Neural Turing Machines

David Nelson [drnelson]
CIS700 Neural Program Learning
Semester Project
05/14/2021

Abstract

This paper discusses the design of a Neural Turing Machine (NTM), as described by Alex Graves, Greg Wayne, and Ivo Danihelka [GWD14][1]. The paper begins by describing the components of an NTM and continues by describing the different operations performed by an NTM during the training process, along with context for why the different operations are performed.

This paper does not include any analysis of results at the present time because at the moment I do not have a working implementation of a full NTM. Instead, the code that I've provided implements some of the operations that an NTM would use during training.

Future work related to this project would be to leverage PyTorch[2] to implement the remaining neural components to bring the code that I have in place up to a full-fledged NTM.

Link to GitHub repo: https://github.com/drnelsoniv/CIS700_NPL_NTM

[1] [GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014. https://arxiv.org/pdf/1410.5401.pdf

[2] PyTorch [Computer software]. Retrieved from https://pytorch.org

Introduction

A Neural Turing Machine is a type of inductive neural network architecture which accepts input/output pairs as input and results in a network which can then be leveraged to produce outputs based on previously unseen inputs. More specifically, by providing a series of matrix pairs, where each pair is composed of a matrix representing an input memory at a time $t = 0$, along with a matrix representing the goal output memory at a later time t^+, a neural network can be trained such that supplying a new, unseen input to the neural network will result in generating the appropriate output matrix representing the memory associated with the intended goal. As an example, an NTM could be trained with input/output pairs representing the memory state of an unsorted list as input, along with a sorted list as output. From that training data, the NTM can infer that the goal is to take an unsorted list as input and produce a sorted list as the output. The NTM is not limited to strictly to sorting, rather the NTM can be programmed to complete a variety of algorithmic tasks, limited only according to the ability to represent the necessary input/output pairs necessary to complete the objective.

Architecture

The NTM is a neural network architecture which features a large, addressable memory, in combination with a Long-Short Term Memory (LSTM) [HS97][3] neural network which handles controlling how that memory is read from and written to. In that sense, an NTM can be thought of as extending a standard LSTM with a large-addressable memory bank capable of storing arbitrary-length input and providing controls for accessing or manipulating that data over time with the intent of producing the correct output according to the algorithm it was programmed to perform.

There are two main components to an NTM, the external memory bank and the LSTM controller which handles how to read and write from memory. The external memory is presented as an N x M matrix, where N represents the number of memory locations available and M represents the data elements available at that memory location. As an example, a 3 x 5 memory bank represents 3 memory locations with 5 data elements at each memory location. The following is an example of a possible memory configuration:

	M[0]	M[1]	M[2]	M[3]	M[4]
N[0]	0	1	0	1	0
N[1]	1	1	1	1	1
N[2]	**0**	**1**	**1**	**0**	**0**

*An example of a possible memory matrix. In this example, memory location 2, denoted **N[2]** contains the data **0 1 1 0 0**.*

Read and write heads are responsible for obtaining data and modifying data in memory. These read and write heads target specific memory locations according to a 1 x N weight matrix whose purpose is to determine which memory location is being accessed by this particular head. The weight matrix represents a probability distribution over all memory locations where a higher probability leads to more information retrieved during a read operation, and a higher degree of modification during a write operation. This also means that an NTM may read from and write to multiple memory locations during a single time step to produce the targeted read/write output.

[3] [HS97] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

Reading from Memory

$$r_t = \sum_i w_t(i) M_t(i) \qquad \text{[GWD14, p. 6, eq. 2]}$$

Data read from memory at time t is defined as the matrix product of the weight matrix w_t (1 x N) representing the memory locations being accessed with the memory matrix M_t(N x M). This yields a 1 x M matrix, r_t representing the data read from memory.

Writing to Memory

Writing to memory in an NTM is composed of two separate operations, an erase, followed by an add. Erase is defined as:

$$\tilde{M}_t(i) = M_{t-1}(i)[1 - w_t(i)e_t] \qquad \text{[GWD14, p. 6, eq. 3]}$$

It can be seen from the equation that if $w_t(i)$ or e_t is 0, then $\tilde{M}_t(i)$ retains the previous value in memory, $M_{t-1}(i)$. As a result, this operation only erases data if both the memory location weight matrix $w_t(i)$ and the erase vector e_t are both nonzero.

Following the erase operation, an add operation is performed:

$$M_t(i) = \tilde{M}_t(i) + w_t(i)a_t \qquad \text{[GWD14, p. 6, eq. 4]}$$

This operation starts with the memory following the erasure above, and then adds in the result of $w_t(i)a_t$. This yields a matrix with the same layout as memory whose additional data is determined by the memory location weight matrix $w_t(i)$ and the add vector a_t.

Determining the Memory Location (Weight) Matrix

The address determined for a read or write operation is determined by two modes of addressing, content-based and location-based. Content-based addressing refers to calculating a location in memory based on the data in that location, using a subset of the data to compare against. One such example could be to use the substring “drn” to locate the full data element “drnelson” in memory, and use its location. Location-based addressing refers directly to the address in memory. NTM uses the composition of both of these addressing modes when determining the weight matrix corresponding to the memory location to read or write.

Content-based Weight Matrix

$$w_t^{\,c}(i) = \frac{exp(\beta_t K[k_t, M_t(i)])}{\sum_j exp(\beta_t K[k_t, M_t(j)])}$$ [GWD14, p. 8, eq. 5]

Content-based addressing is performed by calculating a weight matrix according to the equation above.

- β_t- The key strength to apply for this particular content-based lookup.
- $K[k_t, M_t(i)]$- The cosine similarity of the key matrix (data subset), k_t, and the memory matrix at time t, M_t.

The result is divided by the sum to produce a probability distribution calculating the relative weight which should be applied to each memory location.

Interpolation

$$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1} \qquad \text{[GWD14, p. 8, eq. 7]}$$

The next step towards calculating the final memory location weight matrix is blending together the memory location determined through content-based lookup with the weight matrix from the previous time step.

- g_t- A scalar interpolation gate factor [0, 1], used to determine whether to use the weight matrix from the previous time step exclusively (0), using content-based addressing exclusively (1), or to blend the result from content-based addressing with the weight matrix from the previous time step (between 0 and 1).
- w_t^c- The content-based weight matrix.
- w_{t-1}- The final memory location weight matrix from the previous time step.

Shifting

$$\tilde{w_t}(i) = \sum_{j=0}^{N-1} w_t^g(j) s_t(i - j) \qquad \text{[GWD14, p. 9, eq. 8]}$$

Following interpolation, the next step is to apply a shift factor to the result. This implements the location-based addressing component of the NTM. The shift factor is responsible for rotating the current weight matrix through the Interpolation step. This has the effect of advancing backwards or forwards by a specific amount of memory locations.

- w_t^g- The weight matrix as calculated in the Interpolation step.
- s_t- The shift factor to apply, responsible for performing a circular shift of w_t^g.

Sharpening

$$w_t(i) = \frac{\tilde{w_t}(i)^{\gamma_t}}{\sum_j \tilde{w_t}(j)^{\gamma_t}}$$ [GWD14, p. 9, eq. 9]

The final step for determining the weight matrix at time t is raising the current $\tilde{w_t}$ to a sharpening factor, $\gamma\ (>= 1)$. This has the effect of amplifying, or sharpening, the resulting weight matrix after the convolutional shift. The result is divided by the sum of weights to yield a probability distribution representing the final memory location weight matrix, w_t.

- $\tilde{w_t}$- The resulting weight matrix after performing the convolutional shift.
- γ- The sharpening factor (>= 1)

The resulting memory location weight matrix w_t can represent three different types of address changes:

1. The result from content-based addressing can be used as-is.
2. Content-based addressing can be combined with the convolutional shift to produce a final result where the memory location changes to the start of the content, and is then shifted to a location within that content.
3. Content-based addressing can be ignored, and the system can perform a convolutional shift using the weight matrix from the previous time step exclusively. This is a location-based address change.

Experimental Tasks

[This section blank pending further implementation of NTM architecture, reproducing the results described in [GWD14]].

Results

[This section blank pending further implementation of NTM architecture, reproducing the results described in [GWD14]].

Discussion

The behavior of the write heads, along with the calculated memory location weight matrix, determine how the network evolves over time. All of the factors used to calculate write behavior (what to erase and add) along with the factors for determining how to modify the memory location weight matrix relative to the previous time step, are all trainable parameters of an LSTM controller embedded within the NTM. This LSTM controller is what produces the behavior of the NTM to perform the specific task that it has been trained to perform. Abstractly, it can be seen that a data copy operation, for example, is a combination of starting at a particular memory location, reading the contents, writing the contents to another location, and iterating through the memory locations until complete. This task is a combination of read/write and memory location shifts, all of which the NTM is capable of performing.

References

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014. https://arxiv.org/pdf/1410.5401.pdf

PyTorch [Computer software]. Retrieved from https://pytorch.org

[HS97] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8):1735–1780.

Code in *ntm.py* adapted from an in-class example of NTM architecture produced by Dr. Garrett Katz.