

'Pointed' Question-Answering

Stanford CS224N {Default} Project

Shyamoli Sanghi
Department of Computer Science
Stanford University
`shyamoli@stanford.edu`

March 18, 2021

Abstract

Question-answering - the ability for a machine to infer an answer from a given question and context passage - is an important and keenly-studied application of Natural Language Processing. I built a question answering system for the Stanford Question Answering Dataset (SQuAD) 2.0 by implementing (from scratch) a boundary model of an Answer Pointer layer (introduced in the paper 'Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017) as a replacement of the output layer of the baseline BiDAF end-to-end neural network model. While the baseline model has the probability distribution of the end index computed independently of that of the start index, my new model aims to condition the distribution of the end index prediction on the distribution of the start index prediction, so that it can allow the machine to learn how the end of an answer can depend on its start. My Answer Pointer model manages to achieve an F1 score of 59.60 and an EM score of 55.01 on the development set, which is an improvement on the baseline model (that has an F1 score of 52.19 and an EM score of 52.19).

1 Key Information about the Project

- Mentor: Zihan Wang
- External Collaborators: None
- Sharing project: N/A

2 Introduction

Machine reading comprehension through question-answering is one of the most interesting and significant problems in Natural Language Processing because it

not only measures how well the machine 'understands' a piece of text but also helps provide important informational answers to humans. For this task, given a paragraph and a question relating to that paragraph, the machine's model must select the span from the paragraph that corresponds to the answer using a start index prediction and end index prediction.

My baseline model for this task is a Bidirectional Attention Flow (BiDAF) end-to-end neural network, with embedding, encoder, attention, modelling and output layers. Significantly, the output layers involves the probability distribution of the start index token and end index token to be generated independently. However, in order for the model to learn how the end of an answer can depend on the start of an answer, I implement a Pointer Network, a sequence-to-sequence model proposed by Vinyals et al. (2015) in the form of an Answer Pointer layer that replaces the output layer of the baseline (as is suggested in the paper Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017), that enables us to condition the prediction for the end token on the prediction for the start token of the answer in the input text. Further, since a Pointer Network outputs a probability distribution exclusively over locations in the input paragraph (context) at each step instead of outputting a probability distribution over an entire vocabulary, it allows us to improve the model's efficiency. My experiments with implementing an Answer Pointer layer result in an **F1 score of 59.60 and an EM score of 55.01** on the development set, which is an improvement on the baseline model, which has an **F1 score of 52.19 and an EM score of 52.19**.

In this paper, I present the details of my approach, performance of the model and analysis of results of implementing this Answer Pointer Network in my model for the question-answering task.

3 Related Work

My model is based on the paper Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017), which explains the motivation and equations used to implement an Answer Pointer layer. The idea for pointer networks was first introduced in the seminal paper 'Pointer Networks' (Vinyals et al , 2015) which describes a network that accomplishes the goal of conditioning the end location's probability distribution on the probability distribution for the start location in the question-answering model; instead of predicting the start and end locations independently.

This paper explains that any sequence-to-sequence recurrent Neural Networks with a content-based attention mechanism (such as that used in machine translation) requires that the number of target classes at each step of the output (the output dictionary) has a **fixed** size a priori. However, our SQuAD question-

answering dataset makes it the case that the length of the answer to the question can be any length . Thus, due to the notion of variable sized answer predictions, the above-described RNN framework cannot be applied directly to the address this case. Instead, in order to solve this problem, the paper proposes using a Pointer Network to allow for the output to be selected from the input sequence by using attention as a pointer to the input tokens exclusively (not the entire context).

In fact, the idea of a Pointer net can be used to help several other tasks such as extractive and abstractive text summarization as is described in the paper 'Get To The Point: Summarization with Pointer-Generator Networks' (Abigail See et al, 2017).

4 Approach

I modified the baseline Bidirectional Attention Flow (BiDAF) model to achieve an improvement in the model's performance. In order to implement the Answer pointer layer, I worked on replacing the baseline's output layer to include the boundary model Answer pointer in the network. I referred to the paper 'Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017) for the relevant equations, which I had to modify to implement the boundary model of the Answer pointer layer.

First, I let the input to the output layer remain - the baseline's attention layer output matrix G and modelling layer output matrix M . Next, I computed the distribution (attention weight vector) β_s and β_e for selecting any token from the context as the start and the end token respectively using the following equations (7 and 8 in the paper), written below:

$$\begin{aligned}\mathbf{F}_k &= \tanh(\mathbf{V}\mathbf{H} + (\mathbf{W}^a \mathbf{h}_{k-1}^a + \mathbf{b}^a) \otimes e_N) \\ \beta_k &= \text{softmax}(\mathbf{v}^T \mathbf{F}_k + c \otimes e_N)\end{aligned}$$

Here, 'N' is the length of the context and 'h' is the hidden size of the model. Further, $\mathbf{V}, \mathbf{W}^a, \mathbf{b}^a, \mathbf{v}$ and c are parameters to be learned. Here, $\otimes e_N$ repeats the tensor ahead of it N times in its first dimension. Finally, $\mathbf{h}_{k-1}^a$ represents the hidden vector of the Answer LSTM defined as follows: $\mathbf{h}_k^a = \text{LSTM}(\mathbf{H}\beta_k, \mathbf{h}_{k-1}^a)$.

Significantly, I used the concatenation of the baseline's attention layer output matrix G and modelling layer output matrix M as the input $\mathbf{H}$ to the output (answer pointer) layer, making it the case that the dimensions of $\mathbf{H}$ are batch-size*N*10h, even though the paper stated that $\mathbf{H}$'s dimensions are batch-size*N*2h. This is because we needed $\mathbf{H}$ to be the hidden representation matrix that contained all the information the model has obtained so far - which includes the attention and modelling layer outputs. I modelled $\mathbf{V}$ as a linear layer with no bias of input size 10h and output size h, such that, when given

H as input, it would yield a batch-size*N*h matrix. Next, since W_a is a weight parameter and b_a is a bias parameters to be learned, I modelled W_a s a linear layer with bias of input size = h and output size = h. Further, I chose to take 'v' as a linear layer without bias, of input size = 'h' and output size = 1. I modelled 'c' as a real number parameter that would then be repeated batch-size times along its 0th dimension and context-size times along its 1st dimension. I used an LSTMCell layer to implement the aforementioned answer LSTM that, at each time step, outputs the next hidden and cell states (h_{next}, c_{next}) when given as inputs: the tensor H multiplied with β_{curr} and a tuple of the current hidden state and cell states (h_{curr}, c_{curr}). Here, I initialized the current hidden and cell states at some time step as a vector of dimension = batch-size*hidden size.

After computing the quantities in the above two equations, I used equations 11 and 13 from the Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017) paper to deduce that:

$$p(\mathbf{a}|\mathbf{H}) = p(a_s|\mathbf{H}) * p(a_e|a_s, \mathbf{H}) = \beta_s * \beta_e.$$

This yields a batch-size*N*N matrix P such that for a particular example in the batch k, P_{kij} = probability that i is answer starts at context token i and ends at context token j, for that example. Next, I summed P over the third dimension to get the p_{start} tensor (with dimensions: batch-size, N) where $p_{start}(k,i)$ represents the probability that the answer starts at context token i for the batch example k. I also summed P over the second dimension to get the p_{end} tensor (with dimensions: batch-size, N) where $p_{end}(k,j)$ represents the probability that the answer ends at context token j for the batch example k. Finally, I computed the natural logarithm of these tensors and returned them as the outputs of this layer (and hence my model).

I used the sum of the negative log-likelihood loss (cross-entropy loss)for the start and end locations as the model's training loss function.

5 Experiments

In this section, I will describe the dataset and evaluation metrics I employed as well as the details of experiment attempts and results obtained so far.

5.1 Data

For this project, I used the Stanford Question-Answering Dataset (SQuAD 2.0) provided to us as a part of the default project material. The paragraphs in this dataset are from Wikipedia and it has questions and answers that were crowdsourced using the platform Amazon Mechanical Turk. The official dev and test set have three different answers (from three different crowd workers) for each answerable SQuAD question, such that the answers do not always match. The dataset has a total of 150000 questions. As opposed to the original SQuAD dataset, SQuAD 2.0 has almost half questions that cannot be answered given the input context (paragraph). The dataset provided to us has the following three

splits: a Training set with all examples from the official SQuAD 2.0 training set; a Dev set with examples which are randomly selected from the official SQuAD 2.0 dev set, that make up approximately half of this official dev set; a Test set which consists of the remaining examples from the official SQuAD 2.0 dev set, along with hand-labelled examples. Most importantly, SQuAD is a challenging dataset for evaluating question-answering algorithms because, as opposed to previous datasets, in SQuAD the answers can be anywhere in the context passage and have variable lengths. During setup, we **pre-processes** the data for its efficient loading.

5.2 Evaluation method

I used a quantitative evaluation metric (to measure performance of the model), which is based on the two metrics (Exact Match score and F1 score). The Exact Match (EM) score is a strict, binary metric which measures whether the predicted system output exactly matches the ground truth answer. The F1 score is the harmonic means of the precision and recall, where precision denotes the quotient of the number of positive results that are correctly identified and the total number of positive results. Meanwhile recall refers to the number of positive results that are correctly identified divided by the number of samples that ought to have had positive results. After calculating the EM and F1 scores for each of the 3 human-provided answers for each question, the maximum EM and F1 scores over answers for each question are found and then the EM and F1 scores across the whole evaluation dataset are averaged to compute the final EM and F1 scores.

5.3 Experimental details

First, I experimented with implementing different training loss function to compare the model's performance with these loss functions respectively. They were as follows: For a batch of examples,

$$\text{training-loss-1} = -\sum_{k=1}^{\text{batch-size}} \log P(k)(i)(j)$$

For a single example,

$$\text{training-loss-2} = -\log p_{start}(i) - \log p_{end}(j)$$

For this, I created two Answer Pointer models, whose outputs were compatible with their corresponding training loss functions (for the first model, the output was the logarithm of the tensor P described in the previous section; for the second model, the outputs were the logarithms of p_{start} and p_{end}). However, I soon realized that the first loss function would not work with the baseline model because the model expected as output, two tensors of dimensions p_{start} , and not a tensor of the dimensions that P had. Thus, I decided to stick to the second - cross entropy loss function - which I also described in the previous section.

After this, I trained and tested the baseline model, with the default parameters that included the following: **number of epochs = 30, batch size = 64,**

hidden size = 100, dropout rate = 0.2 and learning rate = 0.5. This took 11 hours on a Microsoft Azure NC6 Ubuntu 18.04 Virtual Machine. Next, I trained and tested the above-described Answer Pointer model with the same default parameters as the baseline model. This took 6 hours on a Microsoft Azure NCv3 Ubuntu 18.04 VM but would have taken 18 hours on a Microsoft Azure NC6 Ubuntu 18.04 VM. While training my Answer pointer model, I encountered GPU excess memory issues, realized that I needed to detach some of my model tensors so as to declare that they should not require gradient computation during backpropagation. I will describe the results of testing both models below:

5.4 Results

The results I obtained on development set for the Baseline Model were the following :

F1 score: 52.19

EM score: 52.19

Negative Log-Likelihood Loss: 5.34

Answer vs No Answer Percentage: 52.14

The results I obtained on development set for the new model with the Answer Pointer layer were the following (on the non-PCE leaderboard) :

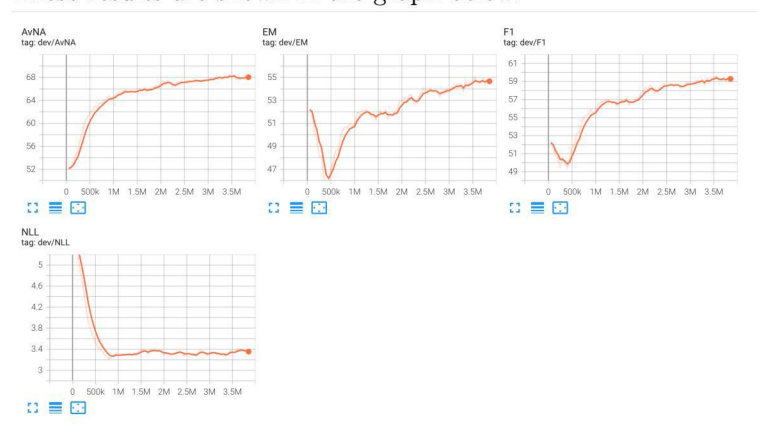
F1 score: 59.60

EM score: 55.01

Negative Log-Likelihood Loss: 3.36

Answer vs No Answer Percentage: 68.34

These results are shown in the graph below:



Importantly, the Answer pointer model that I implemented outperforms the baseline model on the development set as the F1 exact match scores for the new model are higher than that of the baseline model and the new model's Negative Log-Likelihood loss is lower than that of the baseline.

(I could not compute the results on the test set because the test submission CSV file had a few missing predictions, which I do not understand the reason for - I think that the CSV file generation code might have caused this - because I had similar errors for the baseline model too, which trained properly and were based on the provided starter code).

However, this model does not perform as well as the Machine Comprehension using Match-LSTM and Answer Pointer' (Wang et al, 2017) model, which obtains an F1 score of 77 and an EM score of 67.9, as is reported in the paper. This was expected because the Match-LSTM model allows the model to learn textual entailment which would allow it to better 'understand' meaning and hence make more accurate predictions.

6 Analysis

On looking at the output predictions and true answers, it is evident that while the model generated accurate predictions often, there were several instances of the modelling predicting subsets of true answers. In particular, the model generated shorter-than-expected answer predictions, such as is shown in the examples below.

- **Question:** How did the new king react to the Huguenots?
- **Context:** Louis XIV gained the throne in 1643 and acted increasingly aggressively to force the Huguenots to convert. At first he sent missionaries, backed by a fund to financially reward converts to Catholicism. Then he imposed penalties, closed Huguenot schools and excluded them from favored professions. Escalating, he instituted dragonnades, which included the occupation and looting of Huguenot homes by military troops, in an effort to forcibly convert them. In 1685, he issued the Edict of Fontainebleau, revoking the Edict of Nantes and declaring Protestantism illegal.[citation needed]
- **Answer:** acted increasingly aggressively to force the Huguenots to convert
- **Prediction:** convert

However, even in these cases, the model got the end token prediction correct most times. Some examples of this behaviour occur when the answer ought to be 'Giovanni Branca', but instead the model predicts 'Branca'. Similarly, the answer ought to be 'since the 1960s', but the model predicts '1960s'. We can infer that this could occur because the end predictions involved going through additional (answer)LSTM cell layers than the corresponding start predictions.

Further, the model (as well as the baseline model) makes fallacious predictions on adversarial examples such as the one below:

- **Question:** What drugs help T cells respond to signals correctly?
- **Context:** Anti-inflammatory drugs are often used to control the effects of inflammation. Glucocorticoids are the most powerful of these drugs; however, these drugs can have many undesirable side effects, such as central obesity, hyperglycemia, osteoporosis, and their use must be tightly controlled. Lower doses of anti-inflammatory drugs are often used in conjunction with cytotoxic or immunosuppressive drugs such as methotrexate or azathioprine. Cytotoxic drugs inhibit the immune response by killing dividing cells such as activated T cells. However, the killing is indiscriminate and other constantly dividing cells and their organs are affected, which causes toxic side effects. Immunosuppressive drugs such as cyclosporin prevent T cells from responding to signals correctly by inhibiting signal transduction pathways.
- **Answer:** N/A
- **Prediction:** Immunosuppressive drugs such as cyclosporin

In this case, the 'Immunosuppressive drugs such as clorosporin' actually **do the opposite of help** the T cells respond to signals (they prevent them from responding to signals). However, the model does not understanding this notion of opposite meaning and hence generates an incorrect prediction. To ensure that this kind of error is less frequent or does not occur at all, we could replace the answer LSTMCell's current dot product attention mechanism to a multiplicative attention mechanism in the output layer of the model in question. This would allow the model to highly weight the importance of the word 'prevent' as having the opposite sense as that of the word 'help', and hence capture meaning more precisely.

Finally, the Answer pointer Model had fewer no answer predictions than the Baseline model, which points to the fact that the Answer pointer outputs a probability distribution exclusively over context tokens as opposed to the entire vocabulary, which reduces the chances of a no-answer prediction. While, this was desirable in several cases in which the baseline model incorrectly predicted 'No answer', on the other hand, for the answers that were actually not in the context, there were inaccurate context token prediction made by the Answer pointer model.

7 Conclusion

In this project, I implemented and evaluated an Answer Pointer network within the BiDAF Baseline model for the question-answering task. I also analyzed the performance results of the baseline and Answer Pointer models. This allowed me to gain exposure in several unexplored areas such as: implementing layers in neural networks, understanding the importance of different kinds of attention mechanisms, modelling training loss functions, working with CSV files, using Virtual Machines, transferring code from VM to local and vice-versa, and overall being responsible for my model's behaviour. Thus, while this process was challenging, it was a great well-rounded learning experience for me.

Overall, my experiments and analysis demonstrated that it is useful for the BiDAF model to learn how the end token predictions for the answer can depend on the start index predictions, because it is seen that implementing the Answer Pointer layer in the model improves its performance.

Finally, I think there are several ways I can still improve my model. For example, I can include a more sophisticated attention mechanism for the answer LSTM in the output layer so as to allow the model better learn meaning of context tokens. I could also include character embeddings in my model so that it can better learn internal word structure.

8 Acknowledgements

I would like to thank all the instructors in the class CS 224N at Stanford, especially my project mentor, Zihan Wang, for guiding me throughout this project. I am also grateful to have gotten subscription to Microsoft Azure VMs to train and test my models efficiently. Finally, I would like to thank Professor Christopher Manning for teaching us NLP concepts in a clear and accessible manner. I am truly grateful to have got this unique, extremely enriching opportunity to work with the BiDAF model architecture and undertake the IID SQuAD question-answering task.

9 References

- 1) Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. arXiv preprint arXiv:1608.07905, 2016.
- 2) Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In Neural Information Processing Systems
<https://arxiv.org/pdf/1506.03134.pdf>
- 3) Abigail See, Peter J. Liu, Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. arXiv:1704.04368