A Bi-directional Attention Flow (BiDAF) Model for the Stanford Question Answering Dataset (SQUAD)

Helen Xiong
Stanford University
Stanford, CA 94305
hxiong12@stanford.edu

Charles Hale
Stanford University
Stanford, CA 94305
cphalepk@stanford.edu

Abstract

The research community continues to compete to create better models for the Stanford Question Answering Dataset. In this project, we tested several simple adjustments to a high-performing BiDAF model and evaluated their performance benefits. These adjustments include end-conditioned start prediction, policy gradient loss, and character level embeddings.

1 Background

Deep Learning approaches have revolutionized natural language programming (NLP) and have allowed us to feasibly build models that can be applied to much more general tasks than language modeling or sentence parsing. As such, the focus of research has expanded beyond asking the question of how well algorithms can analyze the sentiment of movie reviews or predict which words will come next in a sentence. Researchers are now exploring whether deep learning approaches can "understand" a text, at the level of being able to answer general questions about it.

1.1 Question Format and Dataset

In general, reading comprehension and question answering can be very subjective, as there are often many correct answers for a given question. The Stanford Question Answering Dataset (SQUAD) contains over 100,000 reading comprehension questions. Each question has an associated context and answer. The context is always assumed to contain information that is directly relevant to the question such that the answer can be produced by simply highlighting a section of the context. An example from SQUAD is shown in figure 1.2. This question answering format allows us to greatly reduce the complications associated with the subjectivity of answers, as the problem becomes equivalent to pointing out the starting and ending indices of the answer within the context.

However, even with this rigid answering format, there is still room for subjectivity when choosing which words to include in the answer. In figure 1.2, most would also consider "the San Jose Marriott" to be an appropriate answer. Some of this subjectivity may explain the fact that human performance on this dataset is at about 80% exact match (EM) and 91% F1.

The current best models on the public SQUAD leaderboard have not convincingly surpassed human performance. As such, SQUAD is still considered to be an unsolved problem and active area of research.

1.2 The Problem

To frame the task more formally, we consider a dataset of question, context, and answer pairings, $\{q^{(i)} \in Q, c^{(i)} \in C, a^{(i)} \in A\}_{i=1}^{N}$. q is a vector of words, $w \in \mathcal{V}$, where we consider $\mathcal{V}$ to be

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

an extended vocabulary of the English language. c is the context, which is also a vector of words. The answer $a = (i_{start}, i_{end})$ is a tuple where $0 \leq i_{start} \leq i_{end} \leq |C| - 1$, which correspond to the indices first and last words of the answer within the context. We seek to find a model $f_\theta : (Q \times C) \rightarrow A$, which maximizes the F1 score over the SQUAD test dataset.

Question:
Which hotel did the Broncos use for Super Bowl 50?
Context:
The Panthers used the San Jose State practice facility and stayed at the San Jose Marriott. The Broncos practiced at Stanford University and stayed at the *Santa Clara Marriott*.
Answer:
Santa Clara Marriott

Figure 1: An example from the SQUAD dev set.

2 Approaches

2.1 Attention Baseline

We trained a baseline model which uses a bi-directional RNN encoder to encode both the question and the context. These hidden states are then fed into a basic dot-product attention layer which combines the question and context layers. [1] This attention layer takes the dot product between question and context words and passes the resulting attention scores through a softmax layer to obtain an attention distribution. These attention distributions are used to weight the context hidden states which are then passed into separate softmax layers to produce distributions of start and end indices for the answer.

2.2 Attention over Attention

The first addition to the basic attention model that we implemented was the combined attention layer as described by Cui et. al. [2]. We noticed that the model is relatively simple, and [2] claims it achieved state-of-the-art performance on question answering tasks similar to SQUAD with one attention over attention (AOA) layer.

The AOA layer takes in the context encodings $c_1, ..., c_D$ and the question encodings $q_1, .., q_Q$ and computes a similarity matrix $S \in \mathbb{R}^{D \times Q}$, defined as:

$$S_{ij} = c_i^T q_j \tag{1}$$

We then take the question-to-context attention, A, as the column-wise softmax of S. Additionally, we form the context-to-question attention, B, as the row-wise softmax of S. This tells us the relative importance of each word of the query for each word in the document. We then aggregate these importances by taking the column-wise average, $\beta = \frac{1}{D} \sum_j B_{j,:}$. The result is a single probability distribution representing the overall importance of each query word having taken the document into account. The idea is that A is a basic attention distribution over the context words for each of the query words. We can then weight each query word by its overall relevance to the document to give us a final overall distribution over the importance of each context word. We do this by forming $s = A\beta$. Notice that the columns of A are all distributions, and so is β, so s will also be a distribution.

We formed a basic model using this layer to test its performance. We encoded the context and the query with separate bi-directional RNNs with GRU cells. The context hidden states were then projected with separate linear layers: one for the start index prediction and one for the end index prediction. These two sequences of hidden states were then fed into AOA layers to produce the output distributions over the context positions. We then trained this model using simple cross entropy loss to predict the starting and ending answer indices. We found that this model did not perform significantly better than the baseline.

Since the basic AOA layer does not have any parameters, we believe the model was not powerful enough for the dataset.

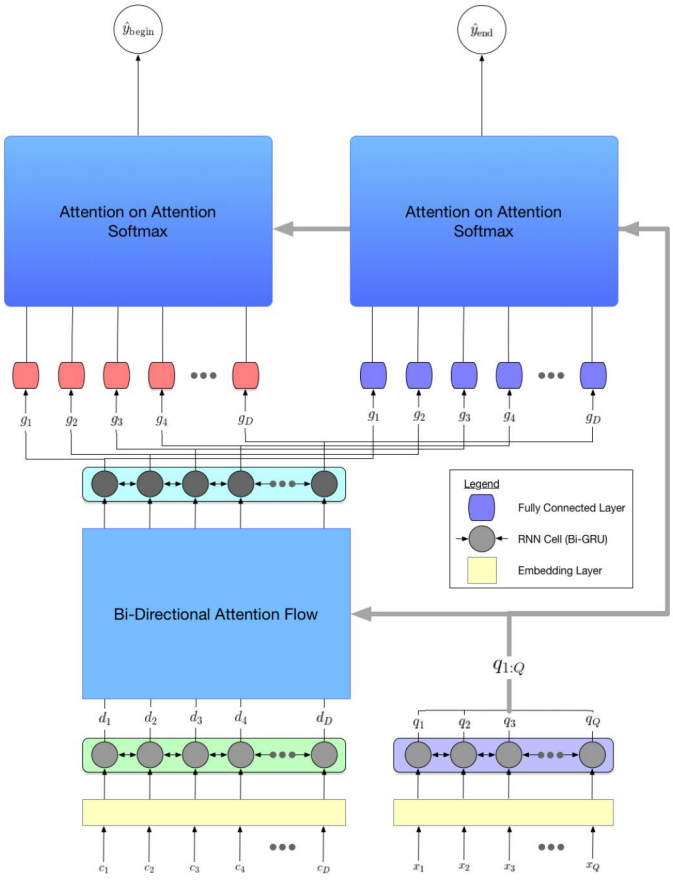

Figure 2: Architecture of BiDAF with AOA attention model

2.3 Bi-Directional Attention Flow

Next, we added a bi-directional attention flow layer (BiDAF) to our model [5]. This layer provides attention weighting from contexts to questions as well as from questions to contexts. Unlike AOA, BiDAF contains trainable parameters, which allows us to calculate a similarity matrix $\boldsymbol{S}$, where $\boldsymbol{S}_{ij}$ is the similarity score between a context and a question hidden state pair, $(\boldsymbol{c}_i, \boldsymbol{q}_j)$. The similarity score is multiplied by a weight vector $\boldsymbol{w} \in \mathbb{R}^{N \times M}$, so $\boldsymbol{S}_{ij} = \boldsymbol{w}^T(\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \circ \boldsymbol{q}_j) \in \mathbb{R}$. Context-to-query attention is calculated using the row-wise softmax of $\boldsymbol{S}$ as the weight for each of the question hidden states $\boldsymbol{q}_j$; i.e. $\alpha^i = \text{softmax}(\boldsymbol{S}_{i,:})$, and $\boldsymbol{a}_i = \sum_{j=1}^{N} \alpha_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h}$. Question-to-context attention is calculated similarly, where we calculate the softmax of the max of every row of $\boldsymbol{S}$, i.e., $\boldsymbol{m}_i = \max_j \boldsymbol{S}_{ij}$, $\beta = \text{softmax}(\boldsymbol{m})$, and $\boldsymbol{c'} = \sum_{i=1}^{N} \beta_i \mathbf{c}_i \in \mathbb{R}^{2h}$. Then, we concatenate the context vectors, context-to-query vectors, the element-wise multiplication of the context and context-to-query vector, and the element-wise multiplication of the context vector and the question-to-context attention to obtain the output of the BiDAF attention layer: $\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c'};] \in \mathbb{R}^{8h}$. Finally, we pass the output of this layer through a bi-directional LSTM layer.

We inserted a BiDAF layer in between the RNN hidden states and the linear layers of our original AOA model. The architecture is shown in figure 2. For the BiDAF model, we were able to implement some computational optimizations that reduced the number of necessary parameters which restricted the size of our overall model. The BiDAF layer led to a significant performance boost which we will describe in the experiments section.

2.4 End Conditioned Start Prediction

Building on our combined BiDAF AOA model, we experimented with conditioned answer choices. The Cross Entropy training loss for a particular answer can be broken up as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(q,c,q)\sim SQUAD}\left[-\log\left(P_\theta(a|q,c)\right)\right] \tag{2}$$

$$\begin{aligned} P_\theta(a|q,c) &= P_\theta(i_{start}, i_{end}|q,c) && (3)\\ &= P_\theta(i_{end}|q,c)P_\theta(i_{start}|i_{end},q,c) && (4)\end{aligned}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{(q,c,q)\sim SQUAD}\left[-\log\left(P_\theta(i_{end}|q,c)\right) - \log\left(P_\theta(i_{start}|i_{end},q,c)\right)\right] \tag{5}$$

In the original case, we treated the two distributions as independent, which is equivalent to the assumption $P_\theta(i_{start}|i_{end},q,c) = P_\theta(i_{start}|q,c)$. But we experimented with allowing our network to condition its choices for the start index based on its choice for the end index.

First, we kept the model the same until after the final RNN, where we have D hidden states, where D is the length of the context, which we denote as $g_1, g_2, ..., g_D$. For the end distribution, we feed these hidden states through a linear layer:

$$\Phi_j^{end} = W^{end}g_j + b^{end} \qquad \forall j = 1, 2, ..., D \tag{6}$$

We then feed these final end feature representations into an AOA Softmax to produce, $\hat{y}_{end}$, as $\hat{y}_{end} = AOA(\Phi_1^{end}, \Phi_2^{end}, ..., \Phi_D^{end}, q_{1:Q})$. This is the same as before.

However, for the start distribution, $\hat{y}_{start}$, we also include the end index choice as an input feature by concatenating $h_{end} = g_{i_{end}}$ to each of the context feature vectors g, before feeding these vectors into the start linear layer. That is:

$$\Phi_j^{start} = W^{start}\begin{bmatrix} g_j \\ h_{end}\end{bmatrix} + b^{start} \qquad \forall j = 1, 2, ..., i_{end} \tag{7}$$

Note that we also drop all indices after i_{end} since we know $i_{start} \leq i_{end}$. Then $\hat{y}_{start} = AOA(\Phi_1^{start}, \Phi_2^{start}, ..., \Phi_{i_{end}}^{start}, q_{1:Q})$.

The inclusion of the end masking for the start distribution makes the start distribution non-differentiable with respect to i_{end}. To train this network, we used the loss

$$\mathcal{L}(\theta) = \mathbb{E}_{(q,c,q)\sim SQUAD}\left[-\log\left(P_\theta(i_{end}|q,c)\right) - \log\left(P_\theta(i_{start}|\hat{i}_{end},q,c)\right)\right] \tag{8}$$

where $\hat{i}_{end} = \underset{j}{\text{argmax}} \quad \hat{y}_j^{end}$. Figure 3

2.5 Policy Gradient Loss

The addition of end-conditioned start prediction introduced a non-differentiability in our model. The most technically sound way of dealing with this non-differentiable loss is to use reinforcement learning (RL) instead of supervised learning to train our model.

For an RL formulation of this problem, we used the F1 score of the prediction as the model's reward. Our prediction network became a policy which takes in the question and context as a "state", and outputs a probability distribution over answers as "actions". Our policy can be rolled out over one episode by producing a single answer prediction. The policy gradient theorem as introduced in [7] states that we can improve our policy network (in the sense that it will learn to achieve better rewards) by following a policy gradient:

$$\frac{\partial V^\theta}{\partial \theta} = \frac{\partial \log(\pi_\theta(a|s))}{\partial \theta} R(s,a) \tag{9}$$

Notice that this equation simply scales the log probabilities of answer choices made in proportion to the rewards we observe from those choices. In that sense, the RL formulation differs from supervised learning in that it does not utilize the correct answer choice in calculating the loss, only the reward from our answer choice. Furthermore, the reward can be non-differentiable as well, meaning we can use the F1 score directly.

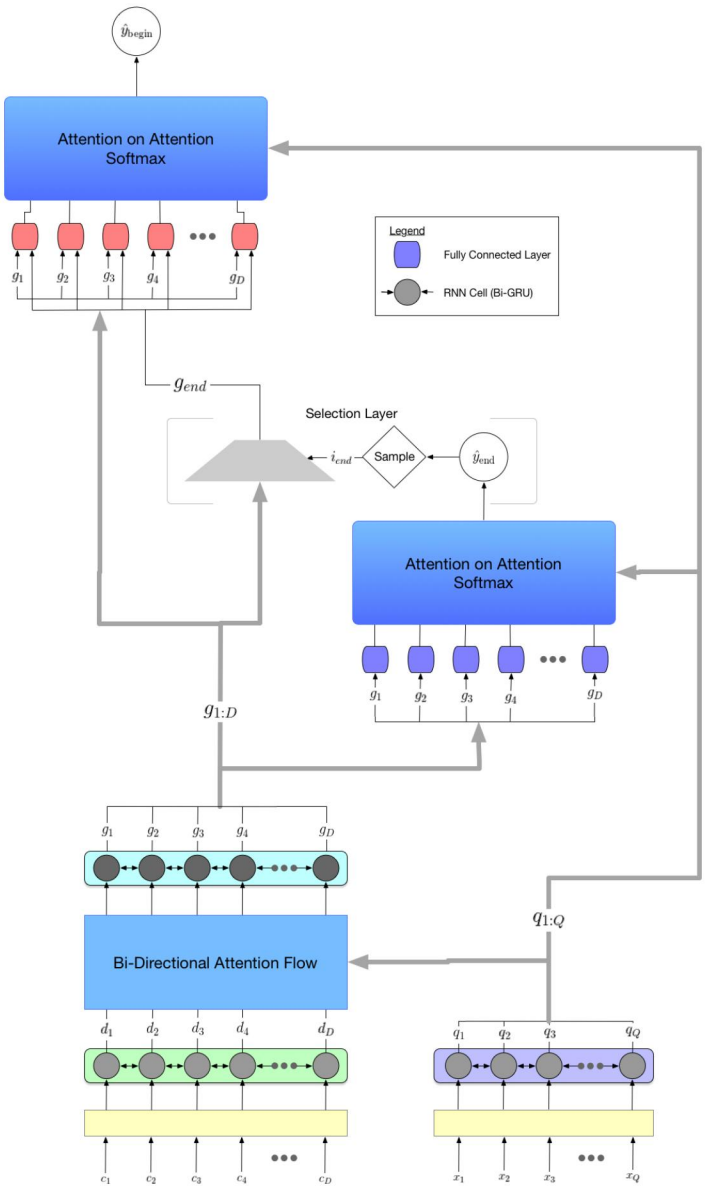

Figure 3: Architecture of BiDAF with AOA attention model and conditioned predictions

In order to evaluate our policy, we feed it examples from our dataset, sample from its answer distribution, and record the F1 scores between the sampled answers and the true answers.

$$(q, c, a) \sim SQUAD, \qquad \hat{a} \sim P_\theta(\cdot|q, c), \qquad r = F1(\hat{a}, a) \tag{10}$$

We can then reinforce our policy to make higher rewarding actions more probable by minimizing

$$\mathcal{L}(\theta) = -\mathbb{E}_{q,c,r,\hat{a}} \left[r \log \left(P_\theta(\hat{i}^{end}|q, c) \right) + r \log \left(P_\theta(\hat{i}^{start}|\hat{i}^{end}, q, c) \right) \right] \tag{11}$$

2.6 Other improvements

We explored some other improvements, but did not add them to our final model due to memory and size constraints. Nonetheless, they either performed as well as our baseline model or improved upon its performance.

2.6.1 Character-level embeddings

To analyze results at the character level and increase robustness against unknown tokens, we implemented character-level embeddings as described in the project handout and by Kim et. al. [3]

The idea behind character-level CNN's is as follows: for each context and question, we encode each word in the passage as a list of character IDs. Thus, each context can be represented by a matrix of dimension context size by word length, where the maximum word length is a hard-coded parameter. Then, we use these character IDs to look up trainable character embedding vectors to obtain the character embedding representation of each word in a context. We pass the character embedding of each word through 1-d convolutional filter with a set window size (also a user-defined parameter) to filter the character embeddings. Finally, we use max-pooling across the convolutional outputs for each character in the word to obtain the character-level representation of the word. Finally, we concatenate the character-level representation of the word to the word embedding, and we use these concatenated embeddings to train the neural network. We did not get a performance boost from character-level embeddings using the same hyperparameters. However, the loss and EM/F1 scores for character-level embedding were unstable relative to the performance of the regular word embedding model, which suggests that we needed to use a smaller learning rate to optimize performance for character-level embeddings.

2.6.2 Dynamic Co-attention

We explored using Dynamic Co-Attention [8] as an alternative to BiDAF. We start with context hidden states $\boldsymbol{c}_1, ..., \boldsymbol{c}_N \in \mathbb{R}^l$ and question hidden states $\boldsymbol{q}_1, ..., \boldsymbol{q}_M \in \mathbb{R}^l$. First, the question hidden states are passed through a linear layer with the tanh nonlinearity to obtain projected question hidden states $\boldsymbol{q'}_j \in \mathbb{R}^l \quad \forall j \in \{1, ..., M\}$. Then, trainable sentinel vectors for both context and questions are added to the set of context and question vectors. Next, the affinity matrix $\boldsymbol{L} \in \mathbb{R}^{N+1 \times M+1}$ is calculated, with each element $\boldsymbol{L}_{ij} = \boldsymbol{c}_i^T \boldsymbol{q}_j \in \mathbb{R}$. Context-to-projected question and question-to-context attention matrices are obtained similarly to the method for BiDAF. Finally we obtain second-level attention outputs $\boldsymbol{s}_i = \sum_{j=1}^{M+1} \alpha_j^i \boldsymbol{b}_j \in \mathbb{R}^l \quad \forall i \in \{1, ..., N\}$. The final step in coattention is to concatenate the second-level attention outputs with the first-level context-to-question outputs, and feed the concatenations through a bidirectional LSTM to obtain coattention encodings:

$$\{\boldsymbol{u}_1, ..., \boldsymbol{u}_N\} = \text{biLSTM}(\{[\boldsymbol{s}_1; \boldsymbol{a}_1], ..., [\boldsymbol{s}_N; \boldsymbol{a}_N]\})$$

. Coattention gave us a performance boost over our baseline attention-over-attention model, but not as much as BiDAF. However it does support our hypothesis that weighted attention mechanisms are very important for improving the performance of the model.

3 Experiments

3.1 Data analysis

We examined the lengths of contexts and questions in the dev set. We discovered that the preset context length and question length were much longer than the majority of contexts and questions, so we reduced the training context size to 250 from 60, and the training question size to 25 from 30.

When character-level encodings were added to the model, a similar analysis showed that maximum word length could be limited to 20 characters. The high frequency of short tokens is due to tokens such as "a", "the", and "'s" for possessives. 3.1

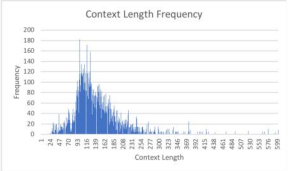

Figure 4: Context length frequencies

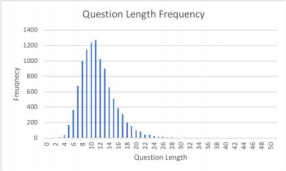

Figure 5: Question length frequencies

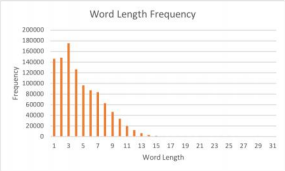

Figure 6: Word length frequencies

3.2 Results

Dev EM/F1 Scores from AoA, AoA + BiDAF, Baseline Attention and BiDAF

Model	Best Dev EM/F1/Iterations
Baseline Attention	0.251/0.356/14.5k
Attention-on-Attention	0.252/0.358/16.5k
AoA + BiDAF	0.421/0.574/10.5k
Baseline Attention + BiDAF	0.493/0.642/8.5k

3.3 Architecture Details

All linear layers had $\tanh$ activation functions. This is because the AOA layers computed the result based on dot products with the query hidden states, which were also the result of $\tanh$ activations. We also experimented with ReLU activations and found it made little difference.

The hidden sizes of the context and query RNNs were always equal. The RNN after the BiDAF layer was typically chosen to be 4 or 5 times the size of the context and query RNNs.

3.3.1 Hyperparameter Tuning

We found the best hyperparameters to use with our model through a combination of trial and error, including a script to run a hyperparameter search on our better-performing models. This search provided optimal values for learning rate (0.00183). We used ADAM to optimize our loss. Memory limitations on the GPU limited the size of parameters such as batch size and hidden layer size. For most of our training, we used batch size = 50 and hidden size = 64.

3.4 Architectural variants

We experimented with several variants of our conditioned answer model, including:

1. An additional RNN layer for the context encodings, after the BiDAF layer.
2. Larger embeddings.
3. An additional RNN layer for both the context encodings and the question encodings (shared weights), after the embedding layer.
4. Not training the embeddings.
5. An additional RNN layer for the query encodings, after the BiDAF layer.
6. Larger hidden sizes at every layer.

The dev F1 and EM scores for of each of these variants are shown in figure [**?**].

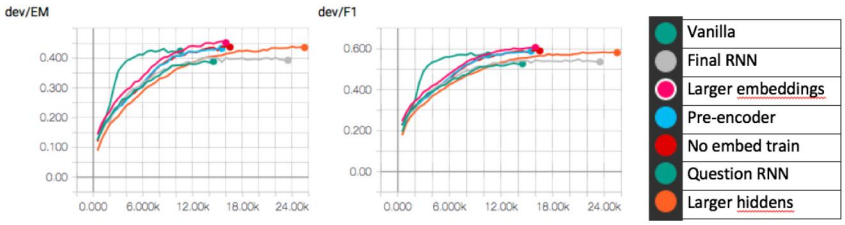

Figure 7: The dev F1 and EM scores of each of the model variants. Not all of these variants were trained with the exact same hyperparameters. However, empirically we found that the learning rate, regularization parameters, etc, had little effect on the asymptotic performance of the model.

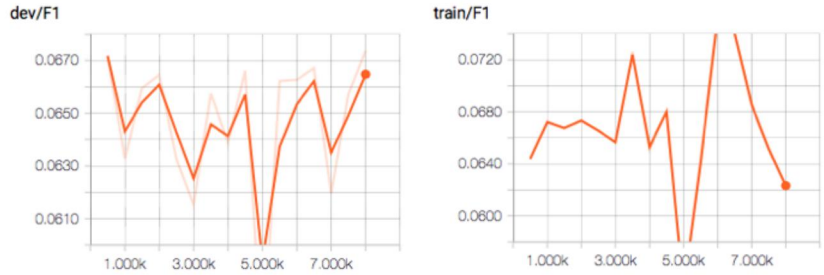

Figure 8: The dev and training F1 scores of the RL model.

4 Error Analysis

4.1 Qualitative

We found that our model made some common mistakes on certain types of questions:

1. Ignoring surrounding words
   - `QUESTION: how many hours can one expect to ride the train from newcastle to king's cross ?`
   - `TRUE ANSWER: about three`
   - `PREDICTED ANSWER: three`
   - `F1 SCORE ANSWER: 0.667`
   - `EM SCORE: False`

   In some cases, the model predicts an answer that is very close but not an exact match with the human answer. This decreases F1 and EM scores, even though the predicted answer largely encompasses the meaning of the true answer. The model may need to be more flexible in order to learn about accompanying words. This also goes to show that the question-answering task is an inherently messy one - even a human could not achieve a perfect score on the tasks so it is impressive that a machine can do this.
2. Poor question understanding
   - `QUESTION: what two member nations of the holy roman empire received huguenot refugees ?`
   - `TRUE ANSWER: electorate of brandenburg and electorate of the palatinate`
   - `PREDICTED ANSWER: england , wales , scotland , denmark , sweden , switzerland , the dutch republic`
   - `F1 SCORE ANSWER: 0.000`
   - `EM SCORE: False` Again, the model demonstrates its limited vocabulary and capacity to understand questions. It either does not understand "electorate" as a "place" entity, or it has ignored the adjective "two" in the question statement itself. We believe this

example illustrates the importance of having both question-to-context attention as well as context-to-question attention.

3. Entity Misclassification

   - `CONTEXT: extra pay is also given for teaching through the irish language , in a gaeltacht area or on an island .`
   - `QUESTION: what does teaching on an island result in ?`
   - `TRUE ANSWER: extra pay`
   - `PREDICTED ANSWER: gaeltacht area`
   - `F1 SCORE ANSWER: 0.000`
   - `EM SCORE: False`

   This example illustrates the complexities of language modeling. Here, model seeks an answer to the query word "what", and presumably seeks an answer that could be classified as a noun, such as "extra pay" or "gaeltacht area". However, the model is merely trying to match prepositions (e.g. "in a gaeltacht area"). These types of errors belie problems in word-level classification and challenges in training machines to truly understand language. Perhaps a model with a phrase-based recognition system would perform better on these types of errors.

4.2 Quantitative

We made several plots of average dev F1/EM score vs context length or query length and saw no obvious correlation between the variables. We believe this is a good sign for the models because correlations here would expose an obvious modeling limitation.

In general, we believe it would be much easier to narrow down on the exact weaknesses of our models if we achieved higher performance.

5 Discussion

As shown from our experimental results, all of the model variants achieved very similar performance. We conclude that most of the model adjustments are likely not worth additional computation required to evaluate them.

The vanilla BiDAF layer provided the greatest power in all of our models. Given the performance of our original model based on the AOA layer, we believe that the AOA model limited the performance of our model, perhaps by limiting the size of our other parameters or diluting the strength of the attention signal.

Future directions of research should focus on refining or expanding the notions of weighted attention.

5.1 Policy Gradient Results

Empirically, we saw that training our model with a reinforcement learning approach was much more volatile than using simple supervised learning, see figure [**?**]. This likely due to the fact that supervised learning directly uses the correct answer to form the gradient update. Conversely, RL algorithms require an agent to explore its options and would need many more context, question, answer, reward tuples in order to learn the correct answers. One-step policy rollouts simply do not scale well with the dataset, as they have very high variance. We believe other RL approaches such as ReasoNet [6] are successful because they make use of the policy many times per episode, which allows them to form stronger gradients.

The high variance of the gradient is a known issue with the REINFORCE algorithm and there is extensive literature on techniques for reducing the variance, such as estimating the performance of an action with respect to a baseline. However, we saw it best not to pursue these approaches extensively further, as they could have exhausted all our time and resources.

Acknowledgments

We would like to thank the CS 224N staff, especially Abigail See, for their direction and guidance in this project, as well as the rest of the Stanford NLP group for providing the starter code and infrastructure for the SQUAD competition. [4]

References

[1] Cs 224n default final project: Question answering. 2018.

[2] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *CoRR*, abs/1607.04423, 2016.

[3] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

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

[6] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *CoRR*, abs/1609.05284, 2016.

[7] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 1057-1063, 2000.

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