My Experiments with Differentiable Neural Computers

CS 726 Course Project

Anmol Kagrecha, Karan Taneja, Pranav Kulkarni, Udayan Joshi

Content

Motivation

- Recurrent neural networks (RNNs) 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.

Motivation

- Recurrent neural networks (RNNs) 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**, which they can interact with by attentional processes.
- The system is differentiable and can be **trained by gradient descent**. With memory, it can learn to represent and manipulate complex data structures.

Motivation

- Recurrent neural networks (RNNs) 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**, which they can interact with by attentional processes.
- The system is like a differentiable and can be **trained by gradient descent**. With memory, it can learn to represent and manipulate complex data structures.
- 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**.

Problem statement

I. To test out DNC model on various **simple algorithmic tasks**:
   A. Top-k sorting
   B. Shortest path in a graph
   C. Connectedness queries

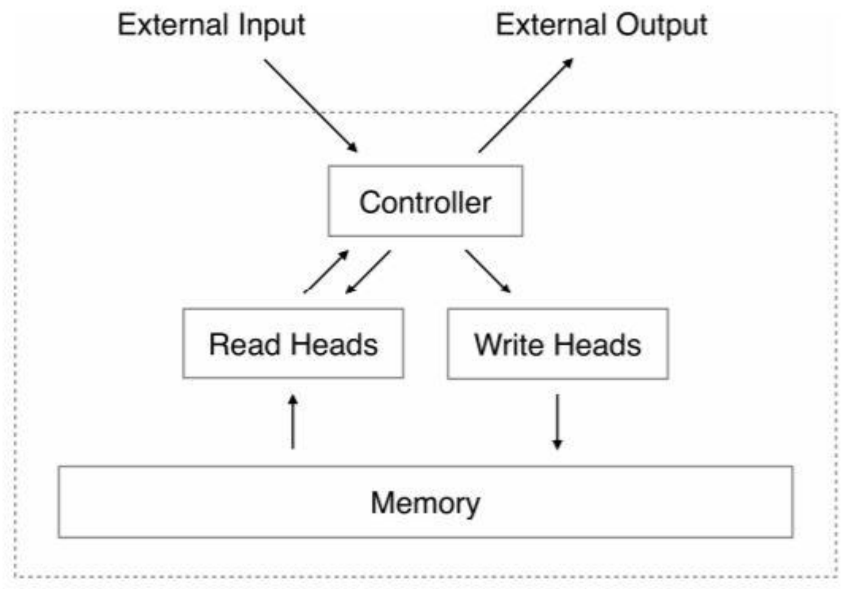
Problem statement

I. To test out DNC model on various **simple algorithmic tasks**:
   A. Top-k sorting
   B. Shortest path in a graph
   C. Connectedness queries

II. **'Thinking'** state
   A. A state where model is not absorbing any input or output.
   B. This allows model to do complex manipulation of memory.
   C. Testing out model on different task to check improvements.

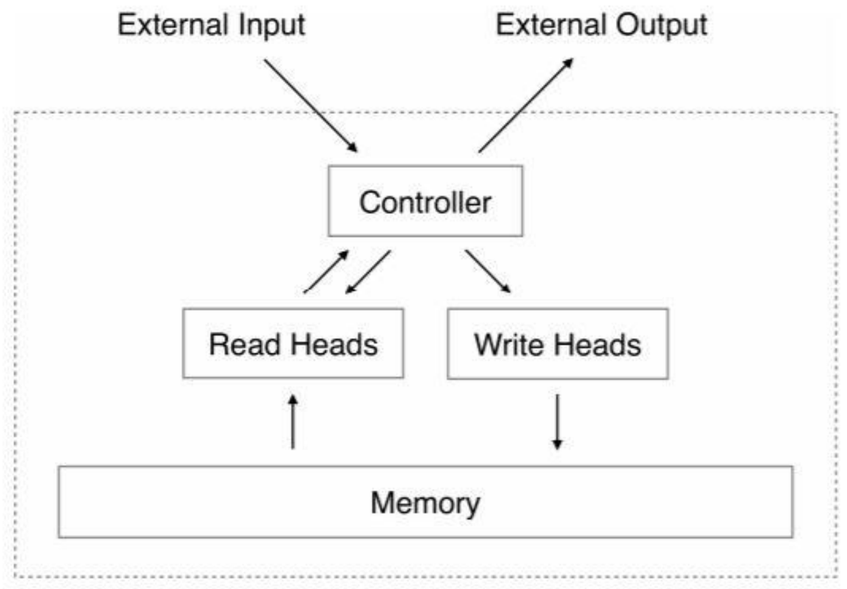
Neural Turing Machine

- 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.



Neural Turing Machine

- 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.



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:

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

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:

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

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

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 $\mathbf{M}_{t1}(i)$ from the previous time-step are modied as follows:

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

where **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.

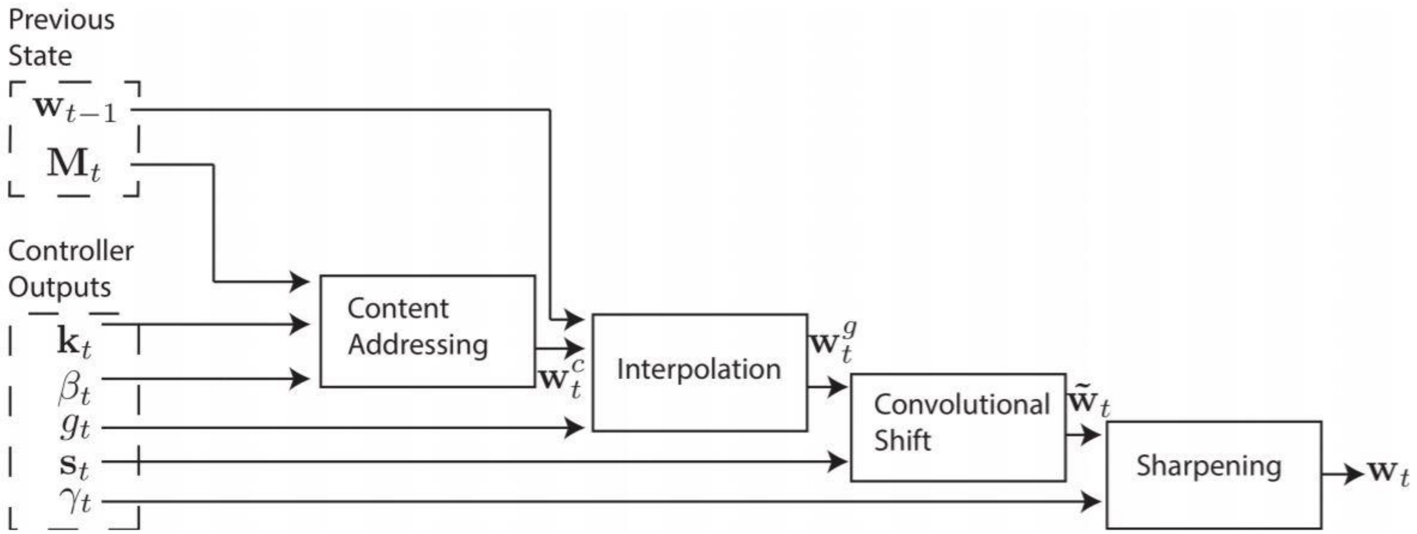
Each write head also produces a length M add vector $\mathbf{a}_t$, which is added to memory after the erase step has been performed:

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

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.

Addressing Mechanisms

- **Content based addressing:** focuses attention on locations based on similarity of current values and values emitted by the controller.
- **Location based addressing:** conventional address based addressing



Limitations of NTM

- The NTM has no mechanism to ensure that blocks of allocated memory do not **overlap and interfere**.

Limitations of NTM

- 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.

Limitations of NTM

- 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 solves these problem!

- 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.

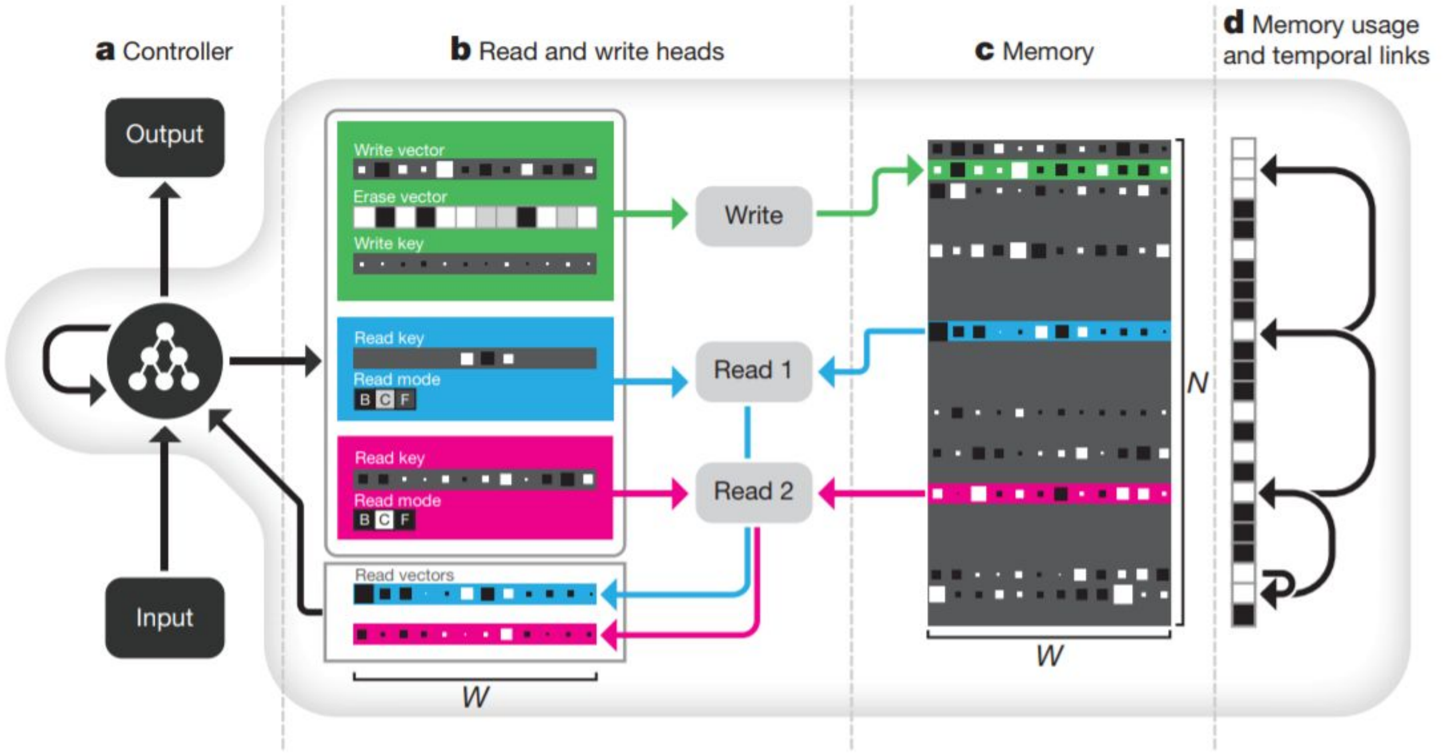
DNC solves these problem!

- 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.

DNC solves these problem!

- 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.

Dynamic External Memory



Implementation in PyTorch at https://github.com/ixaxaar/pytorch-dnc.

Datasets and Evaluation

We generate synthetic data for following tasks:

- **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.

Datasets and Evaluation

We generate synthetic data for following tasks:

- **Shortest path in a graph:** 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.

Datasets and Evaluation

We generate synthetic data for following tasks:

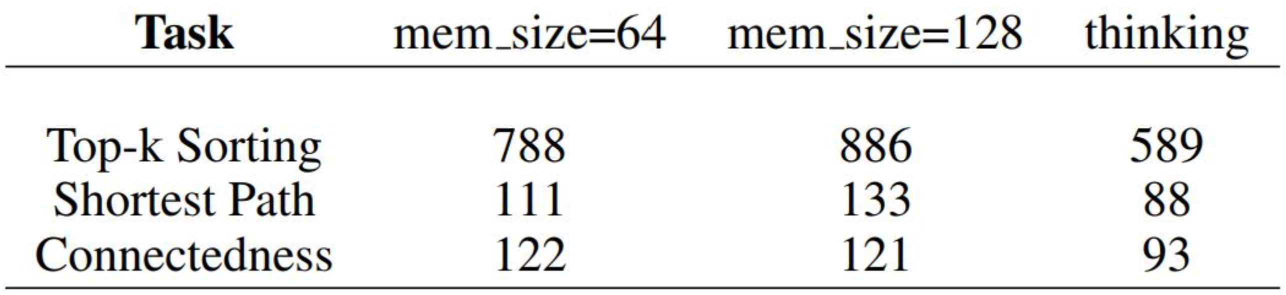
- **Shortest path in a graph:** For shortest path task, we input the model with bit representations of pair of nodes which have edges between them which is followed by an end-of-sequence tag. After this, a pair of query nodes is given between which the path is desired. Finally, we give the end-of-query flag as the input.
- The target output is selected based on the actual output: the path having closest bit representation as output is chosen.

Datasets and Evaluation

We generate synthetic data for following tasks:

- **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.

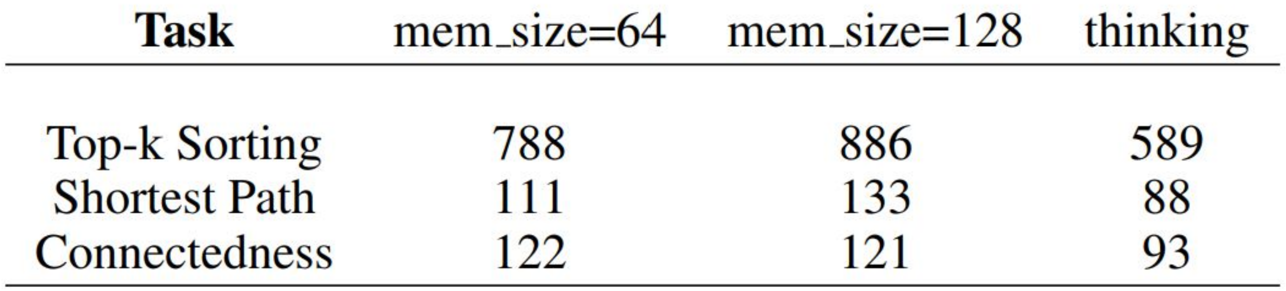
Datasets and Evaluation



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

Average logistic loss in scale of 10^{-4}, trained under three different settings: 1) 64 memory vectors are used while training, 2) 128 memory vectors are used while training, and 3) 10 thinking steps are used after input and before emitting the output.

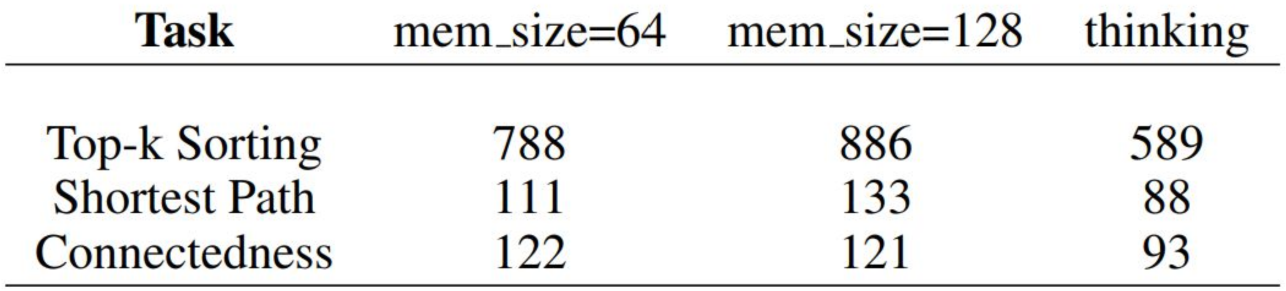
Datasets and Evaluation



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

- 10 thinking steps consistently perform better by almost 20-25%
- the thinking really helps the model to better interpret the input
- surprising that more memory for has adversely affected performance for top-k sorting and shortest path task

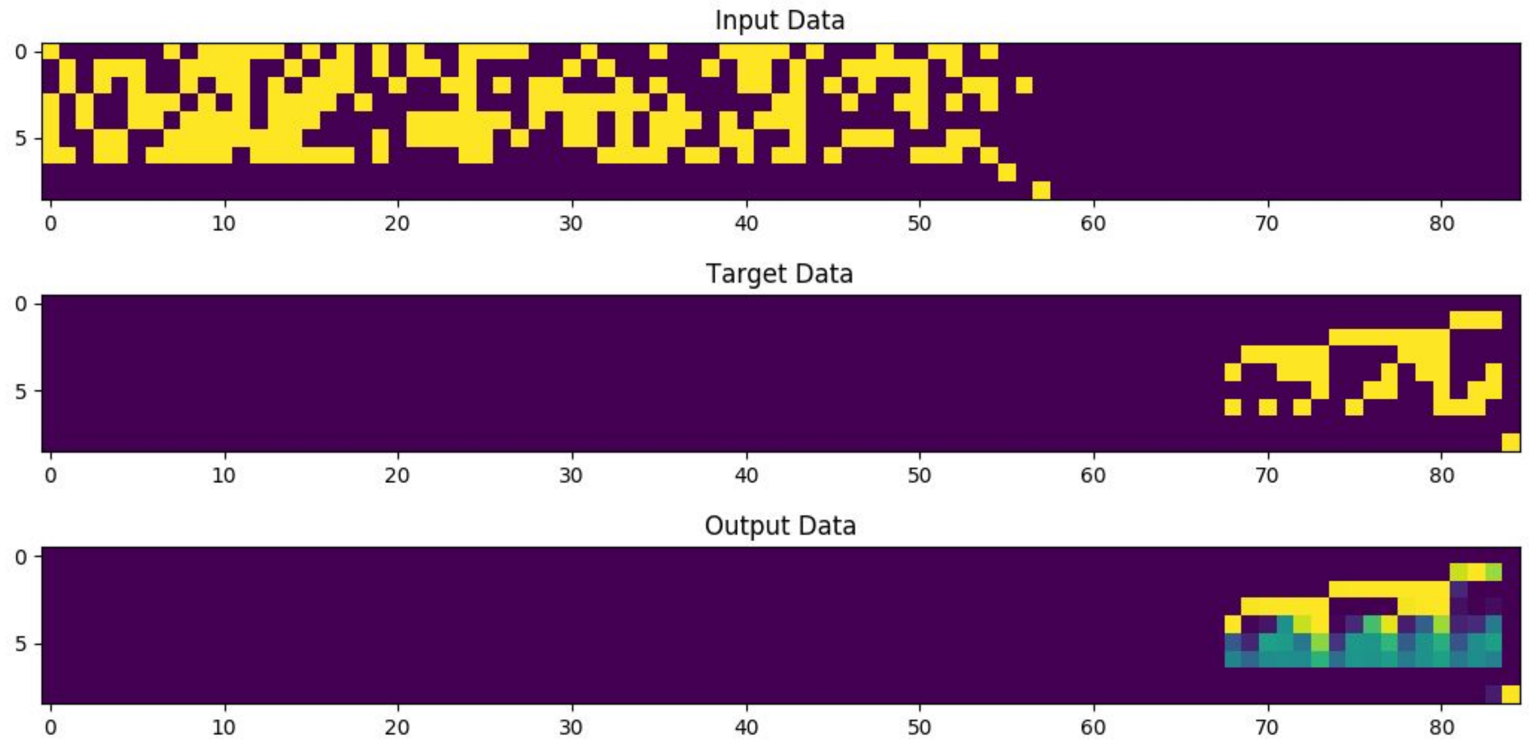
Datasets and Evaluation



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

- 10 thinking steps consistently perform better by almost 20-25%
- the thinking really helps the model to better interpret the input
- surprising that more memory for has adversely affected performance for top-k sorting and shortest path task
- possibly due to the **oversized memory** which doesn’t help the performance but decreases it because controller decisions on reading and writing memory involve **more complexity** now

Datasets and Evaluation (Top-k Sorting)

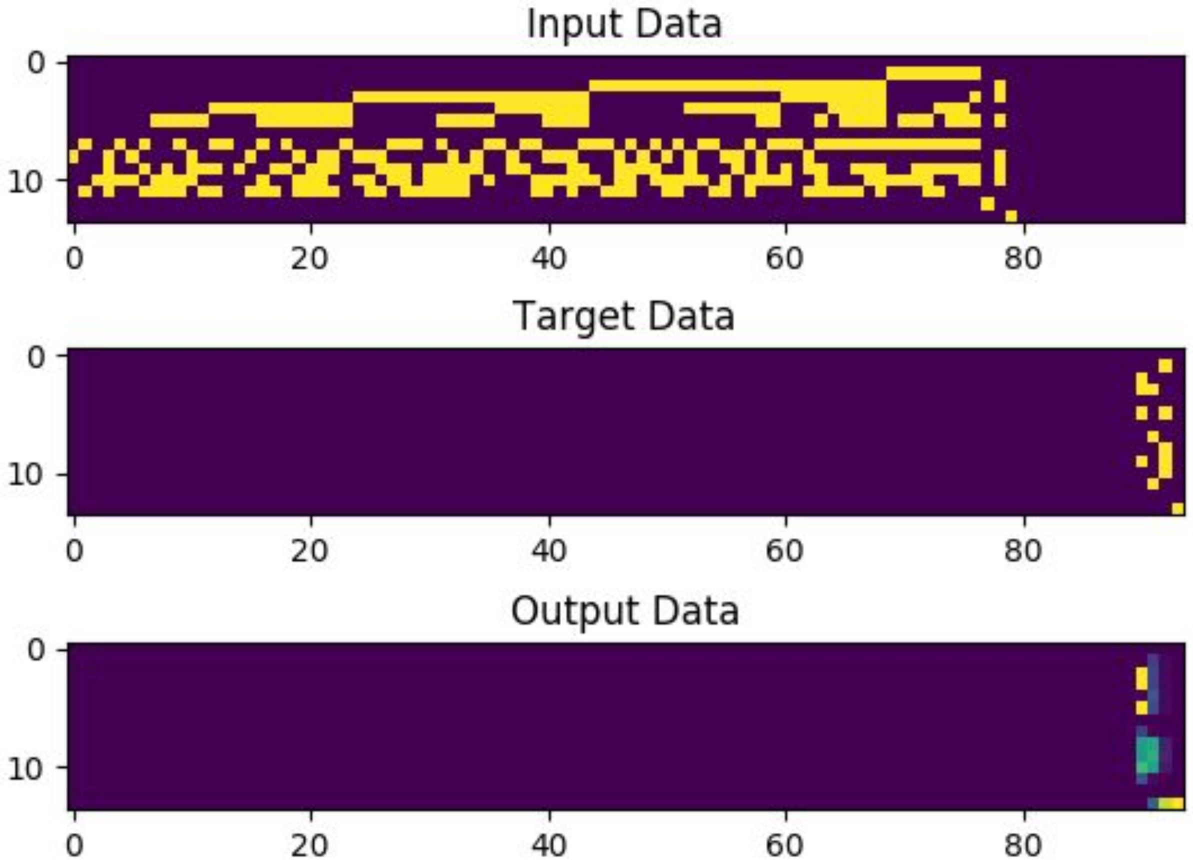


Datasets and Evaluation (Top-k Sorting)

- model is unable to learn the top-k sorting task perfectly
- but it precisely learns to **capture the most significant bit** with very high confidence
- also, the **end marker is correctly predicted**
- indicates that the model has learned to **capture the *k* provided** as the query vector
- internally **implements a counter** to keep a count
- surprising that lower bits are not recalled well
- because bit representations are (intuitively) the simplest representations

Datasets and Evaluation

(Shortest Path)

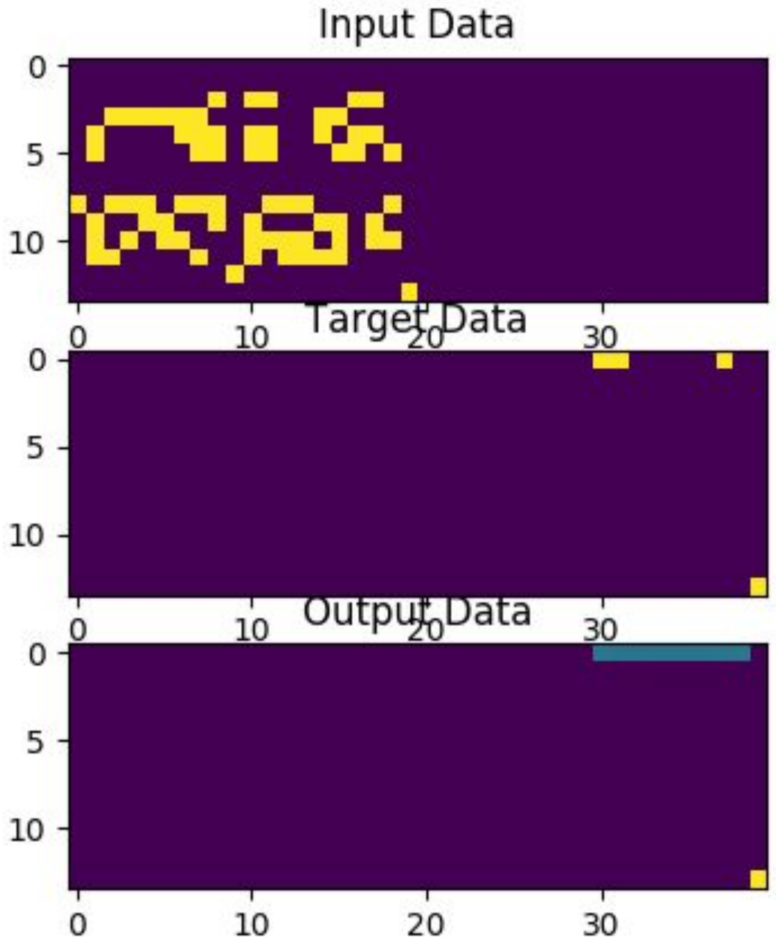


Datasets and Evaluation (Shortest Path)

- model is giving end-of-output tag which is the last bit at an earlier step
- seeing a lot of such examples, we realized model is **always giving two vectors** followed by end-of-sequence tag
- most examples that are input the model have queries with pair of nodes that are two hops apart.

Datasets and Evaluation

(Connectedness)



Datasets and Evaluation (Connectedness)

- model learned an average output for all the queries and converged
- number of outputs exactly equal to the number queries
- model learned to **count the number of queries** and output the number of predictions accordingly

Conclusion

- Understood the working of neural Turing machine and differential neural computer and their major differences.
- Trained and evaluated DNC on top-k sort, shortest-path and connectedness task.
- Highlighted the observed strengths and shortcomings of the DNC model for the three tasks.

THE END