MY EXPERIMENTS WITH DIFFERENTIABLE NEURAL COMPUTERS

Anmol Kagrecha, Karan Taneja, Pranav Kulkarni, Udayan Joshi
Indian Institute of Technology Bombay
{15D070024,karantaneja,pranavdk,udayanjoshi}@iitb.ac.in

1 INTRODUCTION

Recurrent neural networks can learn and carry out complicated transformations of data such as sensory processing, sequence learning and reinforcement learning over extended periods of time. However they are limited in their ability to represent variables and data structures and to store data over long timescales, owing to the lack of an external memory. Neural Turing Machines (NTM) are neural networks coupled with external memory resources, which they can interact with by attentional processes. The system is like a differentiable computer that can be trained by gradient descent. The memory learns to represent and manipulate complex data structures with the help of training data. Differentiable Neural Computer (DNC) is built upon NTM with more reading mechanisms that can use memory more flexible way while keeping track of the temporal information. NTMs have been shown to be capable of inferring simple algorithms such as copying, sorting, and associative recall from input and output examples. DNCs have been shown to successfully answer synthetic questions designed to emulate reasoning and inference problems in natural language.

2 NEURAL TURING MACHINE

Neural Turing machine introduced in Graves et al. (2014) is analogous to a Turing Machine but it is differentiable end-to-end and can be trained efficiently using gradient descent.

The architecture of the Neural Turing Machine contains two basic components: a controller and a memory bank. The controller has the capability to interact with external world as well as the memory bank. Specific details about reading and writing to memory bank are described below.

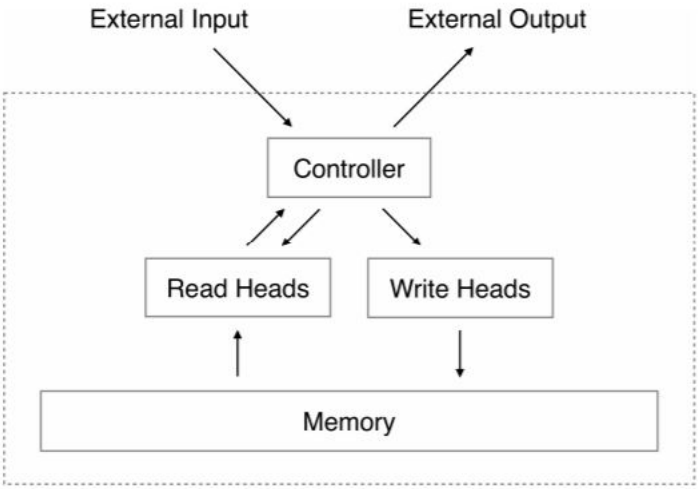

Figure 1: Neural Turing Machine Architecture

2.1 READING

Let $\mathbf{M}_t$ be the contents of the $N \times M$ memory matrix at time t, where N is the number of memory locations, and M is the vector size at each location. Let $\mathbf{w}_t$ be a vector of weightings over the N locations emitted by a read head at time t. As all the weightings are normalized, elements of $\mathbf{w}_t$ obey:

$$\sum_i w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1 \tag{1}$$

The M length read vector $\mathbf{r}_t$ returned by the head is dened as a convex combination of the row-vectors $\mathbf{M}_t(i)$ in memory:

$$\mathbf{r}_t \leftarrow \sum_i w_t(i)\mathbf{M}_t(i) \tag{2}$$

which is clearly differentiable with respect to both the memory and the weighting.

2.2 WRITING

Each write operation is made of two steps: an *erase* followed by an *add*. Given a weighting $\mathbf{w}_t$ emitted by a write head at time t, along with an erase vector e_t whose M elements all lie in the range $(0, 1)$, the memory vectors $\mathbf{M}_{t-1}(i)$ from the previous time-step are modied as follows:

$$\tilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i)[\mathbf{1} - w_t(i)\mathbf{e}_t], \tag{3}$$

where $\mathbf{1}$ is a row-vector of all 1-s, and the multiplication with the memory locations acts point-wise. A memory element becomes zero only if both the weighting at the location and the erase element are one. When multiple write heads are present, the erasures can be performed in any order, as multiplication is commutative.

Each write head also produces a length M add vector $\mathbf{a}_t$, which is added to memory after the erase step has been performed:

$$\mathbf{M}_t(i) \leftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\mathbf{a}_t \tag{4}$$

The order of adds is irrelevant because the operation is commutative. As the erase and add operations are differentiable, the composite write operation is also differentiable.

2.3 ADDRESSING MECHANISMS

The read and write operations require the weightings. These weightings arise by combining two addressing mechanisms: content-based addressing and location-based addressing. Content-based addressing, focuses attention on locations based on the similarity between their current values and values emitted by the controller. Location-based addressing is the conventional form of addressing suitable for iterations and random-access jumps.

2.4 FOCUSING BY CONTENT

For content-addressing, each head (whether employed for reading or writing) first produces a length M key vector $\mathbf{k}_t$ that is compared to each vector $\mathbf{M}_t(i)$ by a similarity measure $K[\cdot, \cdot]$. The content-based system produces a normalized weighting w_t^c based on the similarity and a positive key strength, β_t, which can amplify or attenuate the precision of the focus:

$$w_t^c(i) \leftarrow \frac{\exp\Big(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(i)]\Big)}{\sum_j \exp\Big(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(j)]\Big)} \tag{5}$$

The authors use cosine similarity as the similarity measure:

$$K[\mathbf{u}, \mathbf{v}] = \frac{\mathbf{u} \cdot \mathbf{v}}{\|u\| \cdot \|v\|} \tag{6}$$

2.5 FOCUSING BY LOCATION

Location-based addressing is implemented using a rotational shift of a weighting. Prior to rotation, each head emits a scalar interpolation gate g_t in the range $(0, 1)$. The value of g is used to blend the weightings $\mathbf{w}_{t-1}$ and $\mathbf{w}_t^c$, yielding the gated weighting $\mathbf{w}_t^g$:

$$\mathbf{w}_t^g \leftarrow g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1} \tag{7}$$

After interpolation, each head emits a shift weighting $\mathbf{s}_t$ that defines a normalized distribution over the allowed integer shifts. The shift weightings are generated using a softmax layer of appropriate size attached to the controller.

If the N memory locations are addressed from 0 to $N - 1$, the rotation applied to $\mathbf{w}_t^g$ by $\mathbf{s}_t$ can be expressed as the following circular convolution:

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j) s_t(i - j) \tag{8}$$

where all index arithmetic is computed modulo N. Each head also emits one further scalar $\gamma_t \geq 1$ whose effect is to sharpen the final weighting as follows:

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}} \tag{9}$$

The combined addressing system of weighting interpolation and content and location based addressing can operate in three different ways:

- Weighting chosen by the content system without any modication by the location system.
- Weighting produced by the content addressing system can be chosen and then shifted.
- Weighting from the previous time step can be rotated without any input from the content-based addressing system.

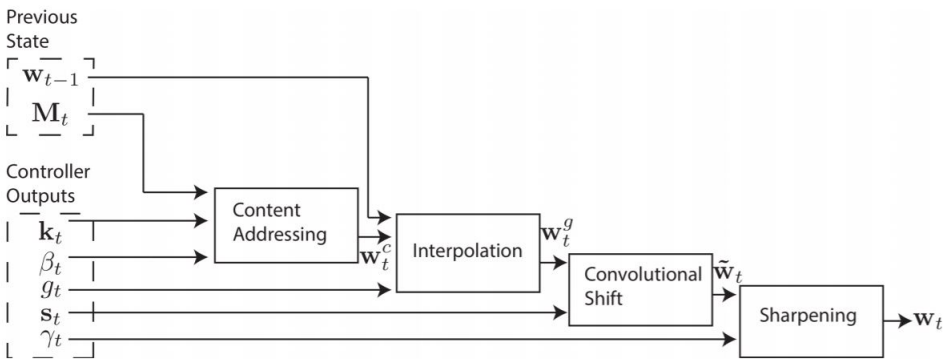

Figure 2: Flow Diagram of Addressing Mechanism

2.6 CONTROLLER NETWORK

The authors experiment with both feedforward and recurrent networks as the controller. A recurrent controller such as LSTM has its own internal memory that can complement the larger memory in the matrix. Feedforward networks are more interpretable but not as flexible as the recurrent networks.

3 DIFFERENTIABLE NEURAL COMPUTER

Differentiable neural computer is presented as follow-up work, in Graves et al. (2016), on neural Turing machine with more reading mechanisms. Also, the memory is dynamic in the sense that it can be extended without re-training the model i.e. using the same model parameters.

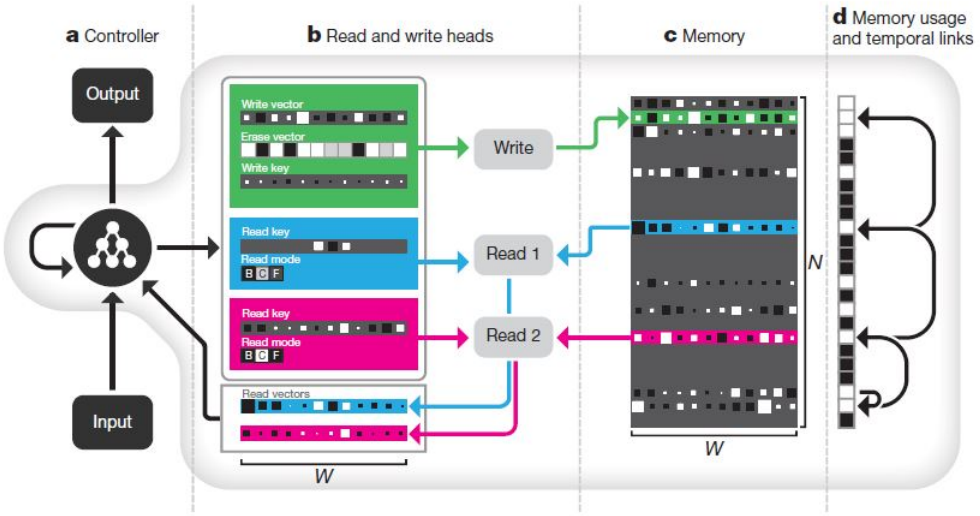

Figure 3: DNC Architecture

In the NTM, in order to iterate through memory locations content-based addressing was combined with location-based addressing. However, this has several drawbacks:

- The NTM has no mechanism to ensure that blocks of allocated memory do not overlap and interfere.
- The NTM has no way of freeing locations that have already been written to and, hence, no way of reusing memory when processing long sequences.
- Sequential information is preserved only as long as the NTM continues to iterate through consecutive locations.

DNC doesn't suffer from these drawbacks:

- Interference is not an issue for the dynamic memory allocation used by DNCs, which provides single free locations at a time, irrespective of index, and therefore does not require contiguous blocks.
- DNC has free gates, one per read head, that determine whether the most recently read locations can be freed. This helps in freeing locations that have already been written to.
- The temporal link matrix used by DNCs blocks tracks the order in which writes were made. Hence, sequential information is always preserved.

4 EXPERIMENTS

4.1 TASKS

We train and test DNC model on following three tasks:

1. **Top-k Sorting**: For top-k sorting, we input a sequence of numbers in form of their bit representation. This is followed by an end-of-sequence flag which is further followed by the k query and an end-of-query flag.
2. **Shortest Path**: For shortest path task, we input the model with bit representations of pair of nodes which have edges between them which is followed by an end-of-sequence tag. After this, a pair of query nodes is given between which the path is desired. Finally, we give the end-of-query flag as the input.

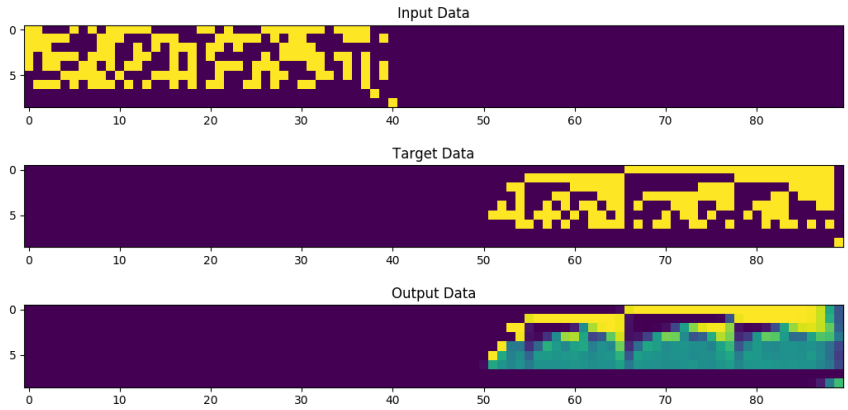

Figure 4: Top-k sorting task example 1

3. **Connectedness**: For connectedness task, we input the graph as done in shortest path task. There are multiple queries to the model in form of pairs of nodes and followed by end-of-queries flag. We train the model to output 1 if pair of nodes are connected in the graph by some path, and 0 otherwise.

4.2 Results and Observations

For the three tasks, the final average logistic loss after training the model is given in Table 1 on scale of 10^{-4}. The three columns correspond to three different settings in which the model were trained. For experiments in first column of the table, we've used 64 memory vectors but 128 for second column. For experiments in the third column, we added 10 blank steps after input stage and before emitting the output where model can process the input to *plan the output* that it wishes to generate subsequently. We refer to these blank controller steps as *thinking steps*.

Task	mem_size=64	mem_size=128	thinking
Top-k Sorting	788	886	589
Shortest Path	111	133	88
Connectedness	122	121	93

Table 1: Average logisitc loss in scale of 10^{-4} for top-k sorting, shortest path and connectedness when trained under three different settings viz. 64 memory vectors are used while training, 128 memory vectors are used while training, and 10 thinking steps are used after input and before emitting the output.

We observe that model trained with 10 thinking steps consistently perform better than ones without it by almost 20-25% which means the thinking really helps the model to better interpret the input. It is a surprising observation that more memory for has adversely affected performance for top-k sorting and shortest path task. This is possibly due to the over-sized memory which doesn't help the performance but decreases it because controller decisions on reading and writing memory involve more complexity now.

For the top-k sorting task, two examples are showed in Figure 4 and 5. We observe that though the model is unable to learn the top-k sorting task perfectly, it precisely learns to capture the most significant bit of the number correctly with very high confidence. Also the end marker is correctly predicted indicating that the model has learned to capture the k provided as the query vector and internally implement a counter to keep a count. As seen in the two examples, the model is unable to

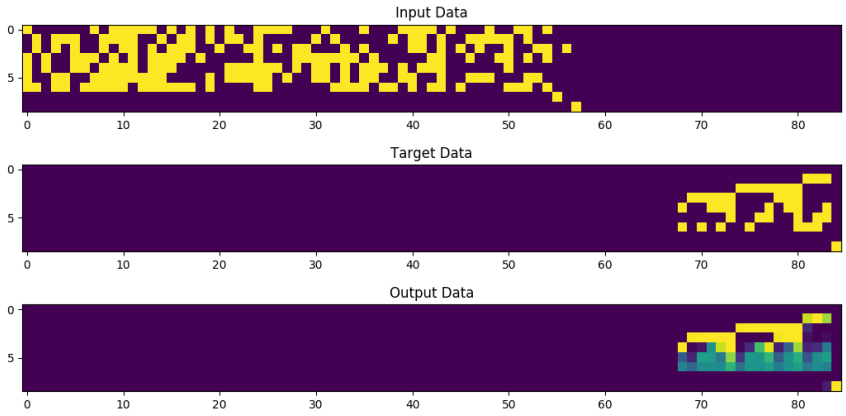

Figure 5: Top-k sorting task example 2

recall the lower bits in the input, which is surprising because the bit representations are (intuitively) the simplest representations to store in a bit array and should have been easy to recall.

An example of shortest path task is given in Figure 6. We observe that model is giving end-of-output tag which is the last bit at an earlier step. On close inspection on a lot of such examples, we realized model is always giving two vectors followed by end-of-sequence tag. This is probably because of the most examples that are input the model have queries with pair of nodes that are two hops apart.

In the connectedness task, we observed that model learned an average output for all the queries and converged. Interesting observation is that number of outputs i.e. number of predictions emitted before end-of-output sequence were exactly equal to the number queries. In other words model learned to count the number of queries and output the number of predictions accordingly.

5 Conclusion

We understood the working of neural Turing machine and differential neural computer and their major differences. We trained and evaluated DNC on top-k sort, shortest-path and connectedness task. With our experiments, we highlighted the observed strengths and shortcomings of the DNC model for the three tasks.

Code: Code and a subset of trained models for the project can be found here.

6 Contributions

Following are the contributions of teams members:

- **Anmol Kagrecha**: Implemented random dataset generation for shortest-path and connectedness tasks and major contribution to report.
- **Karan Taneja**: Wrote the training and evaluation scripts common to all tasks, read the Graves et al. (2016) paper and shared with team.
- **Pranav Kulkarni**: Wrote the code for random dataset generation for top-k sorting, read the Graves et al. (2014) paper and shared with team.
- **Udayan Joshi**: Implemented the travelling salesman problem dataset which, unfortunately, couldn't be trained and delivered here.

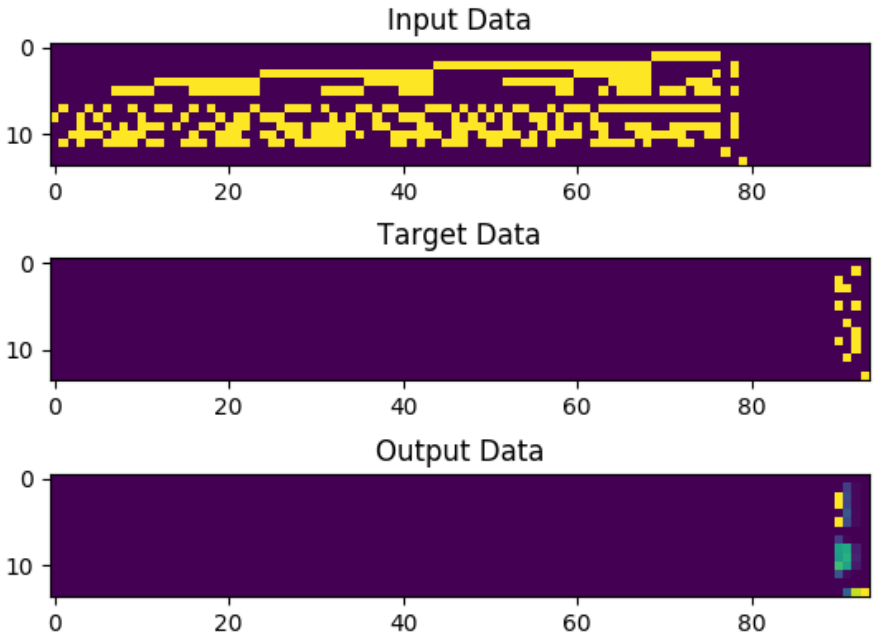

Figure 6: Shortest path example

REFERENCES

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016.