Introduction

We build a question answering (QA) system and apply it on the Stanford Question Answering Dataset (SQuAD 2.0), seeking to achieve high accuracy scores (EM, F1). Given a correct paragraph and a question, we wish the system to output a contiguous span of words that is the answer to the question. We undertake to build a Smart Chunk Reader (SCR) a model based on Dynamic Chunk Reader (DCR) and Stochastic Answer Network (SAN). SCR seeks to augment DCR with an intermediary "candidate chunk" neural model that establishes a probability distribution over chunks (i.e., any text) which could comprise the answer. We chose to implement Stochastic Answer Network (SAN) as our chief candidate chunk selection model due to its multi-step construction of the final answer span distribution coupled with its use of stochastic prediction dropout.

Background

In the present context, transformer-based models achieve superior performance. QANet, for example, moved away from the use of recurrent networks, and relies upon convolution, self-attention mechanisms, and achieved the best published F1 score for its time. However, rather than opting for a transformer-based architecture like many of our peers and previous project teams, we sought to build a model incorporating unique insights that came from approaches independent from transformers. This drew us to Dynamic Chunk Reader and Stochastic Answer Network.

• Chunk-Representation: The model Dynamic Chunk Reader, implements a compelling and intuitive idea of representing answer candidates as chunks instead of word level representations, to make the model aware of subtle differences among candidates.

• Multi-step Reasoning: Answer-span distributions are generated over T-time steps, the final probability distribution is a function of the average of the answer-span distributions that remain after applying stochastic dropout.

• Attention Mechanism: Both Dynamic Chunk Reader and Stochastic Answer Network make use of variants of dot-product style attention methods.

Methods

Our model is primarily an extension of DCR (Dynamic Chunk Recognizer), a model architecture that builds a representative over candidate answer "chunks" of text instead of predicting separate start-of-answer and end-of-answer probability distributions.

Smart Candidate Generation

The original DCR paper naïvely generates candidate chunks by enumerating all possible sub-ranges of the context up to a maximum answer length. However, this is infeasible for many cases. More recently, questions have been generated by taking the top K chunks from the model. These K chunks are the candidate input to our DCR model.

To generate our candidates, we decided to use a model called SAN (Stochastic Answer Network) because it had strong K-Oracle performance and an interesting architecture.

Experiments

Our experiments consisted testing a set of baselines including DCR and then combinations of a candidate model to generate a chunk with DCR and SAN. We discuss the procedure and findings below.

1. DCR: This baseline involved first direct implementations of DCR with the default hidden state size of 128.

2. SAN. This was our baseline implementation of SAN with a hidden size of 128.

3. SAN+DRCR. A candidate model not tested with DCR. We implemented SAN model with the high performing SAN model tested it independent of the DCR baseline.

4. SAN+DRCR=model. Once we got SAN working, we paired our SAN candidate model with DCR itself, DCR with an extended hidden representation and encoding layer. Due to memory constraints we had to shrink the hidden size of both SAN and DCR.

Results

<table>
<thead>
<tr>
<th>Models</th>
<th>F1</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCR baseline</td>
<td>56.60</td>
<td>55.70</td>
</tr>
<tr>
<td>SAN128</td>
<td>62.73</td>
<td>60.94</td>
</tr>
<tr>
<td>SAN12</td>
<td>65.07</td>
<td>61.84</td>
</tr>
<tr>
<td>SCR SAN12</td>
<td>58.24</td>
<td>56.96</td>
</tr>
<tr>
<td>SCR SAN12</td>
<td>58.77</td>
<td>61.03</td>
</tr>
</tbody>
</table>

Table 1. Performances of various models on the SQuAD 2.0 development set.

We were moderately pleased with SAN performance achieving a peak F1 score of 65.07 and EM of 61.84. While these results were top-performing than the published results in the original SAN paper, we had no opportunity to test our contextual embeddings (as was done in the original paper).

We were surprised by the underperformance of our SCR models relative to the DCR baselines. We were also surprised by how passing more candidates to SCR from SAN is associated with even poorer F1 performance.

Analysis

Most of our experiments were centered around reducing our two main sources of error: (i) candidate selection taken and DCR failures. The only metric we cared about for a candidate was its K Oracle performance. As long as the correct answer was within the top K, DCR would have a chance of perfectly answering the question. Reducing any of these sources of error was easy. Increasing K, the number of candidates we selected, would drive candidate selection failures lower (often towards << 5%) at the cost of DCR performance as it selected between more candidates. Decreasing K would increase DCR performance at the cost of more candidate selection failures. So, we needed to train better SAN candidate models that could achieve very high K Oracle performance for smaller K. Then, we needed to train DCR models on the candidate output of these SAN models.

Analysis (cont.)

• DCR performed progressively worse with longer answer lengths with EM dropping as low as 25%. SCR maintained a higher level of performance across answer lengths.

• Similarly, SCR has key variance in performance across question types, likely due to a different range of possible candidates.

• We see a similar improvement in EM score as K increases across SAN sizes. SAN128 is slightly better for higher K, so we mainly trained SCR with SAN128.

Interesting Architectural Pieces

While much of the implementation of SCR (Smart Chunk Reader) was built on basic embeddings, attention, and LSTM, we also worked with some more interesting architectural components.

Question-enhanced Passage Word Embedding

This layer augments the embedding of context words with information about the question being asked, allowing context words to be represented in different ways for different questions.

Let \( g : \alpha \times \beta \rightarrow \delta \) be a \( \beta \)-dim single layer neural network.

\[ y_k = g(x_k, q) \]

Finally, our initial embedding for a context word \( x_k \) is:

\[ f(x_k) = \sum_{i=0}^{n} y_k(x_i) \]

Then, we use a two layer feed forward network \( F_{\theta}(X) \) to bring the dimension of context and question words back down to a common size (the question words are never enhanced in this way).

\[ f_{\theta}(x_k) = F_{\theta}(\sum_{i=0}^{n} y_k(x_i)) \]

Memory Generation

The memory generation works to build a question-attended representation of the context through the use of distal structural mechanisms between words and the context/semantics.

• Given \( P \in \mathbb{B}^{\alpha \times \delta} \) and \( Q \in \mathbb{B}^{\beta \times \delta} \), representations of the passage and the question, respectively.

We first transform our input with a single-layer network \( P \rightarrow \overline{P} = W \cdot PL + \beta Q \rightarrow \overline{Q} = W \cdot QL \).

• We then take basic attention with dropout. \( C \rightarrow \text{dropout}(\alpha, \beta) \cdot P \rightarrow \text{dropout}(\alpha, \beta) \cdot Q \).

• In addition to our usual features, we also attempt to capture connective attention between words by taking attention and then dropping the diagonal (all self attention is removed out).

• Finally, we feed our visual and connective features into an LSTM.

\[ M = \text{LSTM}(\overline{P}, \overline{Q}) \]

Stochastic Answer Module

As an initial state is constructed based on question representation \( Q \) and is passed through a GRU with hidden state \( h_0 \). The answer module will compute over 5 time steps given the state \( h_0 \) and the memory \( M \). We experimented with expanding each state to include the passage representation \( P \) but this did not insignificantly improve performance.

• \( h_{t+1} = \text{softmax}(\alpha_{t+1})(h_t) \cdot W_{h} \cdot \beta Q + \alpha_{t+1} \cdot W_{h} \cdot M \)

• \( s_{t+1} = \text{softmax}(\alpha_{t+1})(h_t) \cdot W_{s} \cdot \beta Q + \alpha_{t+1} \cdot W_{s} \cdot M \)

Evaluating training, we dropped all of these layers to ensure that no single layer output is necessary for our model(s). Finally, \( s_{t+1} \) and \( s_{t+1} \) are set to be the averages of our 5 computed distributions.

References