Question Answering: Examining Top Trends from the SQuAD Task

Edgar G. Velasco
Department of Computer Science
Stanford University
edgarv1@stanford.edu

Anuprit Kale
Department of Computer Science
Stanford University
anuprit@stanford.edu

Abstract

The Stanford Question Answering Dataset (SQuAD)[1] task has created an explosion of progress in extractive machine reading comprehension. We implemented a baseline model and some high performing components such as Bidirectional Attention Flow(BiDAF)[2], Self Attention[6], Character CNN[2], and integrated transfer learning in PyTorch[7]. Combining these approaches we were able to obtain an F1 score of .71 and EM score of .55. In addition, we conducted an ablation study to see which components were most influential to the final model.

1 Introduction

Extractive machine reading comprehension can be very useful in many scenarios such as the self service sector where customers are trying to get answers to questions from a company's knowledge bank. Using SQuAD as our dataset, we start with a simple encoder architecture and successively add components from high performing models to reach our final model. We use a neural network architecture shown in Figure 1 which combines aspects of successful approaches to this task. Our goal was to implement and evaluate these promising components from high performing models such as BiDAF We also experiment with Embeddings from Language Models (ELMo)[4], a new transfer learning method in NLP. Our code can be found at `https://github.com/evg952/coqa_project`

2 Related work

Research groups all over the world have attempted the SQuAD task providing no shortage of techniques to explore. Bidirectional Attention Flow (BiDAF)[2] for Reading Comprehension is one of the most widely known approaches for this task. In addition, we were intrigued by self attention[6] - a technique which seems to have been become popular even outside the machine comprehension task. There seems to be many different approaches, but we chose to focus on the version used in R-Net[6] which is called Self-Matching Attention.

ELMo[4] is a months old transfer learning technique which pre-trains a three layer bi-directional LSTM on the language modeling task. In this way, you can use a massive unlabeled corpus of text to learn good representations of language.

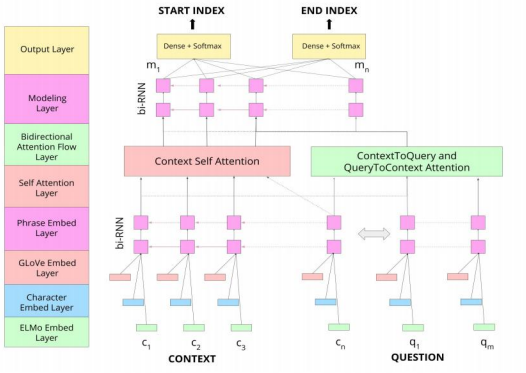
3 Dataset and Features

We used a preprocessed SQuAD v1.1 dataset consisting of about 96,709 total question, answer pairs. The train set was 86,313 examples while the dev set was 10,391. We used part of an existing preprocessing script which tokenized using CoreNLP[9] and lowercased the context and question text. From our analysis we decided to limit the model architecture to a max context length of 300

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

tokens and max question length of 20 tokens as it covered over 99% of the dataset. In Figure 1a. you can see a breakdown of the types of questions found in the dataset that we generated by looking at the first couple starting tokens of the question text. In addition, we performed some preprocessing in order to generate the char-CNN word embeddings(fig. 2b).

(a) SQuAD 1.1 major types of questions



(b) Final Model Architecture

Figure 1

4 Model

The following describes the different components that went into the final model architecture.

1. **GloVe word embeddings**[3] - 100 dimensional GLoVe word embeddings trained on the Wikipedia 14 and Gigaword 5 data.

2. **Char Embedding Layer** fig. 3.b - learns 100 dimensional word embeddings for each word using a convolution neural network operating on character embeddings. The input to the CNN is 8-dim character embeddings. We use 100 filters of size (5,1) operating on neighboring characters in a word, max-pooled over the row resulting in a high dimensional word embedding. We use ReLu activation and a dropout of 0.5.

3. **ELMo** [4] - We use a pre-trained component from the AllenNLP library[11] which provides the embeddings for each token in the sequence of text that is fed in. ELMo provides embeddings by computing a linear combination of all of the internal layers of the bi-LSTM language model at each time step position.

4. **Phrase Embedding Layer** is a 2-layer bidirectional GRU. The three different types of word embeddings for context and question are concatenated and passed as an input to this layer. We add dropout between the 2 layers and on the output hidden states.

5. **Bidirectional Attention Flow Layer** as described in captures the information flow from both context to question and question to context. In Fig. 2, We first calculate the similarity matrix S containing similarity score for each context question pair. To calculate context to question attention we then take rows-wise softmax and then compute weighted sum of question hidden states to get context to question attention.

To calculate question to context attention we take max over similarity each row of matrix S and then take softmax to get attention distribution (U+3B2). We then calculate c' as weighted sum of the distribution (U+3B2) and all context locations to get question to context attention. We finally calculate the output of the bidirectional attention flow as seen in Fig. 2.

6. **Self Attention Layer** as described in [13] attends the context to itself. This helps with co-reference resolution.

7. **Modeling Layer** is a bidirectional 2 layer GRU. The output of BiDAF and self-attention are concatenated and passed as input to this layer. The output of this layer is also concatenated with the BiDAF output and passed to a linear layer.

$$\boldsymbol{S}_{ij} = \boldsymbol{w}_{\text{sim}}^T[\boldsymbol{c}_i; \boldsymbol{q}_j; \boldsymbol{c}_i \circ \boldsymbol{q}_j] \in \mathbb{R}$$

$$\alpha^i = \text{softmax}(\boldsymbol{S}_{i,:}) \in \mathbb{R}^M \quad \forall i \in \{1, \ldots, N\}$$

$$\boldsymbol{a}_i = \sum_{j=1}^{M} \alpha_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h} \quad \forall i \in \{1, \ldots, N\}$$

$$\boldsymbol{m}_i = \max_i \boldsymbol{S}_{ij} \in \mathbb{R} \quad \forall i \in \{1, \ldots, N\}$$

$$\beta = \text{softmax}(\boldsymbol{m}) \in \mathbb{R}^N$$

$$\boldsymbol{c}' = \sum_{i=1}^{N} \beta_i \boldsymbol{c}_i \in \mathbb{R}^{2h}$$

$$\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'] \in \mathbb{R}^{8h} \quad \forall i \in \{1, \ldots, N\}$$

Figure 2: Bidirectional Attention flow equations

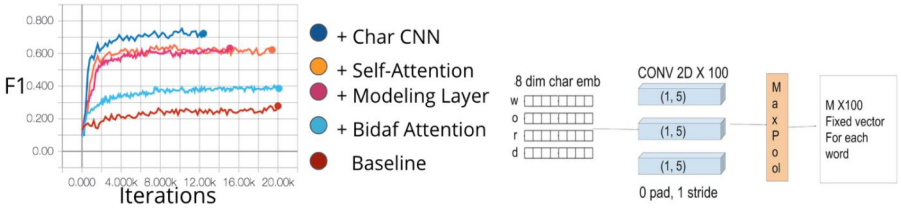
8. **Simple Output Layer** a linear layer followed by a masked softmax. This outputs a vector of probabilities of size max context length for selecting the start and end position as well as the logits before the softmax to be fed to th loss function.

9. **Binary Output Layer** is a sigmoid activation used to predict if the question is answerable or not for the Squad 2.0 task. We swapped the simple output layer with this layer just to get a gauge of the model's language understanding in dealing with adversarial examples in the Squad 2.0 dataset. The model struggled with this task only getting around 60% accuracy.

Loss Function we compute the cross entropy losses for the start and end index using the output layer logits and the 1-hot encoded answer span indices for start and end. We then sum the cross entropy losses for the two positions.

We use an Adam optimizer with a starting learning rate of 0.001. We decayed the learning rate by multiplying by 0.90 every 3 epochs.

To begin, we constructed a baseline architecture that consisted of only components 1,4, a basic additive attention layer that attended the questions hidden states to the context hidden states, and 8. We successively added components to improve the performance until we got to our final architecture. The following plot illustrates our progression throughout the course as we added components and the effect on the dev set F1 score.



(a) Model building progression

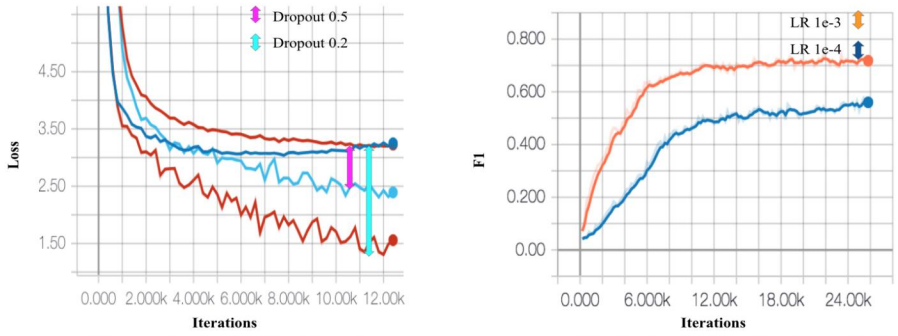
(b) Char CNN Architecture

Figure 3

5 Experiments/Results/Discussion

To evaluate the models we used F1 and Exact Match (EM) as the metrics. Exact Match is simply the proportion of answers which match exactly with at least 1 of the 3 ground truth answers for that example. F1 score, for a single example, computes the average F1 score between the predicted answer span and the 3 potentially correct answers. Then we average over all the examples in the dev set. In the F1 computation the answer and predicted spans are treated as a bag of tokens where the splitting is done on whitespace.

Hyperparameter Tuning. Our best performing model was trained on a batch size of 100 examples, We settled on this after experimenting with batch sizes of 32 and 64. Dropout of 0.5 decreased overfitting but also decreased the F1 and EM scores. We decided to use dropout of 0.2 for better performance on the dev set. The model converged slower with a learning rate of 1e-4 than it did with a learning rate of 1e-3.



(a) Loss vs iterations for different dropouts

F1 vs iterations for different learning rates (dev)

Figure 4: Hyperparameter Tuning

Ablation Study. Although the progression plot hints at the effectiveness of the components we added, we also performed an ablation test of several of the components we added. From our ablation tests we saw that removing the modeling layer had the biggest effect in reducing the F1 score. After this, BiDAF was the most crucial component. Surprisingly, adding ELMo and Self Attention was not as effective as we'd hoped. Perhaps integrating them in some different way would utilize their benefits more. For example, we could have added self attention directly after BiDAF instead of alongside it.[22]

ELMo Experiments We tried a couple ways of incorporating the ELMo embeddings into the model. First we tried concatenating them to the GLoVe embeddings and char-CNN embeddings. This did not provide much benefit as the embeddings became very large and slowed down training. We also tried replacing all the embeddings and the phrase embed layer with the final output of the ELMo layer. Although learning converged much quicker due to the pre-training, it did not reach the same high scores. This method achieved F1 scores between .57 and .63 depending on the size of the pre-trained ELMo layer used.

6 Error Analysis

In 2007 , the kenyan government unveiled vision 2030 , an economic development programme it hopes will put the country in the same league as the asian economic tigers by the year 2030 ...

What did kenya reveil in 2030?

True answer: *vision 2030* Predicted answer: *asian economic tigers*

Figure 5: Model Error

In Figure 5. we see an error produced by our model which exhibits many of the characteristics we saw in other errors. First, due to the lowercasing during pre-processing the text, we lose information that indicates things such as proper nouns. The answer "vision 2030" is a proper noun and is originally capitalized. Finally we see that this is an incorrectly posed question generated by the human labeler since it was revealed (which is spelled incorrectly in the question) in 2007 not 2030. Human error is likely not a large cause of error but it shows that this is not a trivial task.

Attention Visualization We looked into trying to understand what the BiDAF layer was doing on specific examples. In the first part of the BiDAF layer you construct a similarity matrix between the context and question tokens. In Figure 6. you can see a visualization of the similarity matrix output for a specific example. The dark rectangles indicate a higher similarity score between the tokens. On

the left are the question tokens and on the right are the top context words associated with the question token in that row. We also point out on the top several other high similarity activation context words. You can see that the token "when" has a high similarity with the tokens 2009, 2003, 2001, and 2004. This makes sense given that the question type is asking for a date and all the years in the context are being activated. In fact, the correct answer for this example is 2001, which is the token with highest similarity to the token "when". You can also see that the phrases "inducted into" and "hall of fame" are repeated throughout the context and always have similarity with the same words in the question.

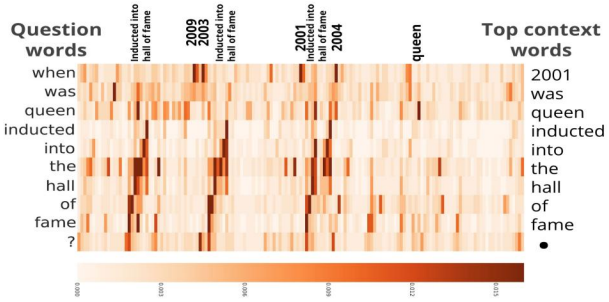
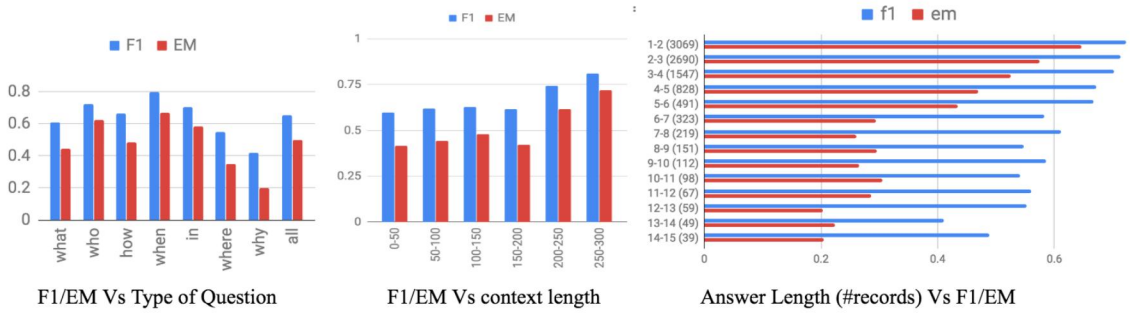


Figure 6: Attention visualization from bidirectional attention flow

Performance Analysis The following bar plots in Figure 7. demonstrate how the model performed on different types of examples. First we can see the score breakdown by the types of questions. Intuitively, these results makes sense. For example, "when" questions tend to be simple to answer as they usually involve finding a specific entity type, a problem which NLP has shown success in. On the other hand, we do poorest on "why" questions as these, presumably, require more complex reasoning and perhaps things such as common sense or real world knowledge. We were surprised to see that our model performed better as the context length increased. This might be misleading as there as fewer data points in the upper intervals of context length. Finally, we plot our performance by answer length and see that we do better on questions that have shorter answers which constitute a majority of the dev set.



F1/EM Vs Type of Question | F1/EM Vs context length | Answer Length (#records) Vs F1/EM

Figure 7: Performance Analysis on dev set

7 Conclusion/Future Work

We learned that the surprisingly simple modeling layer was the most crucial component of the final model. BiDAF provided a more helpful type of attention than self attention which makes intuitive sense since it attends both inputs in both directions. In addition, char-CNN was more helpful than expected, since it helps create meaningful embeddings for out of vocabulary words. The final architecture without the concatenated ELMo embeddings and a large batch size of 100 was the most effective. If we had more time we would explore more how to integrate ELMo more effectively into the model and also explore other transfer learning techniques such as BERT[10]. In addition, it would be cool to submit these model to the leaderboard to see our results on the hidden test set.

8 Acknowledgement

We want to thank Jay Whang for his advice throughout the course.

9 Contributions

All authors contributed equally to the project including architecture review, model development, milestone and poster. Anuprit implemented Bi-directional attention flow and char CNN. Edgar implemented the baseline, self-attention and experimented with different ELMo integrations. And both worked on hyper parameter tuning and ablation study.

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] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603 https://github.com/allenai/bi-att-flow

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word prepresentations. In Proceedings of NAACL, 2018.

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

[6] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. arXiv preprint arXiv:1606.01549, 2016.

[7] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. Automatic differentiation in PyTorch. NIPS-W, 2017.

[8] CS224N default project `http://web.stanford.edu/class/cs224n/default_project/default_project_v2.pdf`

[9] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.

[10] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[11] Matt Gardner and Joel Grus and Mark Neumann and Oyvind Tafjord and Pradeep Dasigi and Nelson F. Liu and Matthew Peters and Michael Schmitz and Luke S. Zettlemoyer, AllenNLP: A Deep Semantic Natural Language Processing Platform. arXiv:1803.07640, 2017.

[12] Evaluation scripts from code for the Default Final Project https://github.com/abisee/cs224n-win18-squad