Look Before You Answer: Separating Answerability Prediction in QA Systems
Intro2IE course mid-term report

Umang Gupta
umanggup@usc.edu

Hanpeng Liu
hanpengl@usc.edu

Abstract
In this project, we are tackling the problem of extractive reading comprehension (RC) where answers may or may not be in the paragraph. We start with QANet (Yu et al., 2018) as our starting model and implement some of the existing modification to handle the case when the question cannot be answered as baselines. We summarize the results of our baselines and discuss further direction.

1 Introduction
We are trying to tackle the problem of reading comprehension where the question may not be answerable. Since, our major focus is on developing better method to predict if the question can be answered or not based on the information from the comprehension, we fix the model architecture for answering. Answer are always continuous phrase from the paragraphs, so it is sufficient to output start and end index of the answer in the paragraph. We use QANet (Yu et al., 2018) as model for answering as it uses convolution instead of recurrent units and is faster to train and test. We report baseline results on SQuAD dataset only (Rajpurkar et al., 2018) so far.

Most of the methods handle case of unanswerable question by adding extra outputs to the model that account for predicting answerability. We use three baselines, that add extra outputs and modify the loss function of typical QA models to learn to predict unanswerable question. We describe QANet architecture, baselines in brief, results for baselines and further work.

Authors contributed equally

We decided to use QANet instead of Bi-DAF model which was mentioned in our survey report because they have similar performance but QANet is faster to train.

2 QANet architecture & representation
In this section, we describe the architecture of the base model QANet. Similar to Bi-DAF (Seo et al., 2016), QANet has embedding layers, context-query attention layers, modeling layers, and output layers. The output layers will predict the start/end probabilities for each word in the context. The major difference between QANet and Bi-DAF is that QANet uses encoders with convolution and self-attention operations to replace RNN-like structure as the main building blocks. The training speed is significantly improved as the convolution and self-attention operations are done in parallel.

2.1 Context/Query representation
QANet uses fixed lengths for contexts (paragraphs) and queries (questions). Most specifically, after tokenization, only the first 400 tokens in the given paragraphs will be used as context and 50 tokens in the given question will be used as query. For each word, first 16 characters are only use to compute character embeddings. If the context/query is shorter than the length threshold, zero-padding is applied. QANet uses pre-trained GLoVE vectors as word and character embeddings for representing both query and context.

2.2 Answer representation
Similar to other pointer networks for reading comprehension, QANet predicts a span from origin context paragraphs as the candidate answer to the question. A span consist of a start index and an end index, indicating the location of the answer in the context paragraphs. Since, the answer is essentially a contiguous phrase in the context, predicting start and end index is sufficient.
2.3 Model architecture

Mainly, QANet is composed of four kinds of layers: embedding layers, encoder layers, context-query attention layers, decoder layers, and output layers. Embedding layer combines word embeddings and char embeddings to convert to a single vector representation for each word. The both context & query are passed through encoder layers. Context-query attention combines context and query and produces output vector for each word in context which is decoder layers. Finally softmax is applied to predict the start and end index probabilities for each word in context. Decoder layer have same structure as encoder layers. Encoder layer has ResNet (He et al., 2016) like connections and adds position embeddings, performs convolution and multi-head attention to the input and finally pass it through a fully connected layer. We give more detailed description of model with figures reproduced from (Yu et al., 2018) in appendix 6.

The major difference is that QANet uses convolutional encoders to replace RNN encoder in encoder layers and decoder layers.

3 Baseline models based on QANet

We discuss how to add answerability prediction into QANet architecture. As described in sec. 2, pointer networks are used to predict the start and end index of answer in the paragraph. Pointer network produce two softmax probability distribution \( p_s, p_e \in \mathbb{R}^N \) of start and end index respectively where \( N \) is the length of context paragraph. We denote corresponding logit score as \( l_s \) and \( l_e \) respectively. Answer phrase is predicted by computing

\[
i, j = \arg \max_{i', j', i' \leq j'} p_s(i') p_e(j')
\]

\[
= \arg \max_{i', j', i' \leq j'} \log \text{softmax}(l_s(i')) + \log \text{softmax}(l_e(j'))
\]

and loss for training the typical QA system (without no-answer case is), when \( i, j \) are indices of correct answer phrase

\[
\text{loss} = -\log \frac{e^{l_s(i)}}{\sum_{i'} e^{l_s(i')}} - \log \frac{e^{l_e(j)}}{\sum_{j'} e^{l_e(j')}}
\]

Apart from predicting the start and end index, we make model produce one more output which is the confidence if the question is unanswerable. \( p(z) \) is the confidence of question being unanswerable, based on which the model will decide to output the predicted answer span or the signal that the question is not answerable.

3.1 Random prediction model

In this model, we use QANet model as it is to predict answer start and end index. Along with that we generate a random number for computing \( p(z) \) and if it greater than 0.5 we say the question in answerable else not. During training, loss is calculated for only those samples for which \( p(z) < 0.5 \).

3.2 Plus-1 model

Given a context of \( N \) tokens, this model predicts start/end probabilities for \( N + 1 \) positions. This model, called as plus-1 model, is based on (Levy et al., 2017). The start/end probabilities in the extra position are used for answerability prediction. More specifically, \( p(z) = p_s(N + 1) p_e(N + 1) \) is a signal indicating whether question is unanswerable. There are two ways to choose threshold for \( p(z) \). The first one is to use a fixed threshold. The second one is to use

\[
t = \max_{i \leq N, j \leq N, i \leq j} p_s(i) p_e(j)
\]

as the threshold and output not-answerable \( (z = 1) \) if

\[
p(z) = p_s(N + 1) p_e(N + 1) > t.
\]

Such simple modification is easy to implement in QANet: we pad the outputs of three decoder blocks with one extra row, so the length of final prediction is one more than that of the input context.

Another benefit of plus-1 model is that we don’t need to tweak the loss function. For a context of \( N \) tokens, when the question is not answerable, we change both the start and end of the answer span to \( N + 1 \) and compute the corresponding negative log-likelihood as the loss.

3.3 No-answer option model

Like (Clark and Gardner, 2018; Rajpurkar et al., 2018), we compute a separate score \( z \) which is computed by weighted combination & attention on decoder outputs and passing through a linear layer that predicts a single output. Decoder output used for predicting start index score is weighted
by $p_s$, decoder output used for predicting end index scores is weighted by $p_e$, and all three decoder outputs are combined using attention. Input to the linear layer for predicting $z$ can be expressed as —

$$\text{concat}(\sum_i p_s(i) h_i, \sum_j p_e(j) h_j, \sum_k a_k h_k)$$

where, $a_k = \text{softmax}(w \cdot h_k)$, $w$ is a trainable parameter, $h_i$ is output of first and second decoder, $h_j$ is output of second and third decoder and $h_k$ is output of all three decoders concatenated (See fig. 2). Loss function is modified to account for $p(z)$ in normalization.

$$\text{loss} = -\log \left(1 - \delta \right) e^z + \delta e^{x_s + s_e} \over e^z + \sum_{i,j} e^{x_i + s_i}$$

$\delta$ is 1 if question is un-answerable.

4 Experiment results

We conducted preliminary experiments to test the performance of these three baselines. We run experiments on SQuAD2.0 (Rajpurkar et al., 2018) and did training and validation over the provided training set and collected the results over the provided dev set. Due to resources limitation, we only let each model run for 10 epochs, approximately taking 8-12 hours on a machine with Titan 1080Ti GPU card. For the evaluation metrics, we used the scripted provided by SQuAD2.0 which computed one exact match (EM) score and one F1 score. We also report the accuracy of answerability prediction.

For training, we use Adam (Kingma and Ba, 2014) and Exponential moving average (EMA). The token limitations are 400 for context paragraphs, 50 for questions, and 30 for answers. We use 1-head attention and the hidden size is 96. We use pre-trained word/char embeddings from GloVe (Pennington et al., 2014). The length of word vectors is 300 and the length of char vectors is 64. The code is available online.

Analysis The experimental results is shown in Table 1. Our results are surprising and we are still investigating why this is so. In particular, Plus-1 model has poor results than random predictions. We strongly think this is due to bug in our code and we are still trying to debug the issue. Our No-answer option model performs better than the other two baselines as has been previously observed by (Rajpurkar et al., 2018) too. However, the EM and F1 score are unexpectedly high. We are not sure how this relates to results on test set, but a similar result on test set can put us near top on SQuAD2.0 leaderboard. We are still investigating these issues and results are only preliminary and inconclusive.

5 Further Plan

To investigate whether our original idea will work out, we need to further work. Next step is to make sure that our baselines are sensible and bug-free. We think that computing answerability confidence and answer from same model is the issue and therefore we want to further investigate by separating the two models.

5.1 Investigate fully separate model

The final goal is to investigate whether and how can a fully separate model (separating answerability predictions and answer span predictions, do answer span prediction conditioned on answerability prediction) can help in reading comprehension dataset with adversarial questions, such as SQuAD2.0. We plan to implement the fully separate model for the two tasks (predicting answerability and answer extraction) based on QANet and run similar tests as we did for baseline methods. We expect that to perform better than the baseline models.

5.2 Investigate loss function and prediction calibration

Given the new introduced answerability variable, the loss function becomes more complex. However, only 50% data is used for training answer span prediction (which is a 400-class classification) and the rest 50% data is used for answerability prediction (which is a binary classification). Baselines have different losses depending on how they predict answerability confidence. Not all of them try to balance the performance of answerability prediction and that of answer span accuracy when it’s answerable.

Another potential issue is about label calibration, i.e., deciding the not answerable threshold. We suspect that the model’s performance can be
Table 1: Performances of three baseline models. The metrics are collected in SQuAD2.0 Dev set after 10 epochs training. EM & F1 are scores for answer span predictions. Acc, Pre, Rec stand for answerability prediction’s accuracy, precision, recall. We also report the EM & F1 got in SQuAD1.1, which contains all answerable question-answer pairs in SQuAD2.

<table>
<thead>
<tr>
<th>Model</th>
<th>EM</th>
<th>F1</th>
<th>Pre</th>
<th>Rec</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random prediction model</td>
<td>65.99</td>
<td>68.67</td>
<td>0.8770</td>
<td>0.7808</td>
<td>0.7808</td>
</tr>
<tr>
<td>Plus-1 model</td>
<td>59.54</td>
<td>63.21</td>
<td>0.7362</td>
<td>0.7011</td>
<td>0.7011</td>
</tr>
<tr>
<td>No-answer option model</td>
<td>74.72</td>
<td>77.44</td>
<td>0.8894</td>
<td>0.8347</td>
<td>0.8347</td>
</tr>
<tr>
<td>QANet (SQuAD1.1)</td>
<td>78.37</td>
<td>68.55</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

improved by calibrating the answerability prediction. We will look into this if time permits.

5.3 Minor improvements

One possible improvement is about training speed. Though QANet is 3x speed-up comparing to BiDAF, our implementation takes around 18 hours to train the model to a decent stage in one machine with Titan 1080Ti GPU card. Improvements may involve reducing the model size and removing non-important layers, which can help us to iterate the model faster and serve as a ablation study to understand which modules are more important.

References


6 Appendix

6.1 Model architecture

Mainly, QANet is composed of four kinds of layers: embedding layers, encoder layers, context-query attention layers, decoder layers, and output layers. The major difference is that QANet uses convolutional encoders to replace RNN encoder in encoder layers and decoder layers.

Embedding layers For both context and query, QANet applies both word embeddings and char embeddings over the tokens. The word embeddings and char embeddings are concatenated as the final vector representation for a token.

Encoder layers QANet uses convolutional layers and self-attention layers as the encoder to replace RNN layers. The encoder block architecture is similar to those used in NMT area (Vaswani et al., 2017; Gehring et al., 2017). Each encoder block contains a position embedding layer, ResNet (He et al., 2016) like convolutional layers, a self-attention layer, and a final fully connected layer. QANet also adds layer normalization and highway connections inside each encoder block.

The architecture of one encoder block is shown in Figure 1.

Context-query attention layers QANet uses standard context-query attention schema, which computes the similarity (by some neural network function) between every pair of context and query tokens’ representations and then applies a Softmax normalization for each row (representing a token in context). The output of context-query attention layer is weighted sum between context-query similarity and learned query representations. This is often called as context-to-query attention.

Decoder layers Similar to the encoder layers, the decoder layers also use convolution and self-attention operators to replace RNN-like structure. The decoder layers contains three encoder blocks in a row. The outputs of the first and second blocks are used for predicting answer span start probability and the outputs of the first and third blocks are used for answer span end probability.

Output layers The output layers are simple fully-connected layers with Softmax activation. The final output is the probability of each token become the start/end of the answer span.

The architecture of QANet is drawn in Figure 2.