PRNN: Recurrent Neural Network with Persistent Memory

Kui Zhao*, Yuechuan Li*, Chi Zhang, Cheng Yang, Shenghuo Zhu
Machine Intelligence Technologies, Alibaba Group
zhaokui.zk, yuechuan.lyc, yutou.zc, charis.yangc, shenghuo.zhu@alibaba-inc.com

Abstract

Although Recurrent Neural Network (RNN) has been a powerful tool for modeling sequential data, its performance is inadequate when processing sequences with multiple patterns. In this paper, we address this challenge by introducing an external memory and constructing a novel persistent memory augmented RNN (termed as PRNN). The PRNN captures the principle patterns in training sequences and stores them in an external memory. By leveraging the persistent memory, the proposed method can adaptively update states according to the similarities between encoded inputs and memory slots, leading to a stronger capacity in assimilating sequences with multiple patterns. Content-based addressing is suggested in memory accessing, and gradient descent is utilized for implicitly updating the memory. Our approach can be further extended by combining the prior knowledge of data. Experiments on several datasets demonstrate the effectiveness of the proposed method.

1 Introduction

Recent years have witnessed the great success of deep learning models. Owing to the increasing computation resources and strong model capacity, neural network models have been applied in numerous applications. Among all neural network models, Recurrent Neural Networks (RNNs) [Williams and Hinton, 1986] have shown notable potential on sequence modeling tasks, e.g. speech recognition [Dario et al., 2016] and machine translation [Cho et al., 2014] [Sutskever et al., 2014], and therefore receive particular attention. With increasing explorations on RNN, several variants, such as Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], Gated Recurrent Unit (GRU) [Chung et al., 2015] have been proposed successively.

The key advantages of RNN come from the recurrent structures, which carry out the same transition at all time steps, and eventually contribute to a satisfactory performance. Yet this merit may validate under the assumption that all sequences follow the same pattern. The conventional RNN may be inappropriate when processing sequences with multiple patterns. As mentioned in [Goodfellow et al., 2016] [Zhu et al., 2017], it is difficult to optimize the network when using the same parameters at all time steps under multiple pattern scenarios. To this end, more adaptive RNN networks are required.

Recently, some extended mechanisms on RNN are proposed to augment model adaptability. The first one is the attention mechanism [Bahdanau et al., 2014], which is a popular technique in machine translation. The attention mechanism suggests aligning data before prediction at each time step, and the encoded contexts are likely to capture different patterns when translating different words. Another attractive mechanism is the memory mechanism [Weston et al., 2014]. The basic idea of the memory mechanism is to setup an external memory for each sequence. However, most memory based approaches build the “temporal” memory and the memory is reset when a new sequence arrives. Thus this “temporal” memory probably cannot capture the principle patterns in training sequences. For instance, when people read documents, their comprehension is based on not only context in the current document, but also knowledge accumulated from previous reading and life experiences. Therefore, besides the “temporal” memory, a persistent memory is immensely needed to capture all historical principle patterns.

In this paper, a novel persistent memory (p-memory) augmented RNN (called PRNN) is proposed. Different from the external memories in existing works, our p-memory holds principle patterns along training and testing phases. The memory is accessed by content-based addressing and it is updated through gradient descent. Each slot in p-memory (the column of memory matrix) denotes one principle pattern. The relationship between the memory accessing and the mixture model is also illustrated. By introducing the p-memory, our PRNN presents a stronger capacity in processing sequences with multiple patterns than the conventional RNNs. Moreover, the PRNN model can be flexibly extended when the prior knowledge of data is provided. The contributions of our work are summarized as follows:

- We propose a novel persistent memory and construct PRNN to adaptively process sequences with multiple patterns.
- We derive content-based addressing for memory access-
ing, and interpret memory accessing from the mixture model perspective.

- We show that our approach can be easily extended by combining the prior knowledge of data.
- We evaluate the proposed PRNN on extensive experiments, including time series prediction task and language modeling task. The experimental results demonstrate significant advantages of our PRNN.

The remainder of the paper is organized as follows. Section 2 reviews related work. In Section 3, the persistent memory is introduced in detail, as well as its extension on LSTM and combination with prior knowledge. Experimental evaluations are included in Section 4, and final conclusion with looking forward comes in Section 5.

2 Related Work

The research on RNN can be traced back to 1990’s [Williams and Hinton, 1986]. In the past decades, a number of variants of RNN model appear. Skip connections are introduced to allow the units to access values from the distant past [Mozer, 1992]. Bidirectional RNN suggests to predict the output based on the entire input sequence [Schuster and Paliwal, 1997]. The gated RNNs belong to another family of RNN variants, in which the most popular one is Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. LSTM networks introduce memory cells and utilize a gating mechanism to control memory accessing. Another classic gated RNN variant, Gated Recurrent Unit (GRU), simplifies LSTM with a single update gate, which controls the forgetting factor and updating factor simultaneously [Chung et al., 2015].

Recently, two advanced mechanisms, attention and memory, appear to modify the RNN structure. The attention mechanism is particularly useful in machine translation [Cho et al., 2014] [Sutskever et al., 2014], which requires extra data alignment, and the similarity between the encoded source sentence and the output word is calculated. Beyond machine translation, the attention mechanism gains notable popularity in other areas including video captioning [Jingkuan et al., 2017], cascade prediction [Yongqing et al., 2017] and so on.

A comprehensive study on attention mechanism can be found in [Vaswani et al., 2017].

In terms of the memory mechanism, its basic idea is to borrow an external memory for each sequence [Weston et al., 2014]. Two popular addressing approaches, i.e. location-based addressing and context-based addressing, are utilized in memory based models. Most existing works use the reading weights, writing weights and usage weights during memory accessing, and update the memory in an explicit manner [Graves et al., 2014] [Santoro et al., 2016]. The memory mechanism is first introduced for few-shot learning tasks [Santoro et al., 2016], and then extended to other applications such as question answering [Kyung-Min et al., 2017] and bug detection in programming [Min-Je et al., 2017].

Although named after “memory”, our approach differs from previous memory approaches in that a “persistent” memory is proposed. The persistent memory works along training and testing phases, and is used to store the principle patterns in training sequences.

3 PRNN Model

3.1 Persistent Memory

In conventional RNNs, the hidden states are updated with a unique cell at all time steps, which can be expressed as:

$$ h_t = g(h_{t-1}, x_t). $$

In this paper, an external persistent memory $M$ (with dimension $m \times n$, called p-memory) is employed to memorize the principal patterns in training sequences, and the new persistent memory augmented RNN (PRNN) can flexibly process sequences with multiple patterns. The structure for PRNN is shown in Figure 1, and the hidden states in PRNN are updated by:

$$ h_t = g(h_{t-1}, x_t, p(h_{t-1}, M)). $$

where $p(h_{t-1}, M)$ denotes the memory accessing via content-based addressing, with the hidden states $h_{t-1}$ and the memory matrix $M$ serving as inputs. Note that when the p-memory is introduced, the function $g(\cdot)$ remains the same, which means the RNN cell need not change its inner structure. In this manner, our persistent memory can equip any RNN cell, and thus is generally applicable.

![Figure 1: The architecture of PRNN.](image)

Memory accessing

The memory matrix $M_{m \times n}$ contains $n$ different slots, and each slot represents certain pattern in training sequences. Given an input sequence $[x_1, x_2, \ldots]$, the hidden state $h_{t-1}$ is able to represent the subsequence $[x_1, \ldots, x_{t-1}]$. Following the content-based addressing, the similarity between $h_{t-1}$ and each slot of p-memory $M_i(i \in \{1, \ldots, n\})$, denoting as $s_i$, can be easily calculated by retrieving each column of the memory matrix. This similarity $s_i$ is used to produce a weight vector $w$, with elements computed according to a softmax:

$$ w_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}. $$

The weight vector $w$ is the strength to amplify or to attenuate the slots in p-memory. Consequently, the memory accessing $p(h_{t-1}, M)$ is formulated as:

$$ p(h_{t-1}, M) = \sum_i w_i M_i. $$
The cosine similarity is widely applied in related works and is given by
\[ L = \frac{{h_{t-1}^T M_i}}{\|h_{t-1}\|_2 \|DM_i\|_2}. \]
The cosine similarity is widely applied in related works [Graves et al., 2014] [Santoro et al., 2016], considering its robustness and computational efficiency.

**Memory updating**

The memory in [Graves et al., 2014] [Santoro et al., 2016] acts as “temporal memory” and is explicitly updated in both training and testing phases. However, the p-memory in this paper is updated in an entirely implicit manner. The updating mechanism in PRNN is completely based on gradient descent, regardless of the read weights, write weights as well as usage weights utilized in previous works. The memory matrix is updated straightforwardly during the training phase, leading to a simple training procedure in memory augmented networks.

Specifically, the procedure for updating the p-memory is based on the gradient of loss function \( \mathcal{L} \) on the \( i \)-th slot of p-memory, which is given by

\[ \frac{\partial \mathcal{L}}{\partial M_i} = \nabla_{p(h_{t-1},M)} \mathcal{L} \frac{\partial p(h_{t-1},M)}{\partial M_i} \]

where

\[ w'_i = w_i (1 - w_i) \nabla_{p(h_{t-1},M)} \mathcal{L} \]

When the similarity is calculated by Eq (5), then:

\[ h'_{t-1} = (h_{t-1} - DM_i)^T, \]

and when the similarity measure follows the cosine similarity:

\[ h'_{t-1} = \left( \frac{h_{t-1}^T}{\|h_{t-1}\|_2} \right) \left( \frac{I}{\|DM_i\|_2} - \frac{(DM_i)(DM_i)^T}{\|DM_i\|_2^2} \right). \]

The gradient in Eq (7) consists of two terms. The first one assigns similarity weight \( w_i \) to the common derivative \( \nabla_{p(h_{t-1},M)} \mathcal{L} \). Thus the first term becomes more smooth when \( h_{t-1} \) is less similar with memory slot \( M_i \). The second term considers how the similarity measure affects the gradient. Taking the measure in Eq (6) for illustration, \( h'_{t-1} \) can be seen as the “difference” between the hidden state and the memory slot \( M_i \). Thus the second part in Eq (7) is a combination of such “difference” and the memory slot \( M_i \). From the above equations, it can be easily found that the gradient in Eq (7) is simply quadratic for \( M_i \) when \( w_i \) is given, which has been computed during memory accessing beforehand. Therefore, memory updating is reasonably expected to be fast during the training process.

\[ \text{Mixture model perspective} \]

As the p-memory \( M \) has \( n \) slots, that is, all hidden states in training sequences are partitioned into \( n \) clusters \( z \in \{1, \ldots, n\} \). Given the hidden state \( h_{t-1} \), the probability that \( h_{t-1} \) belongs to the \( i \)-th cluster is:

\[ P(z = i|h_{t-1}) = \frac{P(z = i) \frac{P(h_{t-1}|z = i)}{P(h_{t-1})}}{\sum_j P(z = j) \frac{P(h_{t-1}|z = j)}{P(h_{t-1})}}. \]

Assume the hidden states \( h_{t-1} \) are Gaussian variables, then

\[ P(h_{t-1}|z = i) = \frac{1}{(2\pi)^{\frac{d}{2}} \det(\Sigma_i)^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (h_{t-1} - \mu_i)^T \Sigma_i^{-1} (h_{t-1} - \mu_i) \right), \]

where \( h \) is the dimension of \( h_{t-1} \), \( \mu_i \) and \( \Sigma_i \) are the mean and covariance matrix of component \( i \) respectively.

For uniformly distributed prior, e.g. \( P(z = i) = \frac{1}{n} \), assume all clusters share the same covariance matrix \( \Sigma (\Sigma^{-1} = P) \) and let \( \mu_i = DM_i \), we have:

\[ P(z = i|h_{t-1}) = \frac{1}{n} \exp \left( -\frac{1}{2} (h_{t-1} - \mu_i)^T \Sigma^{-1} (h_{t-1} - \mu_i) \right) \]

\[ \sum_j \frac{1}{n} \exp \left( -\frac{1}{2} (h_{t-1} - \mu_j)^T \Sigma^{-1} (h_{t-1} - \mu_j) \right) \]

\[ = \frac{\exp(s_i)}{\sum_j \exp(s_j)} = w_i, \]

which is consistent with Eq (3) when the similarity measure follows Eq (5). Thus, the process of accessing p-memory is indeed soft clustering, and \( h_{t-1} \) is assigned to the corresponding center:

\[ p'(h_{t-1}, M) = \sum_i P(z = i|h_{t-1}) \mu_i \]

\[ = \sum_i w_i DM_i = DP(h_{t-1}, M). \]

Let \( D^+ \) be the Moore-Penrose inverse of \( D \), we have:

\[ p(h_{t-1}, M) = D^+ p'(h_{t-1}, M) = \sum_i w_i M_i, \]

and thus the memory accessing in Eq (4) can be easily computed from the center of the mixture model.

As a matter of fact, the mixture model perspective for p-memory accessing can be seen as a variant of the EM algorithm. For given hidden state \( h_{t-1} \) and current memory matrix \( M \), E-step assigns “responsibility” \( w_i \) to each cluster via memory addressing. The M-step optimizes the parameters in each memory slot based on \( w_i \) via gradient descent. Rather than explicit parameter updating mechanism, which appears in conventional EM procedure for Gaussian mixture model, the p-memory updating is implicit and derived from gradient descent.

Instead of Eq (5), when the cosine similarity is selected, and \( h_{t-1} \) follows Von Mises-Fisher distribution \((\kappa = 1)\) with probability density function:

\[ P(h_{t-1}|z = i) = C \exp \left( h_{t-1}^T \mu_i \right), \]

\[ \text{where C is a normalizing constant, } \]
where \( \mu_i = DM_i \) and \( C \) is the normalization constant, we also have \( P(z = i|h_{t-1}) = w_i \), and the memory accessing can be represented as a mixture model.

### 3.2 LSTM with Persistent Memory

As a general external memory mechanism, the persistent memory is able to equip almost all RNN models. In this section, we take LSTM as an example and illustrate how the p-memory is able to equip almost all RNN models. In this section, we take LSTM as an example and illustrate how the p-memory works in practice. Since the persistent memory is added to all gates and cells, the forget gate \( f_t \) and the input gate \( i_t \) are revised as:

\[
\begin{align*}
    f_t &= \sigma(W_f[h_{t-1}, x_t, p(h_{t-1}, M)] + b_f). \\
    i_t &= \sigma(W_i[h_{t-1}, x_t, p(h_{t-1}, M)] + b_i).
\end{align*}
\]

(17) (18)

Then the memory cell \( c_t \) can be updated adaptively:

\[
\begin{align*}
    c_t &= \text{tanh}(W_c[h_{t-1}, x_t, p(h_{t-1}, M)] + b_c). \\
    c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t.
\end{align*}
\]

(19) (20)

Consequently, the output gate \( o_t \) and cell \( h_t \) become:

\[
\begin{align*}
    o_t &= \sigma(W_o[h_{t-1}, x_t, p(h_{t-1}, M)] + b_o). \\
    h_t &= o_t \circ \text{tanh}(c_t).
\end{align*}
\]

(21) (22)

### 3.3 Persistent Memory with Prior Knowledge

In many practical situations, some prior or domain knowledge, either about sample distribution or about data pattern, is known beforehand. As a matter of fact, the prior knowledge can provide much information and help to build more efficient persistent memories, which substantially benefits the training process.

In this subsection, we select an example in text modeling for illustration. The category information (e.g. sports, science) is often provided in advance in text modeling. To leverage prior knowledge, we set up multiple persistent memories for RNN. Assume \( B \) buckets (categories) exist in the training data, an independent p-memory \( M^k \) \( k \in \{1, \ldots, B\} \) is allocated for each bucket. Therefore, the memory accessing is extended to be:

\[
P(h_{t-1}, M^k) = \sum_i w_i M^k_i.
\]

(23)

### 4 Experiments

The performance of the PRNN is evaluated on two different tasks: time series prediction task and language modeling task. Experiments are implemented in Tensorflow [Abadi et al., 2016]. Two different measures for calculating similarity are suggested in Section 3.1, and considering the popularity of cosine similarity, we select this measure in all experiments.

#### 4.1 Time Series Prediction

We conduct time series prediction experiments on two datasets: Power Consumption (PC) and Sales Forecast (SF).

**Dataset**

- **Power Consumption (PC)** [Lichman, 2013]. This dataset contains measurements of electric power consumption in one household over a period of 198 weeks. The global-active-power is aggregated into hourly averaged time series, and the prediction target is the global-active-power at every hour on the next day. The entire dataset is divided into two parts: training part (dates range [2007-04-08, 2010-11-19]) and testing part (dates range [2010-11-20, 2010-11-26]).

- **Sales Forecast (SF)**. This dataset is collected from one of the largest E-commerce platforms in the world, and contains four features (i.e. browse times, customer number, price and sales) of 1 million items from 1500 categories over 13 weeks. The target is to predict the total sales in the next week for each item. As the category information is provided, it can be utilized as the prior knowledge. The training set includes over 3 million samples, and the size of testing set is about 2 thousand (testing sample list is provided by large merchants).

**Setup**

We select the LSTM as the baseline model. The LSTM is first compared with the classical ARIMA model [Hamilton and James, 1994]. Then the p-memory is added to the LSTM to construct our approach. The hyper-parameters are tuned on two datasets as follows:

- **Power Consumption (PC)**. For each hour, the input sequence contains the power consumption on the last 56 days. At every step, the sequence considers values at three adjacent hours centering at the target hour. The goal is to predict global-active-power at every hour on the next day. The LSTM has a single layer with dimension of hidden units equals to 32. In terms of our approach, the p-memory augmented LSTM (PLSTM) is constructed by adding p-memory \( M \) (with dimension \( 4 \times 8 \)) into LSTM. The models are trained for 30 epochs.

- **Sales Forecast (SF)**. For each item, the input is a sequence of four features on recent 56 days, and the target is the total sales in the next week. The LSTM also has a single layer, in which the dimension of hidden units is 64. Similarly, a PLSTM model with a p-memory \( M_{8 \times 16} \) is constructed. In addition, since we have the category information as the prior knowledge, we construct the prior knowledge based memory model PLSTM (PPLSTM), and assign a p-memory \( M_{8 \times 16} \) to each category. The models are trained for 15 epochs.

All trainable parameters are initialized randomly from a uniform distribution within \([-0.05, 0.05]\). The parameters are updated through back propagation with Adam rule [Kinga and Adam, 2015] and the learning rate is 0.001.

*https://archive.ics.uci.edu/ml/datasets/*

| Individual+household+electric+power+consumption |
Results
The results are measured in Relative Mean Absolute Error (RMAE), and it is written as:

\[
RMAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{\sum_{i=1}^{N} y_i},
\]

where \(N\) is the number of testing samples, \(y_i\) is the true value and \(\hat{y}_i\) is the prediction result. All RMAE results are shown in Table 1.

| Dataset | Model | RMAE (%) |
|---------|-------|----------|
| PC      | ARIMA | 40.2     |
|         | LSTM  | 35.4     |
|         | PLSTM | 34.4     |
| SF      | ARIMA | 84.2     |
|         | LSTM  | 56.8     |
|         | PLSTM | 52.5     |
|         | PPLSTM| 42.9     |

From Table 1, it is evident to see that our PLSTM model significantly outperforms LSTM and ARIMA, on both PC and SF datasets. Therefore, the advantages of our p-memory is fully demonstrated. Particularly, on the SF dataset, by utilizing the prior knowledge, the PPLSTM could further enhance the prediction accuracy, which indicates the strong adaptability of our approach as well.

4.2 Language Modeling
We conduct word-level prediction experiments on two datasets: Penn Treebank (PTB) and 20-Newsgroup (20NG).

Dataset
- **Penn Treebank (PTB)** [Marcus et al., 1993]. The PTB is a popular dataset in language modeling. In this experiment, we select the version provided by [Mikolov et al., 2010]. The dataset consists of 1 million words and has 10 thousand words in its vocabulary.
- **20-Newsgroup (20NG)** [Lang, 1995]. This dataset is originally a benchmark for text categorization, where 20 thousand news documents are evenly categorized into 20 groups. In word-level prediction, the group information is considered as the prior knowledge. The preprocessed data can be found in [Cardoso-Cachopo, 2007] and it consists of 5 million words and 74 thousand words are included in its vocabulary.

Setup
There are two different settings for the experiments: simple setting and complex setting. The purpose of the simple setting is to demonstrate the benefit of p-memory based on a simple LSTM model. The complex setting extends the advantages of our approach to more complicated models.

- **Simple setting**. In this setting, each word is embedded into a 32-dimensional representation. The size of hidden units is 128 in the baseline LSTM model. The dimension of the p-memory in PLSTM is \(16 \times 10\). On the 20NG dataset, since the group information is used as prior knowledge, we construct PPLSTM and allocate a p-memory \(M_{16 \times 10}\) for each group. Other settings, such as the parameter initialization, optimization method and learning rate, remain the same with those in time series prediction tasks. The model is trained for 20 epochs.

- **Complex setting**. The “large” network architecture in [Zaremba et al., 2014] provides a strong baseline for language modeling tasks, and there is an open source implementation [http://ana.cachopo.org/master/tutorials/rnn/ptb/ptb_word_lm.py](http://ana.cachopo.org/master/tutorials/rnn/ptb/ptb_word_lm.py). It contains many extensions, including multiple layers stacking, dropout, gradient clipping, learning rate decay and so on. In this setting, we select the “large” network (named by LARGE) as the baseline model. In terms of our approach, a p-memory \(M_{512 \times 32}\) is added into LARGE to construct the persistent memory augmented LARGE (PLARGE) model. Similarly, the prior knowledge based PLARGE (PPLARGE) is established by introducing a particular p-memory \(M_{512 \times 32}\) for each group on the 20NG dataset.

With more sophisticated model [Zilly et al., 2016] or ensemble of multiple models, lower perplexities on the word-level prediction tasks can be achieved. We here simply focus on assessing the impact of p-memory when added to an existing architecture, rather than absolute state-of-the-art performance.

Results
All results are measured in perplexity, which is a popular metric to evaluate language models [Katz, 1987]. The results are summarized in Table 2.

| Dataset | Simple | Perp. | Complex | Perp. |
|---------|--------|-------|---------|-------|
| PTB     | LSTM   | 148.6 | LARGE   | 78.4  |
|         | PLSTM  | 146.2 | PLARGE  | 78.2  |
| 20NG    | LSTM   | 178.9 | LARGE   | 109.1 |
|         | PLSTM  | 168.2 | PLARGE  | 108.4 |
|         | PPLSTM | 157.1 | PPLARGE | 105.4 |

In the simple setting, the benefit from the p-memory is significant on both the PTB and 20NG datasets, especially on the 20NG dataset. One possible explanation is that the content in 20NG is semantically richer, and our model could easily distinguish various semantic patterns in the dataset. Similarly, with the assist from prior knowledge, the efficiency of p-memory is further improved.

Dataset
- [http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz](http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz)
- [http://ana.cachopo.org/datasets-for-single-label-text-categorization](http://ana.cachopo.org/datasets-for-single-label-text-categorization)
The advantages of our approach remain in the complex setting. According to the results in Table\textsuperscript{2} we can see that the LARGE model has already achieved a high accuracy for both PTB and 20NG datasets, and our approach is able to further enhance the model performance. Thus, the general superiority of our approach is proved under both simple setting and complex setting. In addition, it is also noted that another advantage of our approach appears in model convergence. By introducing the p-memory, the PLARGE model is able to provide a better convergence rate. Taking the PTB dataset as an example, during the training procedure, word-level perplexity on the validation set is shown in Figure\textsuperscript{2} We can clearly see that the PLARGE model requires less epochs before it converges. Hence, beyond the superiority on high accuracy, the advantage on convergence performance also makes our approach appealing.

5 Conclusion

Conventional RNN is limited to adaptively process sequences with multiple patterns. In this paper, a novel PRNN approach is proposed and an external p-memory is introduced to memorize the principal patterns in training sequences. Through content-based accessing, the PRNN applies adaptive transition at each time step. The p-memory is updated by gradient descent, and the entire memory accessing can be interpreted as a mixture model. Moreover, the proposed approach can easily combine the prior knowledge of data. Experiments on time series task and language modeling task demonstrate the superiority and effectiveness of our PRNN method.

The proposed p-memory is a universal block, and we look forward to applying it to other types of neural networks, such as feed-forward networks and convolutional neural networks. Another interesting topic for further studies is the memory updating mechanism. The gradient descent based updating mechanism works very well in practice, and it simplifies the end-to-end training. However, it is still necessary to explore more appropriate updating mechanisms, and compare them with existing mechanisms comprehensively.

A Supplementary Material

A.1 Derivation of Updating Persistent Memory

According to the chain rule of calculus, we have:
\[
\frac{\partial \mathcal{L}}{\partial \mathbf{M}_i} = \nabla_p (\mathbf{h}_{t-1}, \mathbf{M}) \frac{\partial p(\mathbf{h}_{t-1}, \mathbf{M})}{\partial \mathbf{M}_i} .
\] (25)

\(\nabla_p (\mathbf{h}_{t-1}, \mathbf{M}) \mathcal{L}\) is the gradient of final loss function \(\mathcal{L}\) on the p-memory accessing \(p(\mathbf{h}_{t-1}, \mathbf{M})\) and \(\frac{\partial p(\mathbf{h}_{t-1}, \mathbf{M})}{\partial \mathbf{M}_i}\) can be calculated by:
\[
\frac{\partial p(\mathbf{h}_{t-1}, \mathbf{M})}{\partial \mathbf{M}_i} = \frac{\partial (w_i \mathbf{M}_i)}{\partial \mathbf{M}_i} + \sum_{i,j} \frac{\partial (w_j \mathbf{M}_j)}{\partial \mathbf{M}_i} .
\] (26)

The gradient of strength \(w_i\) on the \(i\)-th p-memory slot is:
\[
\frac{\partial w_j}{\partial \mathbf{M}_i} = \frac{\partial w_j}{\partial \mathbf{M}_j} \frac{\partial \mathbf{M}_j}{\partial \mathbf{M}_i} = w_j (1 - w_j) \frac{\partial s_j}{\partial \mathbf{M}_i} .
\] (27)

where
\[
\frac{\partial s_j}{\partial \mathbf{M}_i} = \begin{cases} 0 & i \neq j \\ \frac{\partial s_j}{\partial \mathbf{M}_i} & i = j \end{cases} .
\] (28)

So we only need to care about \(\frac{\partial s_i}{\partial \mathbf{M}_i}\). When the similarity is calculated by Eq (5):
\[
\frac{\partial s_i}{\partial \mathbf{M}_i} = \frac{\partial s_i}{\partial (\mathbf{DM}_i)} \frac{\partial (\mathbf{DM}_i)}{\partial \mathbf{M}_i} = (\mathbf{h}_{t-1} - \mathbf{DM}_i)^T \mathbf{D},
\] (29)

and when the similarity is calculated by Eq (6):
\[
\frac{\partial s_i}{\partial \mathbf{M}_i} = \frac{\partial s_i}{\partial (\mathbf{DM}_i)} \frac{\partial (\mathbf{DM}_i)}{\partial \mathbf{M}_i} = \left(\frac{\mathbf{h}_{t-1}^T}{||\mathbf{h}_{t-1}||_2}\right) \left(\frac{\mathbf{I}}{||\mathbf{DM}_i||_2} - \frac{(\mathbf{DM}_i) (\mathbf{DM}_i)^T}{||\mathbf{DM}_i||_2^2}\right) \mathbf{D}.
\] (30)

Then we get:
\[
\sum_j \mathbf{M}_j \frac{\partial w_j}{\partial \mathbf{M}_i} = w_i (1 - w_i) \mathbf{M}_i \mathbf{h}_{t-1}' \mathbf{D} .
\] (31)

where
\[
\mathbf{h}_{t-1}' = (\mathbf{h}_{t-1} - \mathbf{DM}_i)^T ,
\] (32)

or
\[
\mathbf{h}_{t-1}' = \left(\frac{\mathbf{h}_{t-1}^T}{||\mathbf{h}_{t-1}||_2}\right) \left(\frac{\mathbf{I}}{||\mathbf{DM}_i||_2} - \frac{(\mathbf{DM}_i) (\mathbf{DM}_i)^T}{||\mathbf{DM}_i||_2^2}\right) .
\] (33)

Finally, we get the gradient of final loss function \(\mathcal{L}\) on the \(i\)-th p-memory slot by substituting Eq (25) and Eq (31) into Eq (25):
\[
\frac{\partial \mathcal{L}}{\partial \mathbf{M}_i} = w_i \nabla_p (\mathbf{h}_{t-1}, \mathbf{M}) \mathcal{L} + w'_i \mathbf{h}_{t-1}' \mathbf{D},
\] (34)

where
\[
w'_i = w_i (1 - w_i) \nabla_p (\mathbf{h}_{t-1}, \mathbf{M}) \mathcal{L} \mathbf{M}_i .
\] (35)
References

[Abadi et al., 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

[Bahdanau et al., 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[Cardoso-Cachopo, 2007] Ana Cardoso-Cachopo. Improving methods for single-label text categorization. PhD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[Cho et al., 2014] Kyunghyun Cho, Bart Van Merriënoor, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[Chung et al., 2015] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In Proceedings of the International Conference on Machine Learning, pages 2067–2075, 2015.

[Dario et al., 2016] Amodei Dario, Ananthantharayan Sundaram, Anubhai Rishita, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In Proceedings of the International Conference on Machine Learning, pages 173–182, 2016.

[Goodfellow et al., 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[Graves et al., 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014.

[Hamilton and James, 1994] Hamilton and Douglas James. Time series analysis, volume 2. Princeton University Press, 1994.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[Jingkuan et al., 2017] Song Jingkuan, Gao Lianli, Guo Zhao, Liu Wu, Zhang Dongxiang, and Tao Shen Heng. Hierarchical lstm with adjusted temporal attention for video captioning. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 2737–2743, 2017.

[Katz, 1987] Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. IEEE Transactions on Acoustics, Speech, and Signal Processing, 35(3):400–401, 1987.

[Kinga and Adam, 2015] D Kinga and J Ba Adam. A method for stochastic optimization. In International Conference on Learning Representations, 2015.

[Kyung-Min et al., 2017] Kim Kyung-Min, Heo Min-Oh, Choi Seong-Ho, and Zhang Byoung-Tak. Deepstory: Video story qa by deep embedded memory networks. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 2016–2022, 2017.

[Lang, 1995] Ken Lang. Newsweeder: Learning to filter netnews. In Proceedings of the International Conference on Machine Learning, pages 331–339, 1995.

[Lichman, 2013] M. Lichman. UCI machine learning repository, 2013.

[Marcus et al., 1993] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. Computational Linguistics, 19(2):313–330, 1993.

[Merity et al., 2016] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843, 2016.

[Mikolov et al., 2010] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Interspeech, volume 2, page 3, 2010.

[Milan et al., 2017] Anton Milan, Seyed Hamid Rezatofighi, Anthony R Dick, Ian D Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, pages 4225–4232, 2017.

[Min-je et al., 2017] Choi Min-je, Jeong Sehun, Oh Hakjoo, and Choo Jaegul. End-to-end prediction of buffer overruns from raw source code via neural memory networks. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 1546–1553, 2017.

[Mozer, 1992] Michael C Mozer. Induction of multiscale temporal structure. In Proceedings in the Advances in Neural Information Processing Systems, pages 275–282, 1992.

[Santoro et al., 2016] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In International Conference on Machine Learning, pages 1842–1850, 2016.

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

[Sutskever et al., 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[Tao et al., 2017] Lin Tao, Guo Tian, and Aherer Karl. Hybrid neural networks for learning the trend in time series. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 2273–2279, 2017.

[Vaswani et al., 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, et al. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, pages 6000–6010, 2017.

[Weston et al., 2014] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. arXiv preprint arXiv:1410.3916, 2014.

[Williams and Hinton, 1986] D R G H R Williams and Geoffrey Hinton. Learning representations by back-propagating errors. Nature, 323(6088):533–538, 1986.

[Yongqing et al., 2017] Wang Yongqing, Shen Huawei, Liu Shenghua, Gao Jinhua, and Cheng Xueqi. Cascade dynamics modeling with attention-based recurrent neural network. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 2985–2991, 2017.

[Zaremba et al., 2014] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.

[Zhu et al., 2017] Cao Zhu, Wang Linlin, and de Melo Gerard. Multiple-weight recurrent neural networks. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 1483–1489, 2017.

[Zilly et al., 2016] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. arXiv preprint arXiv:1607.03474, 2016.