# Game, Set, Match-LSTM: Question Answering on SQUaD

**Ian Torres**
itorres@stanford.edu

**Eric Ehizokhale**
eokhale@stanford.edu

**CodaLab Username**
eokhale

## Abstract

In this paper, we investigate the problem of answering textual questions based on a context paragraph that contains the answer. For our dataset, we use the Stanford Question Answering Dataset (SQuAD) [1]. Our approach is based on the Match-LSTM and Answer Pointer architecture described by Wang and Jiang [3]. Our final model achieves an F1 of 50.5% and an EM of 38.8% on the test set, which is on par with a strong logistic regression model.

## 1 Introduction

Question answering (QA) is the task of answering a question based on a chunk of context text supporting it. Recently, Rajpurkar, et al. released the Stanford Question Answering Dataset [1], which contains orders of magnitude more questions than previous QA datasets. Correctly identifying answers to the questions in SQuAD takes diverse forms of reasoning, such as reasoning across multiple sentences. For these reasons, the SQuAD dataset allows for training more expressive, large-scale models.

More formally, the task of question answering can be formulated as follows. Given a question represented as a sequence of $n$ words $\boldsymbol{q} = (q_1, q_2, ..., q_n)$ and corresponding context paragraph of $m$ words $\boldsymbol{p} = (p_1, p_2, ..., p_n)$, find the ordered pair $(a_s, a_e)$ for which the span of words from $\boldsymbol{p}[a_s]$ to $\boldsymbol{p}[a_e]$ is the answer. In other words, given a pair of sequences $\boldsymbol{q}, \boldsymbol{p}$, find the start and end indices for where the answer lies.

## 2 Background/Related Work

In "SQuAD: 100,000+ Questions for Machine Comprehension of Text," Rajpurkar et al. provide a strong logistic regression model that achieves an F1 score of 51% [1]. However, the obvious drawback of logistic regression based models is the extensive feature engineering required to correctly parse the context paragraph to find the answer to a question. Rajpurkar et al. use features like part of speech tags, dependency tree parsing, bigram frequencies, and others to successfully train their model. Instead of logistic regression, neural models have achieved state of the art performance on the SQuAD dataset, with F1 scores above 85%.

In "Machine Comprehension using Match-LSTM and Answer Pointer," Wang and Jiang [3] build their question answering model using two important neural frameworks. The first, a match-LSTM layer, is a neural model that, given a "premise" sentence and a "hypothesis sentence," sees if the premise entails the hypothesis by using an attention vector to weight the premise and using this weighted vector on positions of the hypothesis. The second, pointer networks, were proposed by Vinyals et al. [2] as a way to learn about the conditional probability of an output sentence with elements whose tokens correspond to positions in an input sequence.

## 3 Approach

We implemented Wang and Jiang's Match-LSTM model for our question answering system. Below is a quick overview of the model architecture.

### 3.1 Preprocessing

Given the word embeddings for the question and corresponding context paragraph, we encode contextual information by preprocessing the question and context paragraph. We use a one directional LSTM on the question and context paragraph. The resulting matrices $H^p$ and $H^q$ are hidden representations of both layers that are fed into the Match-LSTM layer.

### 3.2 Match-LSTM layer

In this layer, we go through the context paragraph sequentially and attain an attention weight matrix $\alpha$ . In this matrix, the entry $\alpha_{i,j}$ indicates how much a token $i$ in the context paragraph correlates with a token $j$ in the question. We feed this attention vector into a standard one-directional LSTM to form our Match-LSTM in the forward direction. We also build a Match-LSTM model in the reverse direction, and concatenate the outputs of both together to attain our final matrix $H^r$.

### 3.3 Answer-pointer layer

The answer-pointer layer is the decoding layer of our $H^r$ input. At a high level, this layer takes in the representation of our question and context paragraphs with attention, and returns a probability distribution over the positions of the context, in which the probabilities correspond to whether or not that position is part of the answer for the question. We use the boundary model for the answer-pointer layer. Unlike the sequence model, which predicts a series of consecutive words as part of the answer, the boundary model only predicts the start and end index for the answer of the query.

We use the same attention mechanism from the Match-LSTM layer to generate $\beta_s$ and $\beta_e$, the vectors corresponding with the probability distributions over the context paragraphs. More specifically, $\beta_{s,i}$ is the probability that token $i$ in the context length is the start index for the answer, and likewise for $\beta_{e,i}$.

### 3.4 Loss Function

We minimize the cross-entropy loss of our $\beta$ vectors when compared to the ground-truth answer vectors. Recall that for a prediction vector $\hat{y}$ and one-hot label vector $y$, the cross-entropy loss is defined as

$$CE(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i).$$

### 3.5 Answer Selection Improvements

Note that our $\beta$ vectors may generate unintuitive answers for the start and end indices because we do not explicitly force $a_s < a_e$. For example, we might get $a_s = 10$ and $a_e = 8$. Our simple heuristic to fix this is to swap the start and end indices whenever the end index is smaller than the start index.

We also experiment with limiting the number of words to some max length $\ell$, which yielded 2 to 3 percent improvement in the original paper [3]. Formally, we set our boundary indices to

$$a_s, a_e = \text{argmax}_{0 \le i_s \le i_e \le m, i_e - i_s \le \ell}(\beta_{s,i_s}\beta_{e,i_e}).$$

This prevents unreasonably long answers that encompass large portions of the paragraph from being selected as the predicted answer.

### 3.6 Evaluation Metrics

We analyze our model on two metrics, F1 and EM. The F1 score measures the average overlap between words in the predicted answer and the ground-truth answer. Formally, given the true answer

| | F1 | EM |
|---|---|---|
| Training | 71.1% | 56.7% |
| Validation | 51.0% | 35.4% |
| Test | 50.5% | 38.8% |

Table 1: F1 and EM scores of model on dataset splits.

as a sequence of words $a$, and predicted answer $\hat{a}$, we define F1 as

$$\text{F1} = \frac{2PR}{P+R}.$$

where $P$ is the precision (percentage of overlap with the predicted answer) and $R$ is the recall (percentage of ovelap with the true answer).

The exact match score (EM) is simply the percentage of exact string matches between the predicted answers and true answers. That, is, for a single example, the EM is 1 if $a = \hat{a}$ and 0 otherwise.

To evaluate our model, we calculate the average F1 and EM score over all examples in each dataset split.

## 4 Experiments

We train and evaluate our model on the SQuAD dataset. We use a training dataset of roughly 87k question, context, answer triplets. A 5% split of the training set is used as the validation set. We test on the dev training set of roughly 10k triplets. The official test dataset is not publically available. It's kept by the authors of SQuAD to make model evaluation fair.

We use GloVE word vectors trained on the 840B Common Crawl corpus. We limit the max context paragraph length to 300 and the max question length to 30. We use a hidden state size of 200 for all hidden units. Our LSTM parameters are initialized with the Xavier initializer, and the LSTMs have an initial hidden state of all zeros. We use a batch size of 32 and log our training loss every 10 iterations. Our final model used an Adam optimizer with a learning rate of 0.001 and a dropout keep probability of 0.8. We trained our final model for 8 epochs and log the F1/EM score on the training and validation sets, as well as the average loss on the entire validation set, every $1/3$ of an epoch. While we did not have much time for hyperparameter searhc, we also tried decreasing the dropout keep probability to reduce overfitting.

## 5 Results

We achieved our best model after about 15000 iterations (6 epochs) of training. The best scores were achieved by limiting the max length of the answer to $\ell = 13$ using the method described in Section 3. We report the performance of our best model, the training and validation loss history, and the corresponding F1 and EM score history over the training period.

Table 1 shows the performance of our final model on the training, validation, and test sets. The model achieved F1 and EM scores of 50.5% and 38.8% on the test set. However, the F1 and EM scores on the training set were significantly higher at 71.1% and 56.7%, indicating overfitting.

The loss graph in Figure 1 corroborates this, showing that the validation gap began increasing at iteration 10,000. At iteration 15,000, the validation loss itself begins increasing. Following the trend of the graph, if we continued to train past 8 epochs, the validation gap would likely continue to increase.

We also calculated the F1 and EM scores of model checkpoints on 1000 samples from both the training and validation sets. As Figure 2 shows, the scores improve at the same rate for the first two epochs, but then the validation scores plateau soon afterward.

To attempt to reduce overfitting, we also tried training a model with a dropout keep probability $p = 0.5$, compared to to our final model, which used $p = 0.8$. Figure 3 shows the validation and

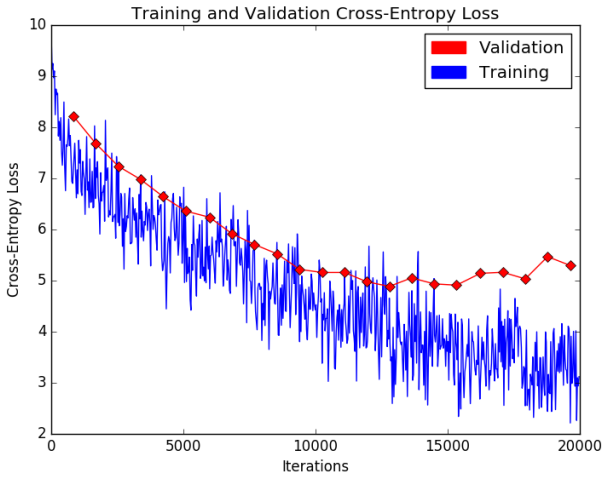
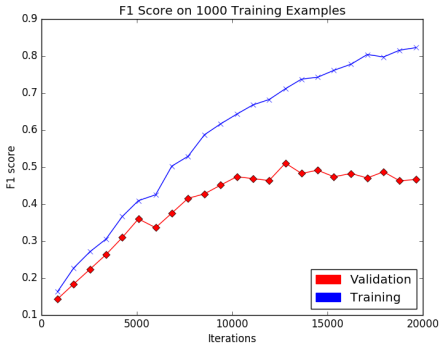
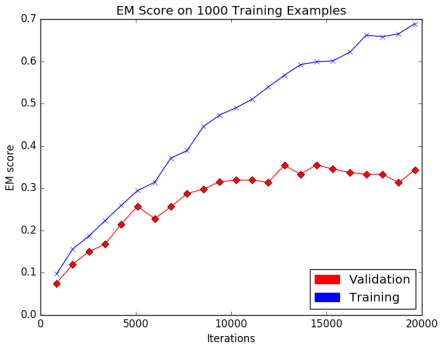



Figure 1: Training and validation loss.




(a) F1




(b) EM

Figure 2: F1 and EM scores on training and validation set

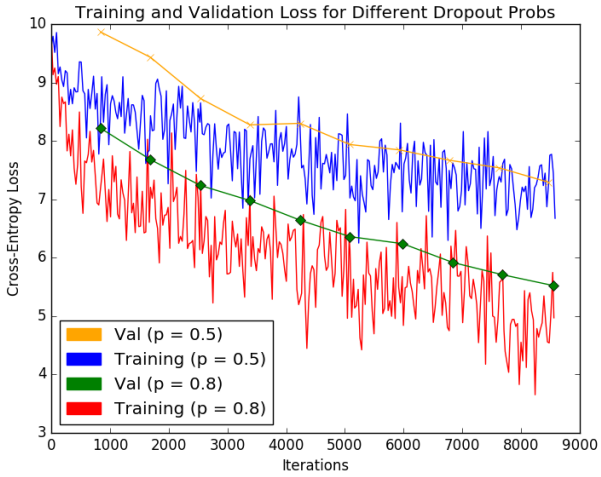



Figure 3: Comparison of different dropout probs.

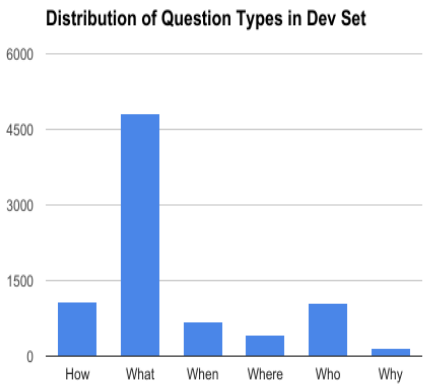



Figure 4: Frequencies of different question types

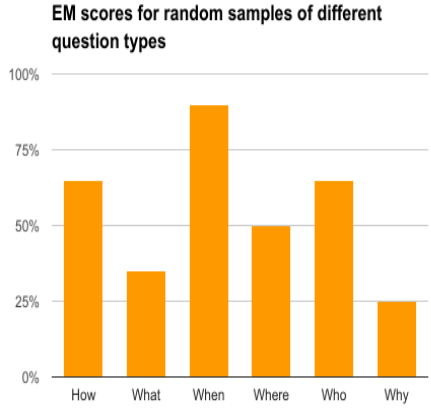



Figure 5: EM scores of different question types

training loss curves for both $p$ values over just under 4 epochs (9000 iterations). As the graph shows, the validation gap was significantly less when $p = 0.5$, indicating better generalization.

However, both the training and validation loss when $p = 0.5$ were both significantly greater htan the corresponding losses for $p = 0.8$. This is expected since dropping out more neurons means that fewer updates are made for each iterations. Thus, it is possible that using a keep probability $p = 0.5$ n would eventually result in a better model than the $p = 0.8$ run, but it likely would likely take much longer to train.

## 6 Discussion and Analysis

While our model performed moderately well on the test set, it heavily overfit the training set. There are many reasons why this could have occurred. The hidden dimension we used for the Match-LSTM and Answer Pointer layers, which we set to 200, may have been too large, giving the model too much representational power. It's also possible that our dropout keep probability of 0.8 was too conservative, as shown in Figure 3. However, the graph also shows that a lower keep probability comes with a time trade-off for training the model. Given more time, we could probably find a better combination of learning rate, dropout keep probability, and hidden dimension that would minimize overfitting without making training too slow.

### 6.1 Qualitative Analysis

To better understand the types of questions our model performed well on, we analyzed the predictions our model made. We separated the dev questions into six general categories (How, What, When, Where, Who, Why). First, we plot the frequencies of the types of the questions in the dataset. Then for each category, we randomly sampled 20 predicted answers and calculated the EM score for that category.

In Figures 4 and 5, we see that the dataset is dominated by "What" questions, which have relatively lower performance than other question categories outside of "How" questions. "When" and "Who" questions have very high performance despite not appearing as much in the dev set. Intuitively, this makes sense given our model. Our attention vector calculated in the Match-LSTM step will give a strong correlation between any tokens related to time in the context paragraph and the word "When" in the question. Even in the instances where our model predicts incorrect answers for "When" and "Who" questions get the incorrect answers, the model will still predict a time frame or person found in the passage. Many of the "How" questions in the dev set were of the form "How many...", so

**Question:** What sort of motion did Watt's steam engine continuously produce?
**Context:** In 1781 James Watt patented a steam engine that produced continuous rotary motion. Watt's <unk> engines enabled a wide range of manufacturing machinery to be powered.
**Predicted Answer:** rotary
**Correct Answer**: rotary

Figure 6: Easier question

**Question:** What types of arts flourished in the Yuan?
**Context:** In the China of the Yuan , or Mongol era , various important developments in the arts occurred or continued in their development , including the areas of painting , mathematics , calligraphy , poetry , and theater , with many great artists and writers being famous today .
**Predicted Answer:** poetry
**Correct Answer**: painting, mathematics, calligraphy, poetry, and theater

Figure 7: Harder question

our model correctly associated numbers as having the highest probability of being answers for these questions.

However, the words "What" and "Why" have weaker correlations to words in the context paragraph. Our model struggles on "Why" questions especially when the token "because" is not found in the context. These are also the questions that often have more partial answers when the answer is wrong. "What" questions are difficult because the answer can be any named entity – whether they're people ("What actress"), actions ("What did X do"), times ("What day"), and so on. "What" questions cover a broader variety of answers, and thus our attention vectors at the Match-LSTM stage of our model don't associate context words as strongly with the question word, "What", instead relying on other words in the question to associate with.

Outside of analyzing our model's performance on question words, we also qualitatively look at the "difficulty" of questions that our model predicts correctly. Questions that have word similarities with paragraphs are more easily predicted. Figure 6 shows a relatively easy question because the answer in the context paragraph is near the same words as in the question. Figure 7 shows a harder question because some words in the question aren't explicitly in the context ("flourished") and other words are farther away from the true answer (Yuan).

We also wanted to visualize the $\beta_s$ and $\beta_e$ vectors we were generating. By looking at the start and end probabilites computed for a given question/paragraph pair, we could get a sense of how confident the model was in certain predictions. Figure 8 shows an example where the model predicts the correct answer, and we see the predicted probabilites are significantly higher than other options. Figure 9 shows an example where the model predicts an incorrect answer. Even though the true answer is 10 tokens away from the predicted answer, the model shows that the predicted answer has the second highest probability of being true. This further supports the claim that the Match-LSTM model is particularly good at answering "When" questions, as the only dates have very high probabilites, and it also suggests that the model can give generally reasonable second or third guesses even when it's wrong.

## 7 Conclusion and Future Work

Our models final performance was on par with the logistic regression baseline, which was reasonable given time constraints.

One of the most significant limiting factors in improving our model was time. Fully training the model on one epoch took about 1.5 hours, so hyperparameter search was limited. Dropout, embedding size, and learning rate are a few hyperparameters we could tune to optimize our model.

Our model heavily overfit the training set, thus, possible improvements include implementing other forms of regularization or decreasing the hidden dimension. Using annealed dropout instead of a fixed dropout probability is another possibility.

We would like to explore other neural models for this task, namely Dynamic Coattention Networks [4]. Match-LSTM models perform worse on questions with longer context paragraphs. However, DCN performance doesnt appear to degrade with context paragraph length, suggesting that the coattentive encoder is good at focusing on relevant parts of the context paragraph while ignoring the rest. We suspect we could further improve our performance by using the DCN encoding layers, and then decoding the probability distribution with our Answer-Pointer layer.

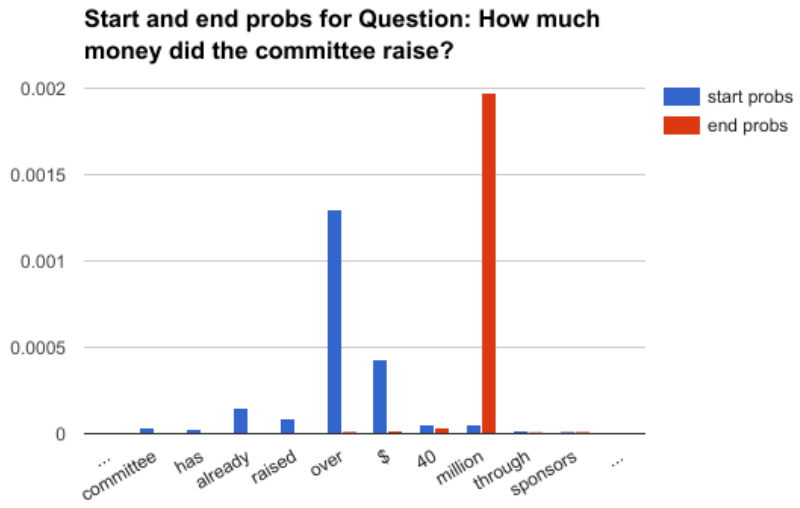



Figure 8: Beta probabilities for a correctly answered question

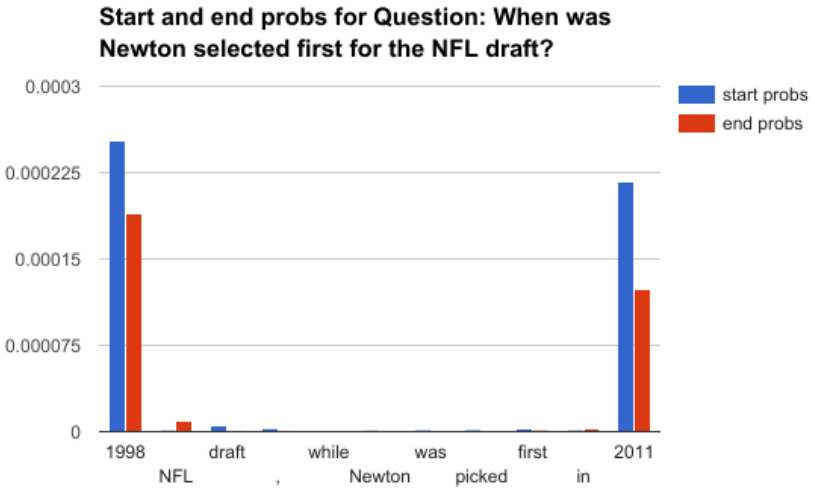



Figure 9: Beta probabilities for an incorrectly answered question (true answer is 2011)

## References


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

[2] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2015.

[3] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[4] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2016.