Neural Transformation Machine: A New Architecture for Sequence-to-Sequence Learning

Fandong Meng\textsuperscript{1*} Zhengdong Lu\textsuperscript{2} Zhaopeng Tu\textsuperscript{2} Hang Li\textsuperscript{2} and Qun Liu\textsuperscript{1}
\textsuperscript{1}Institute of Computing Technology, Chinese Academy of Sciences
\{mengfandong, liuqun\}@ict.ac.cn
\textsuperscript{2}Noah’s Ark Lab, Huawei Technologies
\{Lu.Zhengdong, Tu.Zhaopeng, HangLi.HL\}@huawei.com

Abstract
We propose Neural Transformation Machine (NTram), a novel architecture for sequence-to-sequence learning, which performs the task through a series of nonlinear transformations from the representation of the input sequence (e.g., a Chinese sentence) to the final output sequence (e.g., translation to English). Inspired by the recent Neural Turing Machines [8], we store the intermediate representations in stacked layers of memories, and use read-write operations on the memories to realize the nonlinear transformations of those representations. Those transformations are designed in advance but the parameters are learned from data. Through layer-by-layer transformations, NTram can model complicated relations necessary for applications such as machine translation between distant languages. The architecture can be trained with normal back-propagation on parallel texts, and the learning can be easily scaled up to a large corpus. NTram is broad enough to subsume the state-of-the-art neural translation model in [2] as its special case, while significantly improves upon the model with its deeper architecture. Remarkably, NTram, being purely neural network-based, can achieve performance comparable to the traditional phrase-based machine translation system (Moses) with a small vocabulary and a modest parameter size.

1 Introduction

Sequence-to-sequence learning is a fundamental problem in natural language processing, with many important applications such as machine translation [2, 19], part-of-speech tagging [7, 20] and dependency parsing [3, 21]. Recently, there has been significant progress in development of technologies for the task using purely neural network-based models. Without loss of generality, we take machine translation as example in this paper. Previous efforts on neural machine translation generally fall into two categories:

- **Encoder-Decoder**: models of this type first summarize the source sentence into a fixed-length vector by the encoder, typically implemented with a recurrent neural network (RNN) or a convolutional neural network (CNN), and then unfold the vector into the target sentence by the decoder, typically implemented with a RNN [1, 4, 19];

\*The work is done when the first author worked as intern at Noahs Ark Lab, Huawei Technologies.
• **Automatic Alignment:** with RNNsearch [2] as representative, it represents the source sentence as a sequence of vectors after a processing step (e.g., through a bi-directional RNN [17]), and then simultaneously conducts dynamic alignment through a gating neural network and generation of the target sentence through another RNN.

Empirical comparison between the two schools of methods indicates that the automatic alignment approach is more efficient than the encoder-decoder approach: it can achieve comparable results with far less parameters and training instances [11]. This superiority in efficiency comes mainly from the mechanism of dynamic alignment, which avoids the need to represent the entire source sentence with a fixed-length vector [19].

The dynamic alignment mechanism [2] is intrinsically related to the content-based addressing on an external memory in the recently proposed Neural Turing Machines (NTM) [8]. Inspired by both works, we propose a novel deep architecture, named **Neural Transformation Machine** (NTRAM), for sequence-to-sequence learning. NTRAM carries out the task through a series of non-linear transformations from the input sequence, to different levels of intermediate representations, and eventually to the final output sequence. Similar to the notion of memory in [8], NTRAM stores the series of intermediate representations with stacked layers of memories, and conducts transformations between those representations with read-write operations on the memories. Through layer-by-layer stacking of such transformations, NTRAM generalizes the notion of inter-layer nonlinear mapping in neural networks, and therefore introduces a powerful new deep architecture tailored for sequence-to-sequence learning. NTRAM naturally takes RNNsearch [2] as a special case with a relatively shallow architecture. More importantly NTRAM accommodates many alternative deeper architectures, offering more modeling capability, and is empirically superior to RNNsearch on machine translation tasks.

## 2 Read-Write as a Nonlinear Transformation

We start with discussing read-write operations between two pieces of memory as a generalized form of nonlinear transformation. As illustrated in Figure 1, this transformation consists of memories of two layers (R-memory and W-memory), read-heads, a write-head, and a controller. Basically, the controller sequentially operates the read-heads to get the values from R-memory (“reading”), which are then sent to the write-head for modifying the values at specific locations in W-memory (“writing”). These basic components are more formally defined below, following those in Neural Turing Machines (NTM) [8], with however important modifications for the nesting architecture, implementation efficiency and description simplicity.

- **Memory:** a memory is generally defined as a matrix with potentially infinite size, while here we limit ourselves to pre-determined (pre-claimed) $N \times d$ matrix, with $N$ memory locations and $d$ values in each location. In our implementation of NTRAM, $N$ is always instance-dependent and is pre-determined by the algorithm. In a NTRAM system, we have memories of different layers, which in general have different $d$.

![Figure 1: Read-write as a nonlinear transformation.](image)
- **Controller:** a controller operates the read-heads and write-head, with discussion on its mechanism put off to Section 2.1. For the simplicity in modeling, NTRAM is only allowed to read from memory of lower layer and write to memory of higher layer. As another constraint, reading memory can only be performed after the writing to it completes, following a similar convention as in NTM [8].

- **Read-heads:** a read-head gets the values from the corresponding memory, following the instructions of the controller, which also influences the controller in feeding the state-machine (see Section 2.1). NTRAM allows multiple read-heads for one controller, with potentially different addressing strategies (see Section 2.2).

- **Write-head:** a write-head simply takes the instruction from controller and modifies the values at specific locations.

### 2.1 Controller

The core to the controller is a state machine, implemented as a Long Short-Term Memory RNN (LSTM) [10], with state at time \( t \) denoted as \( s_t \). With \( s_t \), the controller determines the reading and writing at time \( t \), while the return of reading in turn takes part in updating the state. For simplicity, only one reading and writing is allowed at one time step, but more than one read-heads are allowed (see Section 3.1 for an example). Now suppose for one particular instance (index omitted for notational simplicity), the system reads from the \( R \)-memory (\( M^r \), with \( N^r \) units) and writes to \( W \)-memory (denoted \( M^w \), with \( N^w \) units)

- **R-memory:** \( M^r = \{x^r_1, x^r_2, \ldots, x^r_{N^r}\} \)
- **W-memory:** \( M^w = \{x^w_1, x^w_2, \ldots, x^w_{N^w}\} \)

with \( x^r_n \in \mathbb{R}^{d_r} \) and \( x^w_n \in \mathbb{R}^{d_w} \). The main equations for controllers are then

\[
\begin{align*}
\text{Read vector:} & \quad r_t = F_r(M^r, s_t; \Theta_r) \\
\text{Write vector:} & \quad v_t = F_w(s_t; \Theta_w) \\
\text{State update:} & \quad s_{t+1} = F_{\text{dyn}}(s_t, r_t; \Theta_{\text{dyn}})
\end{align*}
\]

where \( F_{\text{dyn}}(\cdot), F_r(\cdot) \) and \( F_w(\cdot) \) are respectively the operators for dynamics, reading and writing \(^1\) parameterized by \( \Theta_{\text{dyn}}, \Theta_r \) and \( \Theta_w \).

In the remainder of this section, we will discuss several relatively simple yet effective choices of reading and writing strategies.

### 2.2 Addressing for Reading

**Location-based Addressing** With location-based addressing (L-addressing), the reading is simply \( r_t = x^r_t \). Notice that with L-addressing, the state machine automatically runs on a clock determined by the spatial structure of \( R \)-memory. Following this clock, the write-head operates the same number of times. One important variant, as suggested in [2, 19], is to go through \( R \)-memory backwards after the forward reading pass, where the state machine (LSTM) has the same structure but parameterized differently.

\(^1\)Note that our definition of writing is slightly different from that in [3].
Content-based Addressing  With a content-based addressing (C-addressing), the return at $t$ is

$$r_t = F_R(M^R; s_t; \Theta_R) = \sum_{n=1}^{N_R} \tilde{g}(s_t, x^R_{n,t}; \Theta_R)x^R_n,$$

and

$$\tilde{g}(s_t, x^R_{n,t}; \Theta_R) = \frac{g(s_t, x^R_{n,t}; \Theta_R)}{\sum_{n'=1}^{N_R} g(s_t, x^R_{n',t}; \Theta_R)},$$

where $g(s_t, x^R_{n,t}; \Theta_R)$, implemented as a deep neural network (DNN), gives un-normalized “affiliation” score for unit $x^R_n$ in $R$-memory. It is strongly related to the automatic alignment mechanism first introduced in [2] and general attention models discussed in computer vision [9]. One particular advantage of the content-based addressing is that it provides a way to do global reordering of the sequence, therefore introduces great flexibility in learning the representation.

Hybrid Addressing: With hybrid addressing (H-addressing) for reading, we essentially use two read-heads (can be easily extended to more), one with $L$-addressing and the other with $C$-addressing. At each time $t$, the controller simply concatenates the return of both read-heads as the final return:

$$r_t = [x^R_t, \sum_{n=1}^{N_R} g(s_t, x^R_{n,t}; \Theta_R)x^R_n].$$

It is worth noting that with $H$-addressing, the tempo of the state machine will be determined by the $L$-addressing read-head, and therefore creates $W$-memory of the same number of locations in writing. As shown later, $H$-addressing can be readily extended to allow read-heads to work on different memories.

2.3 Addressing for Writing

Location-based Addressing  With location-based addressing (L-addressing), the writing is simple. At any time $t$, only the $t^{th}$ location in $W$-memory is updated: $x^W_t = v_t \overset{\text{def}}{=} F_W(s_t; \Theta_W)$, which will be kept unchanged afterwards.

Content-based Addressing  In a way similar to $C$-addressing for reading, the location for units to be written is determined through a gating network $g(s_t, x^W_{n,t}; \Theta_W)$, where the values in $W$-memory at time $t$ is given by

$$\Delta_{n,t} = \tilde{g}(s_t, x^W_{n,t}; \Theta_W)F_W(s_t; \Theta_W), \quad x^W_{n,t} = (1 - \alpha)x^W_{n,t-1} + \alpha \Delta_{n,t}, \quad n = 1, 2, \cdots, N_W,$$

where $x^W_{n,t}$ stands for the values of the $n^{th}$ location in $W$-memory at time $t$, $\alpha$ is the forgetting factor (similarly defined as in [8]), $\tilde{g}$ is the normalized weight (with unnormalized score implemented also with a DNN) given to the $n^{th}$ location at time $t$.

2.4 Read-Write as a Nonlinear Transformation

Clearly the read-write defined above in this section transforms the representation in $R$-memory to the representation in $W$-memory, with the spatial structure embedded in a way shaped

\footnote{This spatial structure is to encode the temporal structure of input and output sequences.}
jointly by design (in specifying the strategy and the implementation details) and the later supervised learning (in tuning the parameters). We argue that the memory offers more representational flexibility for encoding sequences with complicated structures, and the transformation introduced above provides more modeling flexibility for sequence-to-sequence learning.

As the most “conventional” special case, if we use $L$-addressing for both reading and writing, we actually get the familiar structure in units found in RNN with stacked layers [16]. Indeed, as illustrated in the figure right to the text, this read-write strategy will invoke a relatively local dependency based on the original spatial order in $R$-memory. It is not hard to show that we can recover some deep RNN model in [16] after stacking layers of read-write operations like this.

The $C$-addressing, however, be it for reading and writing, offers a means for major re-ordering of the cells, while $H$-addressing can add into it the spatial structure of the lower layer memory. Memory with designed inner structure gives more representational flexibility for sequences than a fixed-length vector, especially when coupled with appropriate reading strategy in composing the memory of next layer in a deep architecture as shown later. On the other hand, the learned memory-based representation is in general less universal than the fixed-length representation, since they typically needs a particular reading strategy to decode the information.

In this paper, we consider four types of transformations induced by combinations of the read and write addressing strategies, listed pictorially in Figure 2. Notice that 1) we only include one combination with $C$-addressing for writing since it is computationally expensive to optimize when combined with a $C$-addressing reading (see Section 3.2 for some analysis), and 2) for one particular read-write strategy there are still a fair amount of implementation details to be specified, which are omitted due to the space limit. One can easily design different read/write strategies, for example a particular way of $H$-addressing for writing.

![Figure 2: Examples of read-write strategies.](image)

### 3 NTram: Stacking Them Together

We stack the transformations together to form a deep architecture for sequence-to-sequence learning (named NTRAM), in a way analogous to the layers in DNNs. The aim of NTRAM is to learn the representation of sequence better suited to the task (e.g., machine translation) through layer-by-layer transformations. Just as in DNN, we expect that stacking relatively simple transformations can greatly enhances the expressing power and the efficiency of NTRAM, especially in handling translation between languages with vastly different syntactical structures (e.g., Chinese and English).
As illustrated in Figure 3 (left panel), the stacking is straightforward: we can just apply a transformation on top of another, with the $W$-memory in lower layer being the $R$-memory of upper layer. Based on the memory layers stacked in this manner, we can define the entire deep architecture of NTRAM, with diagram in Figure 3 (right panel). Basically, it starts with symbol sequence (Layer-0), then moves to sequence of word embeddings (Layer-1), through layers of transformation to reach the final intermediate layer (Layer-$L$), which will be read by the output layer. The operations in output layer, relying on another LSTM to generating the target sequence, are similar to a memory read-write, with however the following two differences:

- it predicts the symbols for the target sequence, and takes the “guess” as part of the input to update the state of the generating LSTM, while in a memory read-write, there is no information flow from higher layers to lower layers;

- since the target sequence is in general of different length as the top-layer memory, it is limited to take only pure $C$-addressing reading, and relies on the built-in mechanism of the generating LSTM to stop (e.g., after generating a End-of-Sentence token).

Figure 3: Illustration of stacked layers of memory (left) and the overall diagram of NTRAM (right).

Memory of different layers could be equipped with different read-write strategies, and even for the same strategy, the configurations and learned parameters are in general different. This is in contrast to DNNs, for which the transformations of different layers are more homogeneous (mostly linear transforms with nonlinear activation function). As shown by our empirical study (Section 5), a sensible architecture design in combining the nonlinear transformations can greatly affect the performance of the model, on which however little is known and future research is needed.
3.1 Cross-Layer Reading

In addition to the generic read-write strategies in Section 2.4, we also introduce the cross-layer reading into NTRam for more modeling flexibility. In other words, for writing in any Layer-$\ell$, NTRam allows reading from any layers lower than $\ell$, instead of just Layer-$\ell-1$. More specifically, we consider the following two cases.

**Memory-Bundle**: A Memory-Bundle, as shown in Figure 4 (left panel), concatenates the units of two aligned memories in reading, regardless of the addressing strategy. Formally, the $n^{th}$ location in the bundle of memory Layer-$\ell'$ and Layer-$\ell''$ would be

$$x_{n}^{(\ell' + \ell'')} = [(x_{n}^{\ell'})^\top, (x_{n}^{\ell''})^\top]^\top.$$ 

Since it requires strict alignment between the memories to put together, MEMORY-BUNDLE is usually on layers created with spatial structure of same origin (see Section 4 for examples).

**Short-Cut** Unlike MEMORY-BUNDLE, SHORT-CUT allows reading from layers with potentially different inner structures by using multiple read-heads, as shown in Figure 4 (right panel). For example, one can use a $C$-addressing read-head on memory Layer-$\ell'$ and a $L$-addressing read-head on Layer-$\ell''$ for the writing to memory Layer-$\ell$ with $\ell > \ell', \ell''$.

3.2 Optimization

For any designed architecture, the parameters to be optimized include \{\(\Theta_{\text{dyn}}, \Theta_{r}, \Theta_{w}\)\} for each controller, the parameters for the LSTM in the output layer, and the word-embeddings. Since the reading from each memory can only be done after the writing on it completes, the “feed-forward” process can be described in two scales: 1) the flow from memory of lower layer to memory of higher layer, and 2) the forming of a memory at each layer controlled by the corresponding state machine. Accordingly in optimization, the flow of “correction signal” also propagates at two scales:

- On the “cross-layer” scale: the signal starts with the output layer and propagates from higher layers to lower layers, until Layer-1 for the tuning of word embedding;
- On the “within-layer” scale: the signal back-propagates through time (BPTT) controlled by the corresponding state-machine (LSTM). In optimization, there is a correction for each reading or writing on each location in a memory, making the $C$-addressing more expensive than $L$-addressing for it in general involves all locations in the memory at each time $t$.

The optimization can be done via the standard back-propagation (BP) aiming to maximize the likelihood of the target sequence. In practice, we use the standard stochastic gradient descent (SGD [14]) and mini-batch (size 80) with learning rate controlled by AdaDelta [22].
4 NTram: Some Special Cases

We discuss four representative special cases of NTram: Arc-I, II, III and IV, as novel deep architectures for machine translation. We also show that RNNsearch [2] can be described in the framework in NTram as a relatively shallow case.

**Arc-I** The first proposal, including two variants (Arc-I$_{hyb}$ and Arc-I$_{loc}$), is designed to demonstrate the effect of $C$-addressing in intermediate memory layers, with diagram shown in the figure right to the text. It employs a $L$-addressing reading from memory Layer-1 (the embedding layer), and $L$-addressing writing to Layer-2. After that, Arc-I$_{hyb}$ writes to Layer-3 ($L$-addressing) based on its $H$-addressing reading (two read-heads) on Layer-2, while Arc-I$_{loc}$ uses $L$-addressing to read from Layer-2. Once Layer-3 is formed, it is then put together with Layer-2 as a Memory-Bundle, from which the output layer reads ($C$-addressing) for predicting the target sequence.

**Arc-II** As a variant of Arc-I$_{hyb}$, this architecture is designed to investigate the effect of $H$-addressing reading from different layers of memory (or Short-Cut in Section 3.1). It uses the same strategy as Arc-I$_{hyb}$ in generating memory Layer-1 and 2, but differs in generating Layer-3, where Arc-II uses $C$-addressing reading on Layer-2 but $L$-addressing reading on Layer-1. Once Layer-3 is formed, it is then put together with Layer-2 as a Memory-Bundle, which is then read by the output layer for predicting the target sequence.

**Arc-III** We intend to use this design to study a deeper architecture and more complicated addressing strategy. Arc-III follows the same way to generate Layer-1, Layer-2 and Layer-3 as Arc-II. After that it uses two read-heads combined with a $L$-addressing write to generate Layer-4, where the two read-heads consist of a $L$-addressing read-head on Layer-1 and a $C$-addressing read-head on the memory bundle of Layer-2 and Layer-3. After the generation of Layer-4, it puts Layer-2, 3 and 4 together for a bigger Memory-Bundle to the output layer. Arc-III, with 4 intermediate layers, is the deepest among the four special cases.
Arc-IV This proposal is designed to study the efficacy of C-addressing for writing in forming intermediate representation. It employs a L-addressing reading from memory Layer-1 and L-addressing writing to Layer-2. After that, it uses a L-addressing reading on Layer-2 to write to Layer-3 with C-addressing. For C-addressing writing to Layer-3, all locations in Layer-3 are initially set to be a linear transformation of that in Layer-2, where this linear transformation is also subject to optimization. Once Layer-3 is formed, it is then bundled with Layer-2 for the reading (C-addressing) of the output layer.

4.1 Special Cases: RNNsearch
Interestingly, RNNsearch [2], the seminal work of neural translation model with automatic alignment, can be viewed as a special case of NTram with shallow architecture. More specifically, as illustrated in the figure right to text, it employs L-addressing reading on memory Layer-1 (the embedding layer), and L-addressing writing to Layer-2, which then is read (C-addressing) by the output layer for generating the target sequence. Essentially, Layer-2 is the only intermediate layer created by nontrivial read-write operations.

5 Experiments
We report our empirical study on applying NTram to Chinese-to-English translation, and compare it against state-of-the-art NMT models and statistical machine translation model.

5.1 Setup
Dataset and Evaluation Metrics: Our training data consist of 1.25M sentence pairs extracted from LDC corpora. The bilingual training data contain 27.9 million Chinese words and 34.5 million English words. We choose NIST MT Evaluation test set 2002 (MT02) as our development set, NIST MT Evaluation test sets 2003 (MT03), 2004 (MT04) and 2005 (MT05) as our test sets. The numbers of sentences in NIST MT02, MT03, MT04 and MT05 are 878, 919, 1788 and 1082 respectively. We use the case-insensitive 4-gram NIST BLEU score as our evaluation metric, and sign-test [6] as statistical significance test.

Preprocessing: We perform word segmentation for Chinese with the Stanford NLP toolkit and English tokenization with the tokenizer from Moses. In training of the neural networks,

---

3 The corpora include LDC2002E18, LDC2003E07, LDC2003E14, Hansards portion of LDC2004T07, LDC2004T08 and LDC2005T06.
4 ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v11b.pl
5 http://nlp.stanford.edu/software/segmenter.shtml
6 http://www.statmt.org/moses/
Table 1: BLEU-4 scores (%) of NMT baselines: RNNsearch (default) and RNNsearch (best), NTram architectures (Arc-I, II, III and IV), and phrase-based SMT system (Moses). The "*" indicates that the results are significantly (p<0.05) better than those of the RNNsearch (best), while boldface numbers indicate the best results on the corresponding test set.

5.2 Comparisons to Other Models

We compare our method with the following models:

- **Moses**: We take the open source phrase-based translation system Moses [12] (with default configuration) as the comparison system of conventional SMT. The word alignments are obtained with GIZA++ [15] on the corpora in both directions, using the "grow-diag-final-and" balance strategy [13]. We adopt SRI Language Modeling Toolkit [18] to train a 4-gram language model with modified Kneser-Ney smoothing on the target portion of training data. For Moses, we use all the vocabulary of training data.

- **RNNsearch**: We also compare NTram against the automatic alignment model proposed in [2]. We use the default setting as in [2], denoted as RNNsearch (default), as well as the optimal re-scaling of the model (on sizes of both embedding and hidden layers, with about 50% more parameters than the default setting) in terms of the best testing performance, denoted as RNNsearch (best). We use the RNNsearch as the NMT baseline, for it represents the state-of-the-art neural machine translation methods with a small vocabulary and modest parameter size (30M~50M).

For a fair comparison to RNNsearch, 1) the output layer in all NTram architectures are implemented as gated-RNN in [2] as a variant of LSTM [5], and 2) all the NTram architectures are designed to have the same embedding size as in RNNsearch (default) with parameter size less or comparable to RNNsearch (best).

5.3 Results

The main results of different models are given in Table 1. RNNsearch (best) is about 1.7 point behind Moses in BLEU on average, which is consistent with the observations made by other

---

*The reported Moses does not include any language model trained with a separate monolingual corpus.*
authors on different machine translation tasks \cite{2,11}. Remarkably, some sensible designs of NTRAM (e.g., Arc-II) can already achieve performance comparable to Moses, with only 42M parameters, while RNNsearch (best) has 46M parameters.

Clearly most of the NTRAM architectures (Arc-I<sub>hyb</sub>, Arc-II and Arc-III) yield performances better than the NMT baselines. Among them, Arc-II outperforms the best setting of NMT baseline RNNsearch (best) by about 1.5 BLEU on average with less parameters. This suggests that the deeper architectures in NTRAM help to capture the transformation of representations essential to the machine translation.

5.4 Discussion

Further comparison between Arc-I<sub>hyb</sub> and Arc-I<sub>loc</sub> (similar parameter sizes) suggests that C-addressing reading plays an important role in learning a powerful transformation between intermediate representations, necessary for translation between language pairs with vastly different syntactical structures. This conjecture is further verified by the good performances of Arc-II and Arc-III, both of which have C-addressing read-heads in their intermediate memory layers.

Clearly the BLEU scores of Arc-IV are considerably lower than the other architectures, while close analysis shows that the translation results are usually shorter than those of Arc-I\textasciitilde Arc-III for about 15%. One possibility is that our particular implementation of C-addressing for writing in Arc-IV (Section 4) is hard to optimize, which might need to be “guided” by another write-head or some smart initialization. This will be one thing for our future exploration.

As another observation, cross-layer reading almost always helps. The performances of Arc-I, II, III and IV unanimously drop after removing the Memory-Bundle and Short-Cut (results omitted here), even after the broadening of memory units to keep the parameter size unchanged.

We have also observed some failure cases in the architecture designing for NTRAM, most notably when we have only single read-head with C-addressing in writing to the memory that is the sole layer going to the next stage. Comparison of this design with H-addressing (results omitted here) suggests that another read-head with L-addressing can prevent the transformation from going astray by adding the tempo from a memory with a clearer temporal structure.

6 Conclusion

We propose NTRAM, a novel architecture for sequence-to-sequence learning, which is stimulated by the recent work of Neural Turing Machine\cite{8} and Neural Machine Translation \cite{2}. NTRAM builds its deep architecture for processing sequence data on the basis of a series of transformations induced by the read-write operations on a stack of memories. This new architecture significantly improves the expressing power of models in sequence-to-sequence learning, which is verified by our empirical study on benchmark machine translation tasks.

References

\cite{1} M. Auli, M. Galley, C. Quirk, and G. Zweig. Joint language and translation modeling with recurrent neural networks. In Proceedings of EMNLP, pages 1044–1054, 2013.
[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

[3] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*, pages 740–750, 2014.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of EMNLP*, pages 1724–1734, 2014.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Proceedings of NIPS Deep Learning and Representation Learning Workshop*, 2014.

[6] M. Collins, P. Koehn, and I. Kučerová. Clause restructuring for statistical machine translation. In *Proceedings of ACL*, pages 531–540, 2005.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[8] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[9] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[12] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL on interactive poster and demonstration sessions*, pages 177–180, Prague, Czech Republic, June 2007.

[13] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of NAACL*, pages 48–54, 2003.

[14] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In *Neural Networks: Tricks of the trade*. Springer, 1998.

[15] F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

[16] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. In *Proceedings of ICLR*, 2014.

[17] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[18] A. Stolcke et al. Srilm-an extensible language modeling toolkit. In *Proceedings of ICSLP*, volume 2, pages 901–904, 2002.

[19] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[20] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. *arXiv preprint arXiv:1412.7449*, 2014.

[21] M. Wang, Z. Lu, H. Li, W. Jiang, and Q. Liu. gcnn: A convolutional architecture for word sequence prediction. *arXiv preprint arXiv:1503.05034*, 2015.

[22] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.