Stochastic Answer Networks for Machine Reading Comprehension

Xiaodong Liu[†], **Yelong Shen**[†], **Kevin Duh**[‡] **and Jianfeng Gao**[†]
[†] Microsoft Research, Redmond, WA, USA
[‡] Johns Hopkins University, Baltimore, MD, USA
[†]{xiaodl,yeshen,jfgao}@microsoft.com [‡]kevinduh@cs.jhu.edu

Abstract

We propose a simple yet robust **s**tochastic **an**swer **n**etwork (SAN) that simulates multi-step reasoning in machine reading comprehension. Compared to previous work such as ReasoNet, the unique feature is the use of a kind of stochastic prediction dropout on the answer module (final layer) of the neural network during the training. We show that this simple trick improves robustness and achieves results competitive to the state-of-the-art on the Stanford Question Answering Dataset (SQuAD).

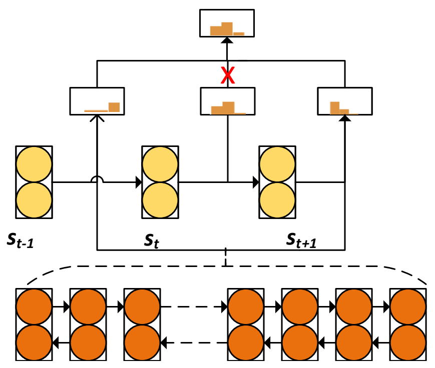


Figure 1: Illustration of "stochastic prediction dropout" in the answer module during training. At each reasoning step t, the model combines memory (bottom row) with hidden states $\mathbf{s_{t-1}}$ to generate a prediction (multinomial distribution). Here, there are three steps and three predictions, but one prediction is dropped and the final result is an average of the remaining distributions.

1 Introduction

Machine reading comprehension (MRC) is a challenging task: the goal is to have machines read a text passage and then answer any question about the passage. This task is an useful benchmark to demonstrate natural language understanding, and also has important applications in e.g. conversational agents and customer service support. It has been hypothesized that difficult MRC problems require some form of multi-step synthesis and reasoning. For instance, the following example from the MRC dataset SQuAD (Rajpurkar et al., 2016) illustrates the need for synthesis of information across sentences and multiple steps of reasoning:

Q: What collection does **the V&A Theator & Performance galleries** hold?

P: **The V&A Theator & Performance galleries** opened in March 2009. ... **They** hold the UK's biggest national collection of <u>material about live performance.</u>

To infer the answer (the underlined portion of the passage P), the model needs to first perform coreference resolution so that it knows "**They**" refers "**V&A Theator**", then extract the subspan in the direct object corresponding to the answer.

This kind of iterative process can be viewed as a form of multi-step reasoning. Several recent MRC models have embraced this kind of multi-step strategy, where predictions are generated after making multiple passes through the same text and integrating intermediate information in the process. The first pioneering models employed a predetermined fixed number of reasoning steps (Hill et al., 2016; Dhingra et al., 2016; Sordoni et al., 2016; Kumar et al., 2015). Later, Shen et al. (2016) showed that dynamically determining the number of steps based on the complexity of the question and answer leads to improvements in MRC; the model uses reinforcement learning to predict the number of reasoning steps for each question-answer pair. Further, Shen et al. (2017) empirically showed that dynamic multi-step reasoning outperforms fixed multi-step reasoning, which in turn outperforms single-step reasoning on two distinct MRC datasets (SQuAD and MS MARCO).

In this work, we derive an alternative multi-step reasoning neural network for MRC. During training, we fix the number of reasoning steps, but perform stochastic dropout on the answer module (final layer predictions). During decoding, we generate answers based on the average of predictions in all steps, rather than the final step. We call this a stochastic answer network (SAN) because the stochastic dropout is applied to the answer module; albeit simple, this technique significantly improves the robustness and overall accuracy of the model. Intuitively this works because while the model successively refines its prediction over multiple steps, each step is still trained to generate the same answer; we are performing a kind of stochastic ensemble over the model's successive prediction refinements. Stochastic prediction dropout is illustrated in Figure 1. We find that SAN is an effective model for multi-step reasoning in MRC.

2 Proposed model: SAN

The machine reading comprehension (MRC) task as defined here involves a question $Q = \{q_0, q_1, ..., q_{m-1}\}$ and a passage $P = \{p_0, p_1, ..., p_{n-1}\}$ and aims to find an answer span $A = \{a_{start}, a_{end}\}$ in P. We assume that the answer exists in the passage P as a contiguous text string. Here, m and n denote the number of tokens in Q and P, respectively. The learning algorithm for reading comprehension is to learn a function $f(Q, P) \rightarrow A$. The training data is a set of the query, passage and answer tuples $< Q, P, A >$.

We will now describe our model from the ground up. The main contribution of this work is the answer module, but in order to understand what goes into this module, we will start by describing how Q and P are processed by the lower layers. Note the lower layers also have some novel variations that are not used in previous work.

As shown in Figure 2, our model contains four different layers to capture different concept of representations: 1) the lexicon encoding layer encodes different types of lexicon variants and its linguistic features; 2) the contextual encoding layer attempts to capture relations between words and phrases in context; 3) the memory generation layer gathers all the different information from the passage and question and forms a "working memory" for the final answer module; 4) the final answer module, a type of multi-step network, searches the answer span. The detailed description of our model is provided as follows.

Lexicon Encoding Layer. The purpose of the first layer is to extract information from Q and P at the word level and normalize for lexical variants. A typical technique to obtain lexicon embedding is concatenation of its word embedding with other linguistic embedding such as those derived from Part-Of-Speech (POS) tags. For word embeddings, we use the pre-trained 300-dimensional GloVe vectors (Pennington et al., 2014) for the both Q and P. Following Chen et al. (2017), we use three additional types of linguistic features for each token p_i in the *passage* P:

- 9-dimensional POS tagging embedding for total 56 different types of the POS tags.

- 8-dimensional named-entity recognizer (NER) embedding for total 18 different types of the NER tags. We utilized small embedding sizes for POS and NER to reduce model size. They mainly serve the role of coarse-grained word clusters.

- A 3-dimensional binary *exact* match feature defined as $f_{exact\_match}(p_i) = \mathbb{I}(p_i \in Q)$. This checks whether a passage token p_i matches the original, lowercase or lemma form of any question token.

- Aligned question embeddings: $f_{align}(p_i) = \sum_j \gamma_{i,j} g(GloVe(q_j))$, where $g(\cdot)$ is a 280-dimensional single layer neural network $ReLU(W_0 x)$ and $\gamma_{i,j} = \frac{exp(g(GloVe(p_j)) \cdot g(GloVe(q_i)))}{\sum_{j'} exp(f(GloVe(p_i)) \cdot f(GloVe(q_{j'})))}$ measures the similarity in word embedding space between a token p_i in the passage and a token q_j in the question. Compared to the *exact* matching features, these embeddings encode *soft* alignments between similar but not-identical words.

In summary, each token p_i in the passage is represented as a 600-dimensional vector and each token q_j is represented as a 300-dimensional vector.

Due to different dimensions for the passages and questions, in the next layer two different bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997) may be required to encode the contextual information. This, however, introduces a large number of additional parameters. To prevent this, we employ an idea inspired by

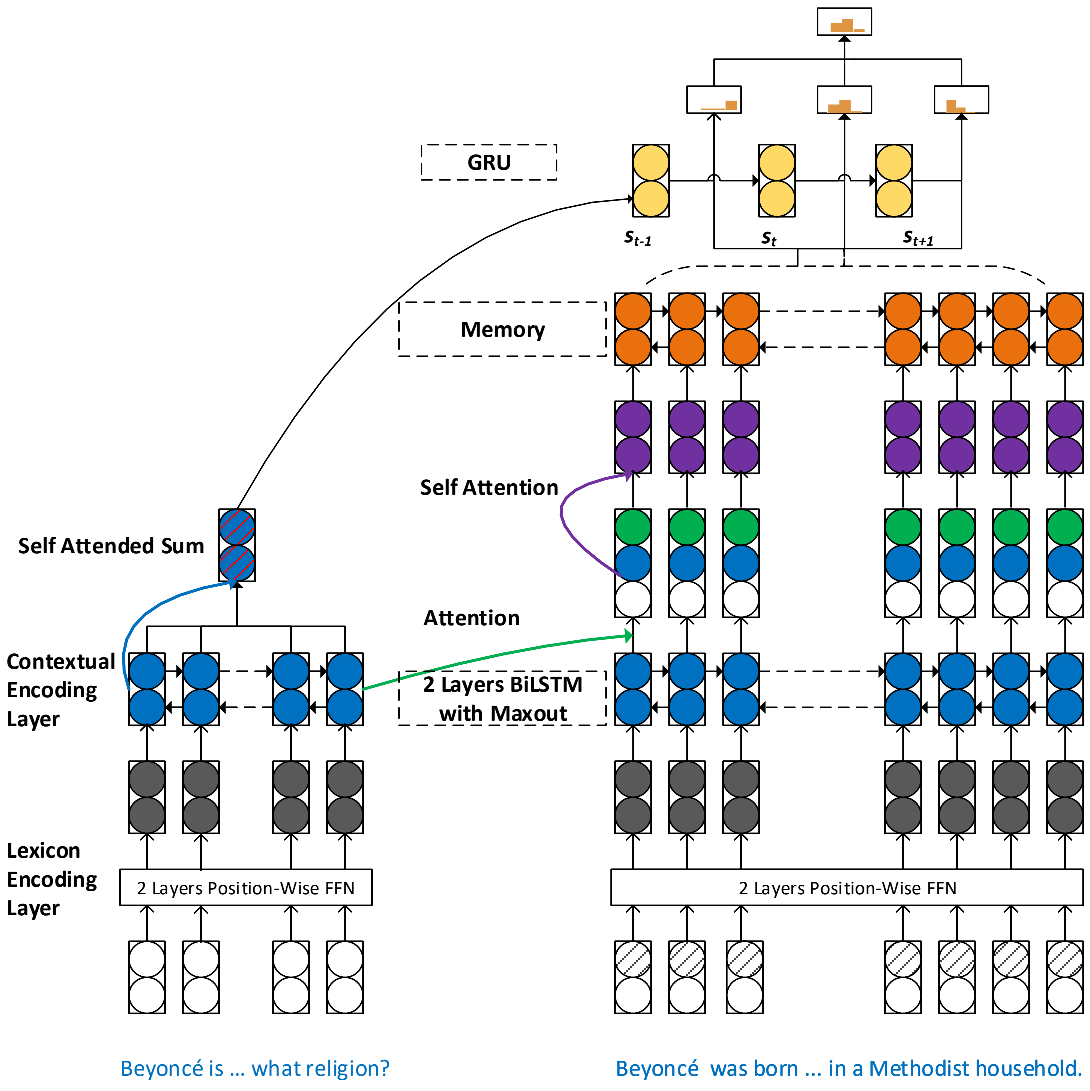

GRU
s_{t-1}
s_t
s_{t+1}
Memory
Self Attention
Self Attended Sum
Attention
Contextual Encoding Layer
2 Layers BiLSTM with Maxout
Lexicon Encoding Layer
2 Layers Position-Wise FFN
2 Layers Position-Wise FFN
Beyoncé is ... what religion?
Beyoncé was born ... in a Methodist household.

Figure 2: **Architecture of the SAN for Reading Comprehension:** The first layer is a lexicon encoding layer that maps words to their embeddings independently for the question (left) and the passage (right): this is a concatenation of standard word embeddings, POS embeddings, etc. followed by a Position-Wise Feed Forward Network (FFN). The next layer is a context encoding layer, where a bidirectional LSTM (BiLSTM) is used on the top of the lexicon embedding layer to obtain the context representation for both question and passage. In order to reduce the parameters, a maxout layer is applied on the output of BiLSTM. The third layer is the working memory: First we compute an alignment matrix between the question and passage using an attention mechanism, and use this to derive a question-aware passage representation. Then we concatenate this with the context representation of passage and the word embedding, and employ a self attention layer to re-arrange the information gathered. Finally, we use another LSTM to generate a working memory for the passage. At last, the fourth layer is the answer module, which is a GRU recurrent network that outputs predictions at each state s_t.

(Vaswani et al., 2017): use two separate two-layer position-wise Feed-Forward Networks (FFN) to map both the passage and question lexical encodings into the same number of dimensions. It is defined as: $FFN(x) = W_2 ReLU(W_1 + b_1)x + b_2$. Note that this FFN has fewer parameters compared to a BiLSTM. Therefore, we obtain the final lexicon embeddings for the tokens in Q as a matrix $E^q \in \mathbb{R}^{d \times m}$ and words in P as $E^q \in \mathbb{R}^{d \times n}$. The fact that both Q and P have embeddings in with the same number of dimension leads to ways to share parameters in following contextual layers. In our experiments, d is set to 128.

Contextual Encoding Layer. Both passage and question use the same context encoding layer based on BiLSTM. To avoiding over-fitting, we concatenate the output of a pre-trained 600-

dimensional contextualized embedding[1] (McCann et al., 2017), which is trained on English-German machine translation dataset, with the aforementioned lexicon embeddings as the final input of contextual encoding layer. Two stacked BiLSTM layers are used to encode the context information for each word. Since it is a bidirectional layer, it doubles the hidden size. To reduce the parameter size, we use a maxout layer (Goodfellow et al., 2013) on the BiLSTM to shrink its output into its original hidden size. Then, a layer normalization (Ba et al., 2016) is applied. By a concatenation of the outputs of two BiLSTM layers, we obtain $H^q \in \mathbb{R}^{2d\times m}$ as representation of Q and $H^p \in \mathbb{R}^{2d\times n}$ as representation of P, where d is the hidden size of the BiLSTM.

Memory Generation Layer. In this layer, we want to construct our working memory, a summary of information from both Q and P. This is accomplished via an attention mechanism. First, a dot-product attention is adopted like in (Vaswani et al., 2017) to measure the similarity between the tokens in Q and P. Instead of using a scalar to normalize the scores as in (Vaswani et al., 2017), we use a flexible learnable layer with a ReLU nonlinearity to transform the contextual information of both questions and documents:

$$C = dropout(f_{attention}(\hat{H}^q, \hat{H}^p)) \in \mathbb{R}^{m\times n} \quad (1)$$

C is an attention matrix, and dropout is applied for smoothing. Note that $\hat{H}^q$ and $\hat{H}^p$ is transformed from H^q and H^p by one layer neural network $ReLU(W_3 x)$, respectively. Next, we gather all the information on passages by a simple concatenation of its contextual information H^p and its question-aware representation $H^q \cdot C$:

$$U^p = concat(H^p, H^q C) \in \mathbb{R}^{4d\times n} \quad (2)$$

Typically, a passage may contain hundred of tokens, making it hard to learn the long dependencies within it. Inspired by (Lin et al., 2017), we apply a self-attended layer to rearrange the information U^p as:

$$\hat{U}^p = U^p drop_{diag}(f_{attention}(U^p, U^p)). \quad (3)$$

In other words, we first obtain an $n \times n$ attention matrix with U^p onto itself, apply dropout, then multiply this matrix with U^p to obtain an updated $\hat{U}^p$. Instead of using a penalization term as in (Lin et al., 2017), we dropout the diagonal of the similarity matrix forcing each token in the passage to align to other tokens rather than itself.

At last, the working memory is generated by using another BiLSTM based on all the information gathered:

$$M = BiLSTM([U^p; \hat{U}^p]) \quad (4)$$

where the semicolon mark ; indicates the vector/matrix concatenation operator.

Answer module. There is a Chinese proverb that says: "wisdom of masses exceeds that of any individual." Unlike other multi-step reasoning models, which only uses a single output either at the last step or some dynamically determined final step, our answer module employs all the outputs of multiple step reasoning. Intuitively, by applying dropout, it avoids a "step bias problem" (where models places too much emphasis one particular step's predictions) and forces the model to produce good predictions at every individual step. Further, during decoding, we reuse *wisdom of masses* instead of *individual* to achieve a better result. We call this method "stochastic prediction dropout" because dropout is being applied to the final predictive distributions.

Formally, our answer module will compute over T memory steps and output the answer span. This module is a memory network and has some similarities to other multi-step reasoning networks: namely, it maintains a state vector, one state per step. At the beginning, the initial state s_0 is the summary of the Q: $s_0 = \sum_j \alpha_j H_j^q$, where $\alpha_j = \frac{exp(w_4 \cdot H_j^q)}{\sum_{j'} exp(w_4 \cdot H_{j'}^q)}$. At time step t in the range of $\{1, 2, ..., T-1\}$, the state is defined by $s_t = GRU(s_{t-1}, x_t)$. Here, x_t is computed from the previous state s_{t-1} and memory M: $x_t = \sum_j \beta_j M_j$ and $\beta_j = softmax(s_{t-1} W_5 M)$. Finally, a bilinear function is used to find the begin and end point of answer spans at each reasoning step $t \in \{0, 1, \ldots, T-1\}$.

$$P_t^{begin} = softmax(s_t W_6 M) \quad (5)$$

$$P_t^{end} = softmax([s_t; \sum_j P_{t,j}^{begin} M_j] W_7 M) \quad (6)$$

where the mark ; indicates the concatenation operator, and j indicates memory index and it only depends on the current begin output P_t^{begin}.

[1]https://github.com/salesforce/cove

From a pair of begin and end points, the answer string can be extracted from the passage. However, rather than output the results (start/end points) from the final step (which is fixed at $T-1$ as in Memory Networks or dynamically determined as in ReasoNet), we utilize all of the T outputs by averaging the scores:

$$P^{begin} = avg([P_0^{begin}, P_1^{begin}, ..., P_{T-1}^{begin}]) \quad (7)$$

.

$$P^{end} = avg([P_0^{end}, P_1^{end}, ..., P_{T-1}^{end}]) \quad (8)$$

Each P_t^{begin} or P_t^{end} is a multinomial distribution over $\{1, \ldots, n\}$, so the average distribution is straightforward to compute.

During training, we apply stochastic dropout to before the above averaging operation. For example, as illustrated in Figure 1, we randomly delete several steps' predictions in Equations 7 and 8 so that P^{begin} might be $avg([P_1^{begin}, P_3^{begin}])$ and P^{end} might be $avg([P_0^{end}, P_3^{end}, P_4^{end}])$. During decoding, we average all outputs as shown in Equations 7 and 8. The use of averaged predictions and dropout during training improves robustness.

Our stochastic prediction dropout is similar in motivation to the dropout introduced by (Srivastava et al., 2014). The difference is that theirs is dropout at the intermediate node-level, whereas ours is dropout at the final layer-level. Dropout at the node-level prevents correlation between features. Dropout at the final layer level, where randomness is introduced to the averaging of predictions, prevents our model from relying exclusively on a particular step to generate correct output. We used a dropout rate of 0.4 in experiments.

3 Experiment Setup

Dataset: We evaluate on the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016). This contains about 23K passages and 100K questions. The passages come from approximately 500 Wikipedia articles and the questions and answers are obtained by crowdsourcing. The crowdsourced workers are asked to read a passage (a paragraph), come up with questions, then mark the answer span. All results are on the official development set, unless otherwise noted.

Two evaluation metrics are used: Exact Match (EM), which measures the percentage of span predictions that matched any one of the ground truth answer exactly, and Macro-averaged F1 score, which measures the average overlap between the prediction and the ground truth answer. Human performance on the test set is 82.3% EM and 91.2% F1.

Implementation details: The spacy tool[2] is used to tokenize the both passages and questions, and generate lemma, part-of-speech and named entity tags. We use 2-layer BiLSTM with $d = 128$ hidden units for both passage and question encoding. The mini-batch size is set to 32 and Adamax (Kingma and Ba, 2014) is used as our optimizer. The learning rate is set to 0.2 at first and decreased by half after every 10 epochs. We set the dropout rate for the word embeddings, all the hidden units of LSTM, and the answer module output layer to 0.4. To prevent degenerate output, we ensure that at least one step in the answer module is active during training.

4 Results

The main experimental question we would like to answer is whether the stochastic dropout and averaging in the answer module is an effective technique for multi-step reasoning. To do so, we fixed all lower layers and compared different architectures for the answer module:

1. Standard 1-step: generate prediction from s_0, the first initial state.

2. 5-step memory network: this is a memory network fixed at 5 steps. We try two variants: the standard variant outputs result from the final step s_{T-1}. The averaged variant outputs results by averaging across all 5 steps, and is like our proposed SAN without the stochastic dropout.

3. ReasoNet[3]: this answer module dynamically decides the number of steps and outputs results conditioned on the final step.

4. SAN: proposed answer module that uses stochastic dropout and prediction averaging.

The main results in terms of EM and F1 are shown in Table 1. We observe that SAN achieves

[2]https://spacy.io

[3]For a fair comparison about the design of the answer module, this is not an exact re-implementation of (Shen et al., 2017). Here, the answer module is the same as (Shen et al., 2017) but the lower layers are set to be the same as SAN, 5-step memory network, and standard 1-step as described in Figure 2.

Answer Module	EM	F1
Standard 1-step	75.139	83.367
Fixed 5-step with Memory Network (prediction from final step)	75.033	83.327
Fixed 5-step with Memory Network (prediction averaged from all steps)	75.256	83.215
Dynamic steps (max 5) with ReasoNet	75.355	83.360
Stochastic Answer Network (SAN), Fixed 5-step	**76.235**	**84.056**

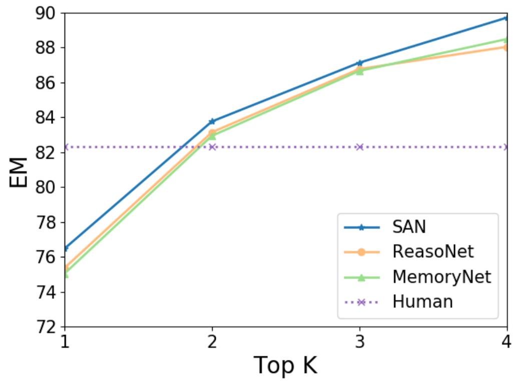
Table 1: **Main results**—Comparison of different answer module architectures. Note that SAN performs best in both Exact Match and F1 metrics.

76.235 EM and 84.056 F1, outperforming all other models. Standard 1-step model only achieve 75.139 EM and dynamic steps (via ReasoNet) achieves only 75.355 EM. We note that SAN also outperforms a 5-step memory net with averaging, which implies averaging predictions is not the only thing that led to SAN's superior results; indeed, stochastic prediction dropout is an effective technique.

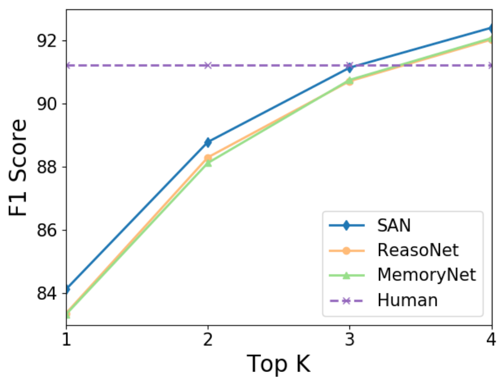
We also show the K-best oracle results in Figure 3. The K-best answer spans are computed by ordering the spans according the their probabilities $P^{begin} \times P^{end}$. We limit K to be 2, 3, or 4 and then pick the span with the best EM or F1 as oracle. SAN also outperforms the other models in terms of K-best oracle scores by a consistent margin. Impressively, these models achieve or surpass human performance at $K = 2$ for EM and $K = 3$ for F1.

Finally, we report the official SQuAD leaderboard results in Table 2.[4]. We see that SAN is very competitive: for the single model case, SAN ranks second in terms both EM and F1 on the test set; for the ensemble model case, SAN ranks third

Note the best-performing model "BiDAF+SelfAttention+ELMo" used a large-scale ELMo (Embeddings from Language Models) (Anonymous, 2017), which is a contextualized language model with two layers of BiLSTM, each with 4096 hidden nodes and 512 dimension projections and trained on a large-scale textual dataset. This resource gave significant improvements, e.g. +4.3% in terms for dev F1; SAN does not currently employ ELMo, and we expect significant improvements when adopting it in future work.



(a) EM comparison on different systems.



(b) F1 score comparison on different systems.

Figure 3: K-Best Oracle results

5 Analysis

To understand our proposed model in more detail, we perform several empirical analyses.

5.1 How robust are the results?

We are interested in whether the proposed model is sensitive to different random initial conditions. Table 3 shows the development set scores of SAN trained from initialization with different random seeds. We observe that SAN results are consis-

[4]For brevity, this is a partial list consisting of only the top systems. The full list is athttps://rajpurkar.github.io/SQuAD-explorer/

[5]As pointed out by Furu Wei (personal communication, Nov. 22, 2017), the r-net result, which is dated Nov. 21, 2017 in the SQuAD leaderboard and is also the one we cited in Table 2, is obtained using a new version of r-net that uses different network structure and attention strategies from what is documented in (Wang et al., 2017), and is not published yet.

Ensemble model results:	Dev Set (EM/F1)	Test Set (EM/F1)
r-net*[5]	-/-	**81.685/87.823**
BiDAF + Self Attention + ELMo*	-/-	81.003/87.432
SAN (Ensemble model)	78.619/85.866	79.608/86.496
Interactive AoA Reader+*	-/-	79.083/86.450
AIR-FusionNet(Huang et al., 2017)	-/-	78.978/86.016
DCN+*	-/-	78.852/85.996
M-Reader(Hu et al., 2017)		77.678/84.888
SLQA*		77.531/84.803
Conductor-net(Liu et al., 2017)	74.8 / 83.3	76.996/84.630
smarnet*		75.989/83.475
ReasoNet++(Shen et al., 2017)	75.4/82.9	75.0/82.6
Individual model results:		
BiDAF + Self Attention + ELMo*	-/-	**78.580/85.833**
SAN (single model)	**76.235/84.056**	76.828/84.396
r-net*[5]	-/-	76.461/84.265
AIR-FusionNet(Huang et al., 2017)	75.3/83.6	75.968/83.900
jNet (Zhang et al., 2017)	-/-	70.6/79.8
Interactive AoA Reader+	-/-	75.821/83.843
RaSoR (Lee et al., 2016)	-/-	75.789/83.261
DCN+*	-/-	75.087/83.081
SLQA_nm*	-/-	74.510/82.604
Ruminate Reader*	-/-	70.6/79.5
ReasoNet++(Shen et al., 2017)	70.8/79.4	70.6/79.36
BiDAF (Seo et al., 2016)	67.7/77.3	68.0/77.3
LR baseline (Rajpurkar et al., 2016)	40.0/51.0	40.4/51.0
Human Performance	80.3/90.5	82.3/91.2

Table 2: Official SQuAD leaderboard performance on December 5, 2017. Asterisk * denotes unpublished works. Results are sorted by Test F1. Note that due to the long list, we only show the top systems.

tently strong regardless of the 10 different initializations. For example, the mean EM score is 76.131 and the lowest EM score is 75.922, both of which still outperform the 75.355 EM of the Dynamic step ReasoNet in Table 1.[6]

We are also interested in how sensitive are the results to the number of reasoning steps, which is a fixed hyper-parameter. Since we are using dropout, a natural question is whether we can extend the number of steps to an extremely large number. Table 4 shows the development set scores for $T = 1$ to $T = 10$. We observe that there is a gradual improvement as we increase $T = 1$ to $T = 5$, but after 5 steps the improvements have saturated. In fact, the EM/F1 scores drop slightly, but considering that the random initialization results in Table 3 show a standard deviation of 0.142 and a spread of 0.426 (for EM), we believe that the $T = 10$ result does not statistically differ from the $T = 5$ result. In summary, we think it is useful to perform some approximate hyper-parameter tuning for the number of steps, but it is not necessary to find the exact optimal value.

5.2 Is it possible to use different numbers of steps in test vs. train?

For practical deployment scenarios, prediction speed at test time is an important criterion. Therefore, one question is whether SAN can train with, e.g. $T = 5$ steps but test with $T = 1$ steps. Table 5 shows the results of a SAN trained on $T = 5$ steps, but tested with different number of steps. As expected, the results are best when T matches during training and test; however, it is important to note that small number of steps $T = 1$ and $T = 2$ nev-

[6]Note the Dev EM/F1 scores of ReasoNet in Table 1 do not match those of ReasoNet++ in Table 2. While the answer module is the same architecture, the lower encoding layers are different.

SAN (5 step)	EM	F1
Seed 1 (official submit)	76.235	84.056
Seed 2	76.301	**84.129**
Seed 3	**75.922**	83.898
Seed 4	75.998	83.952
Seed 5	76.121	83.988
Seed 6	76.225	83.992
Seed 7	**76.348**	84.094
Seed 8	76.074	83.705
Seed 9	75.932	**83.847**
Seed 10	76.149	84.112
Mean	76.131	83.977
Std. deviation	0.142	0.126
Max-Min	0.426	0.424

Table 3: **Robustness on different random seeds for initialization**: best and worst scores are boldfaced.

	EM	F1
SAN (1 step)	**75.383**	**83.294**
SAN (2 step)	75.427	83.413
SAN (3 step)	75.885	83.568
SAN (4 step)	75.922	83.850
SAN (5 step)	**76.235**	**84.056**
SAN (6 step)	75.989	83.717
SAN (7 step)	76.043	83.915
SAN (8 step)	76.026	83.819
SAN (9 step)	75.951	83.752
SAN (10 step)	76.036	83.888

Table 4: **Effect of number of steps**: best and worst results are boldfaced.

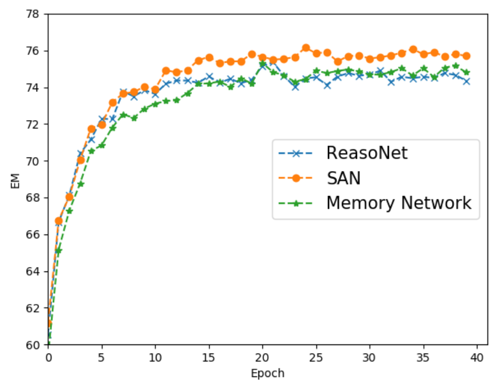
Prediction at	EM	F1
$T = 1$	75.582	83.857
$T = 2$	75.847	83.898
$T = 3$	75.979	83.945
$T = 4$	76.121	83.983
$T = 5$	76.235	84.056
$T = 6$	76.112	84.024
$T = 10$	75.885	83.883
Standard 1-step	75.139	83.367
Dynamic step	75.355	83.360

Table 5: **Prediction on different steps T.** Note that the SAN model is trained using 5 steps.

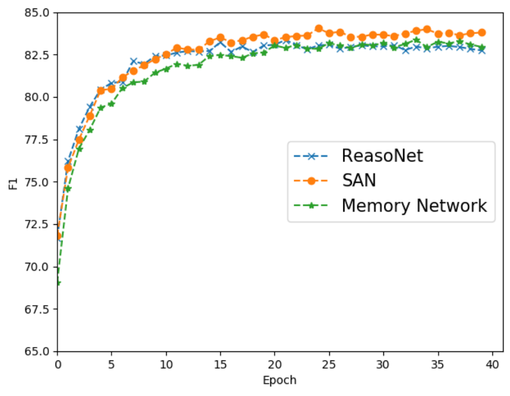
ertheless achieve strong results. For example, prediction at $T = 1$ achieves 75.582, which outperforms a standard 1-step model (75.139 EM) that has approximate equivalent prediction time.

5.3 How does the training time compare?

Does the stochastic dropout lead to longer training times? The average training time per epoch is comparable: our implementation running on a GTX Titan X is 22 minutes for 5-step memory net, 30 minutes for ReasoNet, and 24 minutes for SAN. The learning curve is shown in Figure 4. We observe that all systems improve at approximately the same rate up to 10 or 15 epochs. However, SAN continues to improve afterwards as other models start to saturate. This observation is consistent with previous works using dropout (Srivastava et al., 2014). We believe that while training time per epoch is similar between SAN and other models, it is recommended to train SAN for more epochs in order to achieve gains in EM/F1.



(a) EM



(b) F1

Figure 4: Learning curve measured on Dev set

5.4 How does SAN perform by question type?

We divided the development set by questions type based on their respective Wh-word, such as "who" and "where". We are interested to see whether SAN performs well on a particular type of ques-

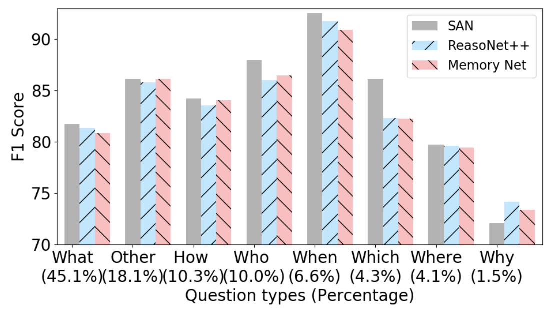


Figure 5: Score breakdown by question/query type.

tion. The score breakdown by F1 is shown in Figure 5. We observe that SAN seems to outperform other models uniformly across all types. The only exception is the Why questions, but there is too little data to derive strong conclusions.

6 Related Work

The recent big progress on MRC is largely due to the availability of the large-scale datasets (Rajpurkar et al., 2016; Nguyen et al., 2016; Richardson et al., 2013; Hill et al., 2016), since it is possible to train large end-to-end neural network models. In spite of the variety of model structures and attenion types (Bahdanau et al., 2015; Chen et al., 2016; Xiong et al., 2016; Seo et al., 2016; Shen et al., 2017; Wang et al., 2017), a typical neural network MRC model first maps the symbolic representation of the documents and questions into a neural space, then search answers on top of it. We categorize these models into two large groups based on the difference of the answer module: single-step and multi-step reasoning. The key difference between the two is what strategies are applied to search the final answers in the neural space.

A single-step model matches the question and document only once and produce the final answers. It is simple yet efficient and can be trained using the classical back-propagation algorithm, thus it is adopted by most of systems (Chen et al., 2016; Seo et al., 2016; Wang et al., 2017; Liu et al., 2017; Chen et al., 2017; Weissenborn et al., 2017; Hu et al., 2017). However, since humans often solve question answering tasks by re-reading and re-digesting the document multiple times before reaching the final answers (this may be based on the complexity of the questions/documents), it is natural to devise an iterative way to find answers as multi-step reasoning.

Pioneered by (Hill et al., 2016; Dhingra et al., 2016; Sordoni et al., 2016; Kumar et al., 2015), who used a predetermined fixed number of reasoning steps, Shen et al (2016; 2017) showed that multi-step reasoning outperforms single-step ones and dynamic multi-step reasoning further outperformed the fixed multi-step ones on two distinct MRC datasets (SQuAD and MS MARCO). But these models have to be trained using reinforcement learning methods, e.g., policy gradient, which are tricky to implement due to the instability issue. Our model is different in that we fix the number of reasoning steps, but perform stochastic dropout to prevent step bias. Further, our model can also be trained by using the back-propagation algorithm, which is simple and yet efficient as single step reasoning.

7 Conclusion

We introduce Stochastic Answer Networks (SAN), a simple yet robust model for machine reading comprehension. The use of stochastic dropout in training and averaging in test at the answer module leads to robust improvements on SQuAD, outperforming both fixed step memory networks and dynamic step ReasoNet. We further empirically analyze the properties of SAN in detail. The model outperforms all the published methods by the time this is published, and achieves results competitive with the state-of-the-art on the SQuAD leaderboard. Due to the strong connection between the proposed model with memory networks and ReasoNet, we would like to delve into the theoretical link between these models and its training algorithms. Further, we also would like to explore SAN on other tasks, such as text classification and natural language inference for its generalization in the future as well.

Acknowledgments

We thank Pengcheng He, Yu Wang and Xinying Song for help to set up dockers. We also thank Pranav Samir Rajpurkar for help on SQuAD evaluations.

References

Anonymous. 2017. Deep contextualized word prepresentations. *International Conference on Learning Representations* https://openreview.net/pdf?id=S1p31z-Ab.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* .

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR2015)* .

Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 2358–2367. `http://www.aclweb.org/anthology/P16-1223`.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Bhuwan Dhingra, Hanxiao Liu, William W Cohen, and Ruslan Salakhutdinov. 2016. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549* .

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. *arXiv preprint arXiv:1302.4389* .

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2016. The goldilocks principle: Reading children's books with explicit memory representations. *ICLR* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Minghao Hu, Yuxing Peng, and Xipeng Qiu. 2017. Mnemonic reader for machine comprehension. *arXiv preprint arXiv:1705.02798* .

Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. 2017. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *arXiv preprint arXiv:1711.07341* .

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *CoRR* abs/1506.07285. `http://arxiv.org/abs/1506.07285`.

Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436* .

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* .

Rui Liu, Wei Wei, Weiguang Mao, and Maria Chikina. 2017. Phase conductor on multi-layered attentions for machine comprehension. *arXiv preprint arXiv:1710.10504* .

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. *arXiv preprint arXiv:1708.00107* .

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* .

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 1532–1543. `http://www.aclweb.org/anthology/D14-1162`.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text pages 2383–2392. `https://aclweb.org/anthology/D16-1264`.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. 2013. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, pages 193–203.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603* .

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284* .

Yelong Shen, Xiaodong Liu, Kevin Duh, and Jianfeng Gao. 2017. An empirical analysis of multiple-turn reasoning strategies in reading comprehension tasks. *arXiv preprint arXiv:1711.03230* .

Alessandro Sordoni, Philip Bachman, Adam Trischler, and Yoshua Bengio. 2016. Iterative alternating neural attention for machine reading. *arXiv preprint arXiv:1606.02245* .

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* .

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 189–198.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. 2017. Fastqa: A simple and efficient neural architecture for question answering. *arXiv preprint arXiv:1703.04816* .

Caiming Xiong, Victor Zhong, and Richard Socher. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604* .

Junbei Zhang, Xiaodan Zhu, Qian Chen, Lirong Dai, and Hui Jiang. 2017. Exploring question understanding and adaptation in neural-network-based question answering. *arXiv preprint arXiv:1703.04617* .