# Question Answering System with Deep Learning

**Jake Spracher**
SCPD
jsprache@stanford.edu

**Robert M Schoenhals**
Stanford Law School
Stanford University
rschoenh@stanford.edu

**Takahiro Fushimi**
Department of Management Science and Engineering
Stanford University
tfushimi@stanford.edu

## Abstract

The Stanford Question Answering Dataset (SQuAD) challenge is a machine reading comprehension task that has gained popularity in recent years. In this paper, we implement various existing deep learning methods with incremental improvements and conduct a comparative study of their performance on SQuAD dataset. Our best model achieves 76.1 F1 and 66.1 EM scores on the test set. This project is completed for Assignment 4 of CS224n.

## 1 Introduction

The Stanford Question Answering Dataset (SQuAD) challenge, a machine comprehension task, has gained popularity in recent years from both theoretical and practical perspectives. The Stanford NLP group published the SQuAD[2] dataset consisting of more than 100,000 question-answer tuples taken from the set of Wikipedia articles, in which the answer to each question is the consecutive segment of text in the corresponding reading passage. The primary task is to build models that take a paragraph and a question about it as an input, and identify the answer to the question from the given paragraph. There has been a lot of research on building a state-of-the-art deep learning system on SQuAD that has been reported to accomplish outstanding performance [3][6][5]. Hence, the objective of this paper is to start with the provided starter code for *CS224n: Natural Language Processing with Deep Learning (2017-2018 Winter Quarter)*, make successive improvements by implementing existing models, and compare their performance on SQuAD.

The rest of the paper is organized as follows. Section 2 illustrates the components and the architecture of our system. Section 3 describes the experiments setting. Section 4 demonstrates error analysis. Section 5 concludes the paper and discusses future work.

## 2 Model

Our model is modular in that different components can be swapped with others due to independent implementation. Thus, we first present all the modules considered in this paper, and then illustrate the specific combinations of the modules that we run in experiments.

### 2.1 Model Components

#### 2.1.1 Encoding Layer

A $d$-dimensional word embeddings of a question $\mathbf{x}_1, \cdots, \mathbf{x}_N \in \mathbf{R}^d$ and context $\mathbf{y}_1, \cdots, \mathbf{y}_M \in \mathbf{R}^d$ are fed into a bidirectional LSTM with weights shared between the question and context. The encoding layer encodes the embeddings into the representation matrix of the context hidden states $\mathbf{H}$ in $R^{N \times 2h}$ and the question hidden states $\mathbf{U}$ in $R^{M \times 2h}$ where $h$ is the size of hidden states.

#### 2.1.2 Bidirectional Attention Layer

Bidirectional Attention Layer [3] is one of the attention layers that we use. Given a set of representations for context hidden states $\mathbf{h}_1, \cdots, \mathbf{h}_N \in \mathbf{R}^{2h}$ and the question hidden states $\mathbf{u}_1, \cdots, \mathbf{u}_N \in \mathbf{R}^{2h}$, a matrix $\mathbf{S}$ in $R^{N \times M}$ is computed according to $\mathbf{S}_{i,j} = \mathbf{w}^T[\mathbf{h}_i; \mathbf{u}_j; \mathbf{h}_i \circ \mathbf{u}_j] \in \mathbf{R}$, where $\mathbf{w} \in R^{6h}$ is a weight vector learned through training.

We first compute *Context-To-Question (C2Q)* attention. *C2Q attention distribution* is obtained by $\alpha^i = \text{softmax}(\mathbf{S}_{i,:}) \in \mathbf{R}^M, \forall\, i \in \{1, \cdots, N\}$. The question hidden states $\mathbf{u}_j$ are then weighted according to $\alpha^i$ to get *C2Q attention output* $\mathbf{a}_i = \sum_{j=1}^{M} \alpha_j^i \mathbf{u}_j \in \mathbf{R}^{2h}$. Next, we compute *Question-To-Context (Q2C)* attention. *Q2C attention distribution* is obtained by $\beta = \text{softmax}(\mathbf{m}) \in \mathbf{R}^N$ for $\mathbf{m}_i = \max_j \mathbf{S}_{i,j}, \forall\, i \in \{1, \cdots, N\}$. The context hidden states $\mathbf{h}_i$ are then weighted according to $\beta$ to get *Q2C attention output* $\mathbf{c}' = \sum_{i=1}^{N} \beta_i \mathbf{h}_i \in \mathbf{R}^{2h}$. Then, we get the *bidirectional attention encoding* $\mathbf{b}_i = [\mathbf{h}_i; \mathbf{a}_i; \mathbf{h}_i \circ \mathbf{a}_i; \mathbf{h}_i \circ \mathbf{c}'] \in \mathbf{R}^{8h}, \forall\, i \in \{1, \cdots, N\}$.

#### 2.1.3 Coattention Layer

Another type of attention layers we implemented is Coattention layer [6]. Given the question hidden states $\mathbf{u}_1, \cdots, \mathbf{u}_M \in \mathbf{R}^l$, we first compute projected question hidden states $\mathbf{u}'_j = \tanh(\mathbf{W}\mathbf{u}_j + \mathbf{b}) \in \mathbf{R}^l, \forall\, j \in \{1, \cdots, M\}$. Also, sentinels $\mathbf{h}_\emptyset$ and $\mathbf{u}_\emptyset$ are appended to the context and question hidden states, which gives us $\{\mathbf{h}_1, \cdots, \mathbf{h}_M, \mathbf{h}_\emptyset\}$ and $\{\mathbf{u}_1, \cdots, \mathbf{u}_M, \mathbf{u}_\emptyset\}$. We then compute a *affinity matrix* $\mathbf{L} \in \mathbf{R}^{(N+1) \times (M+1)}$ where $\mathbf{L}_{i,j} = \mathbf{h}_i^T \mathbf{u}'_j \in \mathbf{R}$.

Using the affinity matrix $\mathbf{L}$, we apply the standard attention mechanism in both directions. *Context-To-Question attention output* is obtained by $\mathbf{a}_i = \sum_{j=1}^{M+1} \alpha_j^i \mathbf{u}'_j \in \mathbf{R}^\ell$ for $\alpha^i = \text{softmax}(\mathbf{L}_{i,:}) \in \mathbf{R}^{M+1}, \forall\, i \in \{1, \cdots, N\}$. *Question-To-Context attention output* is computed in a similar way: $\mathbf{b}_j = \sum_{j=1}^{N+1} \beta_j^i \mathbf{h}_i \in \mathbf{R}^\ell$ for $\beta^j = \text{softmax}(\mathbf{L}_{:,j}) \in \mathbf{R}^{N+1}, \forall\, j \in \{1, \cdots, M\}$. Next, we compute *second-level attention output* $\mathbf{s}_i = \sum_{j=1}^{M+1} \alpha_i^j \mathbf{b}_j \in \mathbf{R}^\ell$. Finally, $[\mathbf{a}_i; \mathbf{s}_i] \in R^{2l}, \forall\, i \in \{1, \cdots, N\}$ is fed into a bidirectional LSTM, and the resulting hidden states are the *coattention encoding*.

#### 2.1.4 Modeling Layer

Following the example of the BiDAF[3], we implement a modeling layer comprised of two layers of bidirectional LSTM's, which outputs $\mathbf{M} \in \mathbf{R}^{d_M \times N}$.

### 2.2 Self-Attention Layer

A self-attention layer [4] is used as an alternative to the modeling layer. Given the context hidden states $\mathbf{H} \in \mathbf{R}^{N \times 2h}$, we apply the attention mechanism to obtain attention distribution $\mathbf{A} = \text{softmax}(\mathbf{H}\mathbf{H}^T/\sqrt{2h}) \in \mathbf{R}^{N \times N}$, where softmax is taken with respect to the rows of $\mathbf{H}\mathbf{H}^T/\sqrt{2h}$. Then, *self-attention output* is computed by $\mathbf{a} = \mathbf{A}\mathbf{H} \in \mathbf{R}^{N \times 2h}$.

#### 2.2.1 Output Layers

The basic output layer we consider has the identical structure of the BiDAF[3]. This module is used in conjunction with the modeling layer. Let $\mathbf{G} \in \mathbf{R}^{d_G \times N}$ denote the output by an attention layer. Then, the probability distribution of the start index is computed by

$\mathbf{p}^{\text{start}} = \text{softmax}(\mathbf{w}_s^T[\mathbf{G};\mathbf{M}]) \in \mathbf{R}^N$, where $\mathbf{w}_s \in \mathbf{R}^{d_G+d_M}$ is a trainable weight. Then, $\mathbf{M}$ is passed to a bidirectional LSTM that outputs $\mathbf{M}_2 \in \mathbf{R}^{d_M \times N}$. Finally, the probability distribution of the end index is obtained by $\mathbf{p}^{\text{end}} = \text{softmax}(\mathbf{w}_s^T[\mathbf{G};\mathbf{M}_2]) \in \mathbf{R}^N$.

Another type of output layer we implemented is Answer-Pointer Layer[5]. Given the blended representation $\mathbf{G}$, the probability distribution of the start index is given by $\mathbf{p}^{\text{start}} = \text{softmax}(\mathbf{w}^T\mathbf{F}_1 + c \otimes \mathbf{e}_N) \in \mathbf{R}^N$, where $\mathbf{F}_1 = \tanh(\mathbf{V}\mathbf{G} + \mathbf{b} \otimes \mathbf{e}_N) \in \mathbf{R}^{d_G \times N}$, and $\mathbf{w} \in \mathbf{R}^{d_G}, c \in \mathbf{R}, \mathbf{V} \in \mathbf{R}^{d_G \times d_G}, \mathbf{b} \in \mathbf{R}^{d_G}$ are parameters to be trained. The operator $\otimes \mathbf{e}_N$ produces a matrix by repeating the element on the left hand side for $N$ times. Then, we compute the hidden vector $\mathbf{h}_1$ by using attention mechanism $\mathbf{G}\mathbf{p}_s \in \mathbf{R}^{d_G}$ and passing it to a standard LSTM. Finally, the probability distribution of the end index is obtained by $\mathbf{p}^{\text{end}} = \text{softmax}(\mathbf{w}^T\mathbf{F}_2 + c \otimes \mathbf{e}_N) \in \mathbf{R}^N$, where $\mathbf{F}_2 = \tanh(\mathbf{V}\mathbf{G} + (\mathbf{W}_h\mathbf{h}_1 + \mathbf{b}) \otimes \mathbf{e}_N) \in \mathbf{R}^{d_G \times N}$, and $\mathbf{W}_h \in \mathbf{R}^{d_G \times d_G}$ is another trainable weight.

The start and end indices $(l^{\text{start}}, l^{\text{end}})$ are selected such that the joint probability $p_i^{\text{start}} p_j^{\text{end}}$ is maximized subject to $i \leq j \leq i + C$ where $C$ is a specified maximum length of answers.

### 2.2.2 Character-level CNN

We choose to implement a character-level CNN for use in generating additional character-level word embeddings. These are then concatenated with the pre-trained glove embeddings to create a hybrid representation for each word. The input to the CNN is a word $w$ broken down into characters $c_1, \cdots, c_L$. The characters are then mapped to character ID's $i_1, \cdots, i_L$, which are used to index into a character embedding matrix to get character embeddings $\mathbf{e}_1, \cdots, \mathbf{e}_L$. These are not to be confused with the character-level word embeddings the network outputs, the character embeddings are a dense vector representation of each *character*.

In order to generate the character ID's, we need to establish a mapping from characters to indices in the character embedding matrix. To do so, we choose to restrict our character embeddings to all of the characters returned by `string.printable` in Python 2.7 plus an *UNK* and a *PAD* character. Our strategy relies on the network to learn to ignore the padding instead of explicitly masking it out. To do so we initially use *UNK* to represent the padding, but determine that adding a separate pad character increases performance. Another key change from the base code is modification of the padding function to allow for a custom pad ID. This allows for padding with empty lists at the word level, and then the *PAD* character ID at the character level which is required to maintain dimensions compatible as numpy array inputs to the CNN. We set our character embedding size $d_c = 20$ at the recommendation of the project handout.

At the CNN layer, we pass the character embeddings $\mathbf{e}_1, \cdots, \mathbf{e}_L \in \mathbf{R}^{d_c}$ through a convolution to produce hidden representations $\mathbf{h}_1, \cdots, \mathbf{h}_L \in \mathbf{R}^f$. We set the hidden/output state dimensionality $f = 100$ and our window width $k = 5$ at the recommendation of the project handout. Finally, the character level-embedding is created via applying elementwise max pooling $\text{emb}_{\text{char}}(w) = \max_i \mathbf{h}_i \in \mathbf{R}^f$.

Additionally, we apply dropout with a keep probability of 80% to the CNN input character embeddings $\mathbf{e}_1, \cdots, \mathbf{e}_L$. We also try sharing the internal CNN weights for the question. Both of these changes seem to boost performance.

## 2.3 Model architectures

Our models are constructed by choosing modules for different layers. The specific combination that we consider is summarized in Table 1. Note that the baseline model uses a basic attention layer in which only the context hidden states attend to the question hidden states, and a fully connected output layer. The coattention models also adopts the fully connected output layer.

| Model | Character | Encoder | Attention | Modeling | Output |
|---|---|---|---|---|---|
| Baseline | - | biGRU | Basic | - | fully-connected |
| BiDAF1 | - | biLSTM | Bidirectional | biLSTM | Basic |
| BiDAF2 | - | biLSTM | Bidirectional | biLSTM | Answer-Pointer |
| BiDAF3 | CNN | biLSTM | Bidirectional | biLSTM | Basic |
| CoAtten1 | - | biLSTM | Coattention | - | fully-connected |
| CoAtten2 | - | biGRU | Coattention | self-attention | fully-connected |

Table 1: Model architecture

# 3 Experiments

## 3.1 Dataset

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset on Wikipedia articles. The answer to each question is a continuous segment of text in the corresponding reading passage. Hence, the primary task of models is to identify the start and end indices, or boundary, that delimit the answer in the passage. The dataset contains more than 100k question-answer pairs on more than 500 articles, and it is split into 90k/10k train/dev question-context tuples with hidden test set.

## 3.2 Word Embeddings

We use pre-trained GLoVe embeddings from the 840B Common Crawl corpus. The word embeddings are not trained in our model as it's clear that the model overfits quickly. Using the word embeddings, however, gives a slight boost over the default glove embeddings supplied for the project which consist of a 400,000 vocabulary size.

## 3.3 Evaluation metrics

Two evaluation metrics are employed: Exact Match (EM) and F1 score. EM is a binary measure of whether the output from a system exactly matches the ground truth answer. F1 score is a less stringent metric, which is the harmonic mean of precision and recall.

## 3.4 Hyper parameters

The choice of hyperparameters is primarily based on the inspection of the SQuAD data. Figure 1 shows the histogram of context, question, and answer length in the training data. We limit the maximum context length to 400, the maximum question length to 30, and the maximum answer length to 15 since more than 90% of examples fit into this length. The size of hidden state is chosen to be 100 in line with the value used in other papers.

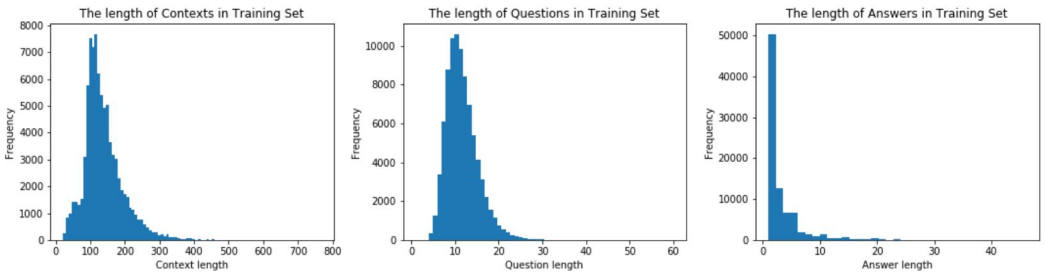


Figure 1: Histogram of context, question, and answer length in the training set

### 3.5 Optimizer

We use Adam Optimizer [1] with learning rate of 0.001. To prevent overfitting, the dropout rate of 0.2 is applied.

## 4 Results

We choose the best model on the basis of its performance on the dev set during the training phase. Table 2 shows F1 and EM on the dev set of each model considered. BiDAF3 turns out to be the best model that achieves 75.9 F1 and 65.8 EM on the dev set and 76.1 F1 and 66.1 EM on the test set in the official evaluation. A comparison with models of other papers are found in Table 3.

| Model | F1 | EM |
|---|---|---|
| | Dev set | |
| Baseline | 39.0 | 28.0 |
| BiDAF1 | 69.2 | 54.5 |
| BiDAF2 | 67.4 | 51.9 |
| BiDAF3 | 70.3 | 55.4 |
| CoAtten1 | 63.8 | 44.7 |
| CoAtten2 | 65.3 | 48.0 |

Table 2: Training phase

| Model | F1 | EM | F1 | EM |
|---|---|---|---|---|
| | Dev set | | Test set | |
| BiDAF3 (our best model) | 75.9 | 65.8 | 76.1 | 66.1 |
| BiDAF [3] | 77.3 | 67.7 | 77.3 | 68.0 |
| Dynamic Coattention [6] | 75.6 | 65.4 | 75.9 | 66.2 |
| Match-LSTM [5] | 77.2 | 67.0 | 77.1 | 66.9 |

Table 3: Official evaluation (single model)

## 5 Error Analysis

In this section, we demonstrate some analysis of our model and discuss possible extension to improve its performance.

### 5.1 Limited Contextual Reasoning

**Example**:

- **Question:** which musical group did the v & a present in july 1973 as part of its youth outreach programme ?
- **Context paragraph:** the v & a became the first museum in britain to present a rock concert . the v & a presented a combined _concert/lecture_ by british progressive folk-rock band gryphon , who explored the lineage of mediaeval music and instrumentation and related how those contributed to contemporary music 500 years later .
- **Answer:** gryphon
- **Predicted Answer:** rock concert

In this example, the model is unable to contextualize the question into the paragraph, which results in a wrong answer. This error is induced by the multimodality of the probability distribution of the predicted start and end indices as shown in Figure 2. The correct answer corresponds to the third spike, while the model chooses the answer from the the second one.

### 5.2 Imperfect Overlap

**Example**:

- **Question:** how much larger would cicada predator populations be if cicada outbreaks occurred at 14 and 15 year intervals ?

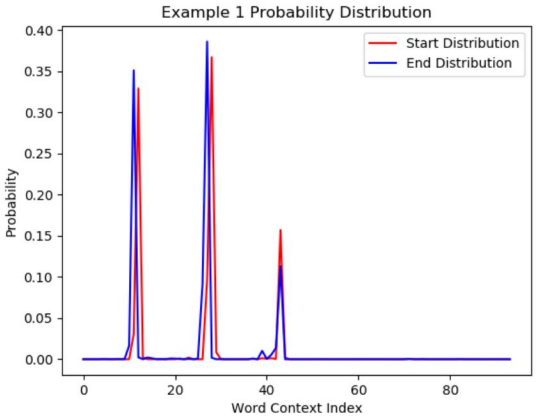


Figure 2: Probability distribution of the predicted start and end indices

- **Context paragraph:** over a 200-year period , average predator populations during hypothetical outbreaks of _14-_ and 15-year cicadas would be up to 2 % higher than during outbreaks of _13-_ and 17-year cicadas . though small , this advantage appears to have been enough to drive natural selection in favour of a _prime-numbered_ life-cycle for these insects .
- **Answer:** up to 2 % higher
- **Predicted Answer:** 2 %

Another type of error our model makes is that the predicted answer incorrectly overlaps with the true answer, e.g., partial overlap or superset/subset of the true answer. It is not clear about how to fix this problem.

### 5.3 Character-Level Word Embedding Quality

We observe a relatively small 3% performance boost on the dev set after implementing the character-level CNN. We hypothesize there are several reasons for this. These include:

1. The character embeddings and CNN weights are only trained over the limited vocabulary in the SQuAD dataset. Because of this, the resulting embeddings are likely worse than they would have been if trained on a larger vocabulary.
2. The character embeddings and CNN weights are trained as part of a task specific model. Since the goal is not to produce the best possible word embeddings and rather to optimize performance on a particular task, the model might have a harder time following gradients toward truly better embeddings during training, and the overall quality of the word embeddings might suffer as a result.

In order to investigate these theories in more detail, we construct a standalone version of the character-level CNN and initialized the character embedding matrix and weights from different checkpoints from our various experiments using `tf.contrib.framework.init_from_checkpoint()`. We then feed a set of arbitrary words through the standalone CNN to generate a set of word embeddings, which we compare via dimensionality reduction using T-SNE [7].

#### 5.3.1 Word Embeddings from Sentences

Figure 3 shows T-SNE plot of the resulting character-level word embeddings from the standalone CNN. Figure 4 illustrates T-SNE plot of the resulting character-level word

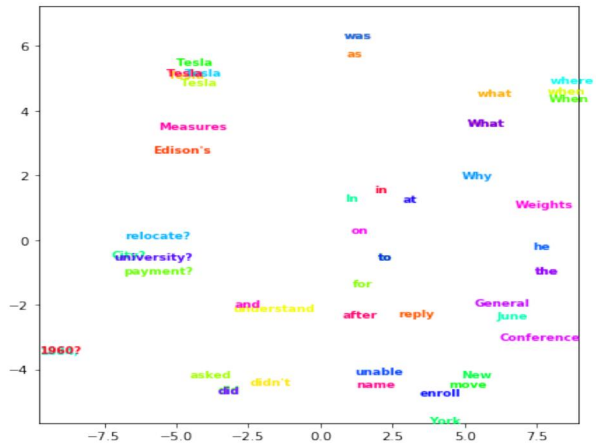


Figure 3: T-SNE plot of resulting of character-level word embeddings for some arbitrary question words

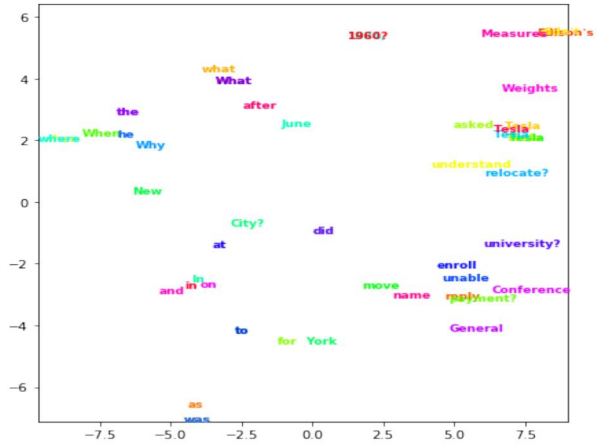


Figure 4: T-SNE plot of resulting of character-level word embeddings from high performing experiment BiDAF3

embeddings from high performing experiment BiDAF3. We use the same T-SNE hyper parameters, and the same set of words to generate both graphs. The T-SNE hyperparameters were: `random_state=42`, `perplexity=5`, `n_iter=1000`, `learning_rate=10`, and `method='exact'`. The words plotted are the vocabulary from the sentence inputs below, which are used to generate all of the plotted word embeddings.

**Sentences used to generate T-SNE plots**:

- "Why was he unable to enroll at the university?"
- "What did the General Conference on Weights and Measures name after Tesla in 1960?"
- "In June 1884, where did Tesla relocate?"
- "When did Tesla move to New York City?"
- "What was Edison's reply as to what Tesla didn't understand when Tesla asked for payment?"

### 5.3.2 Word Embeddings from Context

The same analysis is also performed with the vocabulary from a context. T-SNE plots are shown in Figure 5 and Figure 6.

**Context used to generate T-SNE plots:**

> "in 1881 , Tesla moved to Budapest to work under Ferenc Puskás at a telegraph company , the Budapest Telephone Exchange. upon arrival, Tesla realized that the company , then under construction, was not functional, so he worked as a draftsman in the Central Telegraph Office instead . within a few months , the Budapest Telephone Exchange became functional and Tesla was allocated the chief electrician position . during his employment , Tesla made many improvements to the Central Station equipment and claimed to have perfected a telephone repeater or amplifier , which was never patented nor publicly described . grommet eyelit cavil boorish"

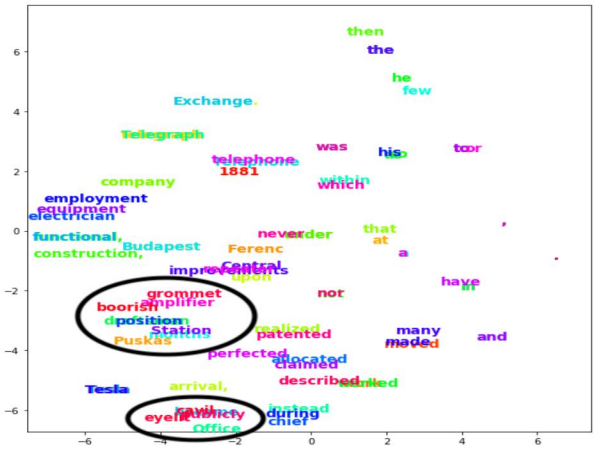


Figure 5: T-SNE plot of resulting character-level word embeddings for some arbitrary context words

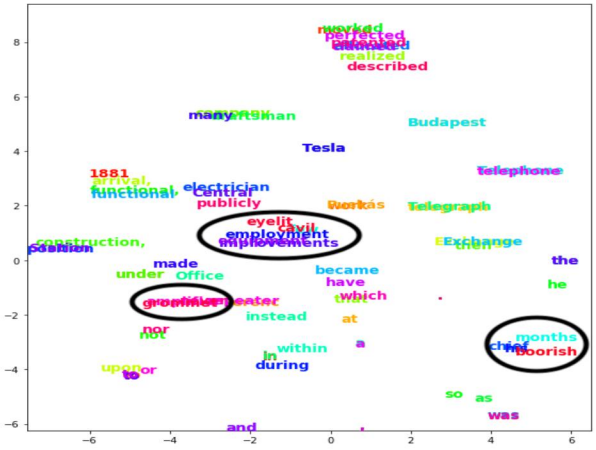


Figure 6: T-SNE plot of resulting character-level word embeddings from context high performing experiment BiDAF3

### 5.3.3 Analysis

In all above plots, we observe that the character level word embeddings seem to do a decent job clustering related words together. However, many mistakes are evident. The context used to generate Figure 5 and Figure 6 above consists of a context that exists in the training data prepended to four rare words. Both "grommet" and "eyelit" do not exist in the train and dev data. Figure 5 and Figure 6 show that these words are not clustered nearby, indicating that the model was not able to infer the meaning of these unseen words from their characters. "Cavil" does not exist in the training set but "boorish" does. Figure 5 and Figure 6 also show that these words were not clustered nearby, indicating that the model was not able to infer the meaning of an unseen word even if provided a similar known word. This supports our first hypothesis. In Figure 3 and Figure 4 the words "Measures" and "Edison's" are clustered nearby in both experiments. This seems to be an example of the embeddings being overfit to the task, as considering these words as similar may result in better answers on the training data, but does not represent an accurate generalization in that these words are not inherently similar. This supports our second hypothesis.

## 5.4 Attention Analysis

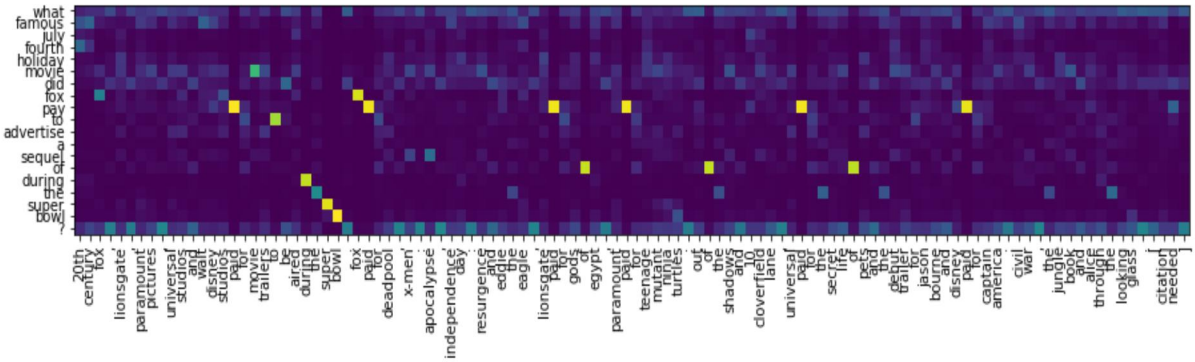


Figure 7: Attention distribution for a sample context and question

In Figure 7 we analyze the attention distribution for a sample context and question. The lighter values indicate stronger question to context attention. For example, as the question mentions "during the super bowl", attention is strongly focused on the context section "aired during the super bowl". Additionally, as the question mentions "did Fox pay", attention is focused on the context section "Fox paid for" This confirms that our attention mechanism is functioning as expected.

# 6 Conclusion and Future Work

In this paper, we implement various deep learning models for building a question answering system on SQuAD by integrating different types of layers. Our experiment demonstrates that the bidirectional attention flow model with the character-level CNN, modeling layer, and basic output layer achieves the best performance among the models considered. However, for the test set, our model underperforms the state of the art models in the previous papers.

Extensions to our work could include different optimizers; for example, other papers have been able to achieve good results using the Adadelta optimizer. The hope is that the optimizer might be able to converge to better local optima. Other extensions could be adding multiple attention layers - such as two levels of bi-directional attention. Limitation in attention to the keys might be one bottleneck of our current best model, and deeper models might be able to better attend to the question itself. Finally, as our model also appears to

have a limited ability to attend to other parts of the context paragraph, experiment with other self-attention layers, such as additive attention, or using gated attention models.

## References


[1] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint* arXiv:1412.6980.

[2] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint* arXiv:1606.05250.

[3] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *arXiv preprint* arXiv:1611.01603.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł& Polosukhin, I. (2017). Attention is all you need. *In Advances in Neural Information Processing Systems* (pp. 6000-6010).

[5] Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint* arXiv:1608.07905.

[6] Xiong, C., Zhong, V., & Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint* arXiv:1611.01604.

[7] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008.