The Quest for High-Performance Question Answering Neural Net Models

Lauren Blake
Stanford University
lblake@stanford.edu

Abstract

Question answering is a challenging NLP task with wide-ranging applications. This paper analyzes what deep learning model architectures and hyperparameters are effective for this task based on model performance on the Stanford Question Answering Dataset (SQuAD). The results highlight the importance of careful hyperparameter tuning. The best F1 and exact match scores achieved were 51 and 41 respectively, however there is likely room for performance improvement with adding new input features, incorporating iterative reasoning, creating an ensemble model, and fixing my coattention implementation.

1 Introduction

Question answering is a difficult NLP task that tests to what extent machines can learn to understand language. In question answering, models are provided with two inputs: a question and a context paragraph that contains the question's answer. The models must return the answer through selecting the span of text from the context paragraph that corresponds to the answer. This is challenging because there is no clear mapping from the question to the answer. Instead, the model must pick up on "clues" for where the answer is in the context paragraph, which requires recognizes their underlying meaning.

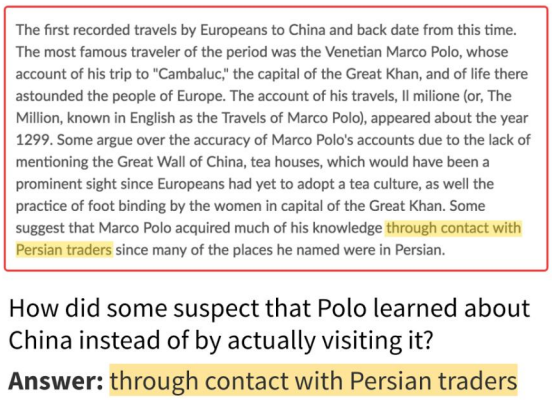

Figure 1: Example context paragraph, question, and answer from the SQuAD dataset. Source: Stanford NLP Blog (https://nlp.stanford.edu/blog/cs224n-competition-on-the-stanford-question-answering-dataset-with-codalab/)

Using techniques from the most successful question answering models, this paper experiments with various neural network architectures and hyperparameters. Model improvements in two phases are tested. The first phase uses an optimized model architecture and hyperparameters with basic attention and the second phase uses an optimized model with a second attention mechanism, specifically coattention.

2 Background

With the introduction of the Stanford Question Answering Dataset (SQuAD) dataset and online leaderboard in June 2016, there has been significant industry and academic research activity around question answering. This literature review focused on the top SQuAD models.

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Jan 22, 2018	Hybrid AoA Reader (ensemble) *Joint Laboratory of HIT and iFLYTEK Research*	82.482	**89.281**
1 Mar 06, 2018	QANet (ensemble) *Google Brain & CMU*	82.744	89.045
1 Feb 19, 2018	Reinforced Mnemonic Reader + A2D (ensemble model) *Microsoft Research Asia & NUDT*	**82.849**	88.764
2 Feb 02, 2018	Reinforced Mnemonic Reader (ensemble model) *NUDT and Fudan University* https://arxiv.org/abs/1705.02798	82.283	88.533
2 Jan 03, 2018	r-net+ (ensemble) *Microsoft Research Asia*	82.650	88.493
2 Jan 05, 2018	SLQA+ (ensemble) *Alibaba iDST NLP*	82.440	88.607

Figure 2: SQuAD Leaderboard on March 18, 2017. Source: SQuAD website (https://rajpurkar.github.io/SQuAD-explorer/)

As on March 18, 2017, the Hybrid AoA Reader Ensemble model has the highest F1 score at 82.482. This is the only model that has exceeded human performance. The Hybrid AoA Reader Ensemble model has an attention-over-attention neural net architecture as described in Cui et al 2017.

The attention-over-attention portion of the architecture includes a second attention mechanism to weight the importance of the attentions from the initial basic attention mechanism. Coattention from Xiong et al 2016 uses a second attention mechanism that is based on attention-over-attention and accomplishes a similar purpose. Similarly, R-net from Wang et al 2017 also uses a second attention mechanism, self-matching attention.

From the submissions in the SQuAD leaderboard, it is also clear that ensemble models outperform individual models. All of the top 5 models are ensemble models. In particular, the F1 score for the Hybrid AoA Reader Ensemble model was 2 higher than the Hybrid AoA Reader Single model.

Looking beyond the leaderboard, other papers demonstrate other techniques for question answering models. Chen et al 2017 improve performance for a simple model with new input features for the context paragraph. Among the features they considered, exact match, whether a word in the context appears in the question, and aligned question embedding, an attention score that measures similarity between context and question words, provided the largest performance improvements. Xiong et al 2016 use iterative reasoning to make their answer predictions. In iterative reasoning, multiple potential predicted answer spans are considered to avoid choosing a local maxima.

3 Approach

At a high-level, question answering models convert a question and context paragraph input into a predicted span output (i.e. predicted start position and predicted end position) of the context

paragraph that correspond to the answer. This paper's model relies on a neural net architecture, coattention, and other optimizations to identify the prediction span output. Each part of the model is described in detail below.

RNN Encoder Layer. The questions and context paragraph are represented by 300-dimensional GloVe embeddings. Both the question and context embeddings are fed into a 1-layer bidirectional LSTM with shared weights. This produces the context hidden states and question hidden states.

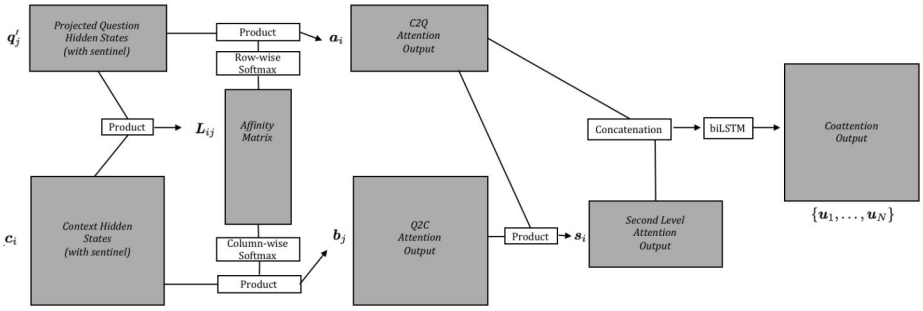

Figure 3: Coattention Layer Diagram

Attention Layer. This paper includes models trained models with either basic attention or coattention. The basic attention layer calculates the attention distribution from the column-wise softmax of the product of the context hidden states. The attention output is the attention distribution weighted by the question hidden states.

The coattention layer starts by calculating an affinity matrix from the context hidden states and projected question hidden states, which are created from question hidden states by applying a fully connected linear layer with a tanh nonlinearity. Separate trainable sentinel vectors are added to both sets of hidden states. The hidden states with sentinels are multiplied together to calculate the affinity matrix. Each entry in the affinity matrix represents the affinity score for a context hidden state, projected question hidden state pair.

The affinity matrix is used to calculate the intermediate and final attention outputs. Context-to-question (C2Q) attention is the row-wise softmax of the affinity matrix weighted by the projected question hidden states. Similarly, the question to context (Q2C) attention is the column-wise softmax of the affinity matrix weighted by the context hidden states. The second-level attention outputs are the Q2C attention outputs weighted by the C2Q attention outputs. The final coattention output are calculated by concatenating the C2Q attention and second-level attention outputs and then feeding them through a bidirectional LSTM.

Output Layer. Separate softmax layers are used to calculate the start position probability distribution and end position probability distribution from the final blended representation. The final blended representation is created by concatenating the context hidden states and coattention output and then feeding it through two fully connected linear layers with ReLu nonlinearities.

Loss. The loss is calculated as the sum of the cross-entropy losses for the gold start and end positions in each training example, averaged across all training examples in a training batch, and then minimized with the Adam optimizer.

Prediction. Based on the approach in Chen et al 2017, the predicted span is chosen to maximize the product of the predicted probability for the start position and the predicted probability for the end position. The relevant algorithm considers all start positions in the context paragraph and then all end positions that are up to 20 positions after a particular start position. This ensures that the predicted end position is *not* before the predicted start position.

4 Experiments

Based on extensive experiments run with the SQuAD dataset, improvements to this paper's model were made in two phases. The first phase transitioned from the baseline model provided to the optimized model with basic attention and the second phase transitioned from the optimized model with basic attention to the optimized model with coattention.

The primary purpose of first phase changes is improving model performance by learning more detailed and relevant hidden features. A secondary consideration was improving training efficiency and reducing the memory required to store parameters.

- Embedding size - The embedding size was increased from 50 to 300 (the largest size available). Larger embeddings allow for more nuanced word information to be provided in the inputs to the model.
- Dropout rate - The dropout rate was increased from 0.15 to 0.5. 0.5 dropout rate is considered best practice and, in general, higher dropout rates help prevent overfitting.
- Hidden size - The hidden size was increased from 200 to 500. Larger hidden sizes can encode more granular hidden features, which often lead to improved model performance.
- Bidirectional LSTM for RNN Encoding - The baseline model used a bidirectional GRU for the RNN encoding. The GRU was replaced with LSTM, which is considered a better default choice and allows for more flexibility.
- More nonlinearity for blended representation output - A second fully connected layer with ReLu nonlinearities was added to the model's output layer. Through introducing additional nonlinearities more complex hidden features can be represented. This complexity may be helpful since the same blended representations are used to calculate both the start and end position probability distributions.
- Maximizing joint start and end predicted probability - Instead of maximizing the probability for the predicted start position and predicted end position separately, the model maximizes the joint predicted probability for both the start and end position across a range of potential start and end positions. (More details are in the approach section.) As mentioned earlier, this eliminated the baseline model's problem that the predicted end position would often occur before the predicted start position.
- Context length and question length - Context length was shortened from 600 to 300 while the question length was shortened from 30 to 20. This reflected the actual lengths in the training data. Over 97% of context paragraphs included less than 300 words. Similarly, over 96% of questions included less than 20 words. Using shorter lengths reduces the number of embeddings required to represent either context paragraphs or questions, allowing for smaller trainable parameters and less computation required for each training iteration.

The second phase substitute coattention for basic attention in the optimized model described above. Due to space limitation, the hidden size had to be reduced to 300. All other changes versus the baseline model were retained.

Results for all three models are listed below. As expected, optimizing the architecture and hyperparameters increased both the F1 and exact match scores. Error analysis (in the appendix) shows that the predicted answers are plausible while still not the best possible or correct answers. This motivated the improvements listed in the conclusion.

However, contrary to my expectations, the second phase did not improve F1 and exact match scores. I believe this is due to a problem with my coattention implementation that I was not able to fix over many iterations of the code. Either the coattention model has a bug or does not fit well with the rest of the model. Based on the large amount of research supporting second attention mechanisms, coattention should have improved model performance if implemented correctly.

Model	Dataset	F1 Score	Exact Match Score
Baseline Model	Test	44.225	34.784
Optimized Model with Basic Attention	Test	50.571	40.921
Optimized Model with Coattention	Dev	28.502	22.460

Table 1: Model Results on SQuAD Data (Codalab issues prevented testing all models on the test data)

5 Conclusion

With its optimized model architecture and hyperparameters, this paper's best model only produces moderately better results than the provided baseline model. This leads me to believe this paper's model would likely benefit from numerous improvements. Based on the literature review, additional input features like exact match or aligned question embeddings could be introduced for the context paragraphs and would help the model learn even more detailed hidden states. Iterative reasoning could improve predicted span selection. And, ensemble models could leverage several separately trained models for a final boost to F1 and exact match scores. Of course, I also would want to iron out my coattention layer implementation and conduct more extensive hyperparameter tuning.

It is a quest to find high-performance question answering neural net models. Due to the day-long training time and GPU resource constraints, a NLP research must travel along a long and unclear path to make a model change and to see its impact on F1 and exact match scores. For me, this emphasizes the importance of building efficient, fast running models. Not only does speed improve prediction runtime, but it also improves the researcher's ability to iterate on the model architecture and hyperparamters and build a fundamentally better model. On my next neural net model quest, I'll weight speed and efficiency more heavily in making model architecture decisions, hoping this may allow for more fine-tuning and ultimately better model performance.

6 References

[1] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Lui, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *arXiv reprint arXiv: 1607.04423*, 2017.

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv reprint arXiv: 1704.00051*, 2017.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv reprint arXiv:1161.01604*, 2016.

[4] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilaterial multi-perspecitve matching for natural language sentences. *arXiv repreint arXiv:1702.03814*, 2017.

7 Appendix: Error Analysis

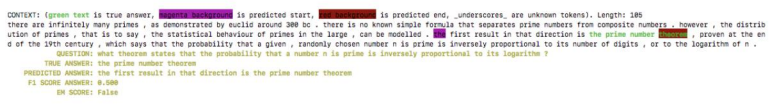

Figure 4: Example 1: Predicted span is too long (i.e. it includes the answer along with other words).

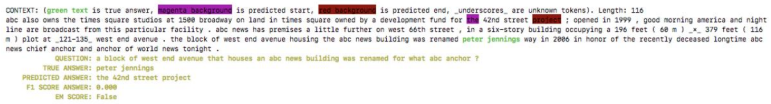

Figure 5: Error Example 2: Predicted span misses that the answer should be a person.

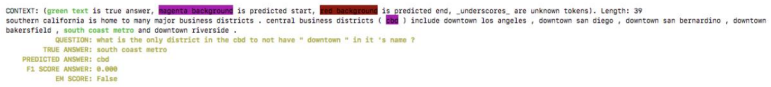

Figure 6: Error Example 3: Predicted span chose the wrong location (although that location technically meets the criteria).

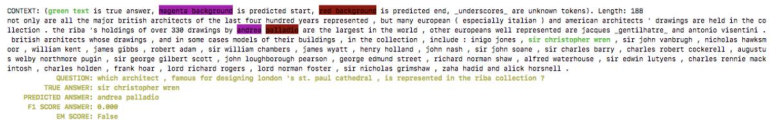

Figure 7: Error Example 4: Predicted span chose the wrong person.

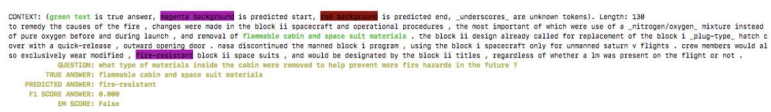

Figure 8: Error Example 5: Predicted span identifies the opposite of the answer.