# Smart Chunk Reader: Stochastic Neural Chunk Selection for SQuAD

Stanford CS224N Default Project

**Antonio Ferris**
Department of Computer Science
Stanford University
antoniof@stanford.edu

**Alex Loia**
Department of Computer Science
Stanford University
alexloia@stanford.edu

**Jonah Wu**
Symbolic Systems
Stanford University
jonahwu@stanford.edu

## Abstract

We build a question answering (QA) system and apply it on the Stanford Question Answering Dataset (SQuAD 2.0) [1], seeking to achieve high accuracy scores (EM, F1, AvNA) given default project constraints. We sought to build an "intelligent" implementation of Yu et al.'s Dynamic Chunk Reader (DCR) [2] that we dub Smart Chunk Reader (SCR), primarily through feeding DCR with top answer candidates and weightings from a performant candidate model, Stochastic Answer Networks (SAN) [3] from Liu et al. Using our performant SAN model, we achieve an F1 score of **60.34** and an EM score of **57.14** on the unseen SQuAD 2.0 test set.

## 1 Key Information to include

- Mentor: Michihiro Yasunaga
- External Collaborators (if you have any): None
- Sharing project: n/a

## 2 Introduction

Reading comprehension-based question answering (RCQA) has been tackled to identify single-entity answers or select from multiple choices but to build robust QA systems, we want them to answer questions pertaining both to entities (factoid) or explanations/reasoning (non-factoids). More specifically, 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. QA is challenging for a variety of reasons. Questions can be unanswerable and ambiguous. Topics and the style of questions can differ widely and good QA systems should be robust to these differing contexts.

Despite the fact that largely RNN-based models are outperformed by non-recurrent transformer-based models like QANet, we were drawn to the highly intuitive approach of generating and ranking full answer chunks rather than probabilities of start and end tokens as presented by Dynamic Chunk Reader (DCR) [2]. Moreover, by intelligently augmenting candidate chunk selection away from naive chunk generation, which is $O(n^2)$ in the length of the answer, we believed we could improve significantly on the results DCR obtained by training a candidate model to generate plausible answer chunks. The Stochastic Answer Networks (SAN) model [3] stood out as a viable candidate model due to the unique nature of its output layer. SAN uses memory in its architecture to generate $T$ different output distributions and averages them to create a final output.

While DCR obtained exceptional results for its time, we seek to streamline the way candidate chunks are generated. DCR struggles with long answers because it naively enumerates candidate chunks and then evaluates all of them in the chunk representation layer. Instead, we propose and implement Smart Chunk Reader (SCR), which 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 SAN as our chief candidate model due to its impressive reported $K$-oracle performance and its unique construction of the final answer span distribution. Then, we use DCR to accurately rank these more intelligently chosen candidate chunks.

Our main contributions are as follows:

1. We rebuilt DCR with its original candidate chunk layer from scratch.
2. To intelligently provide candidates, we rebuilt SAN from scratch.
3. We construct a new model (SCR) that uses the skeleton of DCR with candidate chunk generation provided by the SAN model.
4. We explore various modifications to both SAN and DCR to improve their collective performance.
5. We evaluate the performance of our Smart Chunk Reader, DCR, and SAN for the CS224N SQuAD 2.0 dataset and analyze our models' limitations.

To improve on the original architecture of SAN, we experiment with different attention mechanisms each built from scratch as well as model sizes. To improve on DCR, we experiment with augmenting the embedding layer, changing the candidate sampling process, and appending additional "information" in the form of log probabilities for start and end candidates from SAN.

## 3 Related Work

QA systems seek to locate answers given a question and paragraph of context. In the present context, transformer-based models achieve superior performance. QANet [4], 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, which we then built from scratch.

The model Dynamic Chunk Reader (DCR) [2] implements the 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. A chunk representation layer dynamically extracts the candidate chunks from the given passage, and creates chunk representations supposed to encode the contextual information of each chunk. These chunks are ranked through ranking the relevance between representations of a chunk and the question. However, the selection of chunk candidates in the base architecture appeared to be highly inefficient, which is what we decided to improve.

The Stochastic Answer Network (SAN) [3] engages in multi-step reasoning before selection of a answer candidate given the context and question. In addition, within the answer module, stochastic prediction dropout was applied to the set of answer candidates. The answer module computes over T steps based on an initial state, which is a representation of the question. Answer span distributions are generated through a function of the state, and a memory module M. And given T-answer span distributions, the final probability distribution is a function of the average of all answer span distributions that remain, given a set stochastic dropout probability over the distribution at each time step. It is worth noting that SAN originally used pretrained contextual embeddings from English-German neural machine translation for its context and question encoding.

## 4 Approach

Here we describe the architecture of our baseline (DCR), SCR, and SAN models used to answer question in the aforementioned SQuAD 2.0 dataset.

### 4.1 Baseline

Our baseline implementation is our custom implementation of the Dynamic Chunk Reader model architecture (detailed below) with its naive candidate chunk generation mechanism.

### 4.2 Smart Chunk Reader

We built the original DCR model from scratch in PyTorch, only reusing the IID SQuAD default project start code as scaffolding.

#### 4.2.1 Encoder Layer

Here, we used the same basic features that both DCR and SAN used (POS, NER, and binary exact match features), but also expanded on their embedding features with a question-enhanced passage word embedding for context words inspired by SAN.

Let $g(\cdot)$ is a 280-dim single layer neural network.

We compute $\gamma_{i,j}$ as:

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

Finally, our embedding for a context word is:

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

Then, we use a two layer feed forward network to bring the dimension of context and question words back down to our hidden size.

#### 4.2.2 Attention Layer

We implemented the same attention architecture as DCR. Other than experimenting with different hidden sizes, we kept the structure the same.

#### 4.2.3 Chunk Representation Layer

First, we use our candidate model to generate two probability distributions, $p1$ and $p2$, which are the probability distributions of the start and end of the answer chunk, respectively. We then sample $k$ different chunks from these distributions, where $k$ is a model parameter for the number of candidates.

Then, given the output $\gamma_j$ of the BiGRU from the Attention Layer, we create a chunk representation $\bar{\gamma}_{m,n} = [\overrightarrow{\gamma}_m, \overleftarrow{\gamma}_n]$.

#### 4.2.4 Ranker Layer

Finally, we rank the chunks given their representation. We create a question representation $\left[\overrightarrow{h^{Q_i}_{|Q_i|}}, \overleftarrow{h^{Q_i}_1}\right]$, which mirrors the creation of our chunk representation. We then rank candidate chunks based on the cosine similarity of their chunk representation to this question representation.

For a given candidate $c_i^{m,n}$, its probability would be:

$$\mathbb{P}(c_i^{m,n}|P_i, Q_i) = \text{softmax}\left(\overline{\gamma}^i_{m,n} \cdot \left[\overrightarrow{h^{Q_i}_{|Q_i|}}, \overleftarrow{h^{Q_i}_1}\right]\right)$$

### 4.3 Candidate Model: Stochastic Answer Network

We also built the original SAN model from scratch in PyTorch. Again, we only make use of the default project code as scaffolding. We implemented everything using PyTorch and the `spacy` NLP package for the part-of-speech (POS) and named entity recognition (NER) tags when encoding.

#### 4.3.1 Lexicon Encoding Layer

Here, we used an expanded feature set for the context words as used in the original SAN paper.This includes 9 dimensions of POS tagging information, 8 dimensions of NER tagging, 3 dimensions of binary exact match with the question (exact match, lowercase, lemma), and 280 dimensions of question-enhanced word embedding as in SCR's **Encoder Layer** above.

Both the expanded context word embeddings and the question word embeddings are run through a 2 layer feed forward neural network to equalize the dimensions.

#### 4.3.2 Contextual Encoding Layer

Here, we were forced to depart from the SAN's original model structure. We replaced the pretrained contextual embedding with an RNN encoder that trained along with the rest of the model. This means that our SAN implementation had significantly more parameters to train than the original SAN, which was able to take advantage of pretrained contextual embeddings (CoVe from English-German neural machine translation).

#### 4.3.3 Memory Generation Layer

Here, we construct the working memory for the answer module from the paragraph and question using dot product attention. We implemented multi-headed attention from scratch as well, but it didn't improve performance.

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. $\hat{P} = \text{ReLU}(W_1 P), \hat{Q} = \text{ReLU}(W_2 Q)$
We start by taking a basic attention with dropout: $C = \text{dropout}(f_{att}(\hat{Q}, \hat{P})) \in \mathbb{R}^{m \times n}$.
Let $U = [P; QC] \in \mathbb{R}^{2d \times n}$.
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: $M = \text{BiLSTM}([U; \hat{U}])$.

#### 4.3.4 Answer Module

The Answer Module is the most interesting part of the SAN to us. Instead of performing a single computation to generate the final output probabilities, which tends to concentrate a lot of the computation of a network on one layer operation, SAN instead generates $T$ different distributions and averages them to get the final output. In addition, during training, dropout is applied to the $T$ output distributions so that no single layer output is critical to the network. An initial state is constructed based on the question representation $Q$ and is passed through the GRU with hidden state $x_t$.

Let $s_0 = \sum_j \text{softmax}(w_3 Q_j) Q_j$.

The answer module will compute over $T = 5$ 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.
$s_t = \text{GRU}(s_{t-1}, x_t)$.
$x_t = \sum_j \text{softmax}(s_{t-1} W_4 M) M_j$.
Once we have computed $s_t$ for all $t$, we can then get our output distributions:
$P_t^1 = \text{softmax}(s_t W_5 M)$ and $P_t^2 = \text{softmax}([s_t; \sum_j P_{t,j}^1 M_j] W_7 M)$
(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.

# 5 Experiments

## 5.1 Data

We use the Stanford Question Answering Dataset v.2.0 (SQuAD 2.0) as provided for the CS224N default project, which has different splits than the original SQuAD 2.0 dataset [1]. The training set consists of roughly 130K examples. The dev set and test set are roughly 6000 examples each. Each example consists of a question, a related context, and provided answers (or no answer - N/A).

## 5.2 Evaluation method

We primarily use the F1 score and $K$-oracle EM score for evaluation, using this to compare the performance of our models. We also considered the exact match (EM) accuracy and answer vs. no-answer (AvNA) scores. $K$-oracle score is the score, given consideration of the candidate model's $K$ top answer candidates based on its answer span distribution. For example, a 3-oracle EM score, is the ratio of exact matches within the 3 top answer candidates of the answer span distribution per example.

Evaluating the $K$-oracle scores of our candidate models was important because returning a set of answer chunks containing the right answer in the chunk representation layer is critical. Otherwise, SCR would have no shot at predicting the correct answer.

## 5.3 Experimental details

Here we report how we ran our experiments (e.g. model configurations, learning rate, training time, etc.).

### 5.3.1 Dynamic Chunk Reader

We run our baseline DCR model with a hidden state size of 300, batch size of 64, dropout rate of 0.2, and learning rate of .5. We trained our baseline model for about 30 epochs.

### 5.3.2 Stochastic Answer Networks

We run our baseline SAN model with a hidden state size of 128, batch size of 64, dropout rate of 0.2, stochastic prediction dropout rate of .4, T = 5 (time steps in Answer Module), and learning rate of .5. Our learning rate was significantly higher than the authors (.002), as we did not have pretrained contextual embeddings in our model. We trained this model for 30 epochs over 4 hours.

### 5.3.3 Upsized Stochastic Answer Networks

We also run SAN, 3x larger, with expanded state, and SAN, 4x larger. SAN, 3x larger increases the hidden size from 128 to 384 and SAN, 4x increases the hidden size to 512. We experimented with various learning rates (scheduling halves every 5 or 10 epochs) and weight decay (in the range of 0.01 to 0.001) but ultimately reverted to the default 0.5 learning rate without weight decay.

### 5.3.4 Smart Chunk Reader

We run Smart Chunk Reader with 4 distinct candidate layers - the baseline SAN model which returned the top 20, top 10, and top 3 candidates, and the SAN, 3x larger model returning the top 20 candidates. Due to GPU memory constraints, we generally downsized the DCR-derived components to use a hidden size of 100 so that the SAN (1x and 3x) candidate models could fit in memory alongside SCR.

### 5.4 Results

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

| Models | F1 Percentage | EM Percentage |
|---|---|---|
| DCR (Naive Candidate Selection) | 58.60 | 55.70 |
| SAN (1) | 63.73 | 60.34 |
| SAN, 3x larger (2) | 58.09 | 55.77 |
| SAN, 4x larger (3) | **65.07** | **61.84** |
| SCR, SAN (1), top 3 | **58.24** | **53.96** |
| SCR, SAN (1), top 10 | 55.26 | 51.15 |
| SCR, SAN (1), top 20 | 54.77 | 51.03 |
| SCR, SAN, 3x larger (2), top 20 | 54.20 | 51.62 |

We submitted SAN, 4x larger (3) to the test leaderboard, achieving an F1 score of **60.344** and an EM score of **57.143**.

Our results on the development set can be referenced in the table above.

DCR with naive candidate selection performs at 58.60 F1 and 55.7 EM on the development set. Our SAN models performed better than our baseline, with the 4x larger model achieving a peak F1 score of **65.07** and EM of **61.84**. While these results were significantly less performant than the published results in the original SAN paper, we were still moderately pleased with SAN performance, given that we had no opportunity to utilize pretrained contextual embeddings (as was done in the original paper) and therefore had to learn our context and question representations from the starting point of GLoVe and positional encodings with no representations added from any other dataset.

We were surprised that the larger SAN variants did not achieve expected performance improvements, suggesting that increasing the complexity of SAN may not be significant. However, we were concerned with whether DCR can be improved through intelligent selection of candidate chunks, which is what we sought to test in the third set of models in the table, SCR. In this regard, we were surprised with the underperformance of our SCR models with SAN as the candidate model layer. We analyze and seek to make sense of those suboptimal results below.

## 6 Analysis

### 6.1 Performance breakdown

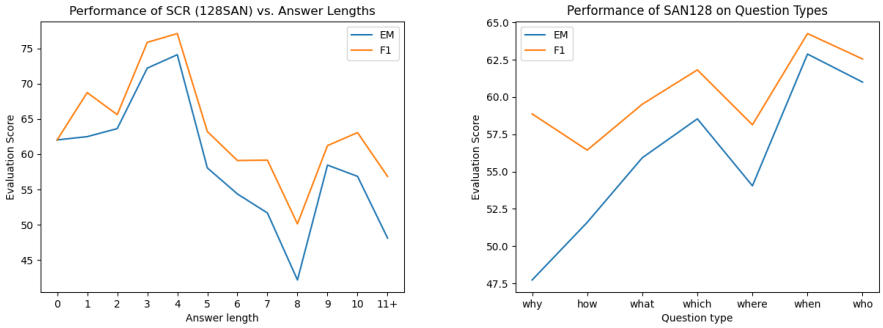


Figure 1: Performance of SCR with SAN (1) Candidate Model vs. Answer Lengths (Plot 1) and Question Head Word (Plot 2)

Our final model did relatively well on "Who" and "When" questions, but much worse on "Why" and "How" questions. This is most likely due to the way answers to these questions relate to the chunk representation step. Names and dates are often unique and distinct from their surroundings, allowing them to be easily separated into "chunks" for our model. However, "How" and "Why" questions are less structured and their answers sometimes involved entire sentences or phrases where the start and end tokens are harder to identify.

The "Why" and "How" answers are also longer. SCR does markedly worse on questions with longer answers than those with shorter answers. However, DCR also struggles in these same ways and SCR generally improves on DCR performance in these key metrics.

In the original paper, 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 narrower range of possible candidates.

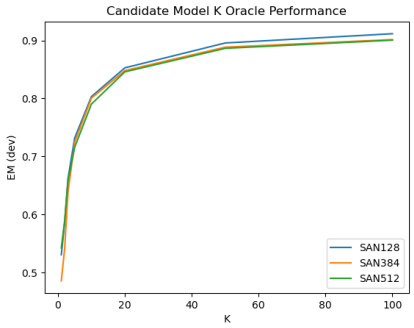


Figure 2: $K$-oracle performance for SAN candidate models from $1 \leq K \leq 100$

### 6.2 Dispositioning of Errors

As observed in the $K$-oracle performance plot, our larger SAN model $K$-oracle performance was impressive, which is promising, as SCR can choose from the right candidates. However, we could not test the 4x SAN model as candidate model for SCR due to the GPU memory limits imposed upon us, meaning that given the compute available, we had to make the difficult choice of splitting our parameters between SCR and and a candidate model with poorer $K$-oracle performance which directly inhibited the upper bound of our performance.

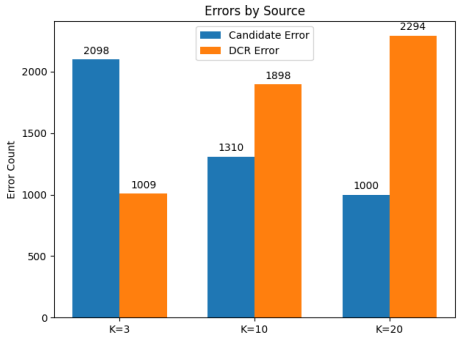


Figure 3: Dispositioning of error source between SCR and SAN candidate model

When iterating on models, we focused on reducing our two main sources of error - candidate error and DCR error. Controlling the value of $K$ let us trade less of one for more of the other, sometimes with greater final results. We paired this search over the parameter space of $K$ with attempts to create better candidate and DCR models that could improve both errors.

### 6.3 Specific example

- **Question:** Packet switching speed is determined by what factor?
- **Context:** Packet switching contrasts with another principal networking paradigm, circuit switching, a method which pre-allocates dedicated network bandwidth specifically for each communication session, each having a constant bit rate and latency between nodes. In

cases of billable services, such as cellular communication services, circuit switching is characterized by a fee per unit of connection time, even when no data is transferred, while packet switching may be characterized by a fee per unit of information transmitted, such as characters, packets, or messages.

- **Answer:** N/A
- **Prediction:** fee per unit of information transmitted, such as characters, packets, or messages

Looking at an example taken from the best performing SCR model (3-oracle SAN (1)), we can see that the model incorrectly predicted a long span when there was in fact no answer. This was observed several times across examples, and we selected this one to analyze. SAN provided SCR with this long candidate chunk as one of three possibilities that is near to "packet switching" and a plausible factor given the question. It is highly probable that SAN did not provide N/A as a plausible chunk. Thus, SCR had no opportunity to select the ground truth. In the future, we could experiment with always providing N/A as a choice to SCR, if not provided by the candidate model, so that SCR can independently evaluate the N/A possibility with respect to the question and context.

## 7 Conclusion

In summary, we implement a chunk reader model similar to DCR which accepts candidate chunks from a stochastic answer network. While SCR does not outperform baseline DCR, it is more efficient, since chunk representations need only be built over the candidate chunks passed by the SAN model. So far, we have only explored the viability of SAN as candidate layer. Moreover, while we have tentative results on how scaling the number of answer candidates passed to chunk representation effects the performance of SCR, we were not able to test this across a robust range. In the future, we would like to examine how SCR interacts with other candidate models with less stringent memory constraints, and extensively test how the number of candidates passed to SCR by the candidate layer affects its performance.

## References


[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[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.

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

[4] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.