Reading Comprehension with SQuAD

Archa Jain
Stanford University
archa@stanford.edu

Abstract

The SQuAD dataset provides a reading comprehension task, with (question, answer, context) pairs. This paper attempts to develop a model to predict the answer for a given question, context pair. In a model inspired by "Multi-Perspective Context Matching for Machine Comprehension" by Wang et al, this paper achieves 0.47 F1 score on a validation set.

1 Introduction

Machine Reading Comprehension is an effort to teach computers how to "understand" a passage, and then to be able to answer questions from it. It is a complex NLP problem with a lot of ongoing research. There are a variety of real world use-cases for this, ranging from assistive educational technology to dynamic information retrieval.

For this work, I am looking at the subsection of Machine Comprehension that, given a context paragraph and a question, attempts to generate the answer to the question from the context. For example, given the context "*In some cases and in some places the edicts were strictly enforced : some Christians resisted and were imprisoned or martyred . Others complied . Some local communities were not only pre-dominantly Christian , but powerful and influential ; and some provincial authorities were lenient , notably the Caesar in Gaul , Constantius Chlorus , the father of Constantine I. Diocletian 's successor Galerius maintained anti-Christian policy until his deathbed revocation in 311 , when he asked Christians to pray for him . " This meant an ofcial recognition of their importance in the religious world of the Roman empire , although one of the tetrarchs , Maximinus Daia , still oppressed Christians in his part of the empire up to 313* ." and question "*What were some provincial governors in enforcement of the Roman edicts ?*", it identifies the answer "*lenient*"

The above example, and the training data for this effort was drawn from the Stanford Question Answering Dataset (SQuAD)[1], a reading comprehension dataset consisting of 100,000+ questions posed by crowdworkers on a set of Wikipedia articles, where the answer to each question is a segment of text from the corresponding reading passage. The dataset provides triples of question, answer and context for training and validation, and model performance is evaluated against a hidden test set by submission to the SQuAD system.

There's a variety of popular Machine Comprehension datasets currently being worked on, with similar patterns of data – like MCTest[2], which contains crowdsourced children's simple narratives, Babl[3], which contains synthetically generated simple narratives, WikiReading[4] that targets slot-filling of Wikipedia infoboxes from Wikipedia pages etc. There are a wide variety of techniques being developed to work on this problem. I discuss some of these in the next section.

2 Related Work

Broadly speaking, a large section of Machine Comprehension techniques utilize Attention/Memory, which helps the model develop an understanding of which parts of one piece of text are relevant to another piece of text. A common pattern that I found in works for this problem was:

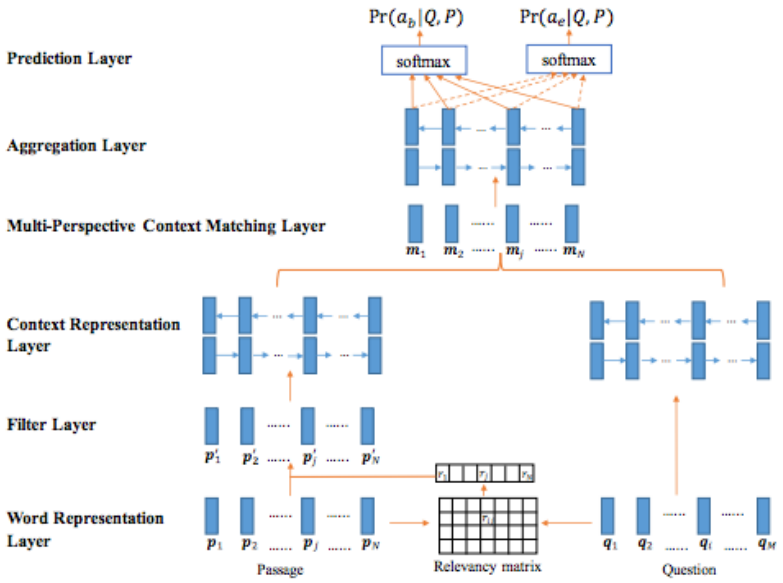
- Encode both the question and the answer
- Combine both encodings using attention/similarity mechanisms.
- Combine the results of the previous step
- Produce a classification over the combined representation.

The finer points of models within each step differ quite a bit. For the encoding step, a bidirectional RNN was a popular choice, with some variation in the cell used within the RNN. LSTM's are a popular choice as the cell, eg in "Multi-Perspective Context Matching for Machine Comprehension" by Wang et al., in "DYNAMIC COATTENTION NETWORKS FOR QUESTION ANSWERING" by Xiang et al. On the other hand, Chen et al. propose using a GRU in "A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task"[5], citing that it sped up their implementation as compared to an LSTM, without significant losses in performance.

The attention/similarity mechanism has a wide range of variations proposed in different works, with this step being the bulk of new innovation. Wang et al. use a Match-LSTM and Answer Pointer[6]. For the encoder attention mechanism, they use attention mechanism to get a weighted representation at each position of the context using a representation of the question. This representation is then used as an attention vector in an LSTM to generate the encoding. "Dynamic Coattention Networks" [7] uses a slightly different approach, utilizing an affinity matrix for each pair of question/answer words to compute an attention vector for each word in the Question and in the Answer.

In a broad sense, the kind of attention mechanism used by these models and where it is applied (in the encoder/decoder/both) determines the characteristic behavior of each of these models.

Figure 1: MPCM Model Architecture



In this work, I explore the performance of an architecture derived from "Multi-Perspective Context Matching for Machine Comprehension" by Wang et al[8]. Their model captures the similarity between the two texts by combining a variety of different similarity functions computed between them. Figure 1 represents the model architecture presented by them, where the "Multi-perspective context matching layer" calculates the output for each of these similarity metrics between the question and answer encoded representation output by the "Context Representation Layer".

The three similarity metrics proposed by the paper are "full-matching", "meanpool-matching" and "maxpool-matching", the details of which and the associated equations can be found in the referenced paper. I utilize this proposed matching layer in my model.

3 Problem Formulation

I formulated the problem as such: given a question Q with words $q_1, ..., q_n$, and a context C with words $c_1, ..., c_m$, where each word is represented as its corresponding word embedding, we want to predict the answer A. A is represented using its start and end span, which is the indices of the start and end word in the context (inclusive): $A = (a_s, a_e)$.

Note that for ease of implementation, I chose to represent the answer as the start and end span, rather than simply a probability distribution over the context, where $P(c_i)$ would be the probability that the ith word is part of the answer. This would be a very valid alternate approach to explore with this architecture in the future.

4 Approach

4.1 Baseline

For a baseline implementation, I implemented the following architecture:

- Use a bidirectional LSTM to generate a question encoding
- Use a different bidirectional LSTM to generate a context encoding, using the question states from the first step to generate an attention vector each time
- Use the concatenated final state output of the second LSTM, input it into a single Feed forward layer
- Run a softmax over the feedforward output to produce a probability distribution over the context words, each for the start and end word.

While this was a good model for proof of concept, it underfitted to the training data, producing a Dev F1 score of 0.17.

4.2 Advanced

On a high level, I implemented a modified version of the architecture proposed in "Multi-Perspective Context Matching for Machine Comprehension". I used the proposed the match layer to generate modified versions of the context word represenatations. The major modifications from the original model were:

- RNN Cell: The paper proposes using an LSTM for the RNN Cell, however as proposed by Chen et al., I replaced it with a GRU, which significantly reduced the training time of the model (see Table X), without changing the F1 score trends significantly. GRUs are further explained in section 4.3
- I removed the filter layer, both for speed-up of training time, and because the original paper's ablation analysis did not suggest it was a crucial component.

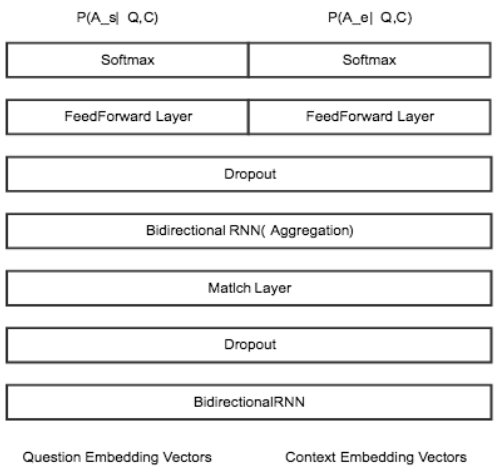
4.3 Implementation

In this section, I discuss the implementation details for the model, and the various optimizations I made to reduce the run times. Before some of the optimizations listed below, a single epoch ran in about 1.5 hours, and after them it sped up to about 30 minutes per epoch. The speedup allowed me to experiment more with the model.

4.3.1 GRU

As I mentioned above, I use GRUs in stead of LSTM's in all three instances of RNNs in the above model. GRUs or Gated Recurrent Units are a form of recurrent cell, that are capable of holding context over a longer range of timesteps than vanilla RNN cells. They have been found to be similarly performant as LSTM's, but require significantly less computation due to a simpler mathematical formulation. As a result of this, switching my model implementation over to GRUs caused roughly a 10 minute speedup per epoch.

Figure 2: Implemented Architecture



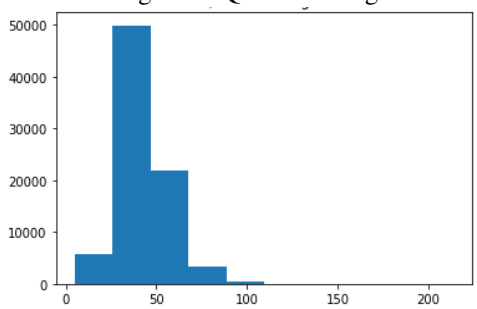
4.3.2 Cosine Similarity

In the paper, the cosine similarity is measure on a per-vector level for each word in context. Setting up the calculation as a batch multiplication significantly sped up the model and cleaned up the implementation.

4.3.3 Masking

Since the questions and context paragraphs are of variable sizes, it was important to pick the correct lengths for clipping each data point. Looking at the distribution of both question and context lengths (Figure 3,4), I picked max lengths that were past the median length, and so included most of the data. The model was trained with a question length of 60 and context length of 500.

Figure 3: Question Lengths

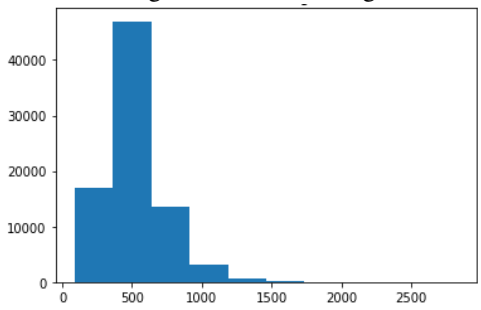


To account for the max lengths, in the data preprocessing, I padded answers that were shorter than the max length, and clipped the longer ones. I used the corresponding actual sequence length to optimize the RNNs. Additionally, before reporting the final prediction logits, I applied the individual masks to set the out of bounds predictions to a large negative number, so it is not factored in the loss calculation.

4.3.4 Gradient Clipping

I implemented gradient clipping to keep the model from experiencing exploding gradients. It is interesting to note, the model is especially susceptible to exploding gradients after adding the aggregation layer.

Figure 4: Context Lengths



4.3.5 Regularization

The model, when implemented as is was extremely prone to overfitting. I implemented two layers of dropout, before the matching and Aggregation layers, with a dropout probability of 0.5. In addition, I implemented L2 Regularization on all the trainable variables I initialized, with a coefficient of 0.3. (See Section for 5) for experimental data supporting these hyperparameter choices.

4.3.6 Batch Size

I trained the model with a batch size of 30, which optimized the memory usage on the VM, and sped up the overall training process. A larger batch size saves on processing overhead, though may introduce noise in the training process, and has a larger memory footprint.

5 Experiments/Results

5.1 Overall Results

The final model achieves an F1 score of 0.47 on the validation set, and 0.71 on the training set, after training for 10 epochs.

5.2 Answer Analysis

Consider the following answers:

Actual Answer: God 's House Tower
Predicted Answer: House Tower

Actual Answer: Windows 8 , Windows Phone 8 , and the company 's suite of online services
Predicted Answer: Windows 8

Actual Answer: The Dutch Republic , long-time British ally , kept its neutrality intact
Predicted Answer: long-time British ally

Actual Answer: the degree of Master Mason
Predicted Answer: Master Mason

As we can see, the model is able to pick up on both backward an forward context, owing to the matching layer, which captures the relevance of each context word, and identify the correct part of the context, however, it does not do a good job of predicting the actual "boundary" of the answer, and often exceeds or underpredicts the answer length. Theoretically, I think adding the filter layer proposed in the original model might help in this case, since that will cause the model to focus, as a first pass, on the relevant area of the context, and force it to learn more fine tuned relationships in that part.

Table 1: Overfitting and Dropout (at Epoch 4)

Num Layers	Keep Prob	Train F1	Val F1
0	-	0.59	0.3
1	0.5	0.71	0.38
2	0.5	0.58	0.31

Table 2: Overfitting and Regularization Coefficient (at Epoch 4)

α	Train F1	Val F1
0.05	0.59	0.38
0.1	0.53	0.38
0.2	0.47	0.41
0.5	0.06	0.07

However, the problem is not limited to this issues, as we can see in the following examples, the model does underfit in general, predicting answers that are very different from the actual answer.

Actual Answer: near St. Sophias Cathedral Predicted Answer: liturgy

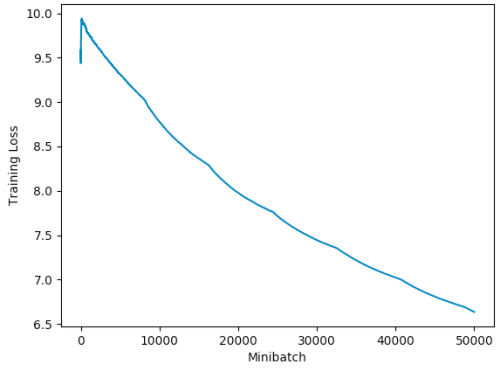
Actual Answer: Such pins as these are rarely seen by the referee Predicted Answer: heels

Actual Answer: wingless Predicted Answer: Apterygota

5.3 Hyperparameter Tuning/Analysis

In this section, I'll present some experimental results that informed architecture and hyperparameter choices.

Figure 5: Training Loss VS Minibatch



Tables 1-5 provide experimental results from the tests I conducted. As we can see,

6 Codalab Submission

As of last night (3/20) I could not get codalab submission to work satisfactorily for my model – due to what I imagine is a bug in my reporting code, the validation F1 score on the leaderboard is much lower than what I am seeing on my VM. I am including the log file from my latest run with the submission. However, due to time constraints posed by other classes I dont anticipate being able to debug the codalab report code.

Table 3: Number of Perspectives (at Epoch 4)

Num Perspectives	Val F1
1	0.38
2	0.41
4	0.44
8	0.47

Table 4: Ablation Analysis (at Epoch 4)

Architecture	Training F1
w/o aggregation layer	0.35
w/o matching layer	0.10
w/o full matching	0.38
w/o maxpool matching	0.41
w/o meanpool matching	0.41
full model	0.47

7 Conclusion

In general, this interpretation of Multi-Perspective Matching is a quick early model to tackle this problem with. While it does not get a very high F1 score, it is quick enough that it enables experimentation to understand the domain better. The model is also extremely prone to overfitting, with regularization only helping slightly.

8 Sharing Disclaimer

Please do not post this report on the course site for future iterations of the course.

Acknowledgments

I would like to thank Professors Manning and Socher, along with the course staff for their guidance and support throughout this quarter.

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] Matthew Richardson. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. Proceedings of the 2013 Conference on Emprical Methods in Natural Language Processing (EMNLP 2013)

[3] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrinboer, Armand Joulin and Tomas Mikolov. Towards AI Complete Question Answering: A Set of Prerequisite Toy Tasks, arXiv:1502.05698.

[4] Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, David Berthelot. WikiReading: A Novel Large-scale Language Understanding Task over Wikipedia. arXiv:1608.03542.

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

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

Table 5: GloVe Embedding Size (Without regularization)

Size	**Training F1**
100	0.58
300	0.71

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

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