Neural Turing Machines

Jake Grigsby

University of Virginia

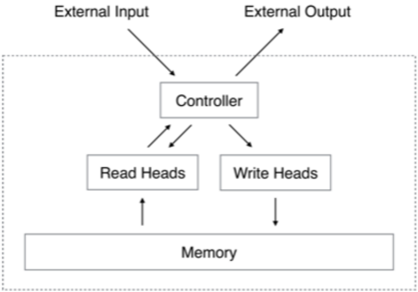
April 19, 2020

Introduction

- Neural Turing Machines [1] (NTMs) aim to equip neural networks with external memory
- They have a memory matrix that they can read and write from at each timestep
- This reading and writing process is fully differentiable and can be trained with gradient descent
- Similar to Software 2.0 [2] but a much less constrained program search.

NTM Overview

NTMs broadly consist of an LSTM 'Controller' network and several Read/Write Head networks. The controller takes the task input and produces an output, while the read and write heads interact with the external memory.



External Memory

- The memory is just an *NxM* array
  - *N* is the number of memory locations
  - *M* is the size of each address
- The contents of the memory at time t are written M_t

Reading from Memory

At each time t, a Read Head network calculates a weight vector $\mathbf{w_t}$ s.t.

$$\sum_i \mathbf{w_t}[i] = 1, 0 \leq \mathbf{w_t}[i] \leq 1, \forall i$$

The final read vector r_t is calculated:

$$\mathbf{r_t} = \sum_i^{N-1} \mathbf{w_t}[i] M_t[i, :]$$

This is basically an attention weighting over the memory, which was a technique that was just starting to take off in 2014...

Reading from Memory

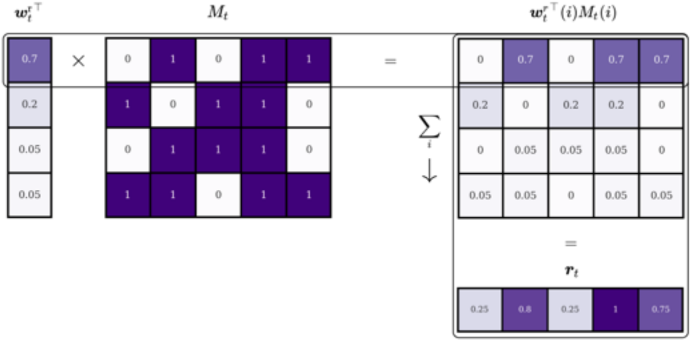


Figure: Reading mechanism diagram [4]

Writing to Memory

Two steps:

1. Erase parts of the memory
   - the Write Head calculates another weight vector $\mathbf{w_t}$ along with an erase vector $\mathbf{e_t}$ with each elements in $(0, 1)$

$$M_t[i, :] = M_{t-1}[i, :] - (M_{t-1}[i, :])(\mathbf{w_t}[i]\mathbf{e_t})$$

     When the weighting and erase element are 1, we're wiping the memory. If either are 0, nothing is changed.

2. Add new data to the memory
   - the Write Head also outputs an add vector $\mathbf{a_t}$, which is partially inserted into memory:

$$M_t[i, :] = M_t[i, :] + \mathbf{w_t}[i]\mathbf{a_t}$$

Addressing Mechanisms

- The location in memory where we read and write is determined by the weight vectors $\mathbf{w_t}$ of each head.
- We want to be able to focus on different addresses based on both what's in that address and where it is in memory.

Focusing by Content

The 'content weighting' $\mathbf{w_t^c}$ is determined by similarity of a key vector $\mathbf{k_t}$ and each row of memory.

$$\mathbf{w_t^c}[i] = \frac{exp(\beta_t \mathbf{K}[\mathbf{k}_t, M_t[i,:]])}{\sum_j exp(\beta_t \mathbf{K}[\mathbf{k}_t, M_t[j,:]])}$$

Where $K[\cdot,\cdot]$ is some similarity measure. Original paper uses cosine similarity. β_t is the 'key strength' (another trainable param)

- This concept shows up often in modern NLP models with attention

Focusing by Location

Once we've focused based on content ($\mathbf{w_t^c}$) we can adjust this weighting based on the location of each address, if the problem requires it.

- If the program we're trying to learn is addition, we care about where the numbers are stored in memory, not what they are.
- Many programs need to find some content (like an object in Python), and then index into a particular attribute of that object.

There are a few steps...

Focusing by Location

First we choose how much of the content-based weighting we care about using an 'interpolation gate' g_t

$$\mathbf{w_g^t} = g_t \mathbf{w_t^c} + (1 - g_t)\mathbf{w_{t-1}}$$

So if the gate is zero we don't care about the content weighting at all and just revert to w_{t-1}

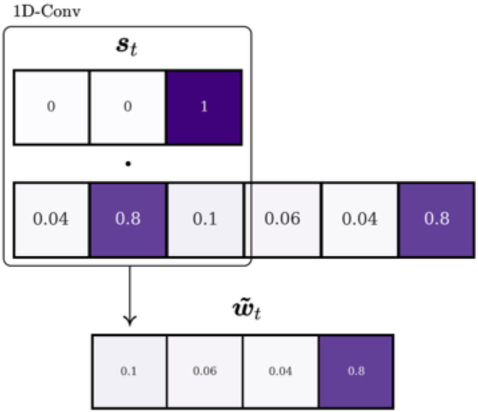
Focusing by Location

Next, we use a 'shift kernel' s_t

$$\mathbf{w_t}[i] = \sum_{j=0}^{N-1} \mathbf{w_t^g}[j]\mathbf{s_t}(i-j)$$

Which is just a circular 1D convolution over the weight vector. If we want to shift by a max of n slots, then $\mathbf{s_t}$ is a length $2n+1$ kernel.

Focusing by Location



Example of a simple 'shift backwards' kernel. We've made $\mathbf{w_t}$ circular by appending the last element to the front and the first element to the back [4].

Focusing by Location

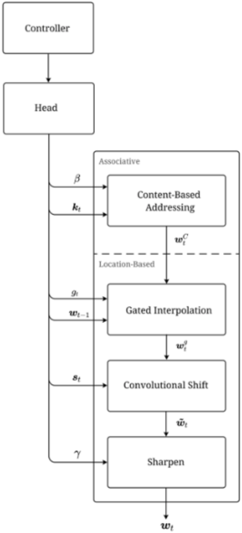
Finally, we 'sharpen' the focus of $\mathbf{w_t}$ with another trainable parameter γ_t:

$$\mathbf{w_t}[i] = \frac{\mathbf{w_t}[i]^{\gamma_t}}{\sum_j \mathbf{w_t}[j]^{\gamma_t}}$$

Addressing Summary

In summary, we calculate $\mathbf{w_t}$ by:

1. Weighting based on content similarity
2. Forgetting parts of that content we don't want to use
3. Shifting the focus based on location
4. Sharpening the focus vector to adjust the scope of the changes



The Controller

The controller is typically an LSTM that takes the task's input $\mathbf{x}$ and outputs an 'instruction vector' that is sent to each of the read and write heads.

The Neural Turing Machine Cell

A summary of the whole process:

1. The task input $\mathbf{x_t}$ is passed to the controller along with all the $\mathbf{r}_{t-1}$s
2. The LSTM controller returns an instruction vector $\mathbf{h_t}$.
3. $\mathbf{h_t}$ is passed to each of the read and write heads, along with that heads' $\mathbf{w_{t-1}}$
4. Write Heads use their networks to turn $\mathbf{h_t}$ and $\mathbf{w_{t-1}}$ into ($\mathbf{k_t}$, β_t, g_t, $\mathbf{s_t}$, γ_t, $\mathbf{e_t}$, $\mathbf{a_t}$), and use these to calculate $\mathbf{w_t}$ and write to memory.
5. Read Heads use their networks to turn $\mathbf{h_t}$ and $\mathbf{w_{t-1}}$ into ($\mathbf{k_t}$, β_t, g_t, $\mathbf{s_t}$, γ_t), and use these to calculate $\mathbf{w_t}$ and read from memory (r_t).
6. A final network computes the output $\hat{\mathbf{y}}_\mathbf{t}$ from the controller output $\mathbf{h_t}$ and all of the read vectors.

Experiments: Copy Task

Given a binary sequence $\mathbf{x}_{\{0:T\}}$, output an exact copy. [1]

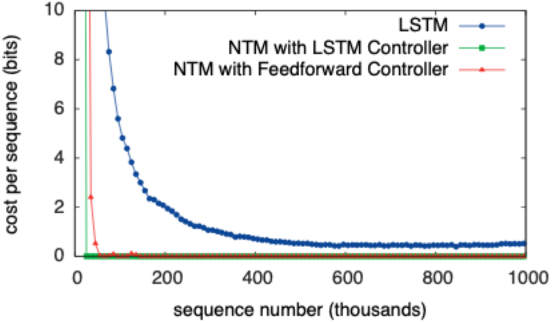
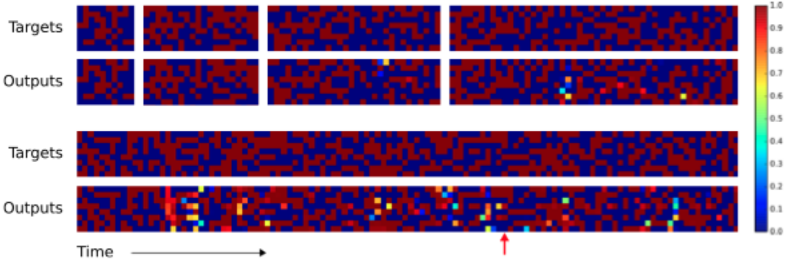


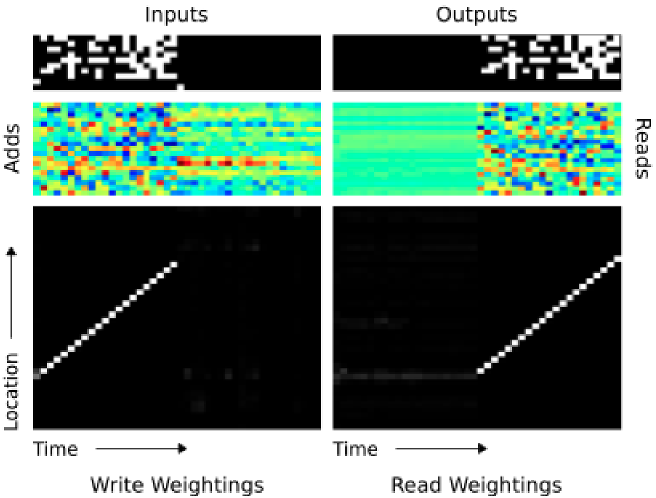
Figure 3: Copy Learning Curves.

Experiments: Copy Task



Sample outputs on the copy task, with errors highlighted. [1]

Experiments: Copy Task



Read and write activity during copy task. [1]

More Experiments, Shortcomings

- The original paper [1] includes more examples of simple programs NTMs can learn.
  - That section is pretty easy to follow once you have the general idea behind the training loop and the format of these diagrams.
- Why aren't we all using NTMs?
  - They are difficult/unstable to train [4] [3]
  - Code was never released by the orignal authors, and the paper is so light on details about how the model is actually trained that it took 4 years [3] to figure out a correct open source implementation!

References I

- Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: arXiv preprint arXiv:1410.5401 (2014).
- Andrej Karpathy. Medium. 2017. URL: https://medium.com/@karpathy/software-2-0-a64152b37c35.
- Mark Collier and Jöran Beel. "Implementing Neural Turing Machines". In: CoRR abs/1807.08518 (2018). arXiv: 1807.08518. URL: http://arxiv.org/abs/1807.08518.
- Niklas Schmidinger. Niklas Schmidinger: Exploring Neural Turing Machines. 2019. URL: www.niklasschmidinger.com/posts/2019-12-25-neural-turing-machines/.