Exploring Deep Learning Models for Machine Comprehension on SQuAD

Junjie Zhu
Department of Electrical Engineering
Stanford University
jjzhu@stanford.edu

Yifei Feng
Department of Electrical Engineering
Stanford University
yife@stanford.edu

Joseph Charalel
Department of Genetics
Stanford University
jcharalel@stanford.edu

Abstract

This paper explores the use of multiple models in performing question answering tasks on the Stanford Question Answering Database. We first implement and share results of a baseline model using bidirectional long short-term memory (BiLSTM) encoding of question and context followed a simple co-attention model [1]. We then report on the use of match-LSTM and Pointer Net which showed marked improvements in question answering over the baseline model [2]. Lastly, we extend the model by adding Dropout [3] and randomization strategies to account for unknown tokens.

Final test score on Codalab under Username: yife, F1: 66.981, EM: 56.845.

1 Introduction

Artificial question-answering systems are a key step toward machine comprehension of natural language. Recently, the Stanford Question Answering Dataset was established as a state-of-the-art resource for QA system development [4]. The SQuAD developers established a strong logistic regression model with and F1 score of 51% and a baseline model achieving 20% whereas humans achieved 86%. This performance gap left a clear opportunity for researchers to develop better models. The main task of question-answer systems we focus in this report can be described as follow: given a context paragraph and a question, find the answer in the paragraph that is a span of consecutive words. The accuracy of model's predicted answer is then evaluated by how well it overlaps with the word span of true answer.

Over the last decade, neural network models have outperformed others across a variety of NLP tasks including machine translation [5], part-of-speech tagging, and dependency parsing [6]. Currently, top performers on SQuAD are ensemble neural network models based on multilayer encoder-decoder BiLSTM frameworks (Fig. 1) [7, 8, 1]. These models both apply various versions of Recurrent Neural Networks (RNNs) [5], which is a class of models that captures dynamic temporal behaviors, such as sentence structures.

In order to further understand question-answering systems and state-of-the-art neural network models developed for these systems, we implemented and analyzed two different models: a simplified bidirectional encoder-decoder model and a more sophisticated match-LSTM model [1, 2]. Through comparative analysis, we assessed the similarities and dissimilarities of the two models in terms of design and empirical performance. We further improved the match-LSTM model with strategies to account for unknown tokens to handling unknown words (or out-of-vocabulary words).

Developing state-of-the-art models on any task is non-trivial. With a limited time and computational budget, we carefully designed our experiments, building off of our previous results, in order to arrive at our final model. While it is not comparable the newest state-of-the-art models, it was close to the performance to the original match-LSTM model [2] (67.98 *vs.* 70.695 `F1` score on the `test` set).

2 Models

Generally speaking, most state-of-the-art question-answering systems rely on a common architecture consisting of three main components (Fig. 1):

- Representation Layer: a layer that takes the context paragraph and the question paragraph and represents each of them independently via hidden embeddings. RNNs are typically used to capture the dynamic temporal relationship in the questions and contexts. In all our implementations, we first encode the question and context using bi-directional RNNs to create independent hidden representations of H_q for the question and H_c for the context .
- Attention Layer: a layer that mixes the relationship between the context representation and the question representation and outputs a question-to-context representation. This is a critical step for question-answering systems because it where the "machine understanding" happens. There are two different models we consider for this layer: the **Bi-directional Attention Flow Model** and the **Match-LSTM Attention Model**.
- Output (and Modeling) Layer: a layer that takes the output from the attention layer and outputs the a probability distribution of the start position of the question and a distribution for the end position. There are two different models we consider for this layer: the **LSTM Decoder** and the **Answer Pointer Decoder**.

For example, the original framework proposed in [1] (Appendix Fig. 5) and [2] (Appendix Fig. 6) have their own specific designs and implementations for these layers.

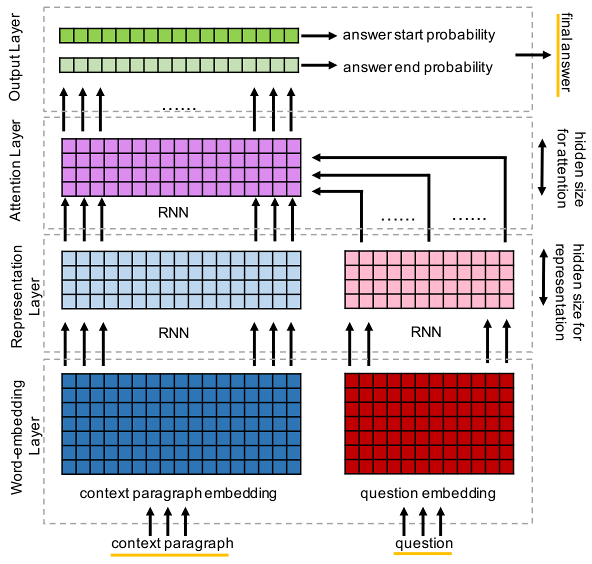

Figure 1: General framework for state-of-the-art methods.

Our baseline model includes combines the Bi-directional Attention Flow Model and the LSTM Decoder, and our final model combines the Match-LSTM Attention Model and the Answer Pointer Decoder, which additional designs. For both implementations we used the "boundary model" for prediction, where we construct a probabilistic model for the start and end positions for the predicted position. Let $p(a = i, e = j|\mathcal{M})$ be the probability that the start and end position are the ith and jth position of the context paragraph of length P_L, given a model $\mathcal{M}$. Then, we solve the following objective function

$$\text{minimize}(-\sum_{n=1}^{N} \sum_{i=1,j=1}^{P_L} y_{i,j} \log(p(s = i, e = j|\mathcal{M})))$$

where ground-truth label $y_{i,j} = 1$ if the true start and end positions are i and j respectively, and $y_{i,j} = 0$ otherwise. (Note that this is a 2-D one-hot matrix). Thus we minimize $\log(p(s = i, e = j|\mathcal{M})) = -\log(p(s = i|\mathcal{M})) + -\log(p(e = j|s = i, \mathcal{M}))$. For both models, each term can be independently optimized. Given the hidden layer capturing dependencies between question and context, we compared two different models: naive decoding and pointer decoding, to construct the length P_L probability vectors $p(\mathbf{s}|\mathcal{M})$ and $p(\mathbf{e}|\mathcal{M})$.

2.1 Bi-Directional Attention Flow Model

The Bi-Directional Attention Flow Model combines information between the context and the question by taking the inner products of the respective representations H_q and H_q:

$$G = \text{softmax}(H_c H_q^T); \;\; H_r = W_g[G; H_q] + b$$

where H_r represents the mixing of the two representation, and allowing for the entire question to mix with each context position.

2.2 Match-LSTM Attention Model

A match-LSTM cell can be mathematically defined as follows:

$$\overrightarrow{G}_i = \tanh(W^q H^q + (W^p h_i^p + W^r \overrightarrow{h}_{i-1}^r + b^p) \bigotimes e_Q)$$

$$\overrightarrow{\alpha}_i = \text{softmax}(w^\intercal \overrightarrow{G}_i + b \bigotimes e_Q) \quad \overrightarrow{h}_i^r = \overrightarrow{\text{LSTM}}(\begin{bmatrix} h_i^p \\ H^q \overrightarrow{\alpha}_i^\intercal \end{bmatrix}, \overrightarrow{h}_{i-1}^r)$$

where $\overrightarrow{H}_r = [\overrightarrow{h}_1^r; \overrightarrow{h}_1^r, .., \overrightarrow{h}_p^r]$, and H_r is obtained by concatenating $\overrightarrow{H}_r$ with its reversed RNN.

2.3 LSTM Decoder

The output probability of the LSTM decoder is given by:

$$p(\mathbf{s}|\mathcal{M}) = \text{softmax}(w_s H_r)$$

$$p(\mathbf{e}|\mathcal{M}) = \text{softmax}(w_e H_r^{(2)}),$$

where $H_r^{(2)} = LSTM(H_r)$, is used to model forward dependency from start to end. The weight vectors w_s and w_e capture the importance of each context position.

2.4 Answer Pointer Decoder

The answer pointer decoder is similar to an match-LSTM, and captures forward relationships between the start of and answer and the end of an answer. The basic cell is similarly defined (full details in [2], and the output of this RNN is given by:

$$h_k^a = \overrightarrow{\text{LSTM}}(\widetilde{H}^\Gamma \beta_k^\intercal, h_{k-1}^a).$$

The modeling probabilities are given as:

$$p(\mathbf{s}|\mathcal{M}) = \beta_1, \; p(\mathbf{e}|\mathcal{M}) = \beta_2,$$

and note that the RNN is only unrolled twice to point the start and the end positions.

2.5 Randomization Strategies to Handle Unknown Words

We noticed that many words on the dev or test set may be words that are not in GloVe or our vocabulary table. By default, these unknown tokens are mapped to the same embedding vector, meaning that they appear to be the same to the model. To handle the unknown words, we used two randomized procedures:

Random embedding selection: This allows us to spread out the unknown words so that when a context paragraph has many unknown tokens and the multiple questions are queried against this paragraph, the model identifies these unknown tokens as different words.

Random dropout during training: Dropout [3] is a common strategy in Deep Learning that prevents the model over-fitting to the training data. In the case of unknowns, we introduced dropout to improve the ability of the model to generalize to newer data sets.

3 Experiments

For all experiments, the word embeddings were fixed, and we used an Adam optimizer. All the experimental results discussed below are summarized in Table in the Appendix.

3.1 Dataset

To trained our different models, we used a dataset of (question, context paragraph, answer) tuples from the Stanford Question Answering Dataset (SQuAD)[4]. SQuAD contains 107785 crowd-sourced question-answer pairs based on 536 Wikipedia articles. This dataset was split into 81381 training entries 4284 and validation entries, and preprocessed along with 100-dimensional GloVe embeddings [9]. Based on the size distributions of the data (Appendix Figs. 7 and 8), we removed a training example if its question length was greater than 30 or if its context length was greater than 300.

3.2 Baseline Experiments

When starting any new problem, setting up an initial, bug-free pipeline is crucial for allowing fast iteration. Thus, our initial goal was to establish that our pipeline was working and capable of training models. The first strategy was to utilize a "`tiny`" dataset, consisting of 100 and 10 random samples from the `train` split for `tiny-train` and `tiny-val`, respectively. If a sufficiently complex model could not overfit and achieve 100% accuracy on `tiny-train` (and probably close to 0% on `tiny-val`), it would indicate that there is a bug in our pipeline, *e.g.*, the label are not correct, the model's equations are not computed correctly. Our initial model, as suggested by the CS224n TAs, is the simple baseline model described in Section 2.1. With our pipeline overfitting in this setting, we could be more confident in our pipeline when moving onto more complex models with the full `train` and `val` sets.

In our first set of experiments (Experiments 1-6), we experimented with 3 factors, which guided our next experiments. We used the baseline model (Section 2.1) with the following (`state size`, `batch size`, `learning rate`) tuples: (50, 32, 0.001), (50, 32, 0.005), (50, 64, 0.001), (50, 64, 0.005). (Other parameters used can be found in the Appendix.) We had a higher `F1` score with a larger batch size and smaller learning rate. Next, we increased the state size and tried to train additional models with the following settings: (100, 64, 0.001) and (100, 64, 0.005). With the larger state size, the gap between the learning rates (preferring smaller) was even more pronounced, with a difference of 10 points. This made it clear to us that the learning rate is critical in accessing how good a given model could be, thus we continued to experiment with multiple learning rates as we changed our model.

3.3 Match-LSTM Experiments

With the baseline model performing in the expected range, we implemented our match-LSTM model with a Pointer Answer decoder [2]. We tried our preliminary experiments (Experiments 7 and 8) with two version of this model with a higher learning rate (0.03) and quickly discovered that this learning rate was too high, causing the training loss to quickly start increasing, thus we terminated the experiments early and decided on a different set of parameters (Experiments 9-12). We decided to try a larger state size (150), keep the batch size at 64, and try 4 different learning rates (0.0001, 0.0005, 0.001, 0.005). The best learning rate, 0.001, achieved a maximum `F1` score of 67.00, significantly outperforming the other learning rates, so we ended up using this setting frequently in later settings. One key realization during designing this experiment was that our models were not utilizing all of the available GPU memory. Thus, using TensorFlow's features, we learned how to control how much GPU memory and which GPU (if multiple were present) the model training would occur on. This was crucial in allowing us to carry out multiple experiments simultaneously, letting us explore the hyperparameter space more effectively. In addition to this experiment, we wanted to see how important the Answer Pointer decoder was over a naive decoder, so we carried out 3 experiments (13-15) with learning rates of (0.0005, 0.001, 0.005), dropping 0.0001 since it was much too slow in the previous experiment, yielding terrible performance. While the best of these experiments achieved comparable `F1` accuracies to our best Answer Pointer decoder model, we noticed that the Answer Pointer decoder had significantly fewer parameters (see the Table in the Appendix), which also did not grow as the maximum context length parameter was increased. Thus, we proceeded to use the matchLSTM + Answer Pointer decoder for further experiments.

We also experimented with different learning rate decay schemes (Experiments 18-20). The previous experiments used an step exponential decay, *i.e.*, the learning rate would be scaled by 0.1 every `x` iterations (with `x` starting at 6, but moving to 7 when we realized we were training for 15 epochs). We tried three model settings where we had the learning rate scaled by 0.9 every epoch. Since high learning rates are helpful in the beginning of training but detrimental near the end, we tried higher learning rates. We found that this strategy did not perform any better than the previous strategy, so we abandoned it in favor of our initial strategy.

3.4 Beyond Match-LSTM Experiments

At this point, we noticed that we were having a problem with generalization. Our models would achieve relatively low losses *and* have great scores on the `val` set (with all words in our vocabulary), but do much worse when evaluated on the `tiny-dev` and `dev` sets. Multiple strategies have been proposed to reduce overfitting and help generalization, such as weight normalization, batch normalization, noisy gradients, decorrelating features, *etc.*. We decided to experiment (Experiments 21-28) with Dropout [3], due to it being simple to implement in TensorFlow and widely used in many state of the networks across different disciplines. We added dropout between the word embedding and the Q and P LSTMs (to simulate having noisy word embeddings caused by out-of-vocabulary words), before the decoder layer, and, if applicable, before the linear classifier that predicted the final start and end indices. Our first experiments, we tried three different dropout rates, parameterized by the "keep probability" rates (*i.e.*, 1 - dropout) that TensorFlow uses: 0.95, 0.90, and 0.85. We found that the 0.85 performed best, achieving comparable `val` accuracy as before, but noticed significant gains in `dev` performances. With a regularization method in place, this also allowed us the try increasing the state size to 300 from 150 (Experiment 24), which ended up yielding our best performing model (Fig. 9).

For our final set of experiments (25-28), we tried increasing the size of the maximum content length and also slightly increasing the state size to 200 for the last two. These experiments yielded a model with identical `val` performance as our best model, but performed slightly worse on `dev` and `test`.

4 Results

Our baseline model result showed that increasing complexity by doubling state size, allowed a smaller learning rate to better results (Fig. 2). However, the best baseline model run yielded an F1 score of merely 47 on validation set suggesting a lack of expressive power needed to capture the intricacy of relationships between context and question.

With a much more sophisticated attention mechanism, the match-LSTM model significantly outperformed the baseline model (Fig. 3). However, for both baseline and match-LSTM, the best validation set results were attained fairly early during training, and the training loss continued to decrease for a long time. This indicates that both models were overfitting to the training data. To address this, we implemented dropout which significantly prevented the model from overfitting (Fig. 4). We also noticed that in order to compensate for the loss of representation capability from dropout, we needed to increase the state size to boost performance. For two experiments with the same dropout rate, the one with larger state size performed notably better.

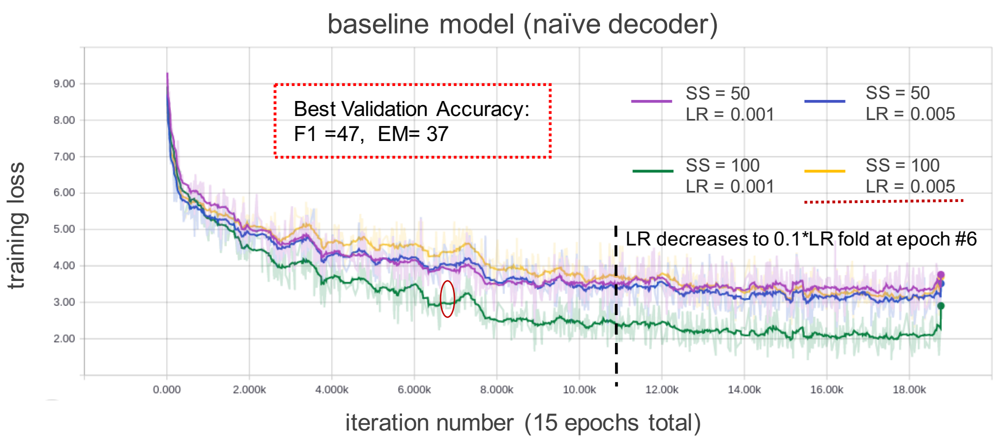

Figure 2: Loss curve for baseline models.

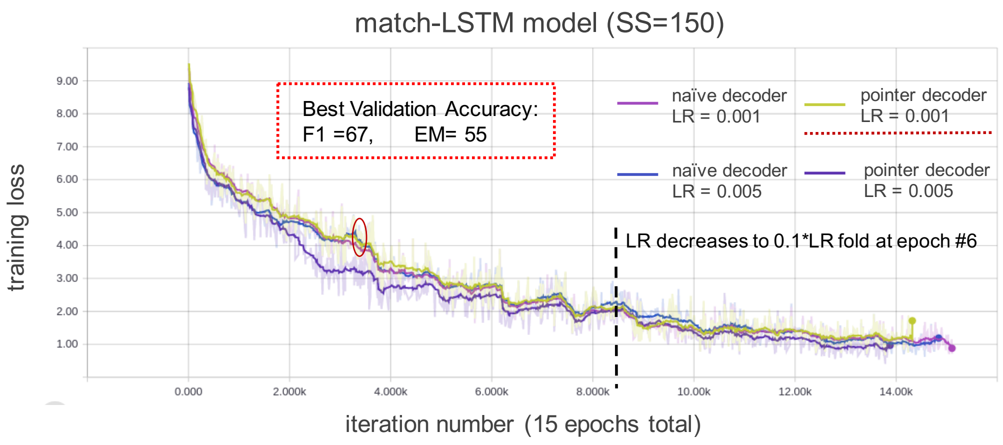

Figure 3: Loss curve for match-LSTM models.

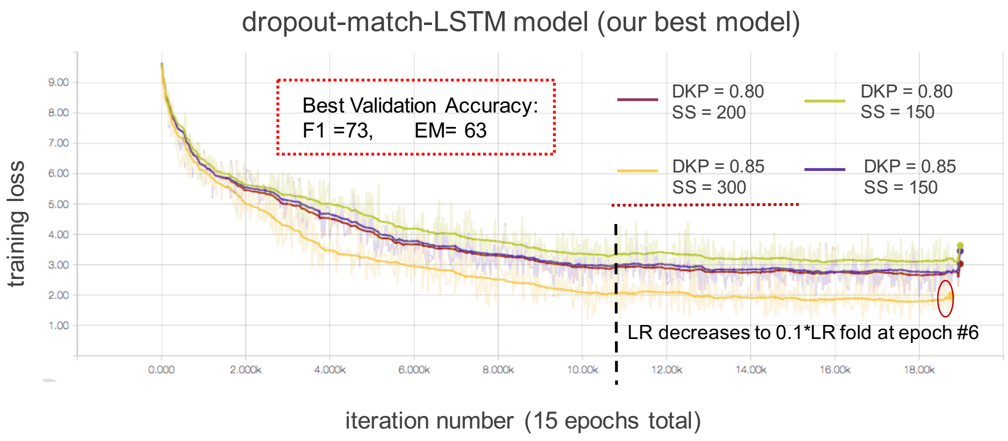

Figure 4: Loss curve for match-LSTM models with different dropout rates.

5 Error Analysis

Following are some interesting examples of wrong/inaccurate predictions from our final model.

-
  ```
  Context: Most of the G3P molecules are recycled back into RuBP using
  energy from more ATP, but one out of every six produced leaves the
  cycle.
  Question: How many G3P molecules leave the cycle ?
  Predicted Answer: Most
  True Answer: one out of every six
  ```

  In the above example, our model predicted the answer 'Most' because it learned that preposition 'of' can imply an amount or number, which corresponds to the question of 'how many'. It also learned that 'of' indicates a reference to 'G3P molecules', which was the object of the question. However, it missed the fact that the question was asking about the event 'leave the cycle', and failed to put enough attention on the second half of the context. It is possible that the model had trouble disambiguating the multiple meanings of the phrase 'out of', which can describe a spatial relationship or frequency.

-
  ```
  Context: Denver linebacker Von Miller was named Super Bowl MVP.
  Question: What position does Von Miller play ?
  Predicted Answer: Super Bowl MVP
  True Answer: linebacker
  ```

  In the above example, our model inferred that 'Super Bowl MVP' modifies 'Von Miller' from the word 'was'. However, it very likely did not understand that 'linebacker' is a position in football.

-
  ```
  Context: ...recording five solo tackles, 2 sacks, and two forced fumbles
  # Wrong Prediction
  Question: How many tackles did Von Miller get during the game ?
  Predicted Answer: two
  True Answer: 5
  # Correct Prediction
  Question: What was the number of solo tackles that Von Miller had in
  Super Bowl 50 ?
  Predicted Answer: five
  True Answer: 5
  ```

  The above set of examples shows that when the question asks about 'solo tackles', our model connected the number '5'. However, when the question omitted the word 'solo', it got confused and chose the number word '2' which was the closest to 'tackle'.

6 Conclusion

We implemented two neural architectures and were able to perform preliminary tuning on each to improve model performance. In particular, we saw performance increases by keeping learning rate at .001, increasing batch size from 32 to 64, and increasing state size from 50 to 150. It is likely that further tuning and training an ensemble of models would yield significant improvement over our current implementation. Overall, we saw major performance gains in the match-LSTM model compared to the bidirectional attention flow model implementation. However, our randomization strategies to handle unseen word tokens did not have major impact on our prediction results.

Looking at the mistakes that even our best model makes, it is clear that longer range word-word interactions are still difficult to resolve. In the first mistake example given above, close proximity of the quantifier "most" (compared to the correct answer) to the noun of interest is likely a major cause for this mistake. Moreover, cases where phrases may take on multiple meanings or where correct answer strings have attributes that are also found in other local context elements appear problematic. However, we did not identify clear patterns in our error making further model development based on such analysis difficult.

An interesting potential extension of match-LSTM would be to adapt the model to predict responses to other reading comprehension datasets. For example, the Cornell Movie-Dialog Corpus contains over 200K conversational exchanges, many of which are likely to contain questions [10]. Questions could be extracted from this dataset along with preceding text of a given length or structure (i.e. some number of quotes or up to some word limit) and the corresponding response from characters. Moreover, the MCTest database contains an additional set of reading comprehension questions for which match-LSTM could be adapted to predict multiple choice style responses which also contain strings from the context [11]. Given multiple datasets from which to learn it would be of great interest to explore the feasibility and impact of transfer learning across QA datasets.

Contribution

All team members analyzed the papers and their implementations. Junjie Zhu and Yifei Feng led the model implementation in Tensorflow. All members drafted the report and poster, and submitted the final version to CodaLab.

Acknowledgments

We would like to thank the CS224n course staff including Professors and TAs for their instruction and guidance throughout the course.

References

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

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

[3] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

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

[5] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[6] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *CoRR*, abs/1603.06042, 2016.

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

[8] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *CoRR*, abs/1612.04211, 2016.

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[11] *MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text*, October 2013.

7 Appendix

Experiments, Parameters and Results

	Model	D	SS	LR	BS	DKP	MCL	LRD	EPD	Max F1	Max EM
1	baseline	naive	50	0.001	32	1	300	0.1	6	0.41	0.29
2	baseline	naive	50	0.005	32	1	300	0.1	6	0.39	0.29
3	baseline	naive	50	0.001	64	1	300	0.1	6	0.41	0.3
4	baseline	naive	50	0.005	64	1	300	0.1	6	0.39	0.28
5	baseline	naive	100	0.001	64	1	300	0.1	6	0.47	0.37
6	baseline	naive	100	0.005	64	1	300	0.1	6	0.36	0.23
7	matchLSTM	pointer	50	0.03	32	1	300	0.1	10	0.11	0.09
8	matchLSTM	pointer	100	0.03	64	1	300	0.1	10	0.14	0.1
9	matchLSTM	pointer	150	0.001	64	1	300	0.1	7	0.67	0.55
10	matchLSTM	pointer	150	0.005	64	1	300	0.1	7	0.64	0.51
11	matchLSTM	pointer	150	0.0005	64	1	300	0.1	7	0.53	0.41
12	matchLSTM	pointer	150	0.0001	64	1	300	0.1	7	0.20	0.13
13	matchLSTM	naive	150	0.0005	64	1	300	0.1	7	0.56	0.46
14	matchLSTM	naive	150	0.001	64	1	300	0.1	7	0.65	0.55
15	matchLSTM	naive	150	0.005	64	1	300	0.1	7	0.67	0.58
16	matchLSTM	pointer	150	0.01	64	1	300	0.1	7	0.36	0.27
17	matchLSTM	pointer	200	0.005	64	1	300	0.1	7	0.61	0.5
18	matchLSTM	naive	150	0.01	64	1	300	0.9	1	0.30	0.23
19	matchLSTM	pointer	150	0.005	64	1	300	0.9	1	0.63	0.51
20	matchLSTM	naive	150	0.005	64	1	300	0.9	1	0.65	0.54
21	matchLSTM	pointer	150	0.001	64	0.95	300	0.1	7	0.68	0.56
22	matchLSTM	pointer	150	0.001	64	0.9	300	0.1	7	0.68	0.59
23	matchLSTM	pointer	150	0.001	64	0.85	300	0.1	7	0.69	0.58
24	matchLSTM	pointer	300	0.001	64	0.85	300	0.1	7	0.73	0.63
25	matchLSTM	pointer	150	0.001	64	0.8	350	0.1	7	0.64	0.55
26	matchLSTM	pointer	150	0.001	64	0.85	350	0.1	7	0.64	0.55
27	matchLSTM	pointer	200	0.001	64	0.8	350	0.1	7	0.73	0.65
28	matchLSTM	pointer	200	0.001	64	0.85	350	0.1	7	0.71	0.59

D=Decoder, SS=Hidden State Size, LR = Learning Rate, BS = Batch Size,DKP = Dropout Keep Probability,MCL = Max Content Length,LRD = Learning Rate Decay Factor,EPD = Epochs Per Decay,Max F1 = Max F1 Score ,Max EM = Max Exact Match Score

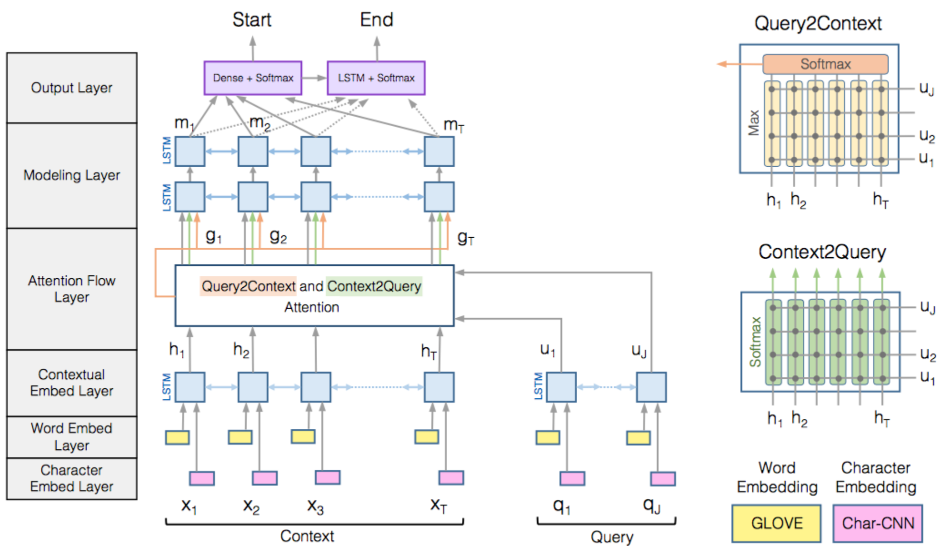

Figure 5: Bi-direction Attention Flow Architecture [1]

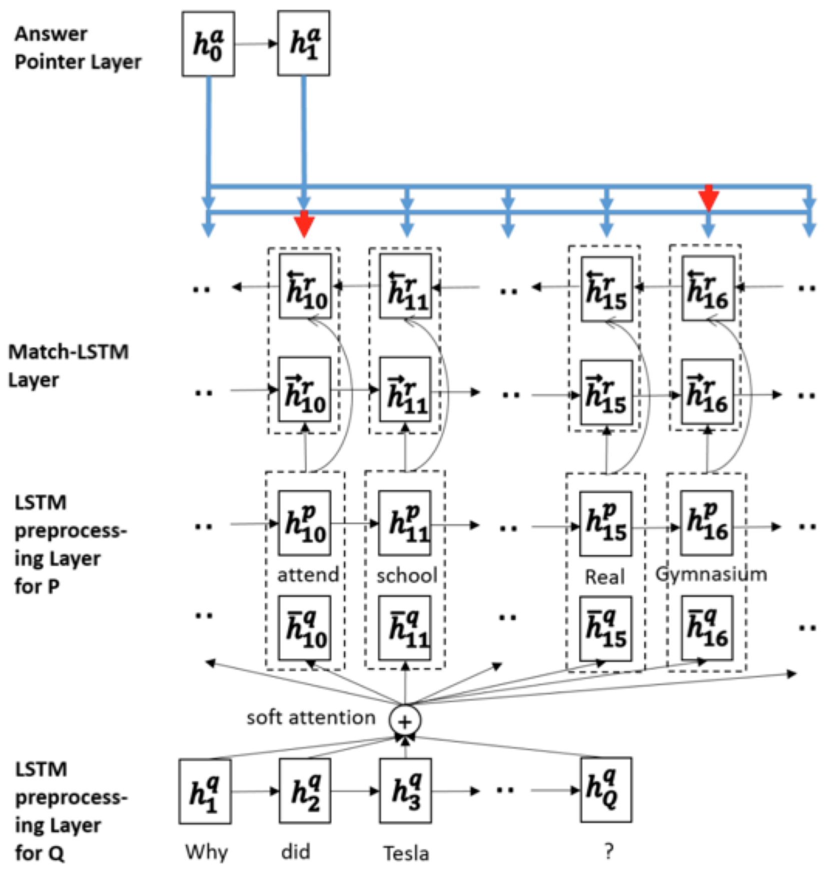

Figure 6: Match-LSTM Boundary Model Architecture [2]

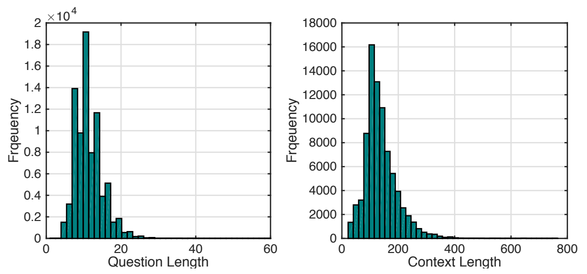

Figure 7: Distribution of the question, context and answer lengths.

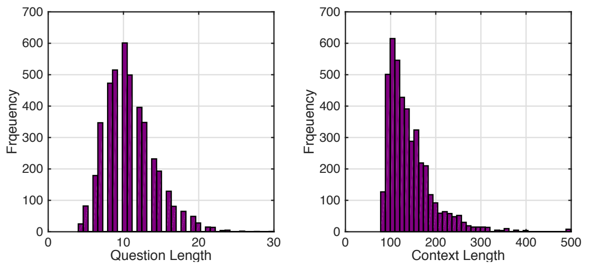

Figure 8: Distribution of the question, context and answer lengths.

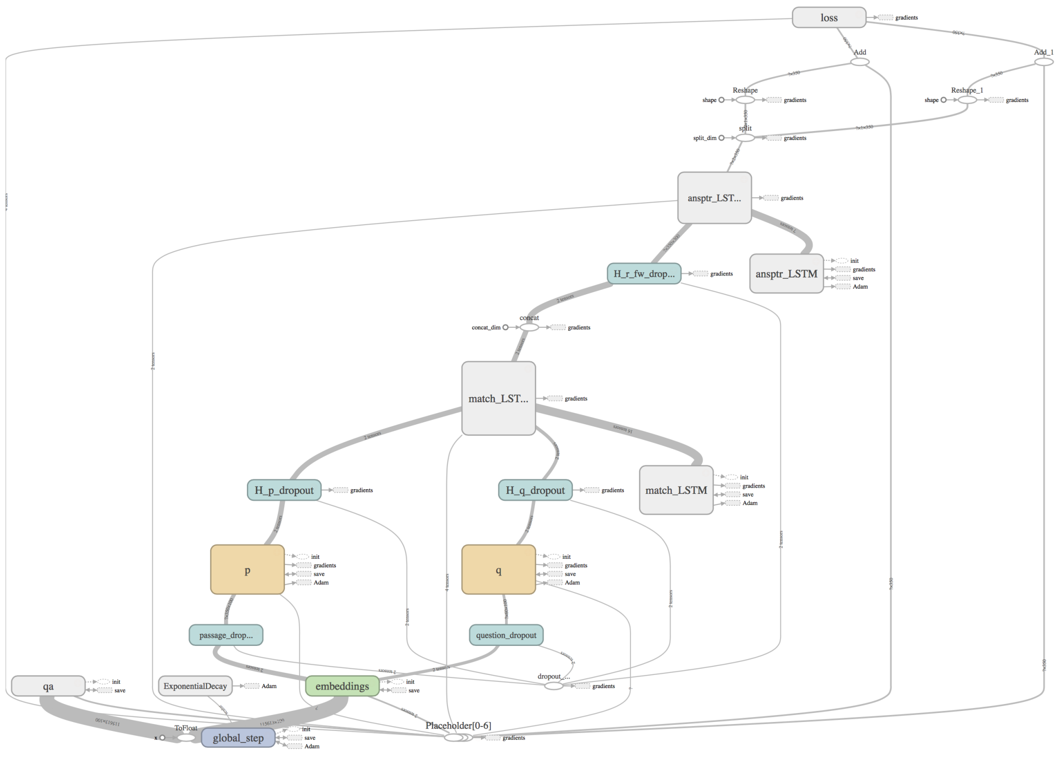

Figure 9: Tensorflow graph of our final match-lstm model with dropouts.