# Stanford CS224n Project: Reading Comprehension

**Joris van Mens**
jorisvm@stanford.edu (jorisvm)

**Ilya Kuleshov**
ikul@stanford.edu (ikuleshov)

**Nick Westman**
nwestman@stanford.edu (nwestman)

## Abstract

In this paper we explore Machine Comprehension, a subset of Natural Language Processing, using data from SQuAD. We take an iterative approach to designing our prediction model, starting with a very basic foundation, extending it and replacing components piece by piece while testing for effectiveness. Our model achieves a 62.3 F1 score on the SQuAD test set.

## 1 Introduction

Machine Comprehension (MC), or more specifically Reading Comprehension, is the ability to read a passage and answer questions about it. It requires a model to both understand the language in question and the passage in order to make a prediction about the answer. Since the release of the Stanford Question and Answer Dataset (SQuAD), teams across the world have been able to make significant progress towards effective MC models (Rajpurkar et al. 2016). In this paper, we explore some of these models and build out our own model.

Given a two week time frame and minimal starter code, our goals were as follows:

1. Build a functional end-to-end Machine Comprehension Model (done)
2. Attempt to implement several state-of-the-art models (done)
3. Exceed the given baseline model (done)
4. Learn how to build ML systems (done)

### 1.1 Task

The task is to identify the answer to a question given a context sentence. SQuAD is a dataset of about 100,000 context / question / answer triplets. The set is split 80/10/10 into train, validation, and test sets. Of these, the team which released SQuAD refused to publicly release the test set as a reserve for evaluation. A baseline logistic regression model achieves a F1 score of roughly 0.51 1, whereas human inference achieves an F1 score of roughly 0.90.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{1}$$

## 2 Background

Virtually all MC models start by encoding input words as word-level embeddings, and some also use character-level embeddings. All use recurrent neural networks to contextualize the inputs, then use some sort of attention mechanism to further encode the representation between question and

context. Finally, models will use a dense layer, another recurrent neural network or more complex systems to generate an answer prediction.

### 2.1 Related Work

There are a range of papers addressing SQuAD. The first model we explore was "Dynamic Coattention Networks For Question Answering" (Xiong, Zhong, and R. Socher 2016). This model proposes two novel techniques to respectively encode attention information and dynamically decode the knowledge representation of the inputs. It defines a co-attention matrix, which uses a cross-product contextualized inputs and a BiLSTM to act as attention matrix. The model then uses a "dynamic decoder" which is a recurrent neural network which takes up to four guesses at an answer, terminating early if the answer converges early.

The model described in "BiDirectional Attention Flow for Machine Comprehension" (BiDAF) takes a similar approach, but instead uses a BiLSTM to decode the output. Additionally, BiDAF adds extra information about the input by using a character-level CNN to help with outside of vocabulary (OOV) errors. Later in this document, we will describe the iterative approach we took to construct a somewhat similar (albeit simpler) model.

We also drew inspiration from RaSoR (Lee et al. 2016). RaSoR has a different neural attention mechanism, which aligns question words to passage words. From that, the authors of the paper create a set of candidate "spans" which are prospective answers identified by the knowledge representation of the start and end words concatenated. These spans are then scored with a dense layer, and the span with the highest score is the answer candidate.

## 3 Approach

We explored several of the state-of-the-art models and iteratively tried approaches from several models in our search. Rather than just implementing one of the models in the paper, we attempted (and sometimes failed!) at implementing several different models and components, using the learnings from each to come up with a final model loosely inspired by BiDAF (Seo et al. 2016).

Although we were able to successfully implement and apply most of the major components described in BiDAF and "Co-attention Networks" (Xiong, Zhong, and R. Socher 2016) models, such as Bi-LSTM encoder/decoder, Char-CNN and Co-attention mixer, we could not produce a working implementation of the Highway-Maxout Network. The first naive approach to HMN implementation resulted in an inefficient code that would take weeks to train. Our second, optimized solution, was promising but failed to achieve high F1 scores. We believe the issues were proper variable initialization, and did not have the time to debug the issue.

Additionally, we attempted to implement RaSoR, but became stuck trying to efficiently generate spans, which is $O(n^2)$, where n is the maximum size of a context paragraph. We started with a naive implementation to generate all spans, which was impossible due to RAM limitations. At that point, we became stuck trying to find a more efficient implementation, but could not find one which was effective.

### 3.1 Word embeddings

For the embedding layer, we used a combination of pretrained GLoVe word vectors and trained character-level embedding vectors generated using a convolutional neural net described in BiDAF (Seo et al. 2016). For the word embeddings, we use the provided 300-dimensional GLoVE vectors trained on 6 billion tokens. The vocabulary size was about 115,000 words.

For the most part, we kept our pretrained GLoVE vectors as constants in our system, which sped up training time significantly. Towards the end of the project, we revisited training special tokens, such as ¡UNK¿ (Unknown Word), so the model could develop its own logic to handle them. Given more time, we would have continued to actively look into training these special tokens to increase model accuracy.

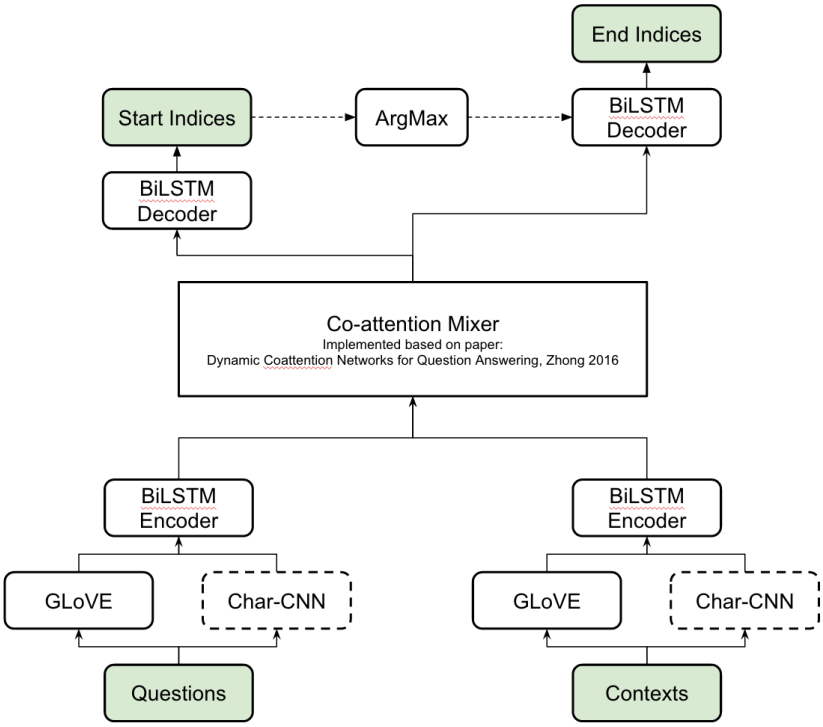


Figure 1: Final model architecture

### 3.2 Character Convolutional Neural Network

In addition to the numerical token ids, we also included a character representation for each word in the dataset, filtering out non-ASCII characters.

#### 3.2.1 Character embedding matrix

Character embedding matrix has dimensions of *(char-vocab-size, char-emb-dim)*, where *char-vocab-size* spans the range of the character numeric codes (250 by default), *char-emb-dim* is the embedding dimension of each character (8 by default). Character level embeddings were initialized using Xavier initializer

#### 3.2.2 Character convolution layer

As per the BiDAF paper, we applied 1-d convolution to the character embedding matrix with kernels (filters) having dimensions of *(1, kernel-length, char-emb-dim, filters)*, where *filters* is the number of convolution filters (100 by default), and *kernel-length* represented the length of the 1-d convolution kernel, or *filter height* in terminology of the BiDAF paper (5 by default).

We then apply a max pooling operation (Kim 2014) in order to take the maximum value over words as the feature corresponding to every filter.

Finally, we concatenated the output of the Char-CNN layer with GLoVe vectors corresponding to each word. This allowed the model to deal with Unknown tokens intelligently, which was especially useful as we were constrained by using the fixed trimmed vocabulary during test evaluation. Given more time, we wanted to look at different ways to combine the embeddings, such as a sum or linear combination, like ”Fine-Grained Gating” does(Yang et al. 2016).

### 3.3 BiLSTM Encoder

To encode contextual information about thRNN (ith LSTM cells) to all the words in each the context and question. This gave each embedding contextual information of the words around it moving forward through the system.

### 3.4 Co-Attention Mixer

For the Co-Attention Mixer, we implemented the exact approach described in (Xiong, Zhong, and R. Socher 2016), creating an attention (similarity) matrix between context and question, and subsequently applying a BiLSTM.

### 3.5 Single-Iteration BiLSTM Decoder

To get to our start token prediction, we take the Co-Attention Matrix (all hidden states of the Mixer) as input to a BiLSTM, which we have produce hidden states of length 1. We use these hidden states as pseudo-logits for the position of the answer start.

For the end token prediction, we use the same approach. In addition, we take an ArgMax of the start vector, turn it into a one hot vector, and concatenate it with the input (the Co-Attention Matrix U):

$$StartInput = U$$
$$EndInput = [e_{argmax(StartIndices)}; U]$$

### 3.6 Loss

The outputs from the Decoder gave scores of both the start and end indices. Given these scores, we used standard softmax cross entropy loss to train our model. This gave us an estimation of how close we were to the ground truth.

## 4 Experiments

Our main experiments came from the iterative model improvements we went through. At each step, we captured F1 scores to see if the additions and alterations were improving the model (note not all F1 scores are perfectly comparable, see footnote).

### 4.1 Model iterations

Table 1: Malformed input data filtering

| Version | Architecture description | F1 (val.) [1] |
|---|---|---|
| V1 | GLoVe → BiLSTM → coattention matrix (final hidden state) → FFNN | 25.5 |
| V2 | GLoVe → BiLSTM → coattention matrix → vector (LogReg) | 49.8 |
| V3 | GLoVe → BiLSTM → coattention matrix → FFNN | 51.4 |
| V4 | GLoVe → BiLSTM → coattention matrix → FFNN (MaxOut) | 53.4 |
| V5 | GLoVe → BiLSTM → coattention matrix → BiLSTM | 62.0 |
| V6 | GLoVe → BiLSTM → coat. mat. → BiLSTM w/ startvec input | 65.3 |
| V7 | GLoVe + CNN → BiLSTM → coat. mat. → BiLSTM w/ startvec input | 66.9 |
| | *Misc. improvements: trainable Special Tokens, L2 Loss, Dropout* | – |

[1] Although not all validation F1 values are directly comparable due to intermediate bug fixes, minor improvements and hyperparameter optimizations all happening in parallel with model architecture improvements, they are still indicative of relative model performance.

### 4.2 Training

Most of the training was done on Google Borg Cluster, using a single TensorFlow worker job per model. Each job was allocated with 8 CPUs (equiv. to Opteron 6 Core 2.6), 2 GPUs (Tesla k80), 10 GB RAM. Each model was trained for 8-24 hrs.

### 4.3 Input data analysis

We show some brief insights from our analysis of the training and validation input datasets.

#### 4.3.1 Answer and context size

We find the question length distribution is most dense around length 5-15, while for contexts this is around lengths 50-200, per figure 4.3.1.


Figure 2: Input data size analysis

#### 4.3.2 Filtering invalid samples

We check and filter for malformed input data. Per table 2, the main type of bad input data we find is data where the question end index is smaller than the question start index, which we filter out.

Table 2: Malformed input data filtering

| Malformed data type | Filtered occurrences |
|---|---|
| Questions with end index $>$ start index | 58 |
| Contexts of length $\leq 2$ | 0 |
| Answers with end index $>$ context size | 0 |

### 4.4 Hyperparameter selection

To find optimal hyperparameter values we experimented with different hyperparameters (mostly by orders of 10) to see their effect on validation accuracy. Where not feasible (due to training time constraints), we studied literature to set sensible values.

**GLoVe dimensions: 300.** We started with the default of 100, and briefly used 50 to have models train faster. Eventually, we chose 300 given the original GLoVe paper (Pennington, R. Socher, and Manning 2014) demonstrates its superior performance.

**Context size: 600.** our input analysis shows context lengths fall primarily within the 50-200 word range and can be trimmed above 600 without significant reduction in explanatory power.

**Hidden layer sizes: 100.** We tested with 20, 100 and 200 sizes. While 20 performed significantly worse than 200 (F1 of 7.34 vs. 10.67 after 1 epoch), we saw no significant difference in explanatory power for 100 and 200.

**Activation Function: ReLU** We picked ReLU (Rectified Linear Unit) in our dense layers because of it is computationally cheap and provided the needed functionality to get the our system working well. While we did see positive results by using a Maxout activation function for a single-layer Feed-Forward Neural Network, we did not have enough time to configure a 'V5' decoder using a Maxout activation.

**Optimizer: Adam.** We briefly used basic SGD, but wanted an optimizer that adapts its learning rate dynamically. Adam (as well as Adagrad) was a solution that seemed to fit the purpose.

**Learning rate: 0.001.** We tested a 0.01 learning rate, but found a learning rate of 0.001 learned faster (F1 of 23.43 vs. 12.76 after one epoch), and let us converge consistently within about 10 epochs.

**Epochs: 10.** We find that the validation set tends to be converged around epoch 10 (see figure 4.4 for an example run), while longer training has a slight tendency of overfitting.

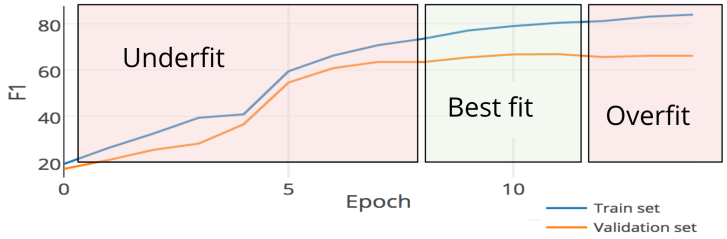


Figure 3: Optimal validation set accuracy around epoch 10

### 4.4.1 Initialization

To initialize almost all of our parameters, we used the Xavier initializer. The exception to that was hidden state of the context paragraph encoding BiLSTM, which used the final states from the question as initial states.

### 4.4.2 Regularization

While we regularly overfitted smaller datasets (100 examples) to build confidence of our model's ability to converge, we explored two methods of regularization to prevent overfitting.

**Dropout probability: 0.5.** We used vertical dropout in our RNN's, and dropout in the hidden layer(s) of our FFNN. While we also tested with lower numbers (0, 0.15, 0.3), (Baldi and Sadowski 2013) shows a dropout probability of 0.5 achieves the maximum regularization effect (albeit with some performance hit). The dropout step is necessary when evaluating any models containing a maxout activation function (Goodfellow et al. 2013).

**L2 regularization constant: 0.0001.** For L2 regularization, we tested with 0, 0.00001, 0.0001 ad 0.001 and found we achieved the best explanatory power at 0.0001, while still prohibiting the model from too much overfitting.

### 4.4.3 Output analysis

Looking at F1 scores does not provide nearly enough satisfaction compared to seeing the actual human readable output of the model. We were surprised how well the model was able to understand context and direct its attention to the relevant parts of the sentence. Below is a sample Q&A session based on the excerpt from the article taken from the CNN website.

According to CNN's ongoing whip count, 19 House Republicans have said they will vote against the bill, while seven more have indicated they are likely to oppose it. GOP leaders can afford to lose 21 members to pass the bill without any Democratic support. A senior House leadership aide said they are confident they will get the votes, and that House Speaker Paul Ryan is 'full involved.' 'This is an all-hands-on deck situation. This is a big bill. It's a big promise Republicans made to the American people,' the aide said.

1. **Question:** How many House Republicans have said they will vote against the bill?
   Answer: 19
   *Correct!*
2. **Question:** How many members gop leaders can afford to lose ?
   Answer: 21
   *Correct!*
3. **Question:** What GOP leaders can afford to lose and still pass the bill without any Democratic support?
   Answer: 21
   *Almost correct. Should be "21 members", which were not recognized as a single entity.*
4. **Question:** Who will vote against the bill?
   Answer: House Republicans
   *Correct!*
5. **Question:** Who will oppose the bill?
   Answer: CNN 's ongoing whip count, 19 House Republicans
   *Different question formulation, less precision.*
6. **Question:** How many House Republicans are likely to oppose the bill?
   Answer: seven
   *Correct!*
7. **Question:** Which company was doing the whip count?
   Answer: CNN
   *Correct!*
8. **Question:** Which company was doing the count?
   Answer: CNN's ongoing whip count , 19 House Republicans
   *Different question formulation, wider answer span.*
9. **Question:** What GOP leaders can afford to lose?
   Answer: 21 members to pass the bill without any Democratic support
   *Correct!*
10. **Question:** Who said that House Speaker Paul Ryan is 'full involved.'?
    Answer: A senior House leadership aide
    *Correct!*
11. **Question:** Who said that Paul Ryan is 'full involved.'?
    Answer: House Speaker
    *Wrong.*
12. **Question:** What did the aide say?
    Answer: they are confident they will get the votes
    *One of the two possible correct answers.*

Overall, the model produces good results as long as the question is stated in terms of the original text. As soon as the question omits certain information (e.g. "Paul Ryan" instead of "House Speaker Paul Ryan", uses synonims ("oppose" instead of "vote against") or adds extra information, precision gets lower.

Apart from the model and implementation deficiencies, this behaviour can be explained by a relatively small training dataset. Not only more question/context pairs need to be added to the dataset, but every answer should be accompanied by multiple variations of the same question.

Nevertheless, even considering question variations, most of the answers seem relevant or at least referencing the correct part of the sentence.

### 4.5 Group Contributions

Each group member did his best effort towards helping the group. In the end, we are happy with the results and contributions of all three group members. :)

### 4.6 Conclusion

Our iterative model improvement approach brought us good results while helping us develop a deep understanding of the workings of the model (things we did not achieve by attempting to implement carbon copies of existing papers). Given the continuous, steady improvements we were able to make with this approach, we are convinced that future improvements such as smart input data processing (cleanup), extending the vocabulary (beyond train + validation set words), and a multi-step decoder approach would bring us well into the 70+ F1 score range.

## References


Baldi, Pierre and Peter J Sadowski (2013). "Understanding dropout". In: pp. 2814–2822.

Goodfellow, I. J. et al. (2013). "Maxout Networks". In: *ArXiv e-prints*. arXiv: 1302.4389 [stat.ML].

Kim, Yoon (2014). "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882. URL: http://arxiv.org/abs/1408.5882.

Lee, K. et al. (2016). "Learning Recurrent Span Representations for Extractive Question Answering". In: *ArXiv e-prints*. arXiv: 1611.01436 [cs.CL].

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global Vectors for Word Representation." In: 14, pp. 1532–1543.

Rajpurkar, P. et al. (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *ArXiv e-prints*. arXiv: 1606.05250 [cs.CL].

Seo, M. et al. (2016). "Bidirectional Attention Flow for Machine Comprehension". In: *ArXiv e-prints*. arXiv: 1611.01603 [cs.CL].

Xiong, C., V. Zhong, and R. Socher (2016). "Dynamic Coattention Networks For Question Answering". In: *ArXiv e-prints*. arXiv: 1611.01604 [cs.CL].

Yang, Z. et al. (2016). "Words or Characters? Fine-grained Gating for Reading Comprehension". In: *ArXiv e-prints*. arXiv: 1611.01724 [cs.CL].