Keeping Notes: Conditional Natural Language Generation with a Scratchpad Mechanism

Ryan Y. Benmalek[†*], **Madian Khabsa**[‡*], **Suma Desu**[¶*], **Claire Cardie**[†], **Michele Banko**[§*]
[†]Cornell University, [‡]Facebook, [¶]Independent , [§]Sentropy Technologies
ryanai3@cs.cornell.edu, mkhabsa@fb.com,
desuma24@gmail.com, mbanko@sentropy.io, cardie@cs.cornell.edu

Abstract

We introduce the *Scratchpad Mechanism*, a novel addition to the sequence-to-sequence (seq2seq) neural network architecture and demonstrate its effectiveness in improving the overall fluency of seq2seq models for natural language generation tasks. By enabling the decoder at each time step to write to all of the encoder output layers, *Scratchpad* can employ the encoder as a "scratchpad" memory to keep track of what has been generated so far and thereby guide future generation. We evaluate *Scratchpad* in the context of three well-studied natural language generation tasks — Machine Translation, Question Generation, and Text Summarization — and obtain state-of-the-art or comparable performance on standard datasets for each task. Qualitative assessments in the form of human judgements (question generation), attention visualization (MT), and sample output (summarization) provide further evidence of the ability of *Scratchpad* to generate fluent and expressive output.

1 Introduction

The sequence-to-sequence neural network framework (seq2seq) (Sutskever et al., 2014) has been successful in a wide range of tasks in natural language processing, from machine translation (Bahdanau et al., 2014) and semantic parsing (Dong and Lapata, 2016) to summarization (Nallapati et al., 2016b; See et al., 2017). Despite this success, seq2seq models are known to often exhibit an overall lack of fluency in the natural language output produced: problems include lexical repetition, under-generation in the form of partial phrases and lack of specificity (often caused by the gap between the input and output vocabularies) (Xie, 2017). Recently, a number of *task-specific attention* variants have been proposed to deal with these issues: See et al. (2017) introduced a coverage mechanism (Tu et al., 2016) to deal with repetition and over-copying in summarization, Hua and Wang (2018) introduced a method of attending over keyphrases to improve argument generation, and Kiddon et al. (2016) introduced a method that attends to an agenda of items to improve recipe generation. Perhaps not surprisingly, general-purpose attention mechanisms targeting individual problems from the list above have also begun to be developed. Copynet (Gu et al., 2016) and pointer-generator networks (Vinyals et al., 2015), for example, aim to reduce input-output vocabulary mismatch and, thereby, improve specificity, while the coverage-based techniques of Tu et al. (2016) tackle repetition and under-generation. These techniques, however, often require significant hyperparameter tuning and are purposely limited to fixing a specific problem in the generated text.

We present here a general-purpose addition to the standard seq2seq framework that aims to simultaneously tackle all of the above issues. In particular, we propose *Scratchpad*, a novel write mechanism that allows the decoder to keep notes on its past actions (i.e., generation, attention, copying) by directly modifying encoder states. The *Scratchpad* mechanism essentially lets the decoder more easily keep track of what the model has focused on and copied from the input in the recent past, as well as what it has produced thus far as output. Thus, *Scratchpad* can alternatively be viewed as an external memory initialized by the input, or as an input re-encoding step that takes into account past attention and generation.

To demonstrate general(izable) improvements on conditional natural language generation problems broadly construed, we instantiate *Scratchpad* for three well-studied generation tasks — Machine Translation, Question Generation, and Summarization — and evaluate it on a diverse set of datasets. These tasks exhibit a variety of input modalities (structured and unstructured

[*] Work performed while at Apple.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4157–4167
Florence, Italy, July 28 - August 2, 2019.

language) and typically have required a variety of computational strategies to perform well (attention, pointing, copying). We find, for each task, that *Scratchpad* attains improvements over several strong baselines: Sequence-to-Sequence with attention (Sutskever et al., 2014; Bahdanau et al., 2014), copy-enhanced approaches (Gu et al., 2016; Vinyals et al., 2015), and coverage-enhanced approaches (Tu et al., 2016; See et al., 2017). *Scratchpad* furthermore obtains state-of-the-art performance for each task. Qualitative assessments in the form of human judgements (question generation), attention visualization (MT) and sample output (summarization) provide further evidence of the ability of *Scratchpad* to generate fluent and expressive output.

2 Background

Scratchpad builds upon a standard attention-based seq2seq neural architecture (Bahdanau et al., 2014) comprised of (a) an *encoder* that operates token-by-token over the *input*, (b) a *decoder* that produces the *output*, and (c) an *attention* mechanism that allows the decoder to focus on different parts of the input. In the subsections below, we first briefly review this architecture (we assume the reader is familiar with the framework). In Section 3, we introduce the *Scratchpad* mechanism.

Encoder Let $X = [x_1, ..., x_n]$ denote an input sequence of length N where x_i is the i-th token. The encoder is a recurrent neural network (RNN) that produces in its final layer a sequence of hidden states $[\mathbf{h}_1, ..., \mathbf{h}_n] = \text{RNN}(\{\mathbf{x_1}, ..., \mathbf{x_n}\})$. These can be viewed as a sequence of token-level feature vectors learned from the input.

Decoder Let the decoding sequence be indexed by the superscript i. The decoder is an RNN whose initial hidden state $\mathbf{s}^0$ is set to the final state(s) of the of the encoder.

Attention At every decoding step i, an attention mechanism, i.e., an attentive read (often termed *attentional context*) ($\mathbf{c}^i$), is derived from the encoder output states ($[\mathbf{h}_1, ..., \mathbf{h}_n]$). Concretely, attention is applied by first computing a *score* for each encoder output, $\mathbf{h}_t$:

$$\text{score}_t^i = \mathbf{W_1}(\mathbf{W_2}[\mathbf{s}^i, \mathbf{h}_t]^\mathbf{T}) \quad (1)$$

where weight matrices $\mathbf{W_1}$ and $\mathbf{W_2}$ are learned parameters. These scores, $\text{score}_{1..T}^i$, are then normalized into a probability distribution:

$$\mathbf{a}^i = \text{softmax}(\text{score}_{1..T}^i) \quad (2)$$

The *attentive read* operation is then the weighted average of encoder outputs according to this distribution, which allows the decoder to focus on different parts of the input at different timesteps i:

$$\mathbf{c}^i = \sum_{t=1}^{T} (\mathbf{a}_t^i * \mathbf{h}_t) \quad (3)$$

3 Scratchpad Mechanism

The above attention mechanism has been widely successful in many generation tasks but the quality of generated text still suffers from caveats and requires significant tuning. We augment attention with a *Scratchpad* mechanism to introduce higher quality generated text with less effort. Intuitively, *Scratchpad* adds one simple step to the decoder: treating the encoder output states, $[\mathbf{h}_1, ..., \mathbf{h}_n]$, as a *scratchpad*, thus it writes to them as if the set of states were an external memory. Exactly how this is done is described next.

Without *Scratchpad*, the decoder's workflow at every output timestep step i is as follows:
1. ***Read*** attentively ($\mathbf{c}^i$) from the encoder outputs ($[\mathbf{h}_1, ..., \mathbf{h}_n]$) using the current state, $\mathbf{s}^i$.
2. ***Update*** $\mathbf{s}^i$ using the most recently generated output token, y^{i-1}, and the results of the attentive read ($\mathbf{c}^i$).
3. ***Output*** a distribution over the output vocabulary $\hat{\mathbf{y}}^i$.

Scratchpad simply adds a fourth step:
4. ***Write*** an update ($\mathbf{u}^i$) to the encoder outputs ($[\mathbf{h}_1, ..., \mathbf{h}_n]$) in an attentive fashion ($\alpha_{1..T}^i$), treating the encoder outputs as if they were cells in an external memory.

More specifically, to calculate both the write-attention and the update, *Scratchpad* uses the concatenation of the decoder state after steps 1-3 ($\mathbf{s}^{i+1}$) and the attentive read ($\mathbf{c}^i$):

$$\mathbf{h}_t^{i+1} = \alpha_t^i \mathbf{h}_t^i + (1 - \alpha_t^i)\mathbf{u}^i \quad (4)$$

$$\alpha_t^i = \sigma(f_\alpha([\mathbf{s}^{i+1}, \mathbf{c}^i, \mathbf{h}_t^i])) \quad (5)$$

$$\mathbf{u}^i = \tanh(f_u([\mathbf{s}^{i+1}; \mathbf{c}^i])) \quad (6)$$

In essence, the *Scratchpad* consists of two components. The first determines what 'notes' to keep

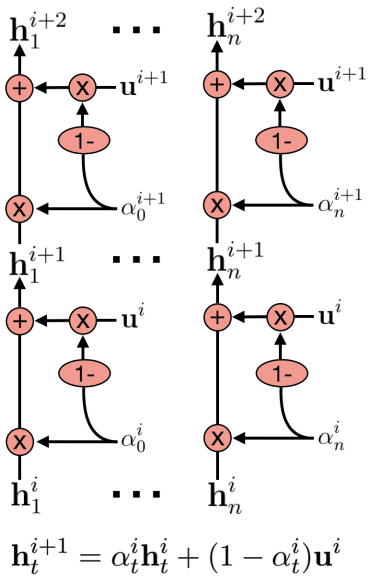

Figure 1: The *Scratchpad Mechanism* first computes the update probability (α_t^i) for each encoder state according to Eq. 5, then computes a global update $\mathbf{u}^i$ according to Eq. 6, and finally updates the encoder states according to Eq. 4.

($\mathbf{u}^i$). The second is similar to the 'forget' gate in an LSTM, where the network decides how much to overwrite a cell ($1 - \alpha_t^i$) versus how much to keep past information (α_t^i) for that cell. These two components are used in concert (see Fig. 1) to provide new encoder states (h_t^{i+1}) to the decoder at each decoding timestep (i). Tanh is used to ensure that $\mathbf{h}_t^{i+1}$ remains in the range $[-1, 1]$, since $\mathbf{h}_t^i \in [-1, 1]$ as $[\mathbf{h}_1, ..., \mathbf{h}_n]^0$ is the hidden states of a GRU or LSTM. We parametrize f_α and f_u as an MLP. Figure 1 shows the outline of the scratchpad mechanism update at multiple timesteps.

4 Experiments

In this section we describe experimental setup and results for Machine Translation, Question Generation, and Summarization tasks which exhibit a variety of input modalities and strategies required to perform well. We work with structured and unstructured language and several sequence to sequences architectures i.e. attention, pointing, and copying. Machine translation is a canonical sequence-to-sequence task where pairwise word-level or phrase-level generation is expected. Question Generation from logical forms requires reasoning about the syntax, parse tree, and vocabulary of the input sequence to infer the meaning of the logical form program and utilize copy-mechanism to copy entities. Lastly, summarization requires understanding both the global and the local context of a sentence within a document, identifying spans that are informative and diverse, and generating coherent representative summaries. Demonstrating a single mechanism that reaches state of the art on such a diverse set of natural language tasks underscores the generalizability of our technique, particularly given the large range in number of training examples (3k, 56k, 153k, 287k) across datasets.

4.1 Translation

We evaluate on the IWLST 14 English to German and Spanish to English translation datasets (Cettolo et al., 2015) as well as the IWSLT 15 (Cettolo et al., 2015) English to Vietnamese translation dataset. For IWSLT14 (Cettolo et al., 2015), we compare to the models evaluated by He et al. (2018), which includes a transformer (Vaswani et al., 2017) and RNN-based models (Bahdanau et al., 2014). For IWSLT15, we primarily compare to GNMT (Wu et al., 2016), which incorporates Coverage (Tu et al., 2016). Table 1 shows BLEU scores of our approach on 3 IWSLT translation tasks along with reported results from previous work. Our approach achieves state-of-the-art or comparable results on all datasets.

Model	IWSLT14		IWSLT15
	De→En	Es →En	En→Vi
MIXER	21.83	✗	✗
AC + LL	28.53	✗	✗
NPMT	29.96	✗	28.07
Stanford NMT	✗	✗	26.1
Transformer (6 layer)	32.86	38.57	✗
Layer-Coord (14 layer)	35.07	40.50	✗
Scratchpad (3 layer)	**35.08**	**40.92**	**29.59***

Table 1: Performance for non-scratchpad models are taken from He et al. (2018) except Stanford NMT (Luong and Manning, 2015). $*$: model is 2 layers.

Experimental Details For IWSLT14, our encoder is a 3-layer Bi-LSTM (Hochreiter and Schmidhuber, 1997), where outputs are combined by concatenation, and the decoder is a 3-layer LSTM as well. For IWSLT15 the encoder and decoder are 2-layers. We follow Luong et al. (2015), using the 'general' score function, input feeding, and combining the attentional context and hidden state. Since we use input feeding, Steps (1) and (2) in Section 3 are switched. All our models have a

	Model	Per-Sentence			Corpus-Level		
		Bleu	Meteor	Rouge-L	Bleu	Meteor	Rouge-L
WebQSP	Baseline	7.51	23.9	47.1	17.96	22.9	47.13
	Copynet	6.89	27.1	52.5	17.42	26.03	52.56
	Copy + Coverage	14.55	33.7	58.9	26.78	30.86	58.91
	Copy + Scratchpad	**15.29**	**34.7**	**59.5**	**27.64**	**31.49**	**59.44**
WikiSQL	Baseline	9.94	26.71	47.96	17.34	25.34	47.96
	Copynet	8.04	24.66	46.82	15.11	23.53	46.82
	Copy + Coverage	15.76	34.04	54.94	25.01	32.38	54.94
	Copy + Scratchpad	**16.89**	**34.47**	**55.69**	**26.10**	**32.76**	**55.69**

Table 2: Methods allowing the model to keep track of past attention (*Coverage*, *Scratchpad*) significantly improve performance when combined with a copy mechanism. The *Scratchpad Encoder* achieves the best performance.

hidden size of 512 (for the LSTM and any MLP's). The internal layers of the decoder are residual, adding their output to their input and putting it through Layer Normalization (Ba et al., 2016). Sentences were encoded using byte-pair encoding (Sennrich et al., 2016), with a shared source-target vocabulary of $10,000$ for De→En and Es→En (En →Vi uses words as tokens to be comparable to Wu et al. (2016)). Source and target word embeddings are dimension 128. We use dropout (Srivastava et al., 2014) in the encoder and decoder with a probability of 0.1. We use the Adam optimizer (Kingma and Ba, 2014), with an initial learning rate of 0.002.We train for 30/20 epochs for IWSLT14/15, decaying the learning rate by a factor of 0.7 whenever the validation loss does not improve from the last epoch. Each training batch contained at most 2000 source or target tokens. We use label smoothing with $\epsilon_{ls} = 0.1$ (Szegedy et al., 2016). We average the last 5 epochs to obtain the final model and run with a beam of size 4.

4.2 Question Generation

We use the task of *question generation*: Given a *structured representation* of a query against a knowledge base or a database (e.g. a *logical form*), produce the corresponding natural language question. We use two datasets consisting of *(question, logical form)* pairs: WebQuestionsSP (Yih et al., 2016) (a standard dataset for semantic parsing, where the logical form is in SPARQL), and WikiSQL (Zhong et al., 2017) (where the logical form is SQL). Both datasets are small, with the former having 3098 training and 1639 testing examples, and the latter being an order of magnitude larger with 56346 training and 15873 testing examples.

We evaluate metrics at both a *corpus* level (to indicate how *natural* output questions are) and at a *per-sentence* level (to demonstrate how well output questions exactly *match* the gold question). BLEU (Papineni et al., 2002), ROUGE (Lin, 2004) are chosen for precision and recall-based metrics. METEOR (Banerjee and Lavie, 2005) is chosen to deal with stemming and synonyms.

We noticed that many tokens that appear in the logical form are also present in the natural language form for each example. In fact, nearly half of the tokens in the question appear in the corresponding SPARQL of the WebQuestionSP dataset (Yih et al., 2016), implying that a network with the ability to copy from the input could see significant gains on the task. Accordingly, we compare our *Scratchpad Mechanism* against three baselines: (1) Seq2Seq, (2) *Copynet* and (3) *Coverage*, a method introduced by Tu et al. (2016) that aims to solve attention-related problems. Seq2Seq is the standard approach introduced in Sutskever et al. (2014). The *Copynet* (He et al., 2017) baseline additionally gives the Seq2Seq model the ability to copy vocabulary from the source to the target.

From Table 2 it is clear that our approach, *Scratchpad* outperforms all baselines on all the metrics.

Experimental Details Our encoder is a 2-layer bi-directional GRU where outputs are combined by concatenation, and our decoder is a 2-layer

Model	Rouge			Meteor	
	1	2	L	exact match	+stem/syn/para
Pointer-generator	36.44	15.66	33.42	15.35	16.65
See et al. (2017)	39.53	17.28	36.38	**17.32**	**18.72**
Scratchpad	**39.65**	**17.61**	**36.62**	17.26	18.63
CopyTransformer + Coverage Penalty	39.25	17.54	36.45	✗	✗
Pointer-Generator + Mask Only	37.70	15.63	35.49	✗	✗
Bottom-up (Gehrmann et al., 2018)	**41.22**	**18.68**	**38.34**	✗	✗

Table 3: The middle third of the table contains the end-to-end models performing the best from (Gehrmann et al., 2018), while the bottom section contains the current state-of-the-art which involves a 2-step training process and is not end-to-end. Scratchpad establishes a state of the art for end-to-end models on summarization without Reinforcement Learning on ROUGE, while remaining competitive with See et al. (2017) on METEOR. Additionally, Scratchpad does not use an auxiliary loss as in See et al. (2017) or the middle third of the table. Gehrmann et al. (2018) do not evaluate on METEOR.

GRU. We use the attention mechanism from 4.1. We train all models for 75 epochs with a batch size of 32, a hidden size of 512 (for the GRU and any MLP's), and a word vector size of 300. Dropout is used on every layer except the output layer, with a drop probability of 0.5. Where Glove vectors (Pennington et al., 2014) are used to initialize word vectors, we use 300-dimensional vectors trained on Wikipedia and Gigaword ($6B.300D$). We use the Adam optimizer with a learning rate of $1e^{-4}$ and we do teacher forcing (Williams and Zipser, 1989) with probability 0.5. These hyperparameters were tuned for our Seq2Seq baselines and held constant for the rest of the models. The vocabulary consists of all tokens appearing at least once in the training set.

4.3 Summarization

We use the CNN/Daily Mail dataset (Hermann et al., 2015; Nallapati et al., 2016b) as in See et al. (2017). The dataset consists of 287,226 training, 13,368 validation, and 11,490 test examples. Each example is an online news article (781 tokens on average) along with a multi-sentence summary (56 tokens, 3.75 sentences on average). As in See et al. (2017) we operate on the original non-anonymized version of the data.

We follow See et al. (2017) in evaluating with ROUGE (Lin, 2004) and METEOR (Banerjee and Lavie, 2005). We report F_1 scores for ROUGE-1, ROUGE-2, and ROUGE-LCS (measuring word, bigram, and longest-common-subsequence overlap, respectively) and we report METEOR in exact and full mode. We compare to the pointer-generator baseline and the coverage variant introduced by See et al. (2017). See et al. (2017) use a multi-step training procedure for the coverage component to improve performance where a pointer-generator model is first trained without the coverage component for a large number of iterations, then trained with the component and a tuned auxiliary coverage loss, finding that the auxiliary loss and pre-training the network without coverage are required to improve performance. As demonstrated in Tab. 3, with *Scratchpad*, we are able to improve performance over See et al. (2017) in all the Rouge metrics, statistically significant for Rouge-2 and Rouge-L, while remaining comparable in METEOR. We reach this performance with half of the training iterations, no pretraining, and without the additional memory outlay and model complexity of including an auxiliary loss.

Experimental Details We use the same setup as in See et al. (2017): The encoder is a single-layer bi-directional LSTM where outputs are combined by concatenation, and the decoder consists of a single-layer LSTM. The encoder states modified by the *scratchpad mechanism* are the outputs of the LSTM at every timestep, i.e. the 'hidden' state. We use the same attention mechanism as in See et al. (2017) to calculate the attentive read and the attentive write probabilities α_t^i for the *scratchpad mechanism*. See et al. (2017) introduce a multi-step training procedure where a pointer-generator model is first trained with the vanilla cross-entropy objective for 230k iterations. Then the coverage component is added and the full model is further

trained for 3k iterations with the combined cross-entropy coverage loss. See et al. (2017) use Adagrad (Duchi et al., 2010) with learning rate 0.15 and an initial accumulator value of 0.1. Early stopping on validation is used to select the final model.

We adopt a simpler procedure, training our full model with the *scratchpad mechanism* for 100k iterations with Adam and a learning rate of $1e^{-4}$ as compared to the two-step procedure in See et al. (2017) taking 230k iterations. We follow See et al. (2017) in using a batch size of 16 and clipping gradient norms to 2.

5 Analysis

To gain insight into the behaviour and performance of our *Scratchpad Mechanism*, we analyze the output for Question Generation and Translation. We start by conducting a human evaluation study on the Question Generation task, since this task is relatively new and it is well known that quantitative metrics like BLEU do not always correlate with human assessed quality of generated text[1]. Later we use the attention heatmaps to visualize how the *scratchpad mechanism* drives the attention weights to be more focused on the relevant source token(s). Additionally, we analize the *entropies* of the attention weights to understand how the *scratchpad mechanism* better allows models to attend to the input. We hypothesize that this is one of the reasons that lead to good performance of the *scratchpad mechanism* as the decoder ends up being more focused than with the standard seq2seq models.

5.1 Human Evaluations

For our human evaluation we use two standard metrics from the machine translation community: *Adequacy* and *Fluency* (Bojar et al., 2017). To compute *Adequacy*, human judges are presented with a reference output and the system proposed output, and are asked to rate the adequacy of the proposed output in conveying the meaning of the reference output on a scale from 0-10. To compute *Fluency*, the judges are asked to rate, on a scale from 0-10, whether the proposed output is a fluent English sentence. We used crowd-sourced judges. Each output is rated by 3 judges.

[1]The relation between BLEU scores and more canonical tasks such as machine translation and summarization have already been studied in the literature.(Bojar et al., 2017; Graham, 2015)

Table 4 summarizes the human evaluation results for our *Scratchpad Mechanism* and two more baselines. As the table shows, the judges assigned higher fluency and adequacy scores to our approach than both the coverage-based and copynet baseline. In the table we also report the fluency score of the gold questions as a way to measure the gap between the generated questions and the expected ones. Our approach is only 2 points behind the gold when it comes to generation fluency.

Model	Fluency	Adequacy
Gold	9.13	✗
Copynet	5.18	5.23
+ Coverage	6.64	6.16
+ Scratchpad	**7.38**	**6.59**

Table 4: Human evaluations show that the *Scratchpad* delivers a large improvement in both *fluency* and *adequacy* over *Copynet* and *Coverage*, accentuating the improvement in quantitative metrics (Bleu, Rouge, Meteor).

Scratchpad vs. Copynet		Scratchpad vs. Coverage	
Both Good	9.26%	Both Good	15.11%
Scratchpad	37.78%	Scratchpad	23.80%
Copynet	6.46%	Coverage	14.99%
Both Bad	46.5%	Both Bad	43.07%
Win Rate	89.44%	Win Rate	61.36%

Table 5: The percentage of times judges preferred one result over the other. In a Head-to-Head evaluation the output of *Scratchpad Encoder* is 9 and 2 times as likely to be chosen vs. *Copynet* and *Coverage*, respectively. Win rate is the percentage of times Scratchpad was picked when the judges chose a single winner (not a tie).

Additionally, we design a side-by-side experiment where judges are presented with 2 generated questions from 2 different systems along with the *reference* and asked to judge which output presents a better *paraphrase* to the reference question. Judges take into consideration the grammatical correctness of the question as well as its ability to capture the meaning of the reference question fluently. In Table 5 We show that in head-to-head evaluations, human judges are nine times as likely to prefer *scratchpad* generated questions over copynet and nearly two times over coverage, accentuating the improved *fluency* and *adequacy*

of *scratchpad* generated questions.

5.2 Attention Visualization and Analysis

In the standard attention setup, the weights assigned to each encoder output is determined by the decoder internal state and the encoder output (s^i and h_t) in Equation 1. Throughout the decoding steps, only s^i varies from timestep to the next. Our *scratchpad mechanism* allows the encoder outputs to change in order to keep track of generated output, so that both s^i and h_t will vary from timestep to the next, hence more focused attention can be generated.

We demonstrate this behavior in Fig 5 where two sentences in a German to English machine translation system are shown. In the top figure, attention weights are shown when the *scratchpad mechanism* is utilized, while in the bottom Figure standard attention is used. As can be seen from the figures, attention weights are more focused especially in the first few steps of decoding that better aligns with word-level translations (e.g. 'hand' is properly attended to with scratchpad, but not with non-scratchpad). Additionally, some words that are never properly translated (e.g. wahrscheinlich - 'probably') by the non-scratchpad model are not heavily attended to, whereas with the scratchpad mechanism, they are.

We also demonstrate this effect quantitatively. Recall the attention distribution $\mathbf{a}_t^i$ over the input $[x_1, ..., x_n]$ generated at each decoding timestep i. By calculating the entropy $\text{ent}^i = -\sum_t \mathbf{a}_t^i * \log(\mathbf{a}_t^i)$ and taking the mean of this value across a set of output sentences we can measure how well the model "focuses" on input sequences (e.g. $[x_1, ..., x_n]$) as it decodes. The *lower* the entropy, the *sharper* the attention distribution. We evaluate this metric on the IWSLT14 De→En test set for the *scratchpad* and non-scratchpad models. By adding the *scratchpad mechanism*, the mean entropy decreases substantially from 1.33 to 0.887 - indicating that it makes the model more selective (focusing on *fewer* input tokens with *higher* weights during generation). Additionally, we plot in Fig. 2 the cumulative frequency of the word-level entropies ent^i for all output timesteps i. Note from the graph that for every value x, the *scratchpad* model produces more attention distributions with an entropy $\leq x$. Finally, the shape of the curve changes to be less sigmoidal, with the proportion of particularly *peaky* or *focused* distributions (very low entropy, e.g. ≤ 0.5) increasing significantly, over $4\times$ that for the non-scratchpad model.

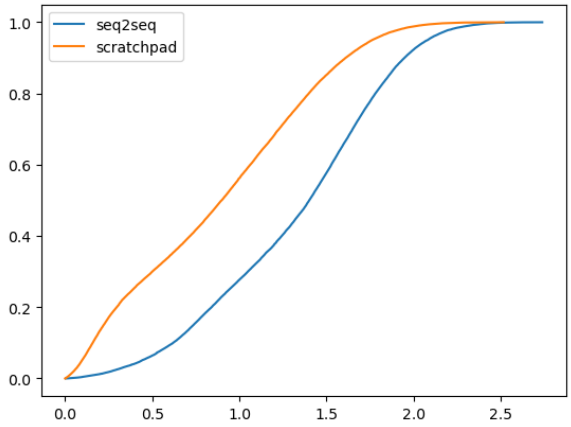

Figure 2: We plot the cumulative frequency of attention distribution entropies. On the Y-axis is the proportion of attention distribution entropies lower than or equal to x.

Previous work based on coverage based approaches (Tu et al., 2016; See et al., 2017) either imposed an extra term to the loss function or used an extra vector to keep track of which parts of the input sequences had been attended to, thereby focusing the attention weights in subsequent steps on tokens that received little attention before. In other words, focusing the attention on the relevant parts of the input. Our proposed approach naturally learns to focus the attention on the important tokens, without a need for modifying the loss function or introducing coverage vectors.

6 Related work

Machine Translation Since Sutskever et al. (2014) introduced the sequence-to-sequence paradigm the approach became the defacto standard for performing machine translation. Improvements over the approach followed, first by the introduction of attention (Bahdanau et al., 2014) which helped seq2seq translation to focus on certain tokens of the encoder outputs. Later on, many improvements were described in the Google neural machine translation system (Wu et al., 2016), including utilizing coverage penalty (Tu et al., 2016) while decoding. The Transformer model was introduced to alleviate the dependence on RNNs in both the encoder and the decoder steps (Vaswani et al., 2017). Our proposed model sits on top of the seq2seq framework, and could

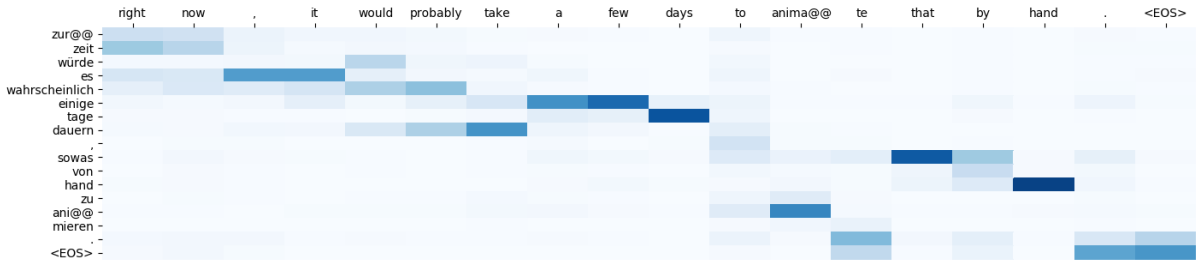

Figure 3: Scratchpad

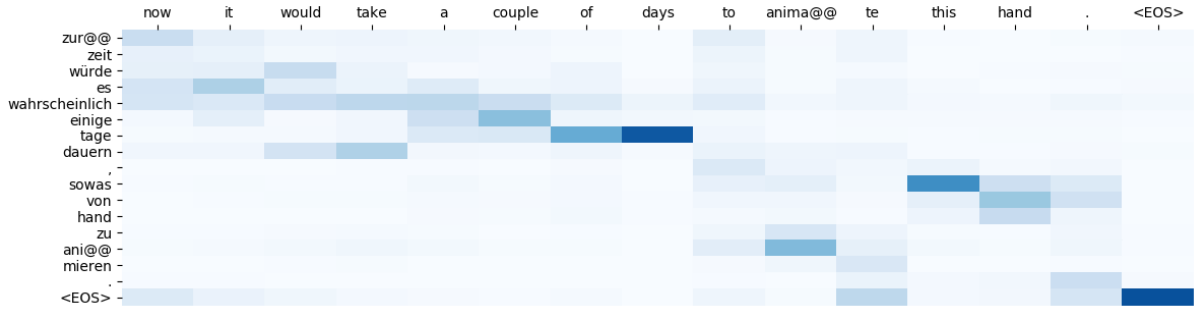

Figure 4: No Scratchpad

Figure 5: Attention over the source sequence visualized at each decoder timestep with and without the scratchpad mechanism. Darker cells mean higher values. "@@" at the end of a bpe token denotes it should be concatenated with the following token(s) to make a word. With Scratchpad, we see sharper attention earlier in the sentence that better aligns with word-level translations (e.g. 'hand' is properly attended to with scratchpad, but not with non-scratchpad). Additionally, some words that are never properly translated (e.g. wahrscheinlich - 'probably') by the non-scratchpad model are not heavily attended to, whereas with the scratchpad mechanism, they are.

be used with any choice of encoder/decoder as long as attention is used.

Summarization Since Rush et al. (2015) first applied neural networks to abstractive text summarization, work has focused on augmenting models (Chopra et al., 2016; Nallapati et al., 2016b; Gu et al., 2016), incorporating syntactic and semantic information (Takase et al., 2016), or direct optimization of the metric at hand (Ranzato et al., 2016). Nallapati et al. (2016b) adapted the DeepMind question-answering dataset (Hermann et al., 2015) for summarization and provided the first abstractive and extractive (Nallapati et al., 2016a) models. See et al. (2017) demonstrated that pointer-generator networks can significantly improve the quality of generated summaries. Additionally, work has explored using Reinforcement Learning, often with additional losses or objective functions to improve performance (Hsu et al., 2018; Paulus et al., 2018; Li et al., 2018; elikyilmaz et al., 2018; Pasunuru and Bansal, 2018). Finally, Gehrmann et al. (2018) demonstrated that a two-stage procedure, where a model first identifies spans in the article that could be copied into the summary which are used to restrict a second pointer-generator model can reap significant gains.

Question Generation Early work on translating SPARQL queries into natural language relied heavily on hand-crafted rules (Ngonga Ngomo et al., 2013a,b) or manually crafted templates to map selected categories of SPARQL queries to questions (Trivedi et al., 2017; Seyler et al., 2017). In (Serban et al., 2016) knowledge base triplets are used to generate questions using encoder-decoder framework that operates on entity and predicate embeddings trained using TransE (Bordes et al., 2011). Later, Elsahar et al. (2018) extended this approach to support unseen predicates. Both approaches operate on triplets, meaning they have limited capability beyond generating simple questions and cannot generate the far more complex compositional questions that our approach does by operating on the more expressive SPARQL query (logical form). In the question generation domain,

there has been a recent surge in research on generating questions for a given paragraph of text (Song et al., 2017; Du et al., 2017; Tang et al., 2017; Duan et al., 2017; Wang et al., 2018; Yao et al., 2018), with most of the work being a variant of the seq2seq approach. In Song et al. (2017), a seq2seq model with copynet and a coverage mechanism (Tu et al., 2016) is used to achieve state-of-the-art results. We have demonstrated that our *Scratchpad* outperforms this approach in both quantitative and qualitative evaluations.

Attention Closest to our work, in the general paradigm of seq2seq learning, is the coverage mechanism introduced in Tu et al. (2016) and later adapted for summarization in See et al. (2017). Both works try to minimize erroneous repetitions generated by a copy mechanism by introducing a new vector to keep track of what has been used from the encoder thus far. Tu et al. (2016), for example, use an extra GRU to keep track of this information, whereas See et al. (2017) keep track of the sum of attention weights and add a penalty to the loss function based on it to discourage repetition. Our approach is much simpler than either solution since it does not require any extra vectors or an additional loss term; rather, the encoder vector itself is being used as *scratch memory*. Our experiments also show that for the question generation task, the Scratchpad performs better than coverage based approaches.

Our idea was influenced by the dialogue generation work of Eric and Manning (2017) in which the entire sequence of interactions is re-encoded every time a response is generated by the decoder.

7 Conclusion

In this paper, we introduce the Mechanism, a novel write operator, to the sequence to sequence framework aimed at addressing many of the common issues encountered by sequence to sequence models and evaluate it on a variety of standard conditional natural language generation tasks. By letting the decoder 'keep notes' on the encoder, or said another way, re-encode the input at every decoding step, the Scratchpad Mechanism effectively guides future generation. The Scratchpad Mechanism attains state of the art in Machine Translation, Question Generation, and Summarization on standard metrics and human evaluation across multiple datasets. In addition, our approach decreases training time and model complexity compared to other leading approaches. Our success on such a diverse set of tasks, input data, and volumes of training data underscores the generalizability of our approach and its conceptual simplicity make it easy to add to any sequence to sequence model with attention.

Acknowledgments

The work was begun while the first author was interning at Apple Siri. We thank Christopher Ré for his helpful comments and feedback on the paper. We thank the reviewers for insightful comments and suggesting mean entropy to measure the distribution of attention weights.

References

Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, et al. 2017. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214.

Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *AAAI*, volume 6, page 6.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2015. Report on the 11 th iwslt evaluation campaign , iwslt 2014.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *HLT-NAACL*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*.

Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1342–1352.

Nan Duan, Duyu Tang, Peng Chen, and Ming Zhou. 2017. Question generation for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 866–874.

John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Hady Elsahar, Christophe Gravier, and Frederique Laforest. 2018. Zero-shot question generation from knowledge graphs for unseen predicates and entity types. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 218–228.

Mihail Eric and Christopher Manning. 2017. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 468–473.

Sebastian Gehrmann, Yuntian Deng, and Alexander M. Rush. 2018. Bottom-up abstractive summarization. In *EMNLP*.

Yvette Graham. 2015. Re-evaluating automatic summarization with bleu and 192 shades of rouge. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 128–137.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *CoRR*, abs/1603.06393.

Shizhu He, Cao Liu, Kang Liu, and Jun Zhao. 2017. Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. In *ACL*.

Tianyu He, Xu Tan, Tao Qin, and Zhibo Chen. 2018. Layer-wise coordination between encoder and decoder for neural machine translation. In *NeurIPS*.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Wan Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. 2018. A unified model for extractive and abstractive summarization using inconsistency loss. In *ACL*.

Xinyu Hua and Lu Wang. 2018. Neural argument generation augmented with externally retrieved evidence. In *ACL*.

Chloé Kiddon, Luke S. Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *EMNLP*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Piji Li, Lidong Bing, and Wai Lam. 2018. Actor-critic based training framework for abstractive summarization. *CoRR*, abs/1803.11070.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries.

Minh-Thang Luong and Christopher D. Manning. 2015. Neural machine translation systems for spoken language domains.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2016a. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *AAAI*.

Ramesh Nallapati, Bowen Zhou, Cícero Nogueira dos Santos, aglar Gülehre, and Bing Xiang. 2016b. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *CoNLL*.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013a. Sorry, i don't speak sparql: translating sparql queries into natural language. In *Proceedings of the 22nd international conference on World Wide Web*, pages 977–988. ACM.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013b. Sparql2nl: verbalizing sparql queries. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 329–332. ACM.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Ramakanth Pasunuru and Mohit Bansal. 2018. Multi-reward reinforced summarization with saliency and entailment. In *NAACL-HLT*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *EMNLP*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1073–1083.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 588–598.

Dominic Seyler, Mohamed Yahya, and Klaus Berberich. 2017. Knowledge questions from knowledge graphs. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 11–18. ACM.

Linfeng Song, Zhiguo Wang, and Wael Hamza. 2017. A unified query-based generative model for question generation and question answering. *arXiv preprint arXiv:1709.01058*.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Sho Takase, Jun Suzuki, Naoaki Okazaki, Tsutomu Hirao, and Masaaki Nagata. 2016. Neural headline generation on abstract meaning representation. In *EMNLP*.

Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou. 2017. Question answering and question generation as dual tasks. *arXiv preprint arXiv:1706.02027*.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218. Springer.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *ACL*.

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *NIPS*.

Yansen Wang, Chenyi Liu, Minlie Huang, and Liqiang Nie. 2018. Learning to ask questions in open-domain conversational systems with typed decoders. In *ACL*.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Ziang Xie. 2017. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534*.

Kaichun Yao, Libo Zhang, Tiejian Luo, Lili Tao, and Yanjun Wu. 2018. Teaching machines to ask questions. In *IJCAI*, pages 4546–4552.

Scott Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Asli elikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. 2018. Deep communicating agents for abstractive summarization. In *NAACL-HLT*.