the word vector, applies a projection onto the embedding, and then applies Highway and Dropout layers to the resulting projection. Highway networks are described in greater detail in another research paper [7]. After we get H_1, we apply a 2 way LSTM to H_1 such that

$$H_2 = b(H_1) \tag{3}$$

where b is a BiLSTM with 2 layers. H_2 includes the forward and backward hidden state for each embedding in H_1. Then we go through another layer

$$H_3 = c(H_{2q}, H_{2c}) \tag{4}$$

The function c represents the Bidirectional Attention Flow layer [2]. We apply the first two layers to both question and context vectors, so H_{2q} represents the state of the question and H_{2c} represents the state of the context. We run H_3 through another 2-way LSTM to get

$$H_4 = d(H_3) \tag{5}$$

where d is a BiLSTM with 1 layer. Finally, I get the output, where

$$P = e(H_3, H_4) \tag{6}$$

P ends up being 2 separate probability distribution matrices, one for the beginning context position and one for the end context position of the answer to our inputted question. The function e runs H_4 through a single layered Bidirectional LSTM. Both H_4 and the states generated by the BiLSTM are concatenated onto H_3. The two resulting matrices undergo separate linear transformations in which the weights are trainable parameters. Finally, a log softmax function is applied to each transformation, producing 2 log probability distributions that is passed into the trainer.

3 Approach

My first model, BiDAF with character embedding, is the same as the baseline model save for one part.

The first layer in the baseline is $H_1 = a(W)$. During layer a, we get the embedding of the word vector W, apply a linear projection to the embeddings, and then, apply Highway and Dropout layers. In my model, which includes character embeddings, we have a different first layer:

$$H_1 = a(W, C) \tag{7}$$

where C is a character vector and W is the word vector. Each step performed on W in the baseline is performed on both W and C. However, let's say that W was projected to dimensionality h in the baseline. In my model, H and C are projected to dimensionality $h//2$ and concatenated together.

My second model, Dynamic Coattention Network with character embedding, replaces the BiDAF layer with the Dynamic Coattention Network layer. For specific details on the Dynamic Coattention Network layer, visit this paper [3] or see section 4.2 in the project instructions.

Both models will be compared to the BiDAF baseline, which is delineated further in Section 2: Related Work.

4 Experiments

4.1 Data

I train the baseline and my models using the Stanford Question and Answering Dataset (SQuAD) 2.0 [1]. I use the official training set for all my models.

I use a provided subset of the official validation set as the dev set, which runs every 50,000 training steps. The dev set is used in this way for all models.

To evaluate the models, I use a provided subset of the official test set. For grading purposes, I am not able to test the entire set.

4.2 Evaluation method

Note: all scores and functions were provided by the template for this project, which in turn, contains elements from the original BiDAF model [2].

AvNA is a normalized benchmark that represents the number of context, question inputs as having an answer or not. Although AvNA and negative log likelihood are evaluated throughout training using TensorBoard, the standard benchmark is the EM and F1 scores achieved by a model.

The EM score for a single prediction outputs 1 if the answer matches the ground truth exactly, and 0 otherwise. The final EM score is the average EM score for all testing examples.

The F1 score can be calculated as follows:

$$F1 = 2 * p * r/(p + r) \tag{8}$$

p is equal to precision, or whether or not the predicted answer is a subset of the ground truth. $p = 1$ if the prediction is a subset of the truth, and 0 otherwise.

r is equal to recall, or the percentage of words in the ground truth that also appear in the prediction. Another way to think about recall is dividing the intersection of the prediction and ground truth by the ground truth.

The evaluation and test sets both include three ground truth labels. When calculating the F1 and EM scores for a single prediction, the score is the maximum value achieved out of all three ground truths.

4.3 Experimental details

I used an Azure Standard NC6 CPU running on Ubuntu 18.0.3 to train my models. The inputs and model layers are represented as PyTorch tensors and layers respectively. TensorBoard was used to track the values of negative log likelihood, AvNA, F1, and EM scores for the dev set. Negative log likelihood was also graphed during training.

My learning rate remained at a constant value of 0.01 for all models.

The framework and environment I used were (modified) programs for the original BiDAF model [2].

4.4 Results

The maximum scores produced for the testing set used the BiDAF with character embedding layout, achieving F1 and EM scores of 62.266 and 58.952 respectively.

5 Analysis

If we look at Figure 5, it is difficult to tell exactly what model performs the best. However, I would like to focus on the variance in all three models while training. The negative log likelihood during training fluctuates quite a bit for all models. This may not be unusual for other Question-Answering models, but one might wonder why this happens. There are other frameworks, such as gradient descent, that always seem to decrease the loss initially.

One reason may be the size of the dataset. Given the large amount of training examples my models process, the trainer feeds inputs of a certain batch size to the model. Given that each batch makes

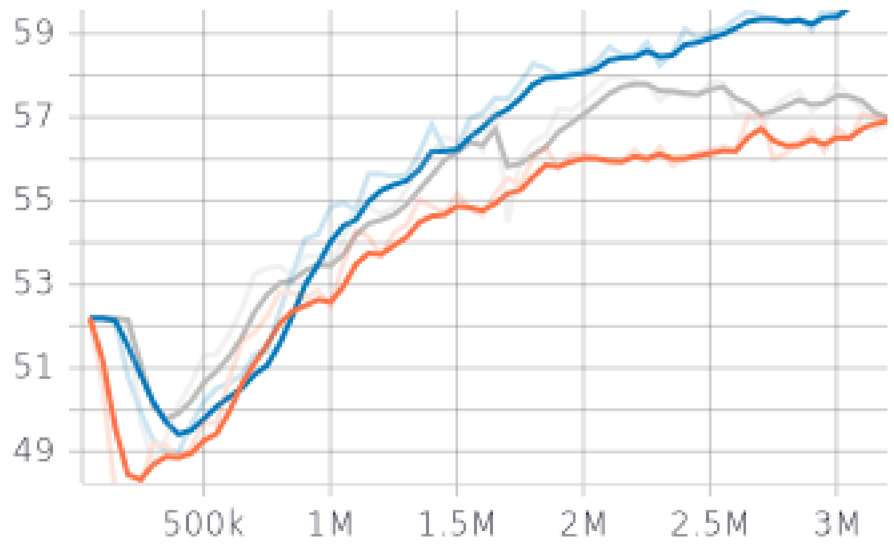

Figure 1: EM scores for the dev set. The orange line is the baseline model, the blue line is BiDAF with character embedding, and the gray line is Dynamic Coattention Network with character embedding. The x axis is the number of iterations the model has completed. The y axis is the score at that iteration.

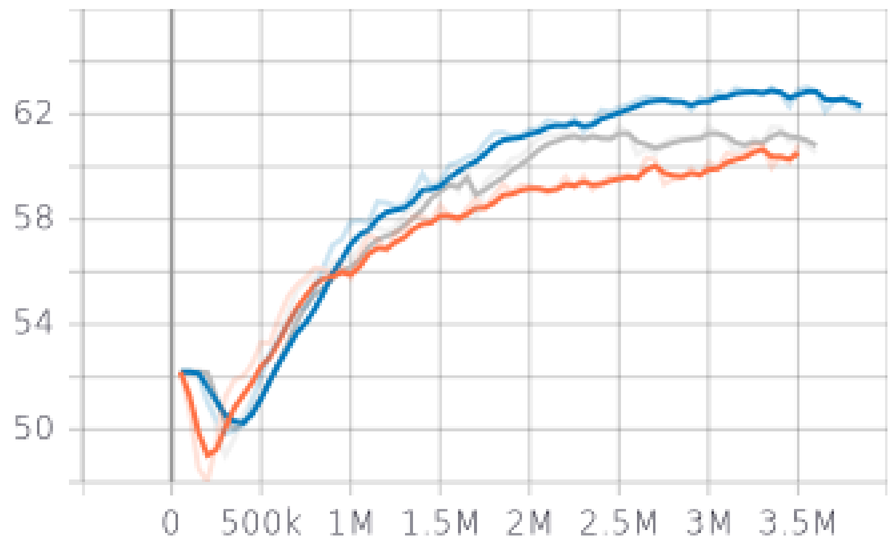

Figure 2: F1 scores for the dev set. The orange line is the baseline model, the blue line is BiDAF with character embedding, and the gray line is Dynamic Coattention Network with character embedding. The x axis is the number of iterations the model has completed. The y axis is the score at that iteration.

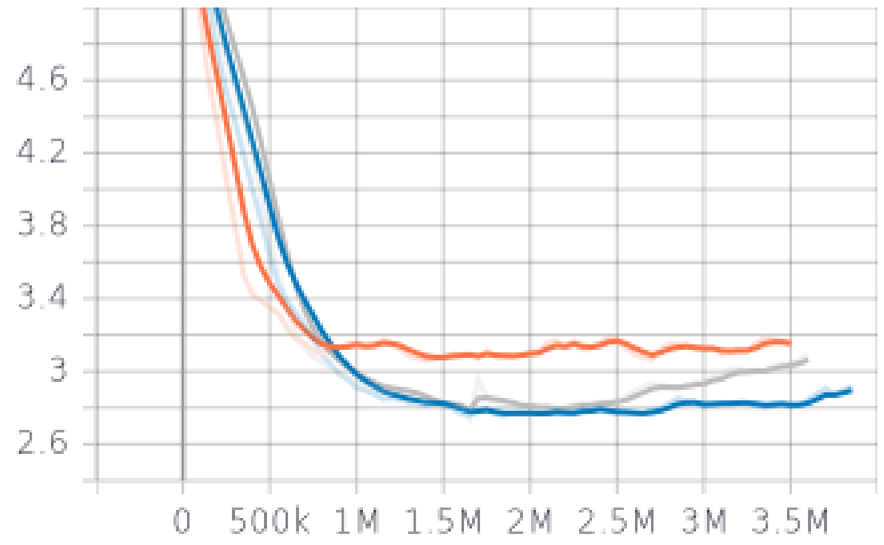

Figure 3: Negative log likelihood scores for the dev set. The orange line is the baseline model, the blue line is BiDAF with character embedding, and the gray line is Dynamic Coattention Network with character embedding. The x axis is the number of iterations the model has completed. The y axis is the score at that iteration.

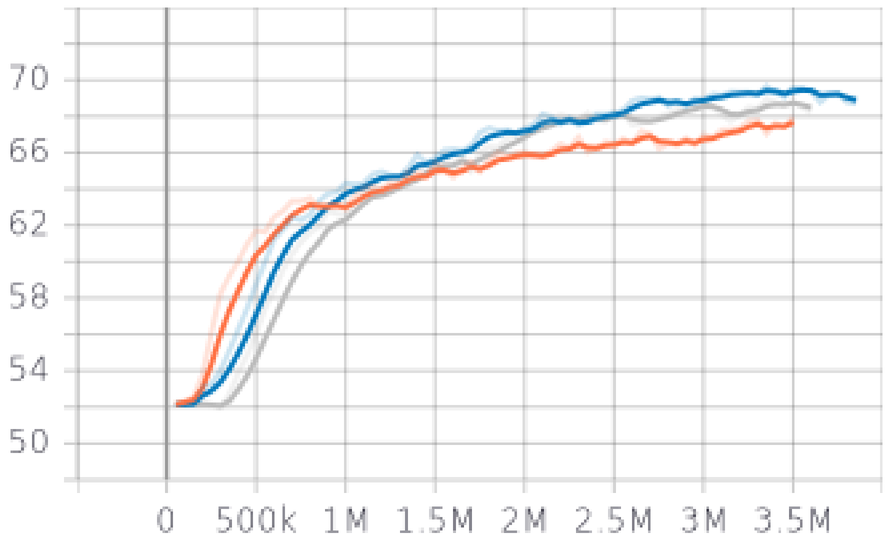

Figure 4: AvNA scores for the dev set. The orange line is the baseline model, the blue line is BiDAF with character embedding, and the gray line is Dynamic Coattention Network with character embedding. The x axis is the number of iterations the model has completed. The y axis is the score at that iteration.

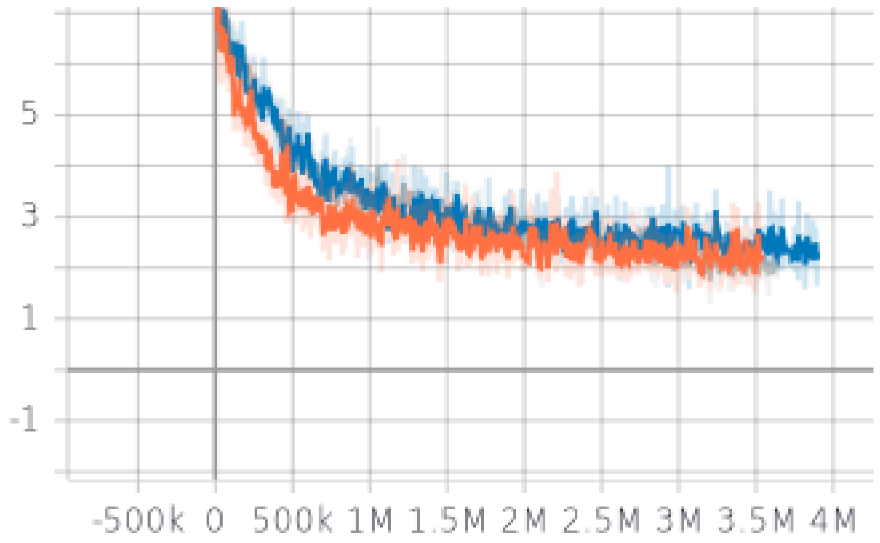

Figure 5: Negative log likelihood scores for the training set. The orange line is the baseline model, the blue line is BiDAF with character embedding, and the gray line is Dynamic Coattention Network with character embedding. The x axis is the number of iterations the model has completed. The y axis is the score at that iteration.

up a very small portion of the SQuAD dataset, each batch probably has properties that are not representative of the entire dataset. Furthermore, due to the nature of Python, the batches may be processed in the same order each epoch, and the contents within each batch may also stay the same. That may explain the seemingly constant variance in the training NLL.

However, there is still a correlation between iteration number and score, as we see in Figures 4, 1, 2, and 3.

6 Conclusion

The BiDAF with character embeddings and the Dynamic Coattention Network both performed better than the baseline in terms of evaluation $F1$ and EM scores by the end of training. Given the resources, my next modification would be to implement span representations delineated in Section 4.5 of the project handout and in the "End-to-end reading comprehension with dynamic answer chunk ranking" paper [6].

To implement Span Representaitons, I will modify the final output layer of the baseline BiDAF model. The Modeling Layer in the Baseline model produces front and backward hidden states for the attention flow at each context position i from $1 < i < n$.

Span representations have a Dynamic Chunk reader that builds a representation for each candidate span. We can simply use the modeling layer as the chunk representation $b_1.....b_n$ described in the project handout.

Let M be the output produced by the modeling layer in the baseline model. We know that $M[:,:,:h//2]$ represent the forward hidden states of the attention flow and $M[:,:,h//2:]$ represent the backward hidden states. Note that h is simply the size of the concatenated forward and backwards hidden states produced by the model layer. Let M_1 equal the forward states and M_2 equal the backward states. Let A be the matrix representing the attention flow before encoding.

In our baseline model,

$$P(start|context, question) = softmax(linear(M) + linear(A)) \tag{9}$$

Using the law of total probability, we know that

$$\sum_{end} P(start, end|context, question) = P(start|context, question) \tag{10}$$

Our definition reframes the probability of $P(start, end|context, question)$ as

$$P(start, end|context, question) = T[start, end] \tag{11}$$

where

$$T = BiLISTM(f(X)) \tag{12}$$

where X is the set of spans from 0 to a given window length. The size of X would be $O(c)$ complexity for small window lengths, but would increase to $O(c^2)$ if the window size gets too large. Therefore, it is critical to keep the window size at a negligible value to preserve runtime.

The function f converts the set of spans to a matrix that can be passed into BiLSTM. The dimensions of $f(X)$ are (batch size, c len, c len, hidden size) where hidden size is the length of the last dimension of the output from the baseline model layout. $f(X)$ would be an upper triangular matrix with respect to dimension 1 and dimension 2. The lower triangle of $f(X)$ would be some constant mask, and the upper triangle and diagonal of $f(X)$ would be the concatenation of $[b_i, b_j]$ where the first index is a forward hidden state and the second index is a backward hidden state. After we get T from the BiLSTM, we can apply softmax with respect to the second dimension so that values across dimension 2 sum to 1. However, given equation 10, values in dimension 2 should actually sum to the corresponding probability represented as log_{p1} in the baseline model. We can achive this by simply using tensor multiplication operations.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] Yang Yu, Wei Zhang, Kazi Saidul Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end reading comprehension with dynamic answer chunk ranking. *CoRR*, abs/1610.09996, 2016.

[7] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.