Smart Chunk Reader: Stochastic Neural Chunk Selection for SQuAD

Antonio Ferris [1] Alex Loia [1] Jonah Wu [2]

[1]Computer Science, Stanford [2]Symbolic Systems, Stanford

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 context paragraph and a question, we want the system to output a contiguous span from the context paragraph that is the answer to the question. We undertook to build **Smart Chunk Reader (SCR)**, a model based on Dynamic Chunk Reader (DCR)[2] and Stochastic Answer Network (SAN)[1]. SCR seeks to augment DCR with an intermediary "candidate chunk" neural model that establishes a probability distribution over what chunks (of any size) 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 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 Mechanisms** Both Dynamic Chunk Reader and Stochastic Answer Network made use of variants of dot-product style attention methods.

Methods

Our model is primarily an extension of DCR (Dynamic Chunk Recognition), a model architecture that builds a representation 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 naively generates candidate chunks by enumerating all possible sub-ranges of the context up to a maximum answer length. However, this is bad for two reasons.

- First, this method involves creating chunk representations over up to $O(n^2)$ different candidates. This results in a very slow runtime of DCR.
- Second, this method is unable to answer questions with long answers correctly. The original DCR doesn't even enumerate answers that are past a certain length.

We decided to solve both of these problems by extending the DCR architecture with another step - candidate generation. Instead of naively generating candidates, we first run a candidate model and choose the top K chunks from the model. Those K chunks are the candidate input to our DCR model.

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

Experiments

Our experiments contained testing a set of baselines including DCR, and then combinations of a candidate model to generate chunks, and a DCR architecture that would rank them.

1. **DCR** This baseline involved our first direct implementation of DCR with the default hidden size of 300.
2. **SAN**^{128}. This was our baseline implementation of SAN with a hidden size of 128.
3. **SAN**^{512}. A candidate model not tested with DCR - this high performing SAN model took so much memory that DCR didn't fit alongside it in RAM.
4. **SAN**^{128} **+ DCR**$_2^{100}$. Once we got SAN working, we paired our SAN candidate model with DCR_2, DCR with an extended feature representation and encoding layer. Due to memory constraints we had to shrink the hidden size of both SAN and DCR.

Results

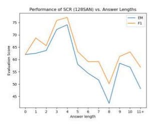
Models	F1	EM
DCR baseline	58.60	55.70
SAN128	63.73	60.34
SAN512	**65.07**	**61.84**
SCR, SAN128-3	58.24	53.96
SCR, SAN128-20	54.77	51.03

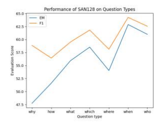
Table 1. Performances of various models on the SQuAD 2.0 development set

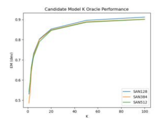
We were moderately pleased with SAN performance achieving a peak F1 score of **65.07** and EM of **61.84**. While these results were less performant than the published results in the original SAN paper, we had no opportunity to utilize contextual embeddings (as was done in the original paper). We were 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: candidate selection failures 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 at perfectly answering the question. Reducing any one of these sources of error was easy. Increasing K, the number of candidates we selected, would drive candidate selection failures low (often towards $< 10\%$), 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 poorly with longer answer lengths with EM dipping as low as 20%. SCR maintained a tighter band of performance across answer lengths.
- Similarly, SCR has less variance in performance across question types, likely due to a tighter 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 basic embeddings, attention, and LSTMs, 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 $g(\cdot)$ is a 280-dim single layer neural network.

$$\gamma_{i,j} = \frac{exp(g(GloVe(c_j))g(GloVe(q_i)))}{\sum_{j'} exp(g(GloVe(c_i))g(GloVe(q_{j'})))}$$

Finally, our initial embedding for a context word is: $f(c_i) = \sum_j \gamma_{i,j} g(GloVe(q_j))$.

Then, we use a two layer feed forward network FFN to bring the dimension of context and question words back down to a common size (the question words are never enchanced in this way)

$f_{embed}(c_i) = FFN(\sum_j \gamma_{i,j} g(GloVe(q_j)))$

Memory Generation

The memory generation seeks to build a question-attended representation of the context through use of dot-product mechanisms between words and the context/question.

- Given $P \in \mathbb{R}^{d\times n}$ and $Q \in \mathbb{R}^{d\times m}$, representations of the passage and question, respectively. We first transform our input with a single layer network. $\hat{P} = ReLU(W_1 P), \hat{Q} = ReLU(W_2 Q)$
- We then take basic attention with dropout. $C = \text{dropout}(f_{att}(\hat{Q}, \hat{P})) \in \mathbb{R}^{m\times n}$
$U = [P; QC] \in \mathbb{R}^{2d\times n}$
- 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 zeroed out).
$\hat{U} = U\text{drop}_{diag}(f_{att}(U, U))$
- Finally, we feed our usual and connective features into an LSTM.
$M = \text{BiLSTM}([U; \hat{U}])$

Stochastic Answer Module

An initial state is constructed based on question representation Q and is passed through a GRU with hidden state x_t. The answer module will compute over 5 time steps given the state s_t and the memory M. We experimented with expanding each state to include the passage representation P, but this did not meaningfully improve performance.

- $s_0 = \sum_j \text{softmax}(w_3 Q_j) Q_j$, $s_t = \text{GRU}(s_{t-1}, x_t)$
- $x_t = \sum_j \text{softmax}(s_{t-1} W_4 M) M_j$ Once we have computed s_t for all t, we can then get our output distributions.
- $P_t^1 = \text{softmax}(s_t W_5 M)$, $P_t^2 = \text{softmax}([s_t; \sum_j P_{t,j}^1 M_j] W_7 M)$

(During training, we dropout 40% of these layers to ensure that no single layer output is necessary for our model.) Finally, P_1 and P_2 are set to be the averages of our 5 computed distributions.

References

[1] Yuwei Fang Aerin Kim Kevin Duh Xiaodong Liu, Wei Li and Jianfeng Gao. Stochastic answer networks for squad 2.0, 2017.

[2] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension, 2016.