Learning to Remember: Using Reinforcement Learning to Augment Neural Networks with Long-Term Memory

Neil Barrett
neil.barrett@genaiz.com
GENAIZ
Montreal, Québec, Canada

Abstract

Memory-augmented neural networks (MANN) are a subset of neural networks that have a read-write memory which can allow the network to access past information other than its current inputs. A new MANN architecture is explored herein. This MANN architecture, called MANN-RL, uses reinforcement learning (RL) to learn what a should be remembered. The MANN-RL architecture defines a neural network called the memory-net which produces the memory at time $t + 1$ given the memory at time t. The MANN-RL architecture is applied to a sequential logic problem, part-of-speech tagging and sentence memorization. This short exploratory work shows that the MANN-RL architecture may be a viable architecture for MANNs. It performed well (0.97 F1 and above) on three relatively simple sequence-based problems.

CCS Concepts

• **Computing methodologies → Reinforcement learning**; **Neural networks**; **Learning latent representations**.

Keywords

memory augmented, neural networks, reinforcement learning, latent representations

1 Introduction

Memory-augmented neural networks (MANN) are a subset of neural networks that have a read-write memory. A memory can allow the network to access information other than its inputs. For example, large language models (LLMs) [7] could encode, store and recall information beyond their input tokens. This might result in smaller LLMs (i.e., fewer parameters) having equivalent capabilities to larger LLMs, or LLMs being more capable without changing their size. Regardless of the potential impact of MANNs, it is interesting to explore MANN architectures and expand knowledge on these memory-enabled machines. This paper does the latter and explores a MANN-RL architecture: a MANN architecture that uses reinforcement learning (RL) [2] to learn what a network should remember.

This paper focuses on the MANN-RL architecture rather than presenting background work. Khosla et al. [4] introduce and discuss MANNs in depth in their survey. Readers are referred to their paper for an introduction to MANNs. Similarly, readers may browse Ding et al.'s book [2] for an introduction to RL.

MANN-RL is presented by first describing its architecture in Section 2. The training recipe for this architecture is then discussed in Section 3. This is followed by some applications of MANN-RL in Section 4. Readers should therefore, at the end of these sections, understand what MANN-RL is and how to apply it. Furthermore, Section 4 hopefully provides sufficient evidence to convince readers on the viability of MANN-RL.

2 Architecture

The MANN-RL architecture defines a neural network called the memory-net which produces the memory at time $t + 1$ given the memory at time t. The memory-net also takes an input and output at time t, which are problem dependent. For example, a memory-net for part-of-speech (POS) tagging [3] could take as input a word embedding and as output the predicted POS tag. This input and output provide visibility on the non memory-net computation of a neural network for which the memory-net is providing and updating a memory. The neural network that consumes the memory is hereafter called the action network[1]. The memory-net may be minimally defined as follows:

$$f_{mn}(m_t, in_t, out_t) = m_{t+1} \tag{1}$$

The memory, input and output are tensors such as a vector or embedding. Additional examples of input and out are

- a LLM which provides an input embedding representing the input tokens and output token,
- a robot which encodes its sensors as an input vector and encodes its actions (e.g., roll forward) as an output vector,
- or a neural network for machine translation [6] which provides the current input token and translated output token.

An interesting consequence of the above definition is that the memory-net is free to use existing neural network layers, activation functions, components and so forth. In other words, there is no explicit restriction on how f_{mn} is built. However, there may be problem-specific restrictions due to, for example, limited machine memory.

3 Training

The memory-net is trained using RL. RL requires a reward as a training signal. The action network errors are converted into memory-net rewards when training. In other words, MANN-RL uses the problem-dependent loss values calculated during training of the action network as rewards (negative of the loss) for training the memory-net. For example, the softmax loss values calculated during POS tagging are converted into rewards (negative of softmax loss) for training the memory-net.

The conversion of action network loss to memory-net reward assumes that the action network is trained on a supervised learning

[1]It is called the action network because it takes the action of producing a problem-dependent output. Although there are some similarities to RL actions, the action network should be understood as an independent concept from RL actions.

task. Training the action network differently (e.g., RL) would require mapping the action network training signal to a memory-net training signal. This is likely feasible but has not been tried, and is outside the scope of this paper.

Given the training signal described above, training is similar to Mnih et al. [5] and their training of a neural network to play Atari games. In particular, the memory-net is trained using Q-learning and experience replay:

- A history of memory-net training states is buffered during training. A memory-net training state is composed of the memory, input, output, correct output or target and the loss value at time t.
- During memory-net training, pairs $(t, t+1)$ of states are randomly selected to create a training batch.
- The memory-net is updated by calculating the Smooth L1 loss [2] of the actual and expected memories of the batch, and propagating the change via a gradient descent step. The actual memory is $f_{mn}(m_t, in_t, out_t)$. The expected memory is $\gamma * f_{mn}(m_{t+1}, in_{t+1}, out_{t+1}) - losses_t$, following from Q-learning.

It is worth noting that RL and Q-learning are typically applied to discrete environments and work on the probability over the discrete RL actions in the environment (e.g., a robot in a maze that is able to move left, right, up and down). In contrast, MANN-RL sees RL applied to a continuous memory. In some sense, each memory dimension is treated as an individual probability of a particular memory subconcept. This discrete legacy is embodied by each dimension of f_{mn} being constrained between $[0, 1]$ and the memory-net outputting the results of the sigmoid activation function.

Action network and memory-net training are coupled. There is a training *dance* that reduces to the following steps:

(1) Settle: Train the action network for a period (e.g., k batches) to allow the action network to learn the updated memory-net memories.
(2) Buffer: Buffer memory-net states for memory-net training, all the while training the action network.
(3) Train memory: Pause the action network training and perform memory-net training as described above.

4 Application

This section discusses various applications of MANN-RL. These applications were constrained by time and budget[3]. The training was performed on a MacBook Pro with Apple silicon in less than a month. These limitations result in interesting, but non-conclusive, evidence supporting the MANN-RL architecture.

Each subsection below describes an experiment: the data used, the action and memory networks, the training parameters and the performance outcome. It is important to understand that in each case, the action network sees only the current input and predicted output. The action network must therefore rely on the memory-net for any information that occurs in the past.

Each experiment presents the following parameters, defined once below for conciseness:

- history: number of states to retain for training the memory
- settle: number of time steps before buffering history
- epochs: number of training epochs
- batch size: action model training batch size
- loss: action model loss function
- optimizer: action model optimizer
- learning rate: action model learning rate
- clipping: action model clipping
- memory size: the size of the memory
- memory train iterations: number of times to train the memory network (at each pause)
- memory optimizer: action model optimizer
- memory learning rate: action model learning rate
- memory clipping: action model clipping
- memory samples: number of samples per training iteration
- memory clamp: restrict loss values to the bounds 0 to memory clamp before training the memory network
- gamma: gamma (RL)

The performance on the training set is reported since the goal is to quantify whether or not the MANN-RL architecture can learn the problem input-output mapping rather than its ability to generalize.

The code for these experiments may be found at https://github.com/nbam/neuralmemory.

4.1 Logic

A simple logic problem was devised for initial testing. The task was to repeat and solve a boolean logic problem specified by a sequential syntax. The operator occurs first followed by the operands. For example, the logical OR of one and zero produces the following input $o, r, 1, 0, =, |$ and output $., o, r, 1, 0, 1$. The action network was fed one symbol and produced one symbol at each time step (at time 0 the input is o and the output is .). The minimum number of operands was 2 and the maximum was 20.

The action network was a simple linear network composed of the following layers:

- linear
- batch norm
- relu
- linear
- softmax

The input and target characters were 1-hot encoded. The memory was used as is.

The memory-net was a simple linear network composed of the following layers:

- linear
- batch norm
- sigmoid

The memory-net input was the concatenation of the input fed to the action network, the action network's output and the current memory. The memory-net's output was the subsequent memory. The training configuration values for both networks are found in Table 1.

The MANN-RL architecture configured and trained, as described above, scored 0.99 F1 on the training set.

[2] https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html
[3] The creation and exploration of MANN-RL had to cost nothing.

Table 1: Logic training configuration

Name	Value
history	32
settle	8
epochs	512
batch size	16
loss	cross entropy
optimizer	adam
learning rate	0.0001
clipping	0.01
memory size	48
memory train iterations	2
memory optimizer	rmsprop
memory learning rate	0.00001
memory clipping	0.001
memory samples	8
memory clamp	10
gamma	0.99

Table 2: POS training configuration

Name	Value
history	16
settle	8
epochs	1024
batch size	16
loss	cross entropy
optimizer	adam
learning rate	0.0001
clipping	0.01
memory size	175
memory train iterations	2
memory optimizer	rmsprop
memory learning rate	0.00001
memory clipping	0.001
memory samples	8
memory clamp	10
gamma	0.99

4.2 Part-of-speech tagging

Existing data sets were used to learn a POS tagging task - predicting a POS tag given the current token (embedding) and any information remembered by the memory. POS tagging employed the Universal Dependencies Version 2 POS Tagged data accessed directly through python/torch[4] and the GloVe embeddings, also supplied with python/torch.

The action network was a simple linear network composed of the following layers:

- linear
- batch norm
- relu
- linear
- softmax

The input to the action network was a pre-computed word embedding and the memory as is. The target tags were 1-hot encoded.

The memory-net was a simple linear network composed of the following layers:

- linear
- batch norm
- sigmoid

The memory-net input was the concatenation of the input fed to the action network, the action network's output and the current memory. The memory-net's output was the subsequent memory. The training configuration values for both networks are found in Table 2.

The MANN-RL architecture configured and trained, as described above, scored 0.97 F1 on the training set. An informal assessment of the errors suggests that many errors are cases where there is annotator disagreement. A baseline AI that memorizes the most frequent POS tag given a token scored 0.92 F1, which demonstrates that the memory is providing important decisional information.

[4]https://torchtext.readthedocs.io/en/latest/datasets.html

4.3 Sentence encoding

A small portion of the WMT 2014 French-English news corpus [1] was used to train a model to read an input sentence, memorize it and reproduce the sentence as output. The sentence was fed to the network, and reproduced by it, character by character. Special characters were reserved to denote the beginning of a sentence, the end of the input sentence and beginning of the output reproduction and the end of all of this. Thus, while the network was reading the input, it was tasked with producing a character that represented an empty output and, when the network was reproducing the memorized sentence, a character representing the empty input was fed to the network.

The full WMT 2014 French-English news corpus was too big and training experiments too lengthy given the compute restriction of training on a MacBook Pro. Consequently, the shortest 500 sentences were selected. These were augmented with sequences where the input character was reproduced in the output for every non-special character (still following the empty input and output pattern above).

The input character (1-hot encoded) was converted to an embedding within the action and memory network. For both networks, the embedding size was 96 using the layers:

- linear
- batch norm
- relu

Furthermore, the memory-net encoded the action network's output character in an identical manner.

The action network was a linear network composed of the following layers:

- linear
- batch norm
- relu
- linear
- batch norm

Table 3: Sentence encoding training configuration

Name	Value
history	10
settle	6
epochs	2048
batch size	16
loss	cross entropy
optimizer	adam
learning rate	0.0001
clipping	0.01
memory size	1024
memory train iterations	1
memory optimizer	rmsprop
memory learning rate	0.0000001
memory clipping	0.001
memory samples	8
memory clamp	10
gamma	0.99

- relu
- linear
- softmax

The input was a word embedding learned during training and the memory as is. The target characters were 1-hot encoded.

The memory-net was a simple linear network composed of the following layers:

- linear
- batch norm
- sigmoid

The memory-net input was the concatenation of the input fed to the action network (encoded as an embedding), the action network's output (encoded as an embedding) and the current memory. The memory-net's output was the subsequent memory. The training configuration values for both networks are found in Table 3.

The MANN-RL architecture configured and trained, as described above, scored 0.97 F1 on the training set.[5]

5 Conclusion

This short exploratory work showed that the MANN-RL architecture may be a viable architecture for MANNs. It performed well (0.97 F1 and above) on three relatively simple sequence-based problems.

More so than other research, the short exploratory nature of this work leaves many open questions. I hope the research community will embrace and explore this work further. In particular, I'm curious to know the following: Are the outputs required for the memory-net to be effective? Can this work be extended to larger data sets and more difficult problems? Is there a more efficient training recipe for MANN-RL? What new or existing theories would provide a more complete foundation?

[5]A similar result could be obtained for English-French machine translation but isn't reported since it isn't compelling new evidence supporting the viability of MANN-RL.

References

[1] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia (Eds.). Association for Computational Linguistics, Baltimore, Maryland, USA, 12–58. https://doi.org/10.3115/v1/W14-3302

[2] Zihan Ding, Yanhua Huang, Hang Yuan, and Hao Dong. 2020. *Introduction to Reinforcement Learning*. Springer Singapore, 47–123. https://doi.org/10.1007/978-981-15-4095-0_2

[3] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing*. Prentice Hall, Upper Saddle River, New Jersey.

[4] Savya Khosla, Zhen Zhu, and Yifei He. 2023. Survey on Memory-Augmented Neural Networks: Cognitive Insights to AI Applications. (12 2023). arXiv:2312.06141 https://arxiv.org/pdf/2312.06141.pdf

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (12 2013). arXiv:1312.5602 https://arxiv.org/pdf/1312.5602.pdf

[6] Haifeng Wang, Hua Wu, Zhongjun He, Liang Huang, and Kenneth Ward Church. 2022. Progress in Machine Translation. *Engineering* 18 (2022), 143–153. https://doi.org/10.1016/j.eng.2021.03.023

[7] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*. https://openreview.net/forum?id=gEZrGCozdqR