Neural Turing Machines: An Overview

Bhuvan Singla*, Debaditya Bhattacharya*, Mohit Kulkarni*

* Equal Contribution

Abstract

One of the salient features of an intelligent agent is its ability to sort a continuous stream of incoming data appropriately (denoising & variable binding) and store this information in a way that facilitates easy retrieval, allowing the agent to make intelligent decisions. Artificial agents can be trained to do so with the addition of memory to their standard architecture.

This paper explores the Neural Turing Machine [1] as a memory augmented neural network. It is fully differentiable due to its unique addressing mechanism. In this paper, we build upon a previous open-source implementation of an NTM in PyTorch [2] to include a wider variety of tasks. We have also implemented GPU support. We have experimented with the initialisation mechanism to enable faster convergence of the model. We have also looked at some of the previous works which helped the NTM to be developed and various other future models that have taken inspiration from the NTM.

The Neural Turing Machine

Neural Turing Machine is a fully differentiable implementation of a Memory Augmented Neural Network. The architecture of NTM consists mainly of four components: Controller, Read Head(s), Write Head(s), and Memory, along with the input and the output. The controller is a feed-forward network or LSTM that interacts with the working memory using the read and the write heads to store/retrieve representations into/from memory. The reads and writes to the memory are rapid, and the representations are read/written at potentially every time step. NTMs are capable of long-term storage (because of slow updates of weights) and short-term storage (because of external memory), making them ideal for meta-learning and low-shot prediction.

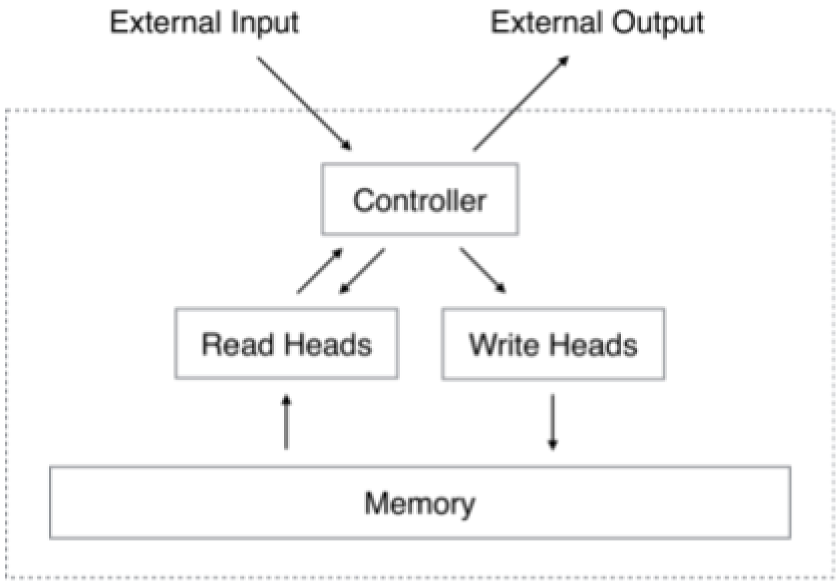


Figure 1: Neural Turing Machine Architecture.

Reading Mechanism

Let M be the memory matrix, with R rows and C columns and M_t be the state of M at time t. We also need an attention mechanism which is a length-R normalised weight vector w_t. Since these weightings are normalised, these two constraints hold:

$$\sum_i w_t(i) = 1 \qquad 0 \le w_t(i) \le 1, \forall i$$

The read vector r_t is a length-C vector, returned by the head, defined as:

$$r_t \leftarrow \sum_i w_t(i) M_t(i)$$

The read vector is differentiable with respect to both memory and weighting.

Writing Mechanism

The writing mechanism involves two steps: (i) Erasing and (ii) Adding.
To erase data, the write head requires a length-C erase vector e_t and the length-R normalised weight vector w_t, and M is then modified as:

$$\tilde{M}_t(i) \leftarrow M_{t-1}(i)[1 - w_t(i)\mathbf{e}_t]$$

After the erase step, the write head uses the length-C add vector a_t to complete the writing as:

$$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)\mathbf{a}_t$$

Addressing Mechanism

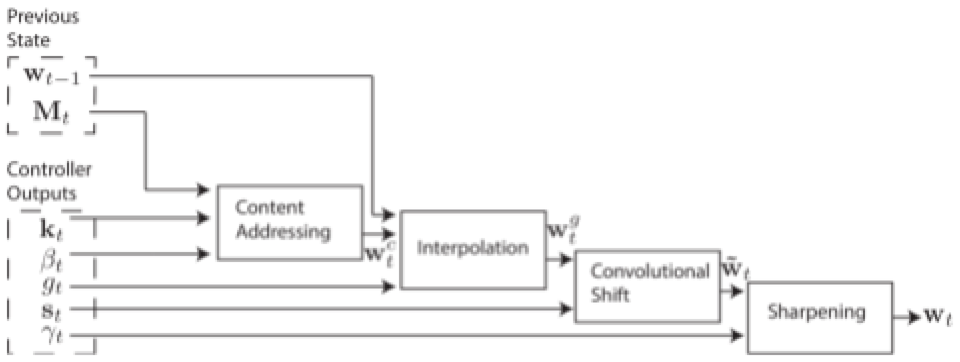


Figure 2: Flow Diagram of the Addressing Mechanism

The addressing mechanism is a four-stage process as represented in the Flow Diagram of Figure 2. Each stage generates an intermediate weight vector that is passed onto the next step.

The first stage generates a weight vector referred to as the content weight vector (w_t^c) based on the similarity of each row in memory with the length-C vector k_t that is emitted by the controller. This content weight vector enables the controller to select values similar to previously seen values i.e. *content-based addressing*. The content weight vector is calculated as follows:

$$w_t^c(i) = \frac{exp(\beta_t K(k_t, M_t(i))}{\sum_j exp(\beta_t K(k_t, M_t(j)))}$$

Where β_t is a positive scalar parameter called the key strength and determines how concentrated the content weight vector should be. K is a similarity measure, which in the case of the original paper is cosine similarity.

To implement the *location-based addressing*, three more stages are needed.

The second stage blends the content weight vector w_t^c with the previous time step's weight vector w_{t-1} to produce the gated weighting w_t^g using a scalar parameter $g_t \in (0, 1)$, called the interpolation gate.

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}$$

By doing this, we enable the system to learn when to use content-based addressing and when to ignore.

To enable the controller to shift focus to other rows, in the third stage, each head emits a normalised shift weighting s_t and performs a convolutional shift to produce shifted weight $\tilde{w}_t$.

$$\tilde{w}_t(i) \leftarrow \sum_{j}^{R-1} w_t^g(j) s_t(i - j)$$

In the fourth stage, the shifted weight is sharpened to prevent it from blurring.
Where $\gamma \geq 1$ is a scalar.

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

This outputs the weight vector.

Controller

The controller is a neural network. It may be a recurrent network or a feedforward network. Recurrent controllers like LSTM have benefits because they also have an internal memory that complements the external memory matrix. In comparison, feedforward network controllers benefit by having greater transparency because the pattern of reading from and writing to the memory matrix is easier to interpret than the internal state of an RNN.

Tasks, Experiments, and Improvements

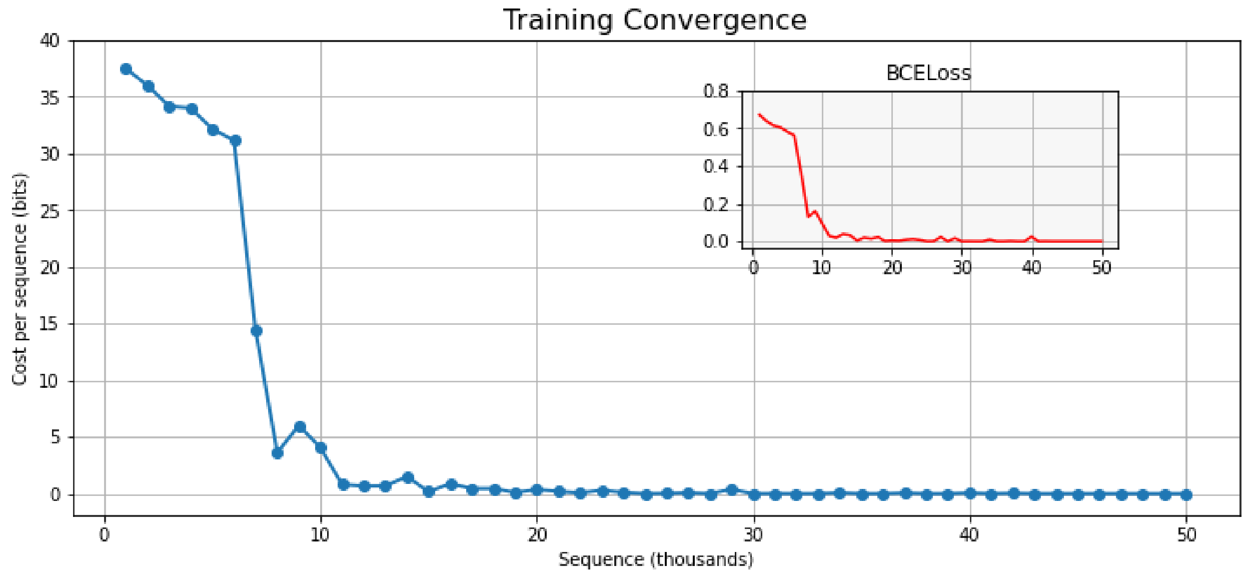
Tasks

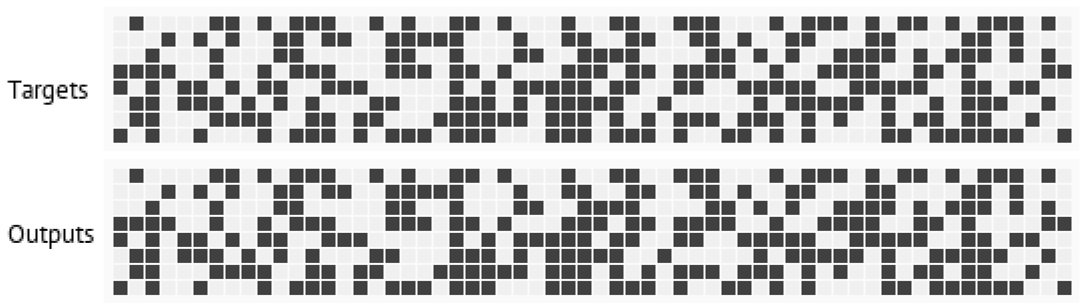
The end goal of an NTM is to be able to produce intelligent behaviour through good performance and generalisation in a task. How do we get the Neural Turing Machine to learn a task? We feed it a particular task that has been encoded in the form of input and output behaviour. The task that we have attempted to replicate are:

1) Copy Task
2) Repeat Copy Task
3) Associative Recall (Novel implementation)
4) Priority Sort (Novel implementation)
5) Lexicographic Sort (Novel implementation)

• Copy Task

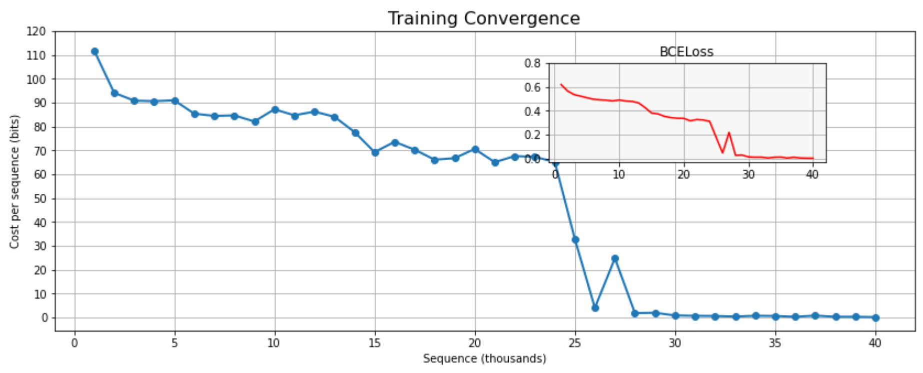
For the copy task, we require the network to output a sequence similar to the one it received as an input. The input sequence is a vector of random bits followed by the end of the sequence marker. This task requires the network to learn two things, one, to start output at the beginning of the sequence, and two, to move along the sequence and output that. In our experiments, we see that our network converges pretty well and also generalises to longer sequences.
GitHub: https://github.com/m2kulkarni/pytorch-ntm/blob/master/tasks/copytask.py

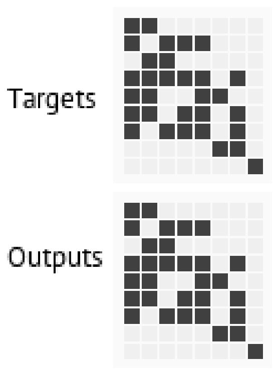




Repeat Copy Task

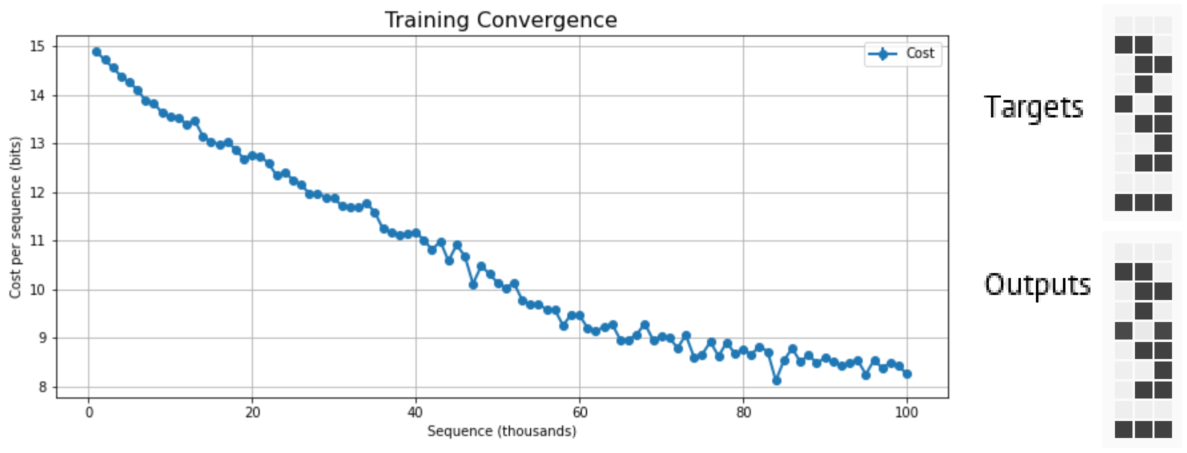
Here we try to make our network learn to output a fixed number of reparations of the input sequence. The input has random sequences followed by a value that represents the number of repetitions the network should output of the input sequence.
GitHub:
https://github.com/m2kulkarni/pytorch-ntm/blob/master/tasks/repeatcopytask.py





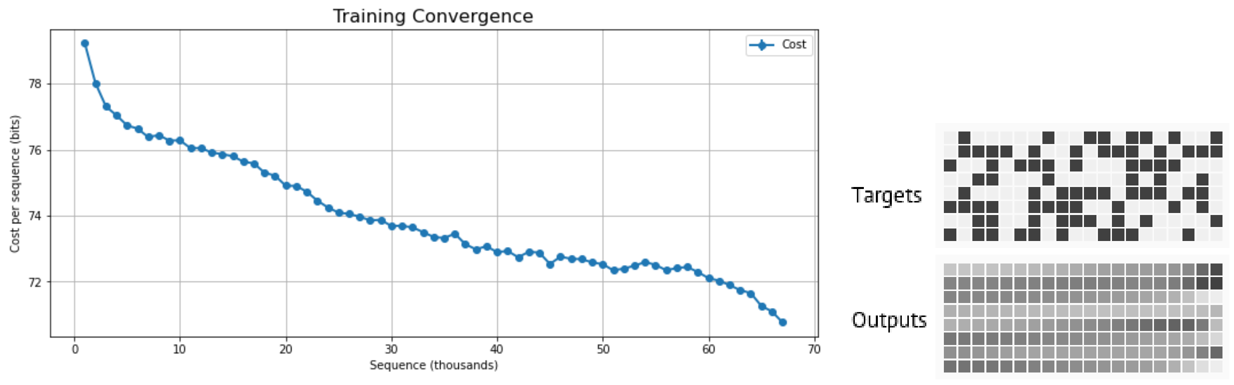
Associative Recall

Here our input is a sequence of binary vectors bounded on the left and right by delimiters. We want our network to output the next item of the query. This means that the network should learn a list of input blocks that point to another block.
GitHub: https://github.com/m2kulkarni/pytorch-ntm/blob/master/tasks/recall.py



- Priority Sort

In this task, we assign a random priority from [-1,1] to each sequence and see if NTM can sort our data according to the priority weights assigned. We see that NTM can sort the data, showing that our model can traverse the memory locations according to a predefined priority order.
GitHub: https://github.com/m2kulkarni/pytorch-ntm/blob/master/tasks/psort.py



We can see that our implementation of the priority sort task converges, but the task could not be run until full convergence due to resource constraints.

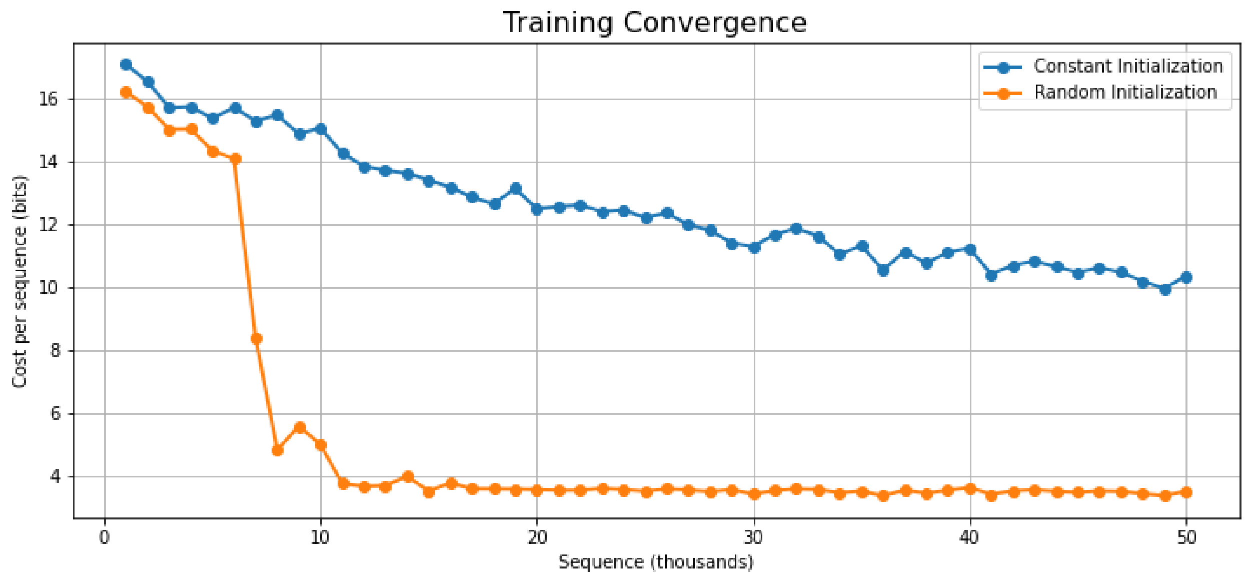
- Lexicographic Sort

Here we try to sort our input not according to a random priority but according to the number of bits in each input sequence. Due to technical resource constraints, we were not able to train this but have implemented the code.
GitHub: https://github.com/m2kulkarni/pytorch-ntm/blob/master/tasks/sort.py

Experiments

- ### Initialisation (Random vs Constant)

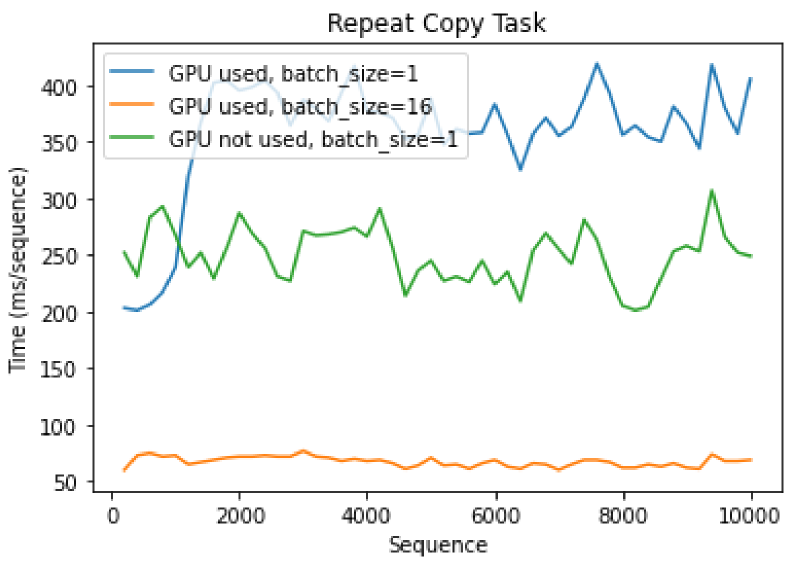
Initial experiments show a worse converging rate for const initialisation of Memory. These results were different from what we were expecting as per [3].



Improvements

- ### GPU

A Neural Turing Machine requires a lot of episodes to converge for tasks and is time-intensive. We have structured and modified the existing code and added CUDA support, allowing the model to be trained on the GPUs. NTM in itself is sequential and, therefore, challenging to parallelise since the functions are dependent on the next step. Increasing batch_size gives it some room for parallelisation, and therefore we can see improvement in the time/sequence for larger batch size.



Why a Neural Turing Machine?

In 1945, Von Neumann proposed his computing architecture (a.k.a, the Princeton architecture). The specific tasks involved in the Von Neumann Architecture

1) I/O
2) Control and logic
3) External memory. <- Ignored in Artificial Neural Networks

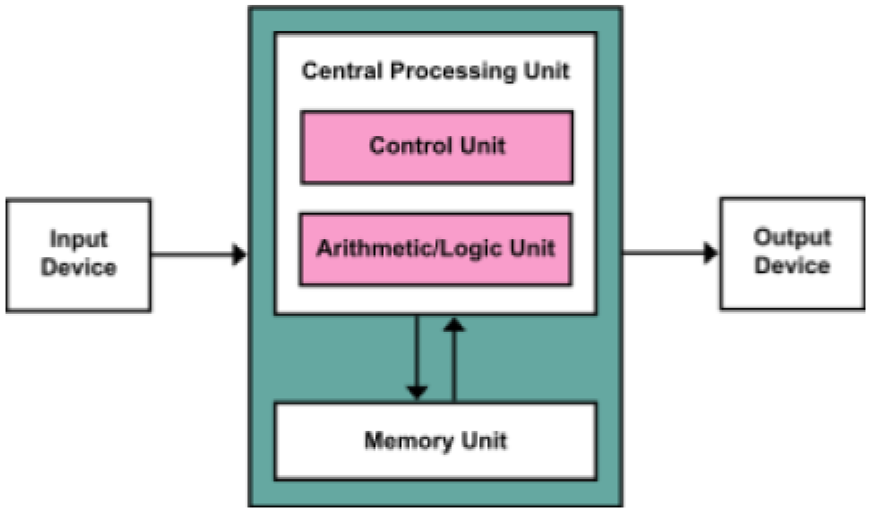


Figure 3: Von Neumann Architecture (Source Wikipedia)

Development in Artificial intelligence has given rise to biologically inspired neural networks. Some of the current state-of-the-art networks, such as RNNs, have demonstrated extremely good control, logical, and arithmetic abilities. However, no memory has been associated with the same. What remained was to connect it up to external memory.

Von Neumann's architecture was inspired by a plausible architecture of the brain. The next logical step is to implement external memory. While designing an external memory controller scheme, taking inspiration from nature, we try to build a Working Memory, implemented as a Long Term Working Memory. Providing a unique addressing mechanism in which cue guides association (focus) and similarity in location (spatial organisation) give us a unique memory interface. This allows it to have a large capacity and amorphous storage and recall methods, making it different from traditional computer memory (RAM). [4]

The turing enhancement for neural networks is: "Add Working memory!" [5]

Literature Review

A broad goal of research in artificial intelligence is the pursuit of a general-purpose computational machine. It is quite natural to turn to our only currently known example of an entity gifted to us by nature for inspiration. The brain (or the mind) can perform such general-purpose computations, and we attribute this ability to the brain being intelligent. To mimic intelligence on silicon, we attempt to create cognitive architectures which try to encapsulate our understanding of the metaphor: The brain is an information processing system. With increased computational power and a large amount of data, neural networks have produced intelligent behaviour.

One of the salient features of an intelligent agent is its ability to sort a continuous stream of incoming data appropriately (denoising & variable binding) and store this information in a way that facilitates easy retrieval, allowing the agent to make intelligent decisions.

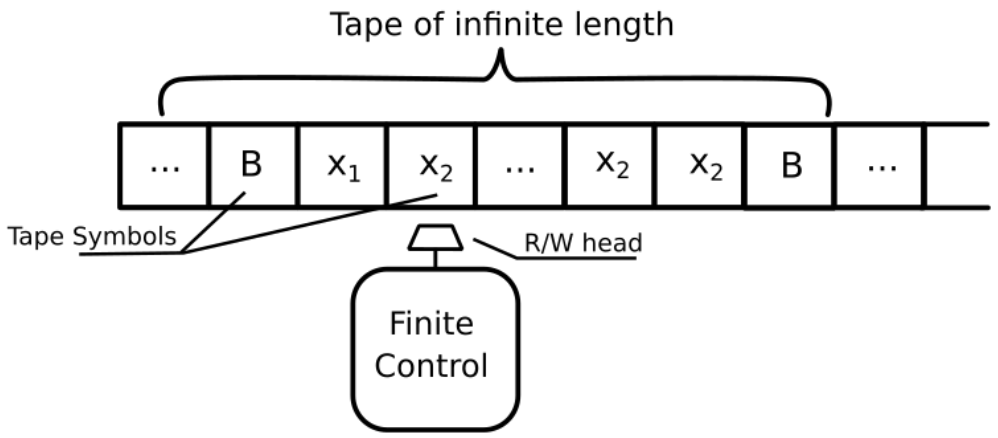
In recent years neural networks which were initially inspired from the brain have been immensely successful in performing 'intelligent' tasks. Taking further inspiration, attempts have been made to accommodate memory alongside the neural network. This is to increase the knowledge base of the network without increasing the size of the network.

We have tried to organise in a chronological order various attempts at memory augmented machines that have led to the NTM, and the future developments developed further due to the NTM.

Previous attempts:

- ### Turing Machine

The Turing machine is an abstract computer proposed by Alan Turing in the early 1930s with a simple architecture consisting of a finite control automaton equipped with a tape divided into squares. The squares hold tokens of a finite set of symbols. The tape is, in principle, of infinite length, and the automaton moves up and down it writing and erasing symbols according to instructions contained in a machine table. Cognitive scientists have used it to explain the human mind's processes. It was shown that most problems could be broken down so that they can be implemented by the Turing machine in finite time, suggesting that the Turing machine, a finite controller with infinite memory, is a good model for an intelligent system. [6]

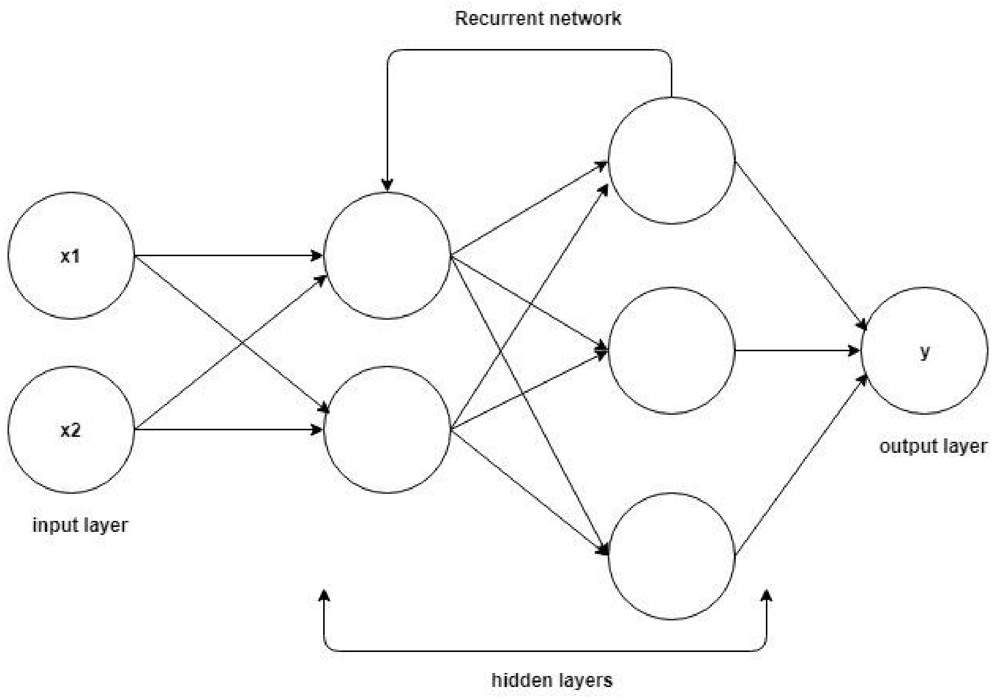


- ### RNN

"A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence."[7] It is able to remember the past inputs, and its decisions are influenced by what it has learned from the previous inputs.

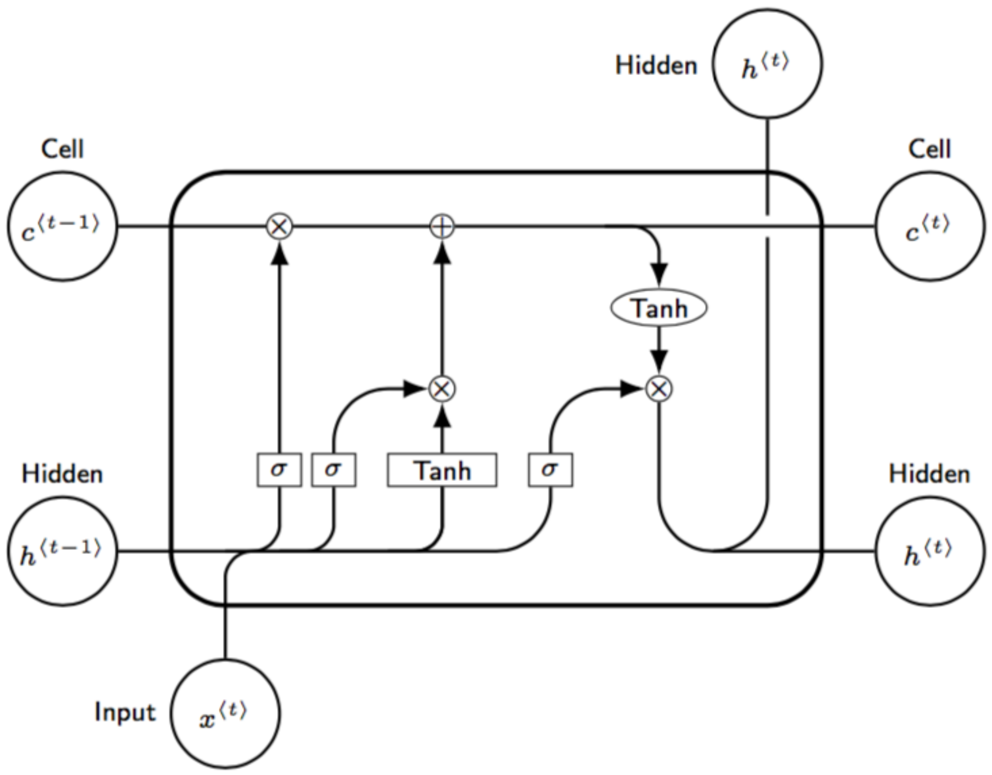
Recurrent neural networks (RNNs) achieve this by modelling the abundant recurrent connectivity among neurons in the cortex. Recently, RNN's have been used in computational neuroscience literature to model high recurrence in cortical areas of the human brain to explain cognitive processes like perception, memory, attention etc. [8]

AN RNN is an improvement over classical feedforward neural networks as it has a recurrent input that allows for maintaining a very simple internal state (and thus having memory. And have the ability to "remember" what is essential. [9]



- LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. These networks have hidden states which are passed on to the next iteration. This allows the LSTM to function like a Hidden Markov Model. This hidden state function enables the LSTM to learn long-term dependencies. Due to its recursive nature and enhanced architecture, LSTMs outperform RNNs in many tasks. LSTMS have also shown robust performance in tasks which involve manipulation of sequential data such as speech or video. [10] [Cite: https://en.wikipedia.org/wiki/Long_short-term_memory]



The Neural Turing Machine

Neural Turing Machine is a fully differentiable implementation of a Memory Augmented Neural Network. The architecture of NTM consists mainly of four components: Controller, Read Head(s), Write Head(s), and Memory, along with the input and the output. The controller is a feed-forward network or LSTM that interacts with the working memory using the read and the write heads to store/retrieve representations into/from memory. The reads and writes to the memory are rapid, that and the representations are read/written at potentially every time step. NTMs are capable of long-term storage (because of slow updates of weights) and short-term storage (because of external memory), making them ideal for meta-learning and low-shot prediction.

Future Attempts:

- ### Neural Random Access Machine:

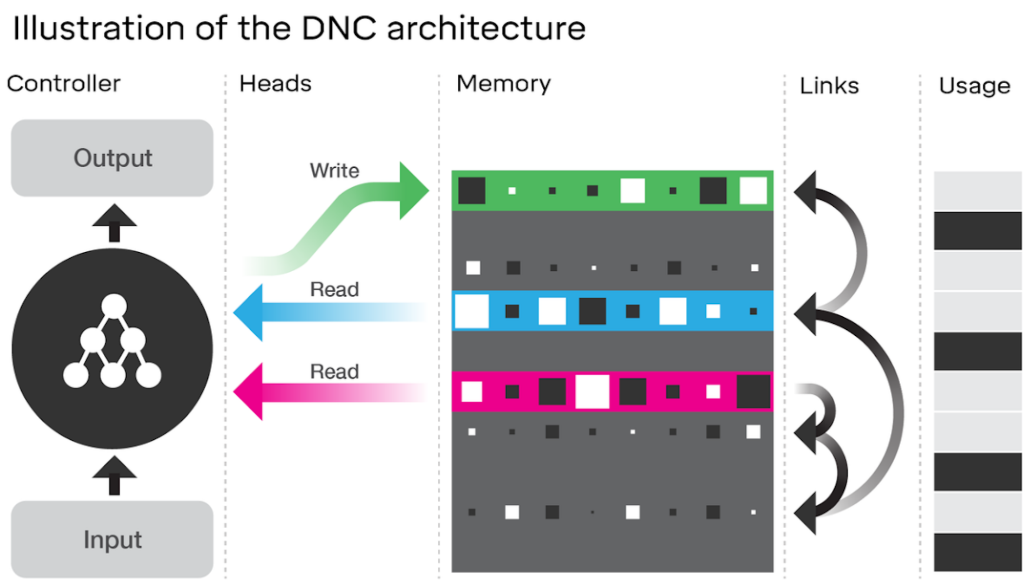
With the demonstrated success of Neural Turing Machines, people have attempted to modify the read and write mechanisms. One such attempt is the Neural Random Access Machine. In a classical computer, pointer manipulation is a very critical feature of the processor. The paper introduces a novel addressing mechanism involving pointer manipulation. This allows the machine to be connected to a variable-sized external memory. The paper has demonstrated a successful ability to manipulate, store in memory and dereference points into its working memory. [11]

- Neural GPU

Neural GPU is based on a type of convolutional gated recurrent unit (cGRU) (Unlike NTMs, which utilise a feedforward or LSTM based controller) and, like the NTM, is Turing complete. Unlike the NTM, the Neural GPU is highly parallel, making it easier to train and efficient to run. The paper shows a remarkable learning experience where the model is able to generalise tasks from a small input to any large arbitrary input. [12]

- Differentiable Neural Computer

DNC is also a kind of Memory Augmented Neural Network where the controller acts analogous to a computer processor. In addition to NTM, DNC contains a memory attention mechanism that uses temporal attention to record the order of events and controls data storage in memory. Each subcomponent of the model is differentiable, hence making it end-to-end differentiable. [13, 14]



Code Structure

NTM Architecture

- NTM:
  - LSTM Controller
  - NTMReadHead
  - NTMWriteHead
  - NTMMemory

Classes

- NTM
  - __init__:
    - self
    - num_inputs
    - num_outputs
    - controller
    - memory
    - heads
  - create_new_state
  - reset_parameters
  - forward
- LSTMController
  - __init__
    - self
    - num_inputs
    - num_output
    - num_layers
    - lstm
    - lstm_h_bias
    - lstm_c_bias
  - create_new_state
  - reset_parameters
  - size
  - forward
- NTMMemory
  - __init__
  - read
  - address
  - _similarity

    - __interpolate
    - __shift
    - __sharpen
    - reset
- NTMReadHead
    - ___init___
    - create__new__state
    - reset__parameters
    - forward
- NTMWriteHead
    - ___init___
    - create__new__state
    - reset__parameters
    - forward

References

[1] Graves, A., Wayne, G. & Danihelka, I. (2014). Neural Turing Machines (cite arxiv:1410.5401)

[2] Neural Turing Machines (NTM) - PyTorch Implementation, GitHub

[3] Mark Collier and Jöran Beel (2018). Implementing Neural Turing Machines. *CoRR, abs/1807.08518.*

[4] Ericsson, K. A., & Kintsch, W. (1995). Long-term working memory. *Psychological Review, 102*(2), 211–245. https://doi.org/10.1037/0033-295X.102.2.211

[5] Graves, A., Wayne, G., Reynolds, M. *et al.* Hybrid computing using a neural network with dynamic external memory. *Nature* 538, 471–476 (2016). https://doi.org/10.1038/nature20101

[6] Wells A.J. (2004) Cognitive Science and the Turing Machine: an Ecological Perspective. In: Teuscher C. (eds) Alan Turing: Life and Legacy of a Great Thinker. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-05642-4_11

[7] In Wikipedia, Retrieved May 4, 2021 from https://en.wikipedia.org/wiki/Recurrent_neural_network

[8] Bitzer, S. and Kiebel, S. J. Recognizing recurrent neural networks (rRNN): Bayesian inference for recurrent neural networks. Biological Cybernetics, 2012, doi.

[9] Mahendran Venkatachalam (Mar 1, 2019) Recurrent Neural Networks Remembering what's important [Blog Post]. Retrieved from https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[10] In Wikipedia, Retrieved May 4, 2021 from https://en.wikipedia.org/wiki/Long_short-term_memory

[11] Karol Kurach, Marcin Andrychowicz, & Ilya Sutskever. (2016). Neural Random-Access Machines. https://arxiv.org/abs/1511.06392

[12] Łukasz Kaiser, & Ilya Sutskever. (2016). Neural GPUs Learn Algorithms. https://arxiv.org/abs/1511.08228

[13] Graves, A., Wayne, G., Reynolds, M. et al. Hybrid computing using a neural network with dynamic external memory. Nature 538, 471–476 (2016). https://doi.org/10.1038/nature20101

[14] Francesco Cicala (May 28, 2018) Differentiable Neural Computers: An Overview. Retrieved from https://towardsdatascience.com/rps-intro-to-differentiable-neural-computers-e6640b5aa73a