Assignment 4: Question Answering on the SQuAD Dataset with Part-of-speech Tagging

Joan Creus-Costa, Philip Hwang, and Nancy Xu
Stanford University
jcreus@stanford.edu, phwang20@stanford.edu, xnancy@stanford.edu

Abstract

This research applies deep learning with bi-LSTMs to train a model that responds to queries on the Stanford Question Answering Dataset (SQuAD). The model design was motivated by Wang et. al.'s December 2016 IBM Research paper on multi-perspective context matching for machine comprehension. Using TensorFlow, we implemented a multi-layer neural network architecture utilizing bi-directional LSTMs and context-based processing to maximize scores in the dataset. Our model achieves a performance of 61% F1 on the hidden test despite its small number of parameters, and offers room for improvement in multiple directions.

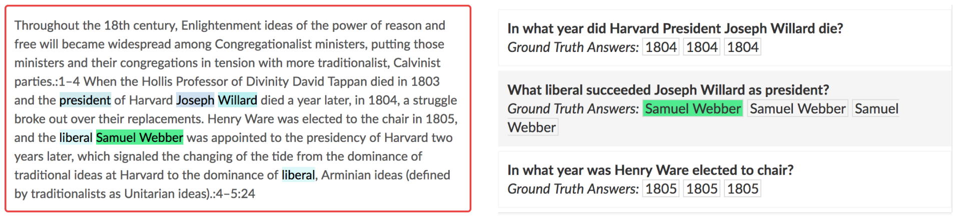
1 Introduction

Since the release of the Stanford Question Answering Dataset (SQuAD) in 2016, training end-to-end models for machine comprehension (MC) has become more accessible than ever before. Previous machine comprehension datasets were either too small to train complex models on or too easy to allow for evaluation of the relative performance of newer models. SQuAD alleviates several of these concerns. We propose a deep learning architecture for machine comprehension based on training feedback on SQuAD. In this work, we build on the Multi-Perspective Context Matching Model of Wang et al., as well as other recent work in machine comprehension by others in developing high-scoring models for SQuAD. In this section, we will outline some previous attempts as well as preluding motivations for the present model.

Some perhaps naive but reasonable baseline models might include some aggregation of bi-LSTMs with no nontrivial intermediate transformations or even variants of Logistic Regression [10]. In practice, these approaches prove to be quite poor in performance in comparison to many of the state-of-the-art. For one, the difficulty of the machine comprehension task can prove too challenging for models that lack the proper expressive power, such as in the case of logistic regression. Furthermore, the amount of unessential "noise" (with respect to the answer of the question) in large passages might appear too frequently for simple aggregations of the standard deep learning arsenal of bi-LSTMs and CNNs to efficiently learn.

Intuitively, one might assume that a distinguishing characteristic between humans and naive models highlighted above is the ability of humans to quickly "skim" through passages to focus on subsets of the passage with demarcating words related to the task at hand. A human might not need completely parse the contents of a passage to find a desired answer. For example, if a question of the form "What date was Barack Obama born?" it would be unnecessary, and perhaps even damaging to performance, for one to completely read the former President's Wikipedia page to find his date of birth. One could easily simplify answering this question by identifying that the answer is most probably in the form of a number and within in some neighborhood of words such as "born." As many high-performing models have been developed since the inception of SQuAD, many of these models appear to mirror this characteristic. For example, the BiDAF model (achieving an F1 of

81.525%) developed by Seo et al. makes use of a so-called attention flow layer which attempts to capture directionally pairwise similarities between words in Q and words in P [3]. Xiong et al.'s recent model ($F1 = 80.4\%$) introduces to the task Dynamic Coattention Networks, which is loosely guided to efficiently locate the important parts of the sentences [8]. Wang et al.'s model, of which our present model is inspired by, uses two intermediate layers composed of variants cosine similarity functions which also have mimicked the "filtering" task (we will later illustrate the effect matching intermediates have on our model in Figure 3). Thus, our main motivation in developing the components to our model was aimed at fine-tuning previous sentence intermediate layers and improving model attention mechanisms to generalize more efficiently.


Throughout the 18th century, Enlightenment ideas of the power of reason and free will became widespread among Congregationalist ministers, putting those ministers and their congregations in tension with more traditionalist, Calvinist parties.:1–4 When the Hollis Professor of Divinity David Tappan died in 1803 and the president of Harvard Joseph Willard died a year later, in 1804, a struggle broke out over their replacements. Henry Ware was elected to the chair in 1805, and the liberal Samuel Webber was appointed to the presidency of Harvard two years later, which signaled the changing of the tide from the dominance of traditional ideas at Harvard to the dominance of liberal, Arminian ideas (defined by traditionalists as Unitarian ideas).:4–5:24

In what year did Harvard President Joseph Willard die?
Ground Truth Answers: 1804 1804 1804

What liberal succeeded Joseph Willard as president?
Ground Truth Answers: Samuel Webber Samuel Webber Samuel Webber

In what year was Henry Ware elected to chair?
Ground Truth Answers: 1805 1805 1805

Figure 1: Sample SQuAD Data Entry [10].

2 Problem Statement

Here, we would like to properly formalize our task. Similarly to the formalization by Wang et al., our machine comprehension task involves taking some passage $P = \{p_1, p_2, \ldots, p_N\}$ and question $Q = \{q_1, q_2, \ldots, q_M\}$ and computing the correct portion of the passage P which produces an answer A for question Q. We simplify the problem by predicting two indices a and b representing the start p_a and end p_b of the answer, assuming that they are independent. Note that we require $1 \leq a \leq b \leq N$. We are therefore trying to maximize the probability of the start and end indices being correct given the question and the passage.

Our architecture implements a multi-layer neural network architecture with bi-directional LSTMs for context representation and vector aggregation. We also use a vector-based relevancy matrix to filter key words in the initial data sample. The full model builds upon previous research by making use of a matching layer to establish dependencies between the question and passage, filtering input using neighborhood similarities, along with part-of-speech (POS) tagging to provide better sentence context for the training model. Training F1 and EM scores are used as a baseline for development.

3 Model Architecture

Our neural network architecture builds on the previous multi-layer network proposed by Wang et. al.'s December 2016 IBM Research paper. Following the standard machine comprehension architecture of word embedding layer, contextual embedding layer, matching layer, and aggregation, Wang et. al. developed a unique matching network based on multiple perspectives building vectors through cosine similarity. Our model consists of a simplified version of that model, that excludes LSTM character embeddings and avoids the full complexity of the original matching layer by using naive cosine similarity. As a way to improve the results we added part-of-speech (POS) tagging and tuned the prediction layer logic, as well as exploring the hyperparameter space and design choices.

Layer 0: Word Embeddings. We transform each word in the input to a vector that represents the word. We used the 6B Wikipedia GloVe vectors with $d = 300$ dimensions. Since it's an uncased dataset, we mapped upper and lowercase versions of the word to the same vector. This gives us two tensors, $\mathbf{q}$ and $\mathbf{p}$, in $\mathbb{R}^{Q \times d}$ and $\mathbb{R}^{P \times d}$ where Q, R are the length of the question and the paragraph. We also concatenate part of speech data, as explained in the section below.

Layer 1: Filter Matrix. As in [6], the model computes a relevancy degree r_j for each word p_j in the passage P. The relevancy score gives a similarity-based estimate of the relevancy of each word

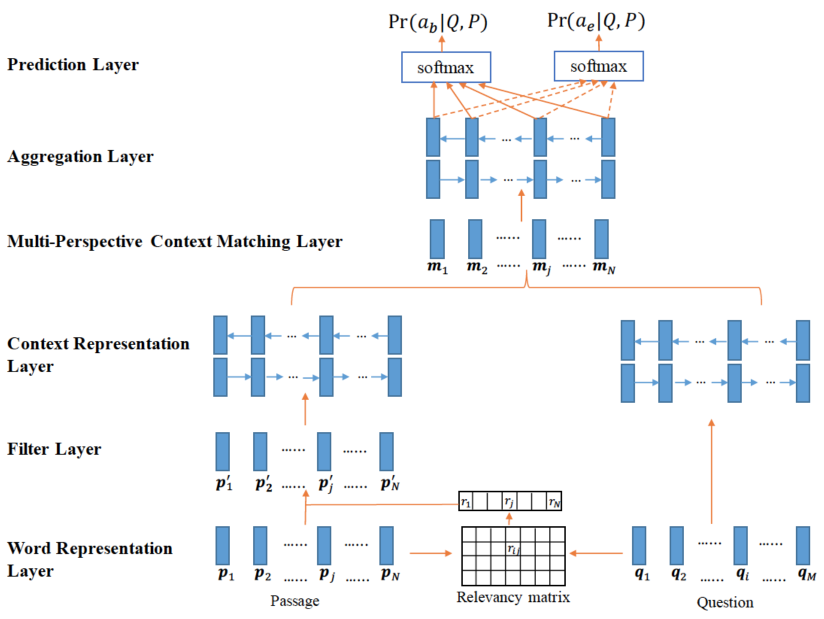


Figure 2: General architecture from *Multi-perspective Context Matching Model* (Wang et al.) [6].

in the passage to the final query answer and is computed as follows, where $r_{i,j}$ is the (i,j) entry of the filter matrix.

$$r_{i,j} = \frac{q_i^T p_j}{||q_i|| \cdot ||p_j||} \tag{1}$$

$$r_j = \max_{i \in M} r_{i,j} \tag{2}$$

The resulting filtered representations are given by

$$\mathbf{p}'_j = r_j \cdot \mathbf{p}_j \tag{3}$$

Essentially, this works as a way to assign preliminary weights to each word in the passage according to how related they are to the question. As explained below, this step has room for improvement that will be explored in future work.

Layer 2: Context Representation. The context representation layer performs a bi-LSTM on both the question and answer vectors given by the filter layer. This layer embeds the context information of the textual input into output vectors of the RNN. The same bi-LSTM cell, with hidden size $h = 150$, is applied to both contexts as shown above.

Layer 3: Context Matching. Here we use a simplified model that does not use the full power of multi-perspective matching in [6]. While the paper showed that this layer had a noticeable effect on scores, we decided to use naive cosine with three matching strategies instead to both simplify the model (and hence training time) and to allow us to have some baseline we could improve upon.

We start with the outputs of the bi-LSTM in the previous step, $\overrightarrow{\mathbf{h}^p}, \overrightarrow{\mathbf{h}^q}, \overleftarrow{\mathbf{h}^p}, \overleftarrow{\mathbf{h}^q}$. The first two come from the forward pass and the last two come from the backward pass: they are as long as each question or passage and their second dimension is h, the state size. The idea is then to compute a similarity matrix between each word in the question and the paragraph.

$$\overrightarrow{R}_{ij} = \overrightarrow{\mathbf{h}_i^p} \cdot \overrightarrow{\mathbf{h}_j^q} \tag{4}$$

$$\overleftarrow{R}_{ij} = \overleftarrow{\mathbf{h}_i^p} \cdot \overleftarrow{\mathbf{h}_j^q} \tag{5}$$

Once we have these two matrices we can perform the same matching strategies described in [6]:

1. Full matching. For each word in the passage we compute two scalars, m_{i1} and m_{i2}, that compare the word to the last and first words in the question.
2. Maxpooling matching. For each word in the passage we compute two more scalars, m_{i3} and m_{i4}, that compare that word with the word in the question that matches the most.
3. Meanpooling matching. For each word in the passage we compute two last scalars, m_{i5} and m_{i6}, that compute the average similarity between that word and those in the question.

We end up having 6 numbers for each word in the passage, that get aggregated into a tensor that gets fed to the next layer.

Layer 4: Aggregation. The aggregation layer runs a final bi-LSTM on the previous tensor to combine all that data, generating forward and backward vectors (each with a state size of $l = 60$).

Layer 5: Prediction. The forward and backward vectors for each word in the passage get concatenated and are fed into a neural network with one hidden layer, of size 50. The activation function is a rectified linear unit (ReLU). The final linear transformation obtains a single number for each word in the passage, that corresponds to a raw probability that gets fed into a softmax. That final transformation does not include a bias, because the softmax function is invariant under constant addition.

4 Experiments

4.1 Model

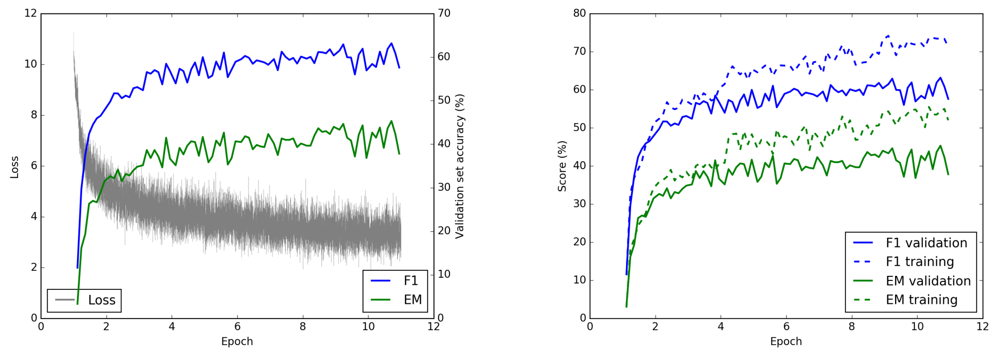
POS Tagging. In an effort to help our model generalize syntactical information about input sequences, we classify each input word based on its part of speech (POS). We provide the POS as a concatenated one-hot vector for each embedding. Python's NLTK provides a quick, pretrained POS tagging system based on a perceptron learning network, which we use to incorporate the additional features in our model. While its performance is somewhat below the state-of-the-art, we expect that the marginal improvement would not be significant enough to warrant the computational expense.

Most SQuAD queries expect noun answers. More intuitively, providing POS augments query answering by using question comprehension to identify and detect the expected POS of the query answer. Essentially, it is equivalent to pre-training part of the context representation layer, allowing the bi-LSTM to use more contextual information about the words it's reading. We implemented POS Tagging and got good results for F1, at very little computational cost and with few additional parameters.

Choice of recurrent cell. While most models use LSTMs (Long Short-Term Memory) as the cell driving the recurrent neural networks, we wanted to explore how well other choices performed. LSTMs are usually chosen because they deal comparatively well with the vanishing gradient problem and have shown good performance on a variety of tasks. However, recently GRUs (Gated Recurrent Units) have been shown to achieve similar performances with fewer parameters. We tried running our model with both cells and achieved a similar final performance (about 61% on the validation set) and learning, so we ended up using simply LSTM cells.

Span generation. An issue with the formalism used above is the fact that the start and end pointers within the passage are clearly not independent of one another—a rigorous, careful Bayesian model would compute the latter conditioned on the former. While, arguably, this partially happens within the aggregation layer, there is still the possibility that the result of the argmax operation will be in very different parts of the passage—if there are two similarly likely answers, for instance. As a way to cope with that, we realized that most answers are short (fewer than 20 tokens) and thus given the most probable start pointer we need only select the most probably end pointer within the next 20.

However, intuitively, an even better approach that's left for future work is to generate the full Bayesian matrix of each pair, with the end pointer conditioned on the start pointer. We then simply take the pair with the greatest combined log probability.



(a) Learning curve throughout the epochs. The F1 score (in blue) tapers off at about 60% and learning slows down significantly after the first epoch, which translates to an asymptote in accuracy.

(b) Plot of the F1 and EM scores for both the training and validation datasets. Note that, despite regularization, overfitting becomes a problem after a few epochs and only gets worse.

Figure 3: Training process for the model that drove the final submission.

4.2 Training

Optimizations. Most questions and passages in the dataset are rather short, with a few outliers. In order to greatly reduce training time, we set a maximum length for questions and passages after looking at the percentiles of each. We chose to cap questions at 25 words and passages at 300. We train our data in batches (usually 48, but ranging from 32–128) with the Adam optimizer.

Learning rate and regularization. Stochastic gradient descent is performed using the Adam optimizer. We found the results to be surprisingly sensitive to learning rate: if the learning rate is too high, the model quickly settles in a bad minimum; however, we found that if it's too small it leads to more pronounced overfitting. Our best result was obtained with $\epsilon = 0.001$. We also used dropout for regularization, with a dropout rate of 0.2. This was applied to each LSTM layer and to the final feedforward neural network for prediction. This attempts to prevent the model from overfitting by randomly dropping units from the network.

Infrastructure. The model was trained on Microsoft Azure GPUs as well as Amazon EC2 (CPU) instances in order to parallelize the training models and perform optimizations faster. Our training times were about 2h/epoch on Amazon CPUs and, after optimizing the GPU instances, about half an hour per epoch for the simpler models. We trained for a maximum of 10 epochs, but often convergence was achieved much faster than that.

4.3 Results

Figure 3 shows an sample training session, from our best model that got submitted to Codalab. We clearly see that most of the learning happens early on in the process and, despite regularization, some overfitting happens in the later stages. Our validation accuracy tapered off at around 60% for many different versions of the model, including different sizes $h = 100, 150, 250$ for the bi-LSTMs and choice of RNN cells.

After submitting to the hidden SQuAD test set in CodaLab, we got an F1 score of 60.8% and an EM score of 49.1%, above the logistic regression model in the original SQuAD paper [10] but well below the current state of the art. For the dev set, we got 60.7% for F1 and 49.2% for EM.

4.4 Analysis

To assess the results of our architecture we tried to visualize the predictions made by the systems, as well as plotting the different learning curves (see Figure 3).

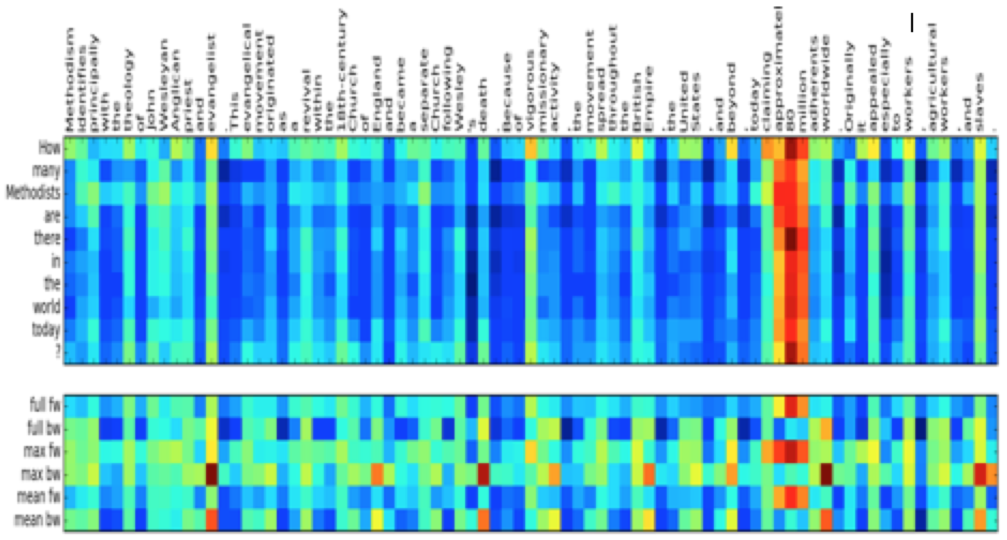


Figure 4: Heatmap of question-answer similarity measures, based on the forward R matrix and associated vectors.

Answer length \ Paragraph length	**0-100**	**101-200**	**200+**
1-3	61% (468)	59% (2393)	55% (353)
4-11	46% (185)	50% (691)	47% (98)
12+	12% (11)	24% (77)	35% (8)

Table 1: F1 score for various combinations of answer length and paragraph length. In parentheses there's the number of datapoints used to estimate each F1 score.

Figure 4 shows the forward similarity matrix from the context matching layer for one of the questions, as well as the aggregated tensor with the different matching strategies at the bottom. We clearly see that, even in that step (before aggregation happens) the model has a good idea of where the answer is going to be. One interesting thing that we noticed is that, for most question-paragraph pairs, *either* the forward or backward similarity matrices were useful, but usually not both at the same time. This indicates that they capture different components, and thus merging their results is useful for the system.

Table 1 shows a matrix that compares our results with different paragraph and answer lengths. It shows that most answers are in fact very short (1–3 tokens) and that the model is able to answer them better. We also see that performance degrades slightly with bigger paragraphs, though more data is required to fully reach that conclusion.

5 Conclusion

We have introduced a model based on multi-perspective matching [6] that incorporates part-of-speech tagging as well as other optimizations to obtain a result of 61% F1 on the hidden SQuAD dataset, thus showing a decent ability to perform Machine Comprehension and Question Answering tasks.

After working with the dataset and exploring different model designs, we have come up with parts where improvements could be made. First, we could include character embeddings and multiple perspectives, as in [6]. However, there are still some weak spots that we hope to be able to improve in the future.

We have considered the usage of neighborhood filtering in the first step (filter matrix) instead of simply looking at each individual word. This is based on the idea that while the answer phrase in the passage often resides near instances of the question words, the answer phrase is rarely every contained within the question. We can instead look at a window of words around to get a better idea of the relevancy.

Another area for improvement is the final prediction layer. By dropping the independence assumption of the start and end pointers, we might be able to compute a matrix of each end position conditioned on each start position, and take the maximum of that. This way, we avoid selecting answers that are too long or that are inconsistent between start and end.

To extend our part of speech tagging addition, another final approach that we can take is to also embed information on the semantic and grammatical dependence of the words, to aid the system in parsing the question and passage. Finally, by creating an ensemble we can slightly boost our performance.

References

[1] Chen, Danqi, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. arXiv preprint arXiv:1606.02858, 2016.

[2] Kenton Lee, Kenton, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. arXiv preprint arXiv:1611.01436, 2016.

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

[4] Shen, Yelong, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. arXiv preprint arXiv:1609.05284, 2016.

[5] Wang, Shuohang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.

[6] Wang, Zhiguo, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective Context Matching for Machine Comprehension. arXiv preprint arXiv:1612.04211, 2016.

[7] Wang, Zhiguo, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching fornatural language sentences. arXiv preprint arXiv:1702.03814, 2017.

[8] Xiong, Caiming, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[9] Yu, Yang, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. arXiv preprint arXiv:1610.09996, 2016.

[10] Rajpurkar, Pranav, Zhang, Jian, Lopyrev, Konstantin, Liang, Percy. Squad: 100,000+ questions for machine comprehension of text. 2013.