# Question Answering on the SQuAD Dataset

**Ivan Suarez Robles**
Department of Computer Science
Stanford University
isuarezr@stanford.edu

**Stephanie Tang**
Department of Computer Science
Stanford University
svtang@stanford.edu

**codalab usernames: isuarezr, svtang**

## Abstract

Machine reading comprehension involves answering a question or query about a context paragraph. To accomplish this task, the system needs to model complex interactions between the query and the context paragraph. We reimplement a modified version of the Bidirectional Attention Flow model and augment it with a simplified self-attention layer, along with a few other minor changes for making predictions. This model achieved 74.779% F1 and 64.002% EM on the dev set and 75.137% F1 and 64.901% EM on the test set.

## 1 Introduction

Recent end-to-end deep neural network research and development has been successful in specific Natural Language Processing tasks; however, researchers are still exploring possible solutions for reading comprehension and question answering tasks. Machine reading comprehension involves answering a question or query about a context paragraph. To accomplish this task, the system needs to model complex interactions between the query and the context paragraph, such as coreference resolution, commonsense reasoning, and causal relations [4].

In this paper, we train several models on the Stanford Question Answering Dataset (SQuAD) [5]. We reimplement a modified version of the Bidirectional Attention Flow model and augment it with a simplified self-attention layer, along with a few other minor changes for making predictions.

## 2 Background/Related Work

The best performing models on the SQuAD leaderboard [7] makes use of some attention mechanism. Attention allows a model to focus on the most relevant subset of the context paragraph and the most relevant subset of a question in order to give a better answer.

One top performing model is the Bidirectional Attention Flow model (BiDAF) [1], a hierarchical multi-stage attention mechanism. BiDAF uses a bidirectional attention flow mechanism to obtain a query-aware representation of the context and a context-aware representation of the query.

Another top performing model is R-net, which uses gated attention-based recurrent networks to obtain query-aware representation of the context and a self-attention mechanism to refine the context representation again itself [2]. The self-attention layer allows the model to encode information from the whole context.

Our model structures are taken and modified from these previous works done on the SQuAD dataset. We further discuss these modified structures in section 3.

## 3 Approach

Our machine comprehension model is comprised of the following seven layers:

### 3.1 Word Embedding Layer

Each word is mapped to a high-dimensional vector space using pretrained GloVe vectors [6]. Let $C_1, ..., C_m$ represent the words in the context paragraph with $m$ words and $Q_1, ..., Q_n$ represent the words in the query with $n$ words. The output would be $\{\boldsymbol{c}_1, ..., \boldsymbol{c}_m\}$ for the context paragraph where $\boldsymbol{c}_i \in \mathbb{R}^d$ and $\{\boldsymbol{q}_1, ..., \boldsymbol{q}_m\}$ for the query where $\boldsymbol{q}_i \in \mathbb{R}^d$. For simplicity, we group the context and query vectors into $\boldsymbol{C} \in \mathbb{R}^{d\times m}$ and $\boldsymbol{Q} \in \mathbb{R}^{d\times n}$, respectively.

### 3.2 Contextual Embedding Layer

We use a bidirectional RNN with LSTM cells on top of the word embeddings provided from the previous layer to model the interactions between words. We concatenate the two outputs of the bidirectional RNN for each context word vector. We obtain $\boldsymbol{H} \in \mathbb{R}^{2h\times m}$ from the context word vectors $\boldsymbol{C}$ and $\boldsymbol{U} \in \mathbb{R}^{2h\times n}$ from the query word vectors $\boldsymbol{Q}$.

### 3.3 Bidirectional Attention Flow Layer

This layer is responsible for capturing the interactions between the context and the query. We use the approach discussed in Seo et. al. [1].

The inputs are $\boldsymbol{H}$ and $\boldsymbol{U}$ from the previous layer. We compute attentions in two directions: from context to query and from query to context. To do so, we first compute a similarity matrix $\boldsymbol{S} \in \mathbb{R}^{m\times n}$ from the context and query matrices that will be later used to compute attention in both directions. $\boldsymbol{S}$ is defined such that for the $i$th context word ($\boldsymbol{H}_{:i}$) and the $j$th query word ($\boldsymbol{U}_{:j}$), their similarity, $\boldsymbol{S}_{ij}$, is as follows:

$$S_{ij} = \boldsymbol{w}_S^T[\boldsymbol{H}_{:i}; \boldsymbol{U}_{:j}; \boldsymbol{H}_{:i} \circ \boldsymbol{U}_{:j}]$$

$\boldsymbol{w}_S \in \mathbb{R}^{6h}$ is a trainable weight vector.

Once the similarity matrix is derived from $\boldsymbol{H}$ and $\boldsymbol{U}$, we now compute attention in both directions:

1. **Context-to-query Attention** (C2Q): We first apply a row-wise softmax on $\boldsymbol{S}$ to compute the C2Q attention distributions $\alpha^{(i)}$. We then compute the C2Q attention outputs $\boldsymbol{a}_i$ by taking a weighted sum on the query vectors from $\boldsymbol{U}$ using the C2Q attention distributions as weights:

$$\alpha^{(i)} = \text{softmax}(S_{i:}) \in \mathbb{R}^n \qquad \forall i \in \{1, ..., m\}$$
$$\boldsymbol{a}^{(i)} = \sum_{j=1}^{n} \alpha_j^{(i)} U_{:j} \in \mathbb{R}^{2h} \qquad \forall i \in \{1, ..., m\}$$

2. **Query-to-context Attention** (Q2C): We first define a vector $\boldsymbol{s} \in \mathbb{R}^m$ where the $i$th element in $\boldsymbol{s}$ is the maximum value taken across the $i$th row in $\boldsymbol{S}$. We then take the softmax of $\boldsymbol{s}$ to generate an attention distribution $\beta \in \mathbb{R}^m$ over the context locations. We use $\beta$ to compute a weighted sum over the context locations given by $\boldsymbol{H}$ to yield the Q2C attention output, $\boldsymbol{c'}$:

$$s_i = \max_j S_{ij} \in \mathbb{R} \qquad \forall i \in \{1, ..., m\}$$
$$\beta = \text{softmax}(\boldsymbol{s}) \in \mathbb{R}^m$$
$$\boldsymbol{c'} = \sum_{i=1}^{m} \beta_i H_{:i} \in \mathbb{R}^{2h}$$

With the C2Q and Q2C attention outputs, we compute a blended representation for the context words, $\boldsymbol{B} \in \mathbb{R}^{8h \times m}$. For the $i$th context word, its blended representation is as follows:

$$B_{:i} = [H_{:i}; \boldsymbol{a}^{(i)}; H_{:i} \circ \boldsymbol{a}^{(i)}, H_{:i} \circ \boldsymbol{c'}]$$

### 3.4 First Modeling Layer

We pass $\boldsymbol{B}$ into a bidirectional RNN comprised of LSTM cells in a similar fashion of the Contextual Embedding Layer to generate what we call the BiDAF representation of the context words, $\hat{\boldsymbol{B}} \in \mathbb{R}^{2h \times m}$.

### 3.5 Self Attention Flow Layer

This layer is responsible for capturing the interactions within the query-aware context. We use an approach similar to that discussed in R-net [2].

The input is the BiDAF representation of the context words, $\hat{\boldsymbol{B}}$, from the previous layer. We use multiplicative attention where each context word attends to all other context words to generate $\boldsymbol{E}$. We then compute the self attention distributions $\gamma^{(i)}$ from $\boldsymbol{E}_{:i}$ for each context word at position $i$. Afterwards, we use the self attention distributions to take a weighted sum of the BiDAF representation of the context words to produce a self-aware blended representation for each context word, $\boldsymbol{z}^{(i)}$:

$$\begin{aligned}
\boldsymbol{E} &= \hat{\boldsymbol{B}}^T \boldsymbol{W}_E \hat{\boldsymbol{B}} \\
\gamma^{(i)} &= \text{softmax}(E_{:i}) \qquad \forall i \in \{1, ..., m\} \\
\boldsymbol{z}^{(i)} &= \sum_{j=1}^{m} \gamma_j^{(i)} \hat{B}_{:i} \in \mathbb{R}^{2h} \qquad \forall i \in \{1, ..., m\}
\end{aligned}$$

$\boldsymbol{W}_E \in \mathbb{R}^{m \times m}$ is a trainable weight matrix.

### 3.6 Second Modeling Layer

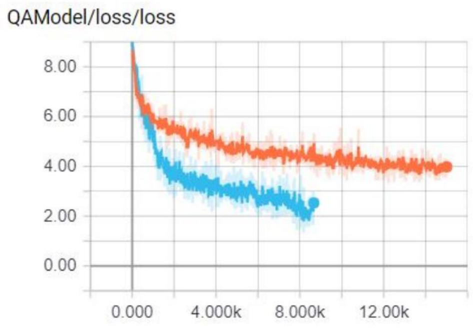
We pass the self-aware blended representations for each context word into a bidirectional RNN comprised of LSTM cells in a manner similar to that of the Contextual Embedding Layer to generate what we will call the self aware representation of the context words, $\boldsymbol{Z} \in \mathbb{R}^{2h \times m}$.

### 3.7 Output Layer

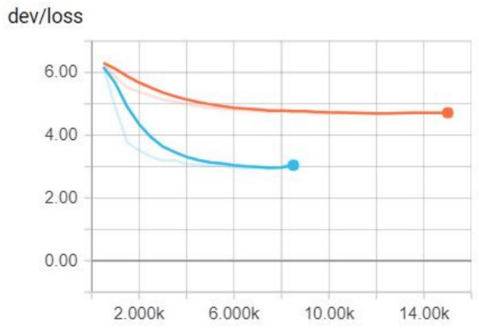
The input is $\boldsymbol{F} = [\boldsymbol{H}; \hat{\boldsymbol{B}}, \boldsymbol{Z}] \in \mathbb{R}^{6h \times m}$ where we concatenate by column. $\boldsymbol{F}$ is fed through a fully connected layer and then passed through a ReLU non-linearity to generate $\boldsymbol{V} \in \mathbb{R}^{h \times m}$. We then perform the following procedure to compute both the probability distribution of the start index and the probability distribution ofthe end index: we take the output of the ReLU non-linearity and we perform a linear downprojection; we then compute the softmax to end up with the final distributions similar to the baseline:

$$\begin{aligned}
V_{:i} &= \text{ReLU}(\boldsymbol{W}_V F_{:i} + \boldsymbol{b}_V) \in \mathbb{R}^h \\
\text{logits}_i^{\text{start}} &= \boldsymbol{w}_{\text{start}}^T V_{:i} + b_{\text{start}} \in \mathbb{R} \\
\boldsymbol{p}^{\text{start}} &= \text{softmax}(\text{logits}^{\text{start}}) \in \mathbb{R}^m
\end{aligned}$$

$\boldsymbol{W}_V \in \mathbb{R}^{h \times 6h}, \boldsymbol{w}_{\text{start}} \in \mathbb{R}^h, b_{\text{start}} \in \mathbb{R}$ are all trainable weights and biases. We perform a similar computation for $\boldsymbol{p}_{\text{end}}$.



(a) Training loss



(b) Dev loss

Figure 1: Comparison of baseline with final model

### 3.8 Prediction

For predicting, after computing $\boldsymbol{p}^{\text{start}}, \boldsymbol{p}^{\text{end}}$ of a given context and query, we compute the positions $l^{\text{start}}, l^{\text{end}} = \arg\max_{\text{i, j}} p_i^{\text{start}} * p_j^{\text{end}}$ such that $i \leq j$. We enforce the restriction $i \leq j$ so that we are always able to get a valid span of the context. By using dynamic programming, we are able to compute $l^{\text{start}}, l^{\text{end}}$ in linear time.

## 4 Experiments

As we built our final ensemble model, we trained several models with different combinations of the structures described above as well as slightly modified versions of those structures. We evaluate how well each model performs by tracking its train loss, dev loss, dev F1 score, dev exact match (EM), and looking at specific examples where the model failed. Additionally, we also paid close attention to the virtual machine's memory usage.

### 4.1 Datasets

We train several models on a recent Machine Comprehension and Question Answering dataset, the Stanford Question Answering Dataset (SQuAD). SQuAD is a collection of context-question-answer triplets. The goal is to predict the start and end index of the answer within the context paragraph.

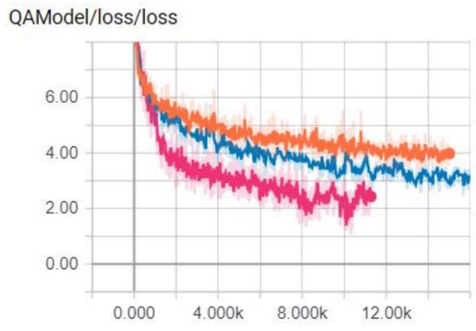
### 4.2 Experimental Model Configurations

#### 4.2.1 Baseline

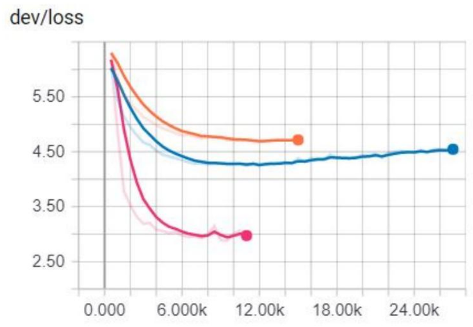
The provided starter code includes a baseline model. The baseline comprises of a RNN encoder layer that encodes the context and question hidden states, an attention layer that combines the context and question representations, and an output layer that applies a fully connected layer and two softmax layers. In Figure 1, we noticed that the baseline model (orange) train F1 score was about 0.58, yet its dev F1 was closer to 0.39. This discrepancy suggests that the baseline is currently overfitting the train data.

#### 4.2.2 Simplified BiDAF (sBiDAF)

This configuration replaces the baseline attention with the BiDAF layer as described in the project specifications. From Figure 2, the simplified BiDAF version (dark blue) does better than baseline (orange), but not by a significant margin. Indeed, after training, we obtain an F1 score of 0.45 and an EM score of 0.33, which is far less than the scores obtained from Seo et. al. [1]. The observation leads us to believe that the modeling layer presented in their work may be important in boosting performance.

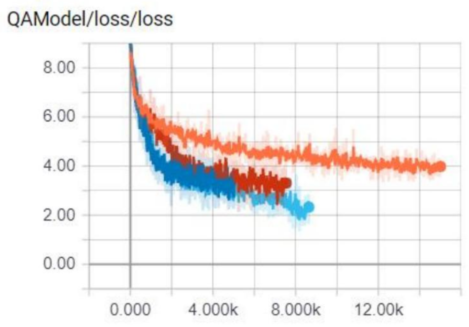


(a) Training loss

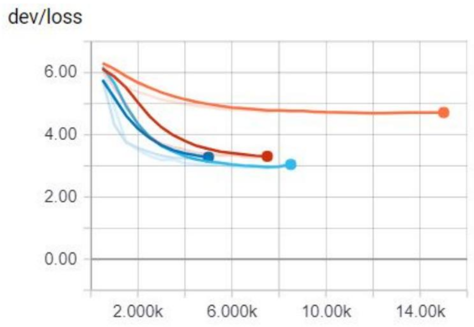


(b) Dev loss

Figure 2: Comparison of simple BiDAF with baseline and final model



(a) Training loss



(b) Dev loss

Figure 3: Comparison of BiDAF + Self Attention with baseline and final model

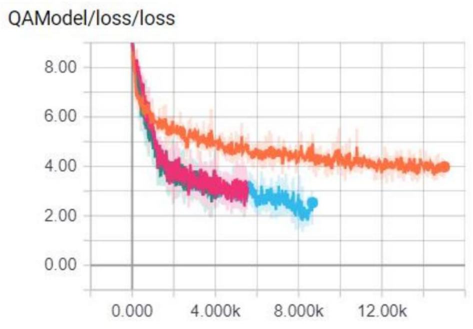
#### 4.2.3 Simplified BiDAF with Self-Attention (sBiDAF + SA)

For these experiments in Figure 3, we inserted a self-attention (SA) layer after the BiDAF layer along with a second bidirectional RNN layer after the SA layer as outlined in the R-Net paper [2]. When implementing SA, we found that the additive attention required more memory than the virtual machine could supply, so we switched out the additive attention for a simpler attention. We initially tried a basic dot product attention (dark blue) and multiplicative attention (red). The dot product attention with simplified BiDAF (sBiDAF + dot.SA) performed better than with multiplicative attention (sBiDAF + mul.SA), which is surprising since multiplicative attention has more parameters and theoretically should be more flexible (see Table 1). However, with more iterations, multiplicative attention performed as well as where the dot product attention model was cut off. We went with multiplicative attention since theoretically it is at least as powerful as dot product attention; i.e., by setting the weight matrix $\boldsymbol{W}$ to be the identity matrix, multiplicative attention would reduce to dot product attention.

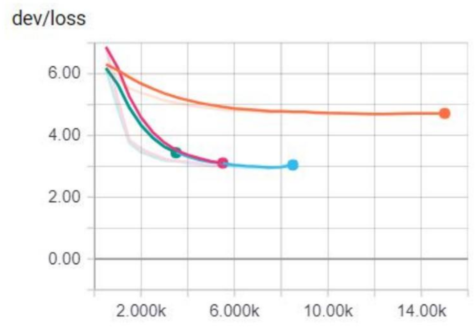
Finally, we inserted another bidirectional RNN layer between the BiDAF and self-attention layers, yielding the submitted model (light blue), which outperforms the experimental models we've tested so far with an F1 score of 0.66 and an EM score of 0.51.

#### 4.2.4 BiDAF with Multiplicative Self-Attention + Additional Layer (BiDAF + mul.SA + RNN)

As we looked closer at the full BiDAF model [1], we noted that our simplified BiDAF uses only one bidirectional RNN for the modeling layer after the BiDAF layer. These experiments in Figure 4 try adding an extra RNN into our current best model: one in the First Modeling Layer, yielding two bidirectional RNNs in the layer (pink; BiDAF + mul.SA + model.RNN) and one in the output layer in a similar manner to Seo et. al [1] (green; BiDAF + mul.SA + output.RNN). However, as

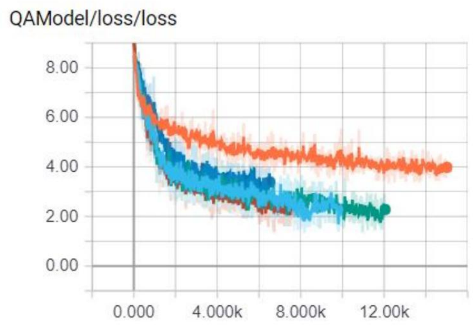


(a) Training loss

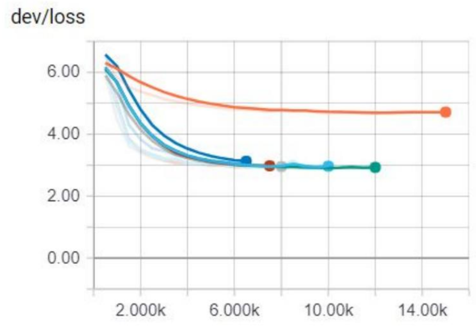


(b) Dev loss

Figure 4: Comparison of BiDAF + Self Attention + Extra Layer with baseline and final model



(a) Training loss



(b) Dev loss

Figure 5: Comparison of Various Hyperparameters with baseline and final model

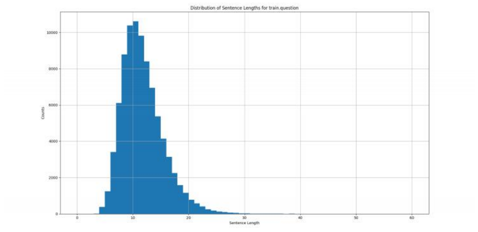
we trained these models, it's performance was only comparable to our submitted model (see Table 1), yet took much longer to train (we stopped output.RNN at 3.5k iterations and model.RNN at 5k iterations). Thus, we did not include the extra layer, even though supposedly the model would be more flexible. This may still have potential with more time and computational power.

#### 4.2.5 Prediction

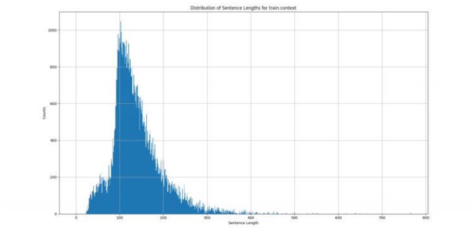
The baseline makes predictions for the start and end positions, $l^{\text{start}}$ and $l^{\text{end}}$, based on $\boldsymbol{p}^{\text{start}}$ and $\boldsymbol{p}^{\text{end}}$ independently. However, since the predictions are made independently, it is quite possible for $l^{\text{start}} > l^{\text{end}}$ to occur, yielding an empty answer. After switching to our mechanism as described in Section 3.8 (Prediction), we guarantee that $l^{\text{start}} \leq l^{\text{end}}$. The change yielded approximately an increase of 0.02 in F1 and EM scores for our model.

#### 4.2.6 Hyperparameter Search

With our model mostly assembled, we experimented with a few different values for drop out (0.25 = dark blue; 0.10 = red), embedding size (200 = grey), and context length (450 = green). The dropout values did about as well or worse as our final model (light blue), so we did not keep these changes. The default embedding size is 100, but with a larger embedding size, the vectors could be more complex, so we tried 200. Increasing the embedding size increases memory usage, and this model did approximately as well as our final model. Next, when looking at the context length and question length distributions (Figure 6), we noticed that most contexts are less than 400 words, while the default model uses a length of 600; thus, we tried an context length of 450 to reduce the memory usage. This performed about as well as our final model (Figure 5).



(a) Query Length Distribution



(b) Context Length Distribution

Figure 6: Distributions of Contexts and Queries Lengths from Training Data

### 4.3 Summary of Results

The table below summarizes the dev F1 and dev EM scores from Tensorboard during training for each of the experimental configurations. Each cell presents these scores as [F1 | EM] at the specified iteration. The dev and test leaderboard results are also included for some configurations. Due to time constraints, not all model configurations were run until 15k iterations.

Table 1: F1 + EM scores during training

| Model Config. | 5k It. | 7.5k It. | 10k It. | 15k It. |
|---|---|---|---|---|
| Baseline | 0.3635 \| 0.2579 | 0.3844 \| 0.2760 | 0.3947 \| 0.2874 | 0.3970 \| 0.2896 |
| sBiDAF | 0.4304 \| 0.3138 | 0.4482 \| 0.3287 | 0.4498 \| 0.3312 | 0.4546 \| 0.3339 |
| sBiDAF + dot.SA | 0.6149 \| 0.4630 | - | - | - |
| sBiDAF + mul.SA | 0.5916 \| 0.4437 | 0.6144 \| 0.4652 | - | - |
| BiDAF + mul.SA | 0.6500 \| 0.4981 | 0.6639 \| 0.5147 | 0.6776 \| 0.5263 | - |
| BiDAF + mul.SA + model.RNN | 0.6631 \| 0.5051 | - | - | - |
| dropout = 0.25 | 0.6445 \| 0.4862 | - | - | - |
| dropout = 0.10 | 0.6527 \| 0.5166 | - | - | - |
| embed size = 200 | 0.6604 \| 0.5156 | 0.6818 \| 0.5244 | - | - |
| context length = 450 | 0.6458 \| 0.5047 | 0.6708 \| 0.5170 | 0.6859 \| 0.5304 | - |

Table 2: F1 + EM scores from dev and test leaderboards

| Model Config. | Dev | Test |
|---|---|---|
| Baseline | 0.43055 \| 0.342257 | - |
| BiDAF + mul.SA | 0.7182 \| 0.6175 | - |
| BiDAF + mul.SA + prediction (final model) | 0.74779 \| 0.64002 | 0.75137 \| 0.64901 |

### 4.4 Example Error Analysis

Our final model mainly struggles with predicting the exact boundaries for answers. Many of the incorrect answers often capture part of the true answer or some extraneous information. For example:

**Question**: what did tesla work on in 1888 ?
**True Answer**: system to power the city 's streetcars
**Predicted Answer**: to create an alternating current system to power the city 's streetcars

We can better understand the model's answer choice from the C2Q Attention Distribution for the first example (see Figure 7). The model primarily pays attention to words found in the query. Beyond that, it pays attention to other words in a weaker fashion. For example, for the word ”work”, it focuses on ”helping”, ”create”, and ”engineers”, suggesting that the model gives more weight to words in the context that are related to the query when selecting an answer. This may partially explain why it included the word ”create” in its predicted answer.

Other times, the predicted answer fails to capture any of the true answer. This may have to do with the question using acronyms and very specific domain knowledge that the model does not recognize. In the example below, the question asks about ”atp” and ”change”. The true answer uses the term ”phosphorylate” to describe ”change” and ”adenosine diphosphate” for ”atp”. However, the word

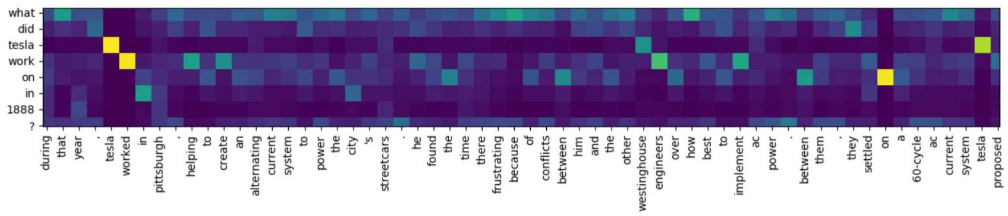


Figure 7: C2Q Attention Distribution for first example

embeddings for these term pairs are likely not too similar as one is a much more specific usage of the other. Also, the predicted answer indices are closer to other instances that use the exact terms in the question. Example:

**Question**: what does atp synthase change into atp ?
**True Answer**: phosphorylate adenosine diphosphate
**Predicted Answer**: the hydrogen ions in the thylakoid space then diffuse back down their concentration gradient

Possible methods to remedy these problems include training word vectors for specific domain knowledge, adding character embeddings to handle unknown vocabulary (very likely in domain specific documents), and also conditioning the probability distribution of the end index to depend on that of the start index (may help eliminate redundant words in the answer).

### 4.5 Conclusion

In this paper, we reimplemented a modified Bidirectional Attention Flow model augmented with multiplicative self attention. We also tested several variations of attention and tested different hyperparameters. The experimental evaluations indicate that the single layer attentions are not complex enough and that the multi-layer attention mechanism is necessary to achieve better results. With more time and computing power, we would extend our model to use trainable character embeddings, initially generated from averaging GLOVE vectors [3], to then be merged with pretrained word vectors; this way, we would be able to handle out-of-vocabulary words more gracefully as well as use morphology as a feature.

### Acknowledgments

The authors would like to thank the CS224N course staff for their help and insights about improving the models as well as providing computational power to run these experiments.

### References

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

[2] https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf

[3] Character embeddings from GloVE. https://github.com/minimaxir/char-embeddings?

[4] S. Sugawara and A. Aizawa, An analysis of prerequisite skills for reading comprehension, EMNLP 2016, p. 1, 2016.

[5] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, SQuAD: 100,000+ Questions for Machine Comprehension of Text, ArXiv e-prints, Jun. 2016.

[6] J. Pennington, R. Socher, and C. D. Manning, Glove: Global vectors for word representation, in Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 15321543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[7] SQuAD leaderboard. https://rajpurkar.github.io/SQuAD-explorer/