Building a QA system (IID SQuAD track): BiDAF with Answer Pointer

Stanford CS224N Default Project (IID SQuAD track)

Catherine Wang
Department of Computer Science
Stanford University
cyw339@stanford.edu

Abstract

Question answering is a central task in NLP that can be broadly used to evaluate machine comprehension of natural language. Improvements in the question answering task are helpful for improving general understanding of text, which has a wide range of applications. This project investigates how character-level word embeddings and Answer Pointer[1] can enhance the performance of a baseline based on the BiDAF model. The addition of both character-level word embeddings and Answer Pointer led to slightly worse performance than the baseline, which highlights how many factors beyond model architecture can have a significant effect on performance.

1 Key Information to include

- Mentor: Lucia Zheng
- External Collaborators (if you have any): None
- Sharing project: No

2 Introduction

The question answering task involves returning the correct answer given a question and a paragraph of context. This simple framework actually provides a powerful way of measuring general machine comprehension of text, since the given questions can range from straightforward factual recall (What year did this occur?) to more complex inference (Why did this happen?) and be about anything. Improving reading comprehension is important for useful applications like answering search queries and summarization. The main idea behind the baseline Bidirectional Attention Flow (BiDAF) model is to have both the context attend to the question and the question attend to the context so our context representations are related to the question being asked [2]. Since the SQuAD dataset used for this project has a particular characteristic where the answer is a section of text within the given paragraph, it is a good situation to look to the Pointer Network architecture as an addition to our model. Pointer Nets were developed to produce an output based on positions in the input, which is exactly what SQuAD requires [3]. Instead of predicting the start and end positions of the output separately, we can use the Answer Pointer architecture based on Pointer Nets to condition the end position on the start position [2]. The original BiDAF model also included character-level word embeddings, which were not implemented in the baseline model and can improve the handling of out-of-vocabulary words and sub-word level meaning [2]. Surprisingly, the addition of character-level embeddings and Answer Pointer did not improve upon the baseline in this project. These results motivate a discussion of how model architecture is only one factor in neural network performance and what additional changes might help.

3 Related Work

BiDAF [4]: The baseline for this project comes from the paper on Bidirectional Attention Flow, which takes embeddings at the character, word, and contextual level and computes bidirectional attention between the context and the query to find which positions in each are most relevant for the other [4]. The model architecture was designed so that the output layer can easily be modified or replaced [4], which further incentivizes the use of Answer Pointer as an addition.

Pointer Network [3]: The Answer Pointer architecture is based on Pointer Network, which is tailored to the case where the output corresponds to positions within the input, and uses an attention mechanism to produce a distribution over input positions as a way to select optimal output positions [3]. The original Pointer Network works in a sequential way, where each element of the output is selected from the input and the resulting output may not be a continuous span within the input [3].

Answer Pointer [1]: The inspiration for this project was the paper introducing Match-LSTM and Answer Pointer. The paper experimented with two types of Answer Pointer. The sequence model works in the same way as the original Pointer Network, whereas the boundary model only selects the start and end positions and everything between the two is part of the answer, producing a continuous span of text from the context [1]. The boundary model performed better[1] and is also better suited to the SQuAD dataset, so this version was used in the project.

R-Net [5]: This paper also utilizes Answer Pointer as its output layer and actually provides a clearer formulation of the Answer Pointer equations and a good example of how to integrate Answer Pointer into a model.

4 Approach

The baseline model consists of an embedding layer using GloVE word embeddings, an encoder layer for contextual-level embedding, the BiDAF attention layer, a modeling layer, and an output layer [2]. This project makes two modifications to the baseline.

Character-level embeddings: The character-level embedding implementation is based on the model details described in the original BiDAF paper [4]. In order to maintain a hidden size H of 100, we first modify the baseline projection step so the word level embeddings have dimensionality $H/2$. Then we run randomly initialized character embeddings through a 2D convolution with $H/2$ output channels and a kernel size of (1, 5). We concatenate the word and character-level embeddings, both of dimensionality $H/2$, and feed the resulting embedding of dimensionality H to the Highway Network.

Answer Pointer: The Answer Pointer implementation for this project is based on the R-Net Answer Pointer output layer [5]:

Given a question representation $\{u_t^Q\}_{t=1}^m$ and a learnable weights parameter V_r^Q, we use the attention mechanism

$$s_j = v^\top \tanh(W_u^Q u_j^Q + W_v^Q V_r^Q) \tag{1}$$

$$a_i = \exp(s_i) / \sum_{j=1}^{m} \exp(s_j) \tag{2}$$

$$r^Q = \sum_{i=1}^{m} a_i u_i^Q \tag{3}$$

to produce a distribution a over the question representation based on V_r^Q and use it to obtain the attention output r^Q. v, W_u^Q, W_v^Q are learnable parameters–v is a vector and W_u^Q and W_v^Q are square matrices, all with size H based on the hidden size.

Then we use r^Q to attend to the context (passage) representation $\{h_t^P\}_{t=1}^n$ and select the start position p^1.

$$s_j^1 = v^\top \tanh(W_h^P h_j^P + W_r^Q r^Q) \tag{4}$$

$$a_i^1 = \exp(s_i^1) / \sum_{j=1}^{m} \exp(s_j^1) \tag{5}$$

$$p^1 = \textit{argmax}(a_1^1, ..., a_n^1) \tag{6}$$

$$c_1 = \sum_{i=1}^{m} a_i^1 h_i^P \tag{7}$$

Next r^Q and the attention output c_1 are used as the initial hidden state and the input, respectively, to an RNN that is run for one timestep to produce the new hidden state h_1^a. In the project implementation, this is an LSTM cell.

$$h_1^a = \textit{RNN}(r^Q, c_1) \tag{8}$$

We use h_1^a to again attend to the context representation $\{h_t^P\}_{t=1}^n$ to get the end position p^2.

$$s_j^2 = v^\top \tanh(W_h^P h_j^P + W_h^a h_1^a) \tag{9}$$

$$a_i^2 = \exp(s_i^2) / \sum_{j=1}^{m} \exp(s_j^2) \tag{10}$$

$$p^2 = \textit{argmax}(a_1^2, ..., a_n^2) \tag{11}$$

5 Experiments

5.1 Data

This project uses the SQuAD 2.0 dataset, which has examples consisting of context paragraphs from Wikipedia, questions, and answers which are sections of text from the context [2]. Half of the questions are unanswerable, meaning the answers are not found within the given context [2]. The inputs to the model are the indices of the context and question words corresponding to the word and character-level embeddings and the start and end positions of the answer within the context. The outputs are the start and end distributions over the context, which we can take the argmax of to find the start and end positions of the answer.

5.2 Evaluation method

Performance on this project was measured using the EM and F1 scores as defined in the project handout [2].

5.3 Experimental details

The following experiments were run:

- Baseline (hidden size: 100, learning rate: 0.5, dropout: 0.2, 30 epochs)
- Character-level embeddings (hidden size: 200, learning rate: 0.5, dropout: 0.2, 30 epochs)
- Character-level embeddings (hidden size: 100, learning rate: 0.5, dropout: 0.2, 28 epochs)
  - The virtual machine crashed at 28 epochs, but finishing the 30 epochs of training actually led to worse performance on the dev set, so we use the results from 28 epochs of training for this experiment
- Character-level embeddings + Answer Pointer (hidden size: 100, learning rate: 0.5, dropout: 0.2, 30 epochs)
- Character-level embeddings + Answer Pointer (hidden size: 100, learning rate: 0.6, dropout: 0.5, 30 epochs)
- Character-level embeddings + Answer Pointer (hidden size: 100, learning rate: 0.6, dropout: 0.35, 30 epochs)

In the first character-level embeddings implementation, both the word and character-level embeddings had dimensionality of hidden size $H = 100$, which led to a hidden size of $2H = 200$ after concatenation. The second configuration where both the word and character-level embeddings were halved to fit the original hidden size $H = 100$ after concatenation worked much better and was used for all other experiments. Attempts to improve the character-level embeddings + Answer Pointer model by increasing the dropout probability (since the dev loss began increasing about halfway through) and learning rate also did not lead to better performance, so the original dropout of 0.2 and learning rate of 0.5 were used on the test set.

5.4 Results

Test Set		
Model	EM	F1
Character Embeddings	57.633	61.137
Character Embeddings + Answer Pointer	56.839	60.431
Dev Set		
Model	EM	F1
Baseline	58.998	62.182
Character Embeddings	58.343	61.365
Character Embeddings + Answer Pointer	57.856	61.242
Character Embeddings + Answer Pointer (lr: 0.6, dp: 0.5)	53.571	56.071
Character Embeddings + Answer Pointer (lr: 0.6, dp: 0.35)	54.966	58.227

As the table shows, none of the additions to the baseline model improved performance, with both evaluation metrics decreasing with each addition. This is unexpected, because intuitively, both character-level embeddings and Answer Pointer should provide more information to the model and lead to better performance. Reasons for these results may include mistakes in the implementation, the hyperparameters, or characteristics of the specific architectures used in the model. This illustrates the difficulty of debugging neural networks, since it can be unclear where or even whether there is an error within the implementation.

6 Analysis

Since the new model does not differ drastically in performance from the baseline model, it is expected that the outputs of the two models will be similar. Here are some analyses of hypothesized differences between the two models:

One hypothesis was that the Answer Pointer model may be biased towards predicting fewer N/As compared to the baseline model. However, when comparing the baseline and Answer Pointer predictions on the dev set, the baseline predicts 2,320 N/As and Answer Pointer predicts 2,299 N/As out of 5,951 predictions, which is fairly similar. However, it seems that quite often, different questions are being predicted N/A by the Answer Pointer model and the baseline. On the dev set, the baseline and Answer Pointer agree on predicting N/A for 1745 out of 5951 examples. But inspecting a few samples where the two models don't agree on an N/A prediction doesn't seem to show any pattern–both the baseline and Answer Pointer seem to be making similar kinds of mistakes when incorrectly predicting an answer when the ground truth is N/A (which seems to be more common than incorrectly predicting N/A for a ground truth answer), where a word or sequence of words in the question is found in the context but does not lead to a correct answer.

Figure 1: Baseline (Answer Pointer correctly predicted N/A)

- **Question:** A routing packet is required under what system?
- **Context:** Connection-oriented transmission requires a setup phase in each involved node before any packet is transferred to establish the parameters of communication. The packets include a connection identifier rather than address information and are negotiated between endpoints so that they are delivered in order and with error checking. Address information is only transferred to each node during the connection set-up phase, when the route to the destination is discovered and an entry is added to the switching table in each network node through which the connection passes. The signaling protocols used allow the application to specify its requirements and discover link parameters. Acceptable values for service parameters may be negotiated. Routing a packet requires the node to look up the connection id in a table. The packet header can be small, as it only needs to contain this code and any information, such as length, timestamp, or sequence number, which is different for different packets.
- **Answer:** N/A
- **Prediction:** the node to look up the connection id in a table

Figure 2: Answer Pointer (Baseline correctly predicted N/A)

- **Question:** Where did China border Kublai's territory?
- **Context:** Instability troubled the early years of Kublai Khan's reign. Ogedei's grandson Kaidu refused to submit to Kublai and threatened the western frontier of Kublai's domain. The hostile but weakened Song dynasty remained an obstacle in the south. Kublai secured the northeast border in 1259 by installing the hostage prince Wonjong as the ruler of Korea, making it a Mongol tributary state. Kublai was also threatened by domestic unrest. Li Tan, the son-in-law of a powerful official, instigated a revolt against Mongol rule in 1262. After successfully suppressing the revolt, Kublai curbed the influence of the Han Chinese advisers in his court. He feared that his dependence on Chinese officials left him vulnerable to future revolts and defections to the Song.
- **Answer:** N/A
- **Prediction:** northeast border

It also appeared from visual inspection that Answer Pointer might be predicting shorter answers on average than the baseline. From the dev set predictions, the average baseline answer is 11.715 characters and the average Answer Pointer answer is 11.591 characters, so this also seems to not be the case.

Finally, it may just be the case that adding Answer Pointer to a model is not enough for significant improvement. Both the R-Net and Match-LSTM papers that use Answer Pointer in their output layer do not show ablation studies, so the bulk of their improvements may come from the other parts of their model.

7 Conclusion

This project found that after implementing character-level embeddings and an Answer Pointer output layer, model performance did not improve over the baseline, with no obvious differences when comparing the outputs. In the event that mistakes in the implementation itself contributed to the slightly worse performance, the project could have benefited from debugging techniques such as using data subsets. More hyperparameter and configuration tuning with batch sizes or embedding dimensions might also have helped.

References

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

[2] CS 224N. Cs 224n default final project: Building a qa system (iid squad track). 2022.

[3] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

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

[5] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.