CS224N Final Project: Question Answering with SQuAD 2.0

Eli Echt-Wilson
eliew@stanford.edu

Kathryn Rydberg
rydbergk@stanford.edu

Abstract

In this paper we explore deep learning techniques for answering questions in the Stanford SQuAD 2.0 dataset. We want to assess the performance of different deep learning architectures for models that can most accurately answer questions based on context paragraphs. We first extend our baseline bidirectional attention flow model to include character level embeddings in addition to word embeddings, and experiment with various hyperparameters to achieve improvements on the baseline model (these improve the baseline performance by about a point for both F1 and EM scores). Second, we implemented a Match-LSTM with Answer Pointer model, trying to leverage a more complex architecture to improve performance. Due to computational and time limitations (training Match-LSTM took upwards of 50 hours), we were only able to train a single version of the Match-LSTM model and it underperformed our baseline architecture on EM and F1 scores. Further work needs to be done to improve the efficiency and accuracy of our Match-LSTM model.

1 Introduction

In 2016, Rajpurkar et al. released the Stanford Question Answering Dataset (SQuAD) [1], which consists of a series of passages from Wikipedia articles, a question corresponding to each passage, and an answer to the question. Human performance on this task is 86.8%, but initial logistic regression models achieve an F1 score of 51.0%. In this paper, we aim to improve the F1 score using deep learning models.

2 Related Work

In Seo et al.'s paper on "Bidirection Attention Flow for Machine Comprehension"[2], the authors approach question answering using the SQuAD 2.0 dataset by constructing a model that uses a Bidirectional LSTM. Their model concatenates a character embedding and a word embedding before feeding it into the model. Our baseline is similar to this model but does not use character embeddings.

Wang and Jiang approach the problem with a completely different model in their paper "Machine Comprehension Using Match-LSTM and Answer Pointer"[3]. The general goal of a Match-LSTM and Answer Pointer model is, given a premise and a hypothesis, to predict whether the premise contains the hypothesis. For question answering, the passage is treated as the premise and the question is treated as the the premise. Our project aims to implement this model.

3 Approach

3.1 Baseline Model

We are using the provided baseline, which is modeled as a bidirectional LSTM. Our experiments will work to improve the performance compared to this baseline. For more information regarding the baseline, see the Default Final Project Handout [1] and the following paper on bidirectional-attention flow for machine comprehension [2].

3.2 Improvements to Baseline Model

3.2.1 Character Embedding Model

Our current iteration of the model differs from the baseline in a couple of ways.

First of all, the baseline Embedding model takes word indices for both the question and the context, looks up the corresponding word embeddings, then feeds both sets of embeddings into a two-layer highway network: given an input $\mathbf{h}_i$, a single highway network is applied, where g indicates gate, t indicates transform, $\mathbf{W}_g, \mathbf{W}_t \in \mathbb{R}^H$, and $\mathbf{b}_g, \mathbf{b}_t \in \mathbb{R}^H$:

$$\mathbf{g} = (\mathbf{W}_g \mathbf{h}_i + \mathbf{b}_g) \in \mathbb{R}^H$$
$$\mathbf{t} = \texttt{ReLU}(\mathbf{W}_t \mathbf{h}_i + \mathbf{b}_t) \in \mathbb{R}^H$$
$$\mathbf{h}'_i = \mathbf{g} \odot \mathbf{t} + (1 - \mathbf{g}) \odot \mathbf{h}_i \in \mathbb{R}^H$$

We first extended this model so that it trains character embeddings and combines the two embeddings (through concatenation) before feeding them into the two-layer highway network. The highway network has an increased input size to accommodate the character and word level embeddings. Character-level word embeddings are produced by first looking up the character embeddings for the characters in a word, then applying a CNN over the character embeddings and a max-pool layer (similar to Assignment 5 embeddings) [2]. The output of the CNN layer is concatenated with our previous input to the highway network ($\mathbf{h}_i$ above) before feeding it into the two layer Highway.

3.2.2 Fine Tuning

In addition to adding character-level embeddings, we did a variety of hyperparameter tuning experiments to improve on the baseline model (see Experiments section). We primarily experimented with changing the dropout probability for regularization and changing the learning rate. The baseline model uses an Adadelta optimizer, and we also experimented with using other optimizers such as an Adam optimizer to see if it improved performance.

Our final change to the baseline model has been switching the encoder and output layers to use GRUs instead of bidirectional LSTMs. We find that doing so allows us to train the model faster while achieving comparable performance.

3.3 Match-LSTM and Answer Pointer Model

Next, we implemented a Match-LSTM and Answer Pointer Model, using Wang's and Jiang's research.[3] This Model had three parts:

1. A Pre-processing layer
2. A Match-LSTM layer
3. An Answer-Pointer Layer

3.3.1 Preprocessing Layer

In the preprocessing layer, a bidirectional LSTM is applied to the context and question separately. This step functions to incorporate the context of both the answer and the question for the next steps. The output of the LSTM, $\mathbf{H}^p \in \mathbb{R}^{l \times P}$ and $\mathbf{H}^q \in \mathbb{R}^{l \times Q}$, where P is the length of the passage, Q is the length of the question, and l is the hidden size of the LSTM.

[1] http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf

[2] http://web.stanford.edu/class/cs224n/assignments/a5.pdf

3.3.2 Match-LSTM Layer

Next, the output from the previous layer, $\mathbf{H}^p \in \mathbb{R}^{l \times P}$ and $\mathbf{H}^q \in \mathbb{R}^{l \times Q}$, is fed into a Match-LSTM layer, which goes through the passage sequentially to calculate the attention, using the following equations:

$$\vec{\mathbf{G}}_i = \texttt{tanh}(\mathbf{W}^q \mathbf{H}^q + (\mathbf{W}^p \mathbf{H}_i^p + \mathbf{W}^r \mathbf{h}_{i-1} + \mathbf{b}^p) \odot \mathbf{e}_Q)$$

$$\vec{\alpha}_i = \texttt{softmax}(\mathbf{w}^T \mathbf{G}_i + b \odot \mathbf{e}_Q).$$

In the above equations, $\mathbf{W}^q, \mathbf{W}^p, \mathbf{W}^r \in \mathbb{R}^{l \times l}$, $\mathbf{b}^p, \mathbf{w} \in \mathbb{R}^l$, and $b \in \mathbb{R}$. We use these value to compute $\mathbf{z}_i$, which is the concatenation of $\mathbf{h}_i^p$ and $\mathbf{H}^q \alpha_i^T$, which is fed into a unidirectional LSTM to produce h_i. On each step, the previous hidden state, $\mathbf{h}_{i-1}$, is used to calculate the next state (see equations above).

The attention calculated in this layer indicates the degree to which the ith word in the passage corresponds to the jth word of the question.

We also constructed a match-LSTM as described above but in the reverse direction so as to encode the contexts of the passage in both directions.

The result of the Match-LSTM layer is a matrix, $\mathbf{H}^r \in \mathbb{R}^{2l \times P}$, which is a concatenation of $\overrightarrow{\mathbf{H}^r}$, the hidden states of the forward layer, and $\overleftarrow{\mathbf{H}^r}$, the hidden states of the backward layer.

3.3.3 Answer-Pointer Layer

The Answer-Pointer Layer take $\mathbf{H}^r$, the output from the Match-LSTM layer, as input. We decided to implement the Boundary Model, which attempts to identify the start and end tokens of the answer from the passage (the alternative presented in the paper was a Sequence Model, which selects the indices of all of the tokens that should appear in the answer). Since the question is a sequence of consecutive words, we felt the boundary model was a better fit for our problem.

The Answer-Pointer layer has a similar framework to the Match-LSTM layer. First, we take the output of the Match-LSTM layer $\mathbf{H}^r$ and calculate a probability distribution over the passage of being the start word in the answer using the following equations:

$$\vec{\mathbf{F}}_s = \texttt{tanh}(\mathbf{V}^q \mathbf{H}^r + (\mathbf{W}^a \mathbf{h}_0^a + \mathbf{b}^a) \odot \mathbf{e}_P)$$

$$\beta_s = \texttt{softmax}(\mathbf{v}^T \mathbf{F}_s + c \odot \mathbf{e}_P).$$

$$p(a_s | \mathbf{H}^r) = \beta_s$$

Then, we calculate a probability distribution β_e for the end token (which is $p(a_e | a_s, \mathbf{H}^r)$) using the same equations, with $\mathbf{h}_0^a$ replaced by the result of passing $\mathbf{H}^r \beta_s^T$ through a unidirectional LSTM.

Finally, we calculate the probability of the answer being bounded by start and end indices $\mathbf{a} = (a_s, a_e)$ by

$$P(\mathbf{a} | \mathbf{H}^r) = P(a_s | \mathbf{H}^r) P(a_e | a_s, \mathbf{H}^r)$$

4 Experiments

4.1 Data

Our data for our experiments is derived from the SQuAD 2.0 dataset, which contains triples of (passage, question, answer). Since the SQuAD test set is secret, we will be splitting the official SQuAD dev set into two parts for our project to form our dev and test sets.

4.2 Evaluation Metric

Our primary evaluation metrics are F1 and EM, since these are what is used for the leaderboard.

4.3 Experimental Details

We had six main categories of experiments, all compared to the provided baseline:

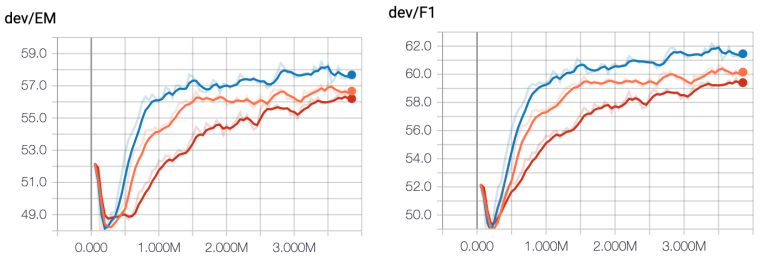
1. **Value for dropout probability**: The baseline uses 0.2 for the dropout probability. We ran experiments to see how the model does compared to the baseline when we use dropout probability of 0.1 and 0.3.
2. **GRU vs LSTM**: The baseline uses a bidirectional LSTM. We experimented with using a GRU instead of a bidirectional LSTM to see if we could get results comparable with the baseline but in less time.
3. **Character embedding**: We tried using a CNN to consider character embeddings. While this makes the model take more time to train, we wanted to see if it would improve F1 and EM scores.
4. **Learning Rate**: We tried tuning the learning rate to see how this alters the results compared to the baseline. The default learning rate is 0.5, and we experimented with learning rates of 1.0 and 0.25.
5. **Match-LSTM and Answer Pointer**: The next experiment was running a completely new model, the Match-LSTM and Answer Pointer model described above, rather than changing small things with the baseline model.
6. **Optimizer**: Finally, we experimented with the optimizer we used. The baseline used an Adadelta Optimizer, and we tried using an Adam Optimizer.

4.4 Results

We are on the Non-PCE Leaderboard. Our leaderboard submission for the dev set has an EM score of 58.293 and an F1 score of 61.675. Our scores for the test set are an F1 score of 60.129 and an EM score of 56.585.

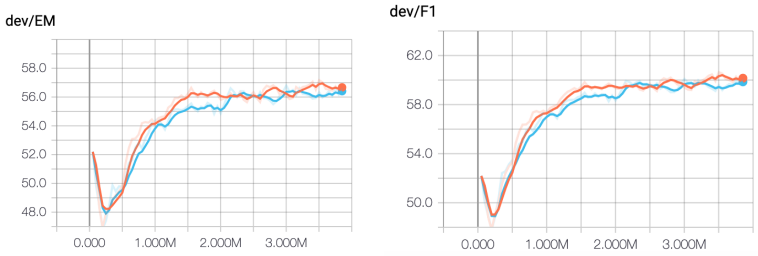
Experiment: Dropout Probability

For the first experiment, we ran the baseline three times with three different dropout probabilities to experiment with which dropout probability gave the best results. In the following graph, the orange line is the default dropout probability (0.2), the red line is a dropout probability of 0.3, and the blue line is a dropout probability of 0.1.



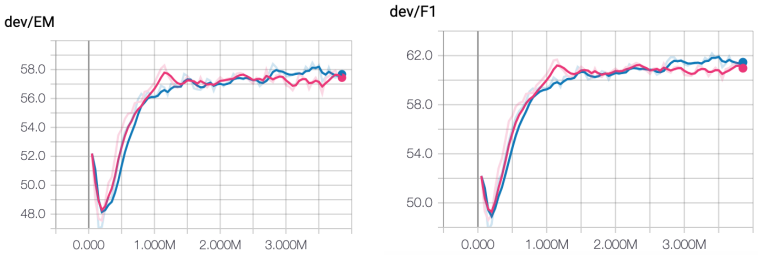
As seen in the graph, using a dropout probability of 0.1 improved both EM and F1 scores. The baseline has EM score of 56.67 and an F1 score of 60.15. Using dropout probability of 0.1 (the blue line) improves both scores, with an EM score of 57.68 and an F1 score of 61.46 and using dropout probability of 0.3 (the red line) decreased both scores, with an EM score of 56.20 and an F1 score of 59.41.

Experiment: GRU vs. LSTM We next ran experiments to see how GRU performs compared to LSTM. We started with running the baseline against a model that uses GRUs instead of bidirectional LSTMs. We wanted to see how just changing this affected the performance, so we started with not changing any other variables (i.e., dropout probability). In the following graphs, the orange lines are the baseline model and the light blue lines are the GRU model.



As seen in the graphs, the GRU model performs just about as well as the LSTM model. However, we found that the GRU model was faster to train than the LSTM model, which is helpful when training these large models.

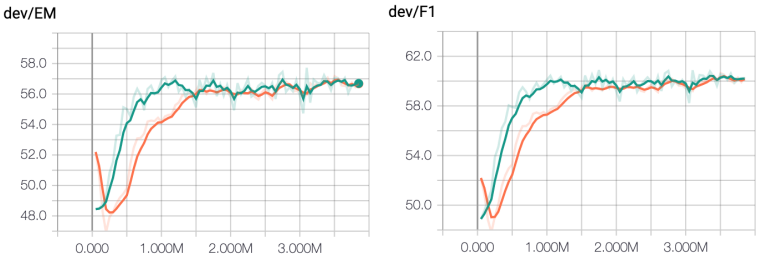
We wanted to then check that we achieved similar results when using a dropout probability of 0.1 instead of 0.2 (since we found that a dropout probability of 0.1 improved performance). In the following graphs, the dark blue line is the LSTM model and the pink line is the GRU model.



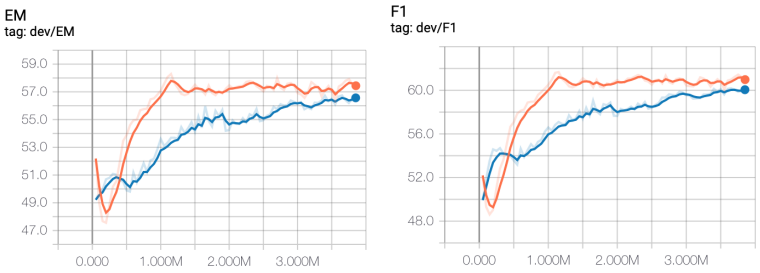
As observed in these graphs, the GRU and LSTM models also achieve comparable results when run with a dropout probability of 0.1. It was again faster to train the GRU than the LSTM.

Experiment: Character Embedding

Our next experiment was to include a character embedding model. We started with a CNN character model, which we then concatenated to the word model. Our results were comparable to the baseline. In the following graphs, the orange line is the provided baseline (on a GRU) and the green line is our model that uses character embeddings.



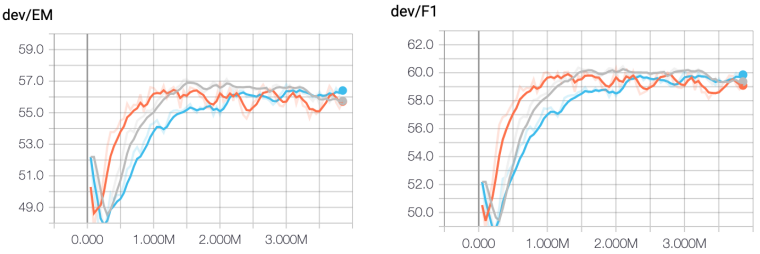
We noticed that the character embedding models started out better than the baseline, but then fluctuated a lot. From observing these results, we decided to implement learning rate annealing in order to start with a high learning rate and decay exponentially over time. In the following graphs, the orange line is the baseline model (but with a GRU) and the blue line is the character embedding model with learning rate annealing.



Learning rate annealing proved to be ineffective at improving performance, so we decided to stick with a constant learning rate for our model.

Experiment: Learning Rate

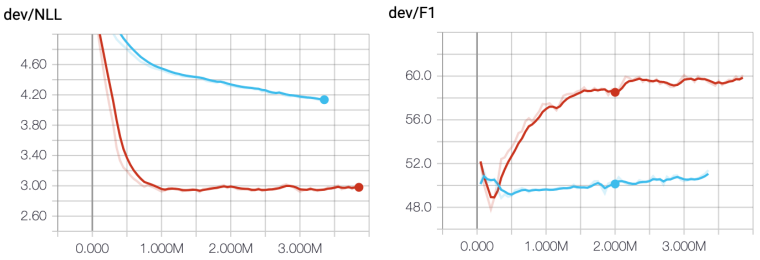
The next experiment was to test different values for the learning rate. In the following graphs, the light blue uses a learning rate of 0.5 (i.e., the baseline), the orange line uses a learning rate of 1.0, and the gray line uses a learning rate of 0.25.



From the graphs, we did not notice a significant difference in performance from the various learning rates, so we stuck with the default learning rate of 0.5 in our final model.

Experiment: Match-LSTM and Answer-Pointer

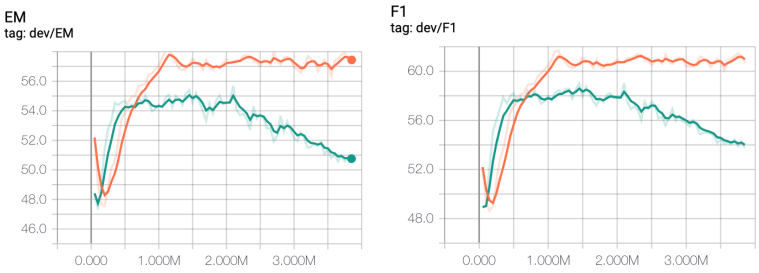
This experiment was the largest change we made since we implemented an entirely new model. While we did find some preliminary positive results (the loss on the dev set decreased over time during training and the model improved at detecting questions with no answer), the model did not achieve improvements on our EM or F1 scores. Additionally, due to the presence of many LSTMs in the model, it took significantly longer to train than the baseline (approximately 60 hours training time). This long training time made it extremely difficult to test different iterations of the model and figure out why it wasn't performing well on our primary evaluation metrics. Further work is necessary to improve the efficiency of the model and conduct more tests to figure out why our results are not matching those achieved in the reference paper. In the following graphs, the red line is the initial baseline and the blue line is the Match-LSTM and Answer Pointer Model.



As observed in the graphs, the loss decreases with the Match-LSTM and Answer Pointer Model, but the F1 score does not improve.

Experiment: Adadelta vs. Adam Optimizers

Our final experiment was with the optimizer we use when training our model. The baseline uses an Adadelta optimizer, and we tried using an Adam Optimizer. In the following graphs, the orange line is the baseline model (using the Adadelta optimizer) and the green line is uses the Adam optimizer.



From the graphs, we noticed that the Adam optimizer started out better, but then had a steep drop off and decrease in both EM and F1 scores. Because of this, we stuck with the Adadelta optimizer.

5 Analysis

5.1 Qualitative Analysis

We analyzed a subset of the questions (using the questions listed in the "Text" tab of Tensorboard). Of those questions, we looked 8 categories: why, what, when, who, how many, how, where, and which. Of the questions sampled for our best model (GRU with dropout 0.1), we had the following in each category:

Category	Num questions	Num correct	Num incorrect
Why	1	0	1
What	55	20	35
When	21	18	3
Who	9	5	4
How Many	7	4	3
How	4	2	2
Where	5	2	3
Which	1	1	0

We noticed that When and What questions were the most popular of any of the categories. When questions were the most accurate of any category. For the less popular categories (why, who, how many, how, where, and which), the number correct was within 1 of the number incorrect. We also noticed that our model did the worst on what questions. Upon further inspection, there were a few common mistakes. The first, and most prevalent, was when the correct answer was N/A, our model was too quick to pick an answer. Of the 35 incorrect what questions, 14 of them were when our model picked an answer when the correct answer was N/A. We also observed that many of the what questions were deemed incorrect even though a human would have the expected answer and predicted answer as equivalent. For example, consider the following two questions:

- Question: What was AUSTPAC
- Context: AUSTPAC was an Australian public X.25 network operated by Telstra. Started by Telecom Australia in the early 1980s, AUSTPAC was Australia's first public packet-switched data network, supporting applications such as on-line betting, financial applications — the Australian Tax Office made use of AUSTPAC — and remote terminal access to academic institutions, who maintained their connections to AUSTPAC up until the mid-late 1990s in some cases. Access can be via a dial-up terminal to a PAD, or, by linking a permanent X.25 node to the network.[citation needed]
- Answer: AUSTPAC was an Australian public X.25 network operated by Telstra
- Prediction: Australian public X.25 network

and

- Question: What German ruler invited Huguenot immigration?
- Context: Frederick William, Elector of Brandenburg, invited Huguenots to settle in his realms, and a number of their descendants rose to positions of prominence in Prussia. Several

prominent German military, cultural, and political figures were ethnic Huguenot, including poet Theodor Fontane, General Hermann von François, the hero of the First World War Battle of Tannenberg, Luftwaffe General and fighter ace Adolf Galland, Luftwaffe flying ace Hans-Joachim Marseille, and famed U-boat captain Lothar von Arnauld de la Perière. The last Prime Minister of the (East) German Democratic Republic, Lothar de Maizière, is also a descendant of a Huguenot family, as is the German Federal Minister of the Interior, Thomas de Maizière.

- Answer: Frederick William
- Prediction: Frederick William, Elector of Brandenburg

The correct answer and the answer given by our model were essentially the same, but not an exact match, which is an issue with selecting the start and end boundaries.

6 Conclusion and Future Work

In this paper we explored multiple deep learning techniques for question answering on the SQuAD 2.0 dataset. We first improved upon a provided baseline bidirectional attention flow model with relatively basic modifications such as adding character embeddings and hyperparameter tuning. These modifications yielded minor improvements to our EM and F1 performance on the dataset (about a point in each).

We then tried to implement Match-LSTM, an entirely different architecture, and encountered unanticipated shortcomings associated with the new architecture. Primarily, we found that the multitude of LSTMs used (in all three layers) significantly slowed training time, making it very difficult to test different iterations of the model given our time and computational resource limits. Given these limits, we may have achieved better performance implementing an architecture that was less computationally expensive to train. This experience highlighted the potential value of convolutional architectures and parallelizable architectures due to their ability to scale and train efficiently.

Additional work needs to be done to improve and potentially fix our Match-LSTM layer. While our basic sanity checks suggest that the model is correctly computing what is described in our reference paper, the results achieved from training the model do not match the results achieved in the paper. Further work needs to be done to correct any potential mistakes in the implementation, but primarily to improve efficiency so that new iterations of the model can be trained and tested in a reasonable amount of time.

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

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