IFT6135 Representation Learning (Programming Assignment 3)

Names: Hossein Askari, Issac Sultan **Due Date:** March 29^{th} 2018

Neural Turing Machines

In this task you will implement the Neural Turing machine as described in [GWD14]. The goal is to better understand attention and memory augmented networks, and the difficulties encountered when training them.

1. Filling in the Gaps

The paper covers the abstract ideas and goes into detail about how the read-/write heads are computed, but does not mention several details. Here, we attempt to cover some of the missing details.

(a) **The output of the controller at each time step consists of parameters that have constraints. For example, $\beta \in (0, \infty)$. Using equations, describe how you have constrained the output to satisfy them. Briefly justify your choice.**

(b) **Present a diagram showing how you think the following are dependent on each other. This will inform your implementation later:**

- **the input x_t**
- **the memory M_{t-1} and M_t**
- **the output of the read head r_{t-1} and r_t**
- **the erase and add vectors e_t and a_t**
- **the output of the controller o_t**

(a) As it is mentioned, at the output of the controller at each time step, we have parameters with constraints. Figure 1 shows the parameters and their constraints that we used in our controller:

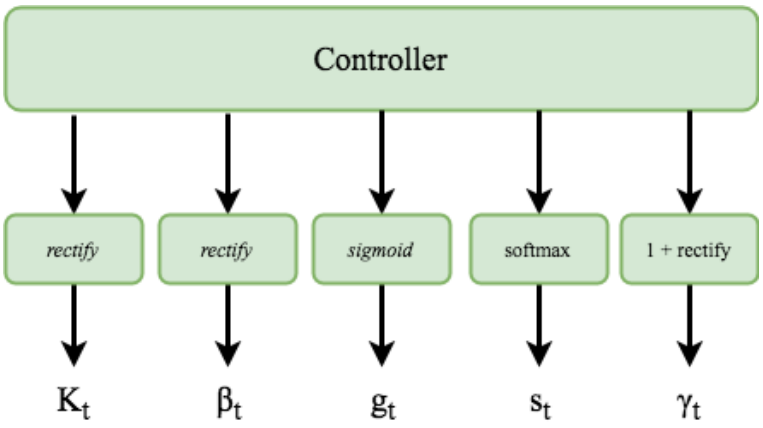

Figure 1: **Neural Turing Machine Controller output. This figure shows the control signals that are generated by the NTM controller to control read/write and interaction with memory. This figure also shows the constraint that is being applied on the parameters.**

We now briefly explain our choices to constraint every parameter:

- β_t: As it is described in the paper, β_t is the key strength. For reference here is Equation 5 from NTM paper:

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

  The above equation shows that a multinoulli decision wants to be made. The β_t can be seen as the inverse of temperature parameter. We want this temperature parameter to be positive. As an example, suppose the output vector is $[2.0, 2.1, 2.2]$ if β is 1 the difference between $exp(2.0)$ and $exp(2.1)$ is negligible compared to a scenario where β is equal to 10 where we compare $exp(20)$ against $exp(21)$. We can see that the β acts as how picky our selection wants to be. So we use a rectifier to make sure the temperature is positive. However, we do not need to bound β since the model will learn how picky it should be.

- g_t: As it is described in the paper, g_t is the interpolation gate. Equation 7 from the paper can give us a better understanding on how to constraint this parameter:

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

  As it can be seen, the value of g is used to blend between the weighting w_{t-1} which was produced by the previous time-step and the weighting w_t^c produced by the content system at the current time-step. Since a binary selection between the two needs to be made, g can be either 0 or 1. Since we want our model to

be differentiable we need a smooth transition between 0 and 1. We already know that sigmoid can provide us a smooth transition which is differentiable, so we selected sigmoid.

- s_t: The shift weighting parameter s_t determines whether and by how much the weighting is rotated. As described in the NTM paper, the shifts are only integer values and are constrained between -1, 0 and 1. Equation 8 from the paper can help us better understand this:

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

The equation above shows a circular convolution. We will discuss later how this convolution acts as a shift operator. We need to choose between the possible three values available (-1, 0 and 1). We also want to make sure this choosing operation is differentiable. As we learned in multi-class classification, softmax can produce a vector that represents the possibility of choosing each element of the vector. So we think softmax is the perfect fit for this task.

- γ_t: Equation 9 of the NTM paper provides a technique to sharpen the blurry weight focus:

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

We need γ_t to be more than one since the weight normalization that is being done in the equation above sharpens the weight. Choosing γ_t less than one will cause smoothing the weights which is not what we want. The equation above constraints γ to be equal or larger than 1. The easiest way to rectify a variable but at the same time be differentiable is to to use a rectifier such as relu. To make sure the value is more than 1 we can simply do:

$$\gamma_t = 1 + relu(\gamma_t)$$

- k_t: As it is described in the paper, k_t is a length M key vector that is compared to each vector M_t (Memory vector) by a cosine similarity measure K which is defined as below:

$$K[u, v] = \frac{u.v}{\|u\| \, . \, \|v\|}$$

For cosine similarity measure we need both u and v to be positive. The easiest way to make sure the values of K are positive is to rectify them. We cannot use a non differentiable function so we used $relu$ for our rectifier.

(b) Figure 2 illustrates the architecture of the implemented Neural Turing Machine. This figure illustrates the connections between different part of the Neural Turing Machine. It also illustrates how control signals are used to control different part of the machine. The architecture consist of five main parts: Controller, Memory, Read head, Write head and the Addressing mechanism.

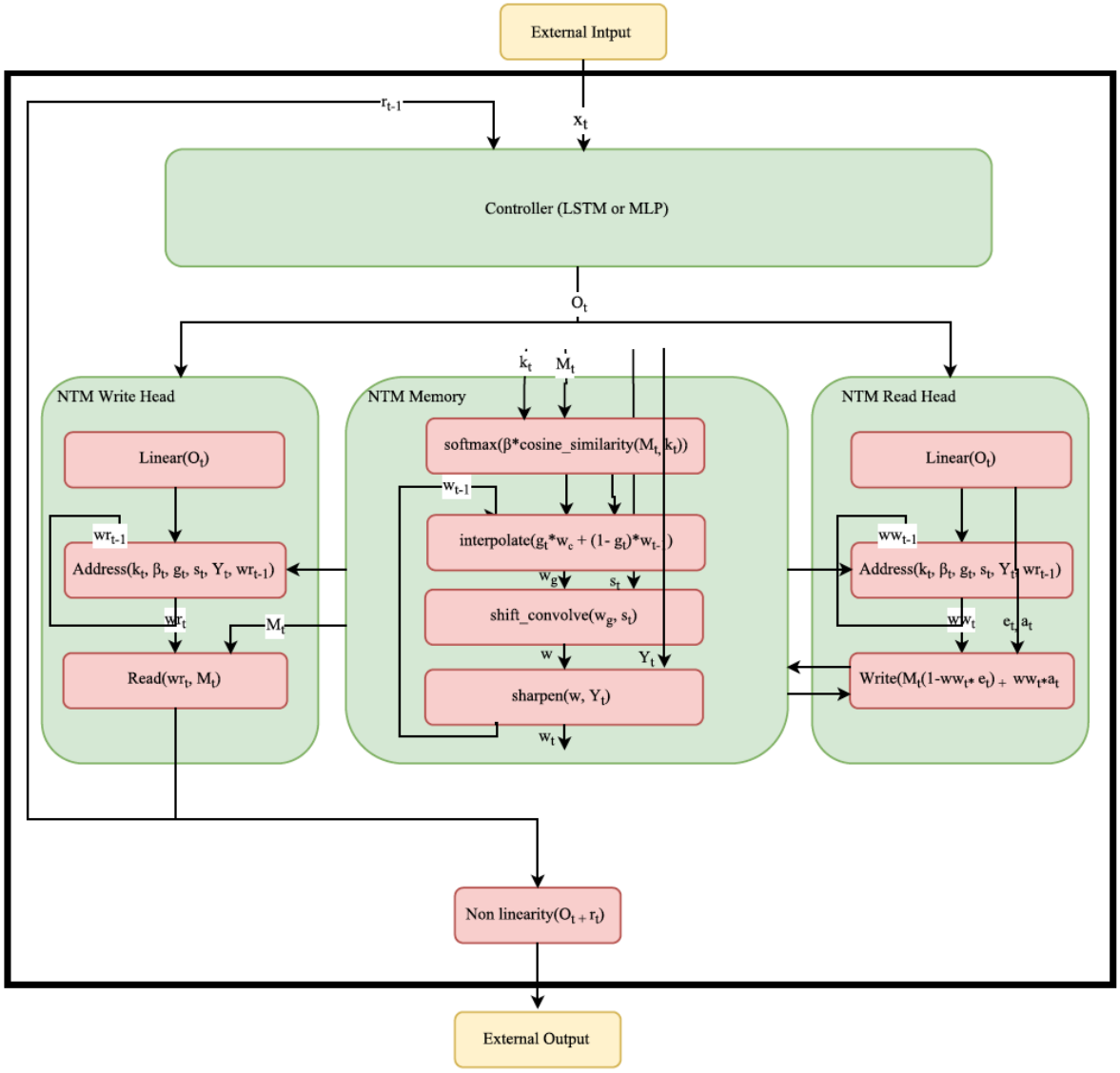

Figure 2: **Architecture of the implemented Neural Turing Machine. This figure illustrates the connections between different part of the Neural Turing Machine. It also illustrates how control signals are used to control different part of the machine. The architecture consist of five main parts: Controller, Memory, Read head, write head and the addressing mechanism. The controller can be a MLP or LSTM model.**

In the above figure, the NTM addressing scheme is illustrated within the Memory unit as it is the memory unit responsibility to handle any memory access such as read and write. We have also illustrated that the controller can be any of the LSTM or Feedforward (MLP) type network. The external input is also illustrated as one the only two inputs to the controller unit.

2. Implement the Neural Turing Machine Implement both a feedforward (Feedforward-NTM) and LSTM controller (LSTM-NTM). Also implement an LSTM for the same task (LSTM). This will be the baseline you will compare the performance of the NTM models with.

- **The task is only the copy task from the paper.**
- **Input data A sequence of random, 8-dimension binary vectors concatenated with a binary indicator for the end of sequence, with sequences no longer than 20 ($T \leq 20$):**

$$(x_1, ..., x_T, x_{T+1})$$

where:

$$\begin{aligned} x_t &= (x_{t,1}, x_{t,2}, ..., x_{t,8}, 0) \\ x_{t,i} &\approx Bernouli(0.5) \qquad for\, t \in 1, ..., T \\ x_{T+1} &= (0, 0, ..., 0, 1) \qquad for\, t = T + 1 \end{aligned}$$

- **Use the cross-entropy loss.**
- **For all models : Use one layer, with a dimension of 100.**
- **For the *-NTM models : Use only 1 read head and 1 write head.**

(a) **Report the total number of parameters of all the models, including the baseline.**

(b) **Perform a training hyper-parameter search (learning rate, batch size, etc.) to ensure that the loss converges. (You do not have to perform an exhaustive grid search, just provide the hyper-parameters you eventually used. Hint : You can download the source of the paper here 1, which contains additional comments pertaining to the hyperparameters used.) Plot the training curves for all three models (using the chosen hyperparameters.)**

(c) **Generalisation to longer sequences One of the benefits of the NTM over a vanilla LSTM is the ability to learn a simple algorithm that generalise to larger sequences. Test your models on sequences of T 2 f10; 20; 30; 40; : : : ; 100g, with 20 different inputs for each T. Plot average loss vs. T. State what you expected of the experiment and why, then comment on the results.**

(d) **Visualising the read and write heads/attention We can visualise the read and write heads to get an idea of what algorithm is learned for the task. Plot the write and read head/attention for an input sequence of T = 10. State what you expected to see and why, then comment on the results.**

(e) **Understanding the shift operator Discuss the relationship between the shift operator and convolutions (Note that you do not have to implement shifts this way.) The purpose of this question is to ensure that students understand the code that they are working with. Modify the code-base so that the shift operator only allows forward shifts. In your answer, show the snippet of code before and after the modification.**

(a) Table 1 shows number of parameters used in each model:

Model Type	Number of Parameters
LSTM Base line	45208
LSTM NTM	62860
MLP NTM	13260

Table 1: Number of parameters used in each network

As it was asked in the question, for all of our models, we used one layer with dimension of 100.

(b) After doing a hyper parameter search to tune the models, we ended up with the following settings for our models:

Model Type	optimizer	learning rate	momentum	Total Number of Batches	Mini Batch Size
LSTM Base line	RM-Sprop	1e-4	0.9	20000	50
LSTM NTM	Adam	1e-5	-	20000	50
MLP NTM	Adam	1e-5	-	20000	50

Table 2: This table shows the different hyper parameters used to train the each model.

We initially had hard time training our NTM model with SGD or RMSProp. After using Adam for $LSTM\_NTM$ and MLP_NTM we never had any issue training our models. We also noticed that using Xavier initialization helps a lot for training NTM models.

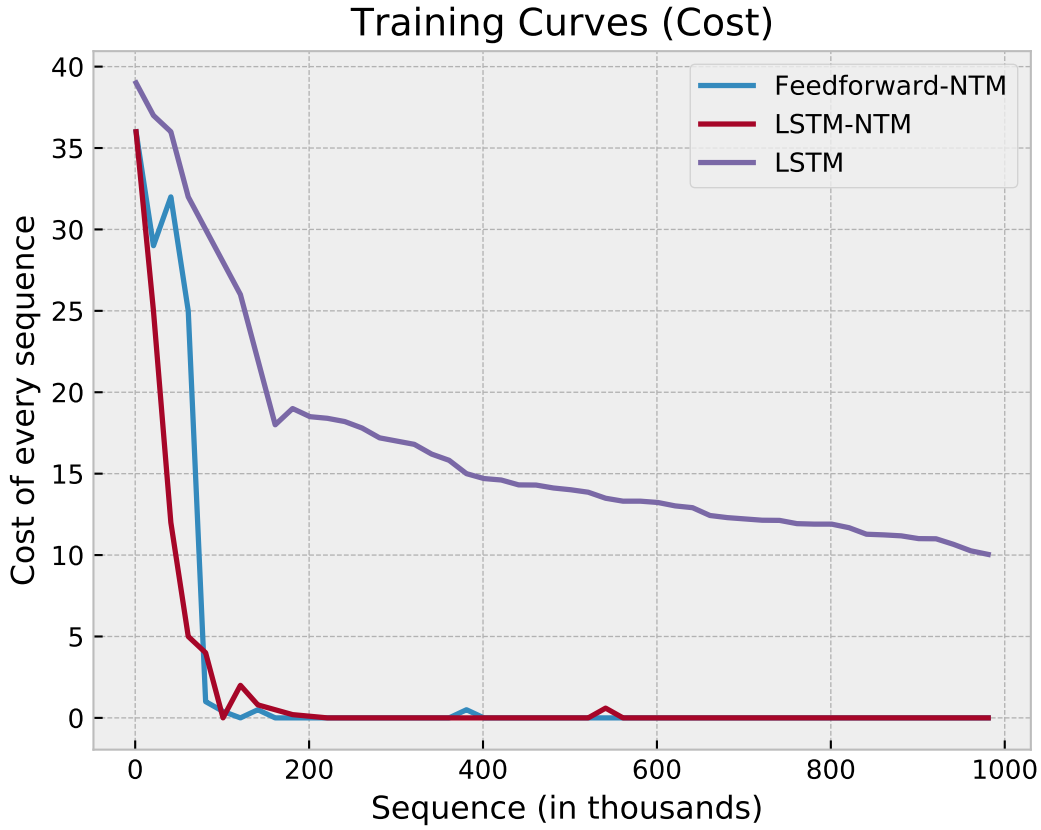

Figure 3: Learning Curve for all three models (MLP_NTM, $LSTM\_NTM$ and $LSTM$). This figure shows the worst performance is achieved by the base line LSTM. It also shows a very small difference between MLP and LSTM in NTM mode. However, for the copy task, the MLP shows that it can learn the task more quickly.

Figure 3 shows the learning curve for the copy task for all three models. We can see that the best performance has been achieved by the MLP model. On the other hand, the base line LSTM shows the worst performance among all other models. This can be explained since MLP has much easier structure over LSTM. Also the copy task is fairly an easy task to perform and it can represented much better with smaller networks such as MLP. All these leads to a faster learning for MLP. On the other hand, the baseline LSTM shows the worst performance since it simply does not have large enough capacity to lean the pattern as fast as MLP.

(c) **Generalization to longer sequences:**
Figure 4 shows the generalization plot for all three models. We initially trained our model with a sequence length of 20. In this experiment, we want to investigate the effect of different sequence length on a model that performed learning on sequences of length 20. We were expecting the base line LSTM to show the worst generalization. This is due to the limited memory resource of the base line LSTM. The base line LSTM can only detect patterns of maximum length 20. On the other hand, both NTM models do not have this issue. Because of accessing to memory, both NTMs show a great generalization over much longer sequences than the original length.

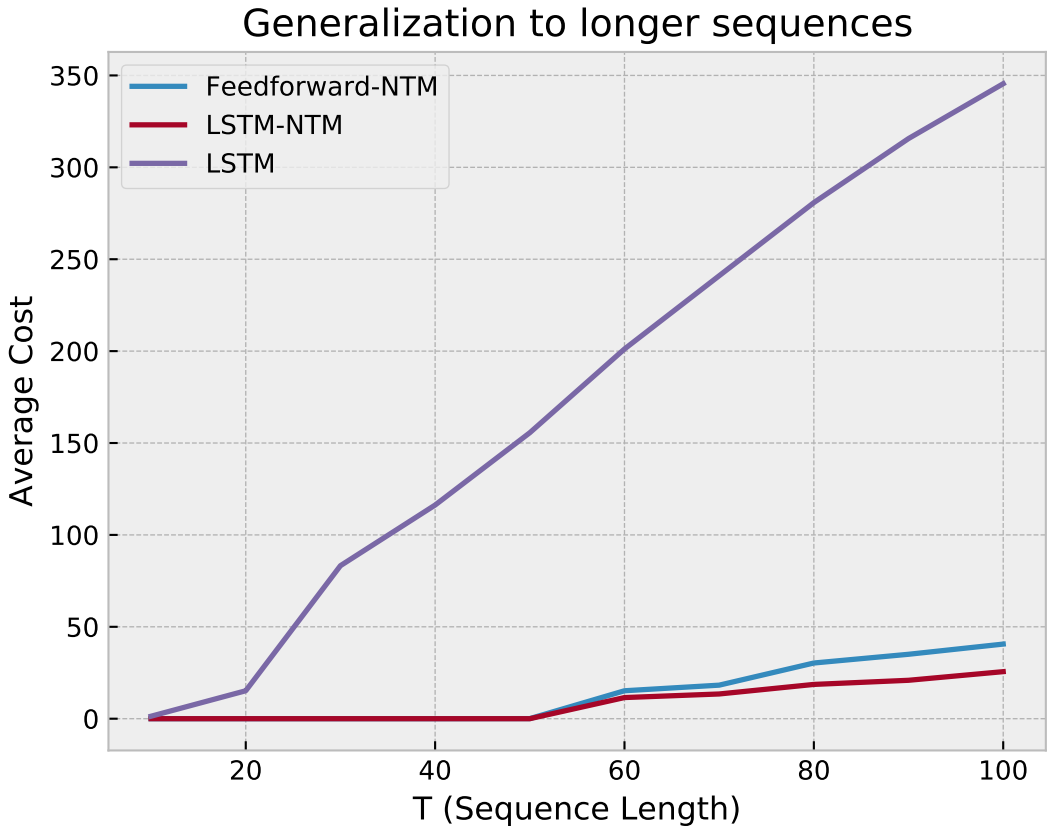

Figure 4: **Generalization to longer sequences**

This can be clearly seen on Figure 4. The purple line shows the average cost for the baseline LSTM. As it can be seen, initially the baseline LSTM shows a fairly good performance. However, the generalization gap between base line LSTM and NTM models opens a lot quicker with sequences of length larger than 20. On the other hand, the NTM models perform amazingly great up to sequence of length 48 and then they start to perform not quite good as before. In general, Figure 4 matches exactly

with our expectation.

(d) **Visualization:**

The copy task is a simple and a fundamental task in a Turing Machine. In this project our task was to design a Neural Turing Machine that could learn to do this task. The most important part is the operation of writing to memory and then learn how to retrieve the written data from memory by the read operation. We expected once the model has been trained successfully, it will be able to write to the memory and then read it back. For that we expected the attention mechanism learn to remember where the specific data has been written and when the times come, read that data back from memory.

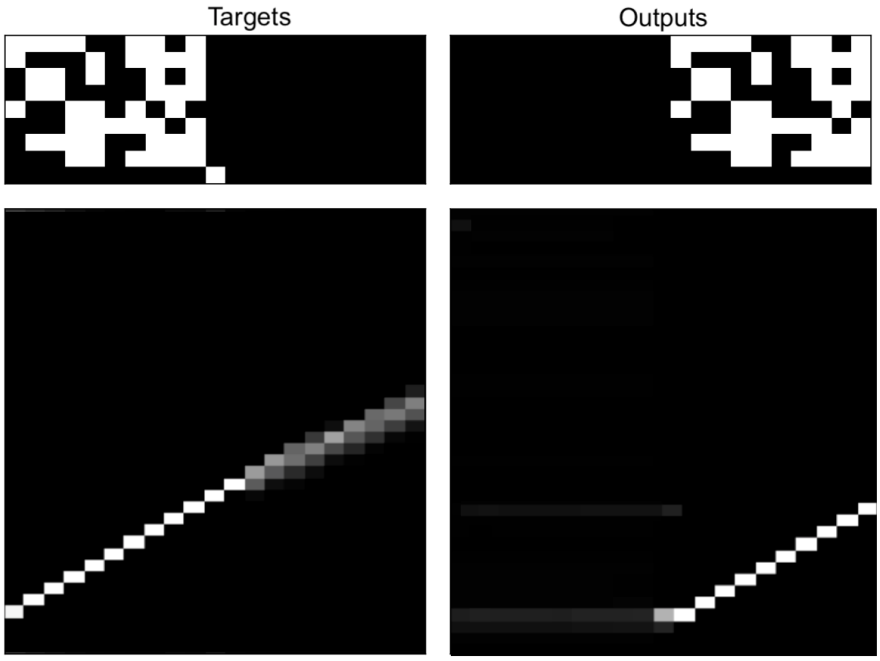

Figure 5: **The left plots depict the inputs to the network (top) and the corresponding write weightings (bottom) during a single test sequence for the copy task. The plots on the right show the outputs from the network (top) and the read weightings (bottom). Only a subset of memory locations are shown. Notice the sharp focus of all the weightings on a single location in memory (black is weight zero, white is weight one). Also note the translation of the focal point over time, reflects the networks use of iterative shifts for location-based addressing, observe that the read locations exactly match the write locations. This suggests that the network writes each input vector in turn to a specific memory location during the input phase, then reads from the same location sequence during the output phase.**

Figure 5 shows a subset of memory where we performed the copy task. This figure shows the passage of time from left to right. It also shows the write weightings as well as read weightings. As it is illustrated, the model learned to start writing data to memory location and to prevent overwriting the data, it learned to increase write memory address. Increasing memory address by one at a time is very important since the model had the option to decrease or not increase the address. This pattern can be seen on the bottom part of Figure 5. On the other hand, the top part of this figure shows the input and output data. It can clearly be seen that the output data matches the exact same input except by the last data which indicates the end of the sequence. This is also a very important feature to be learned by model to perform the copy task exactly as expected.

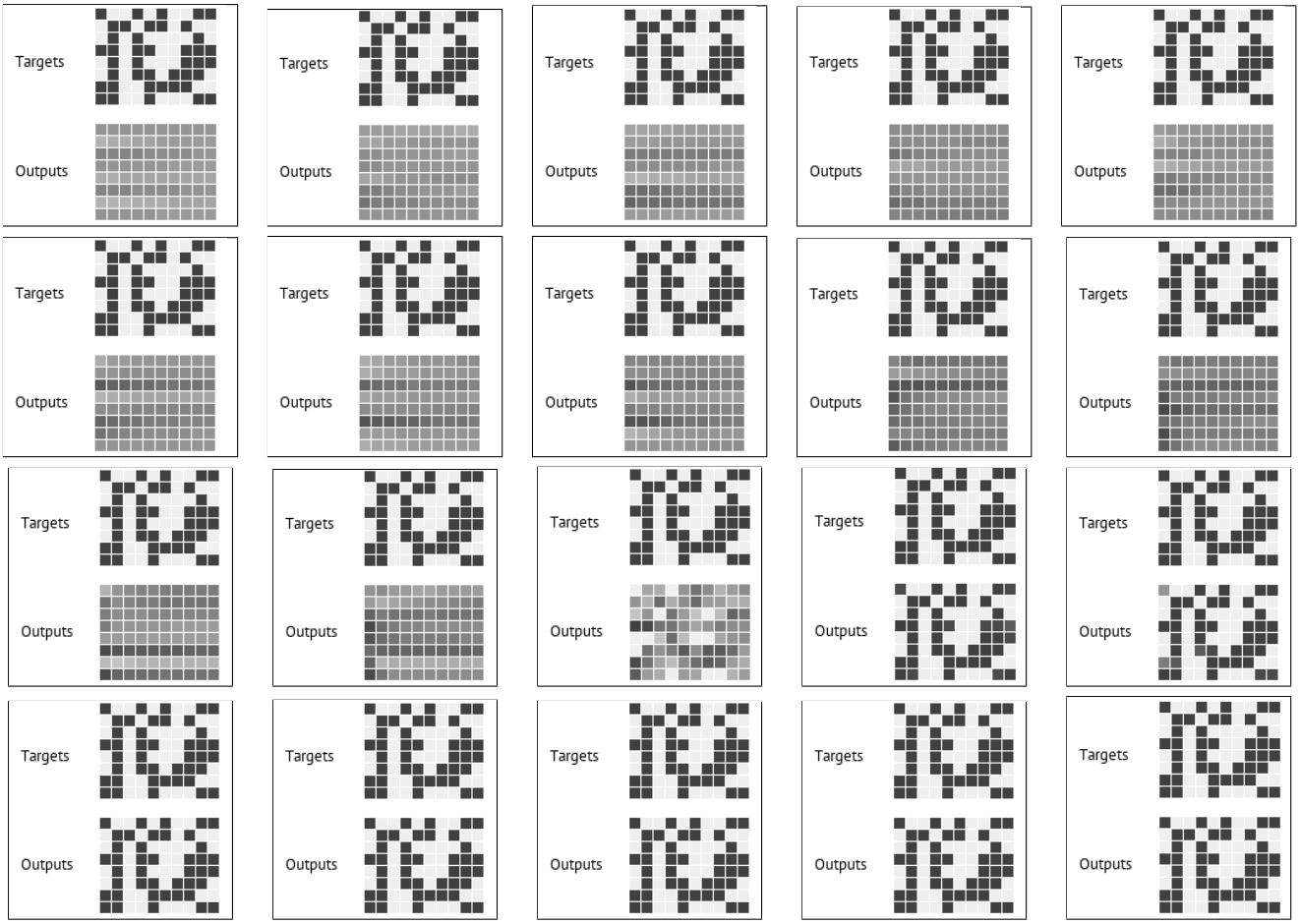

Table 3: From top left to bottom right, this figure shows the process of learning copy task by **LSTM_NTM** model for a sequence of length **10**. Each image shows the target (input) and the read output. Images are taken on different time (sequence number) in the learning process. Each column represents an 8 bit random data. The sequence end indicator is not shown in input.

Figure 5 is a nice image to see but we wanted to know how input and output looks like over time when we are in the process of training. So we sampled input and output at fixed interval and the result is shown in Table 3. As it can be seen, at the beginning, NTM outputs random data (Top Left) since it has no clue what type of operation it needs to learn. By passage of time and after seeing hundreds of patterns, the model starts to capture the pattern and tune the parameters to perform the copy operation (Bottom right).

(e) **Understanding the shift operator:**

The following equation shows the rotational convolution used in the paper:

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

To show how this convolution is actually a rotation shift, we can write down the sum terms inside the convolution as follow. Considering s being a vector with index -1, 0 and 1 we will have:

$$\widetilde{w}_t(i) \leftarrow w(i-1) * s(1) + w(i) * s(0) + w(i+1) * s(-1)$$

Now let us see how this will effect the weight vector to shift depending on the different value of s. Consider a case where $s = [1 \;\; 0 \;\; 0]$ (Not that in this case $s(-1) = 1, s(0) = 0, s(1) = 0$). Now let us assume that the weight vector is as follow: $w_t^g = [0.8 \;\; 0.3 \;\; 0.1]$. Using the equation above, we can see that:

$$\begin{aligned} j = 0 : w_t^g(0) &= 0.1 * 1 + 0.8 * 0 + 0.3 * 0 \\ j = 1 : w_t^g(1) &= 0.8 * 1 + 0.3 * 0 + 0.1 * 0 \\ j = 2 : w_t^g(2) &= 0.3 * 1 + 0.1 * 0 + 0.8 * 0 \end{aligned}$$

From the unrolled convolution equation we know that:

$$\widetilde{w}_t(i) = [w_t^g(0) \;\; w_t^g(1) \;\; w_t^g(2)]$$

Plugging above values and we will get:

$$\widetilde{w}_t(i) = [0.1 \;\; 0.8 \;\; 0.3]$$

Which is a right shifted version of the input vector. Although if the shift weighting parameters are not exactly integers, we will get a blurry output which is a shifted version of the input. The paper have suggested the following equation:

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

The gamma parameter as described earlier is responsible to sharpen the final shifted weights making them closer to the original values as needed. As it was discussed earlier to sharpen the weight we need to use values of more than one for γ. Otherwise the smooth operation will be done by the above normalization equation.

Table 4 shows how to perform only positive shift. The code snippet on top is before and the code snippet on the bottom shows after the change. The arrow points to the part of the code where the softmax is applied on shift weighting. As it can be seen, we used the softplus function in pytorch to perform a rectifying action. This will lead to only performing address increase by 1 or 0 and option -1 will no longer be available.

```
def address_memory(self, k, beta, g, s, Y, w_prev):
    # Handle Activations
    k = k.clone()
    beta = F.softplus(beta)
    g = F.sigmoid(g)
    s = F.softmax(s, dim=1)
    Y = 1 + F.softplus(Y)

    w = self.memory.address(k, beta, g, s, Y, w_prev)

    return w
```

```
def address_memory(self, k, beta, g, s, Y, w_prev):
    # Handle Activations
    k = k.clone()
    beta = F.softplus(beta)
    g = F.sigmoid(g)
    s = F.softmax(F.softplus(s), dim=1)
    Y = 1 + F.softplus(Y)

    w = self.memory.address(k, beta, g, s, Y, w_prev)

    return w
```

Table 4: Figure shows how to perform only positive shift. The code snippet on top is before and the code snippet on the bottom shows after change. The arrow points to the part of the code where the softmax is applied on shift weighting. This will lead to only performing address increase by 1 or 0 and option -1 will no longer be available.