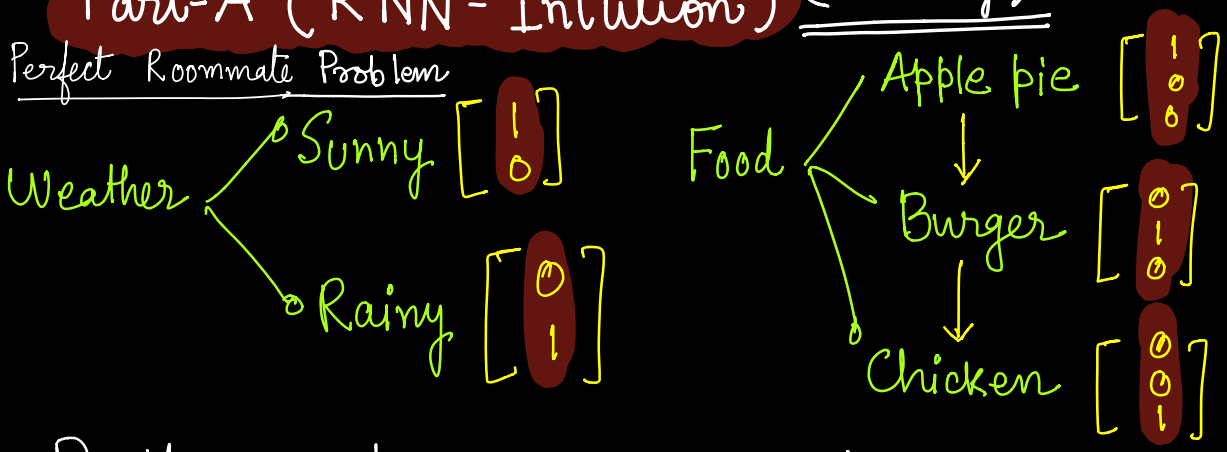
Part-A (RNN - Intution) (Encoding)

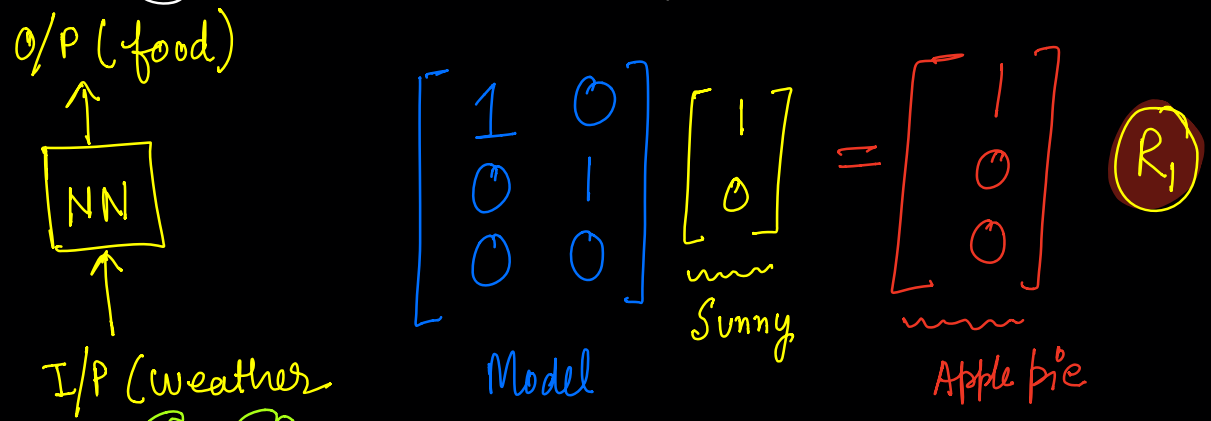
Perfect Roommate Problem

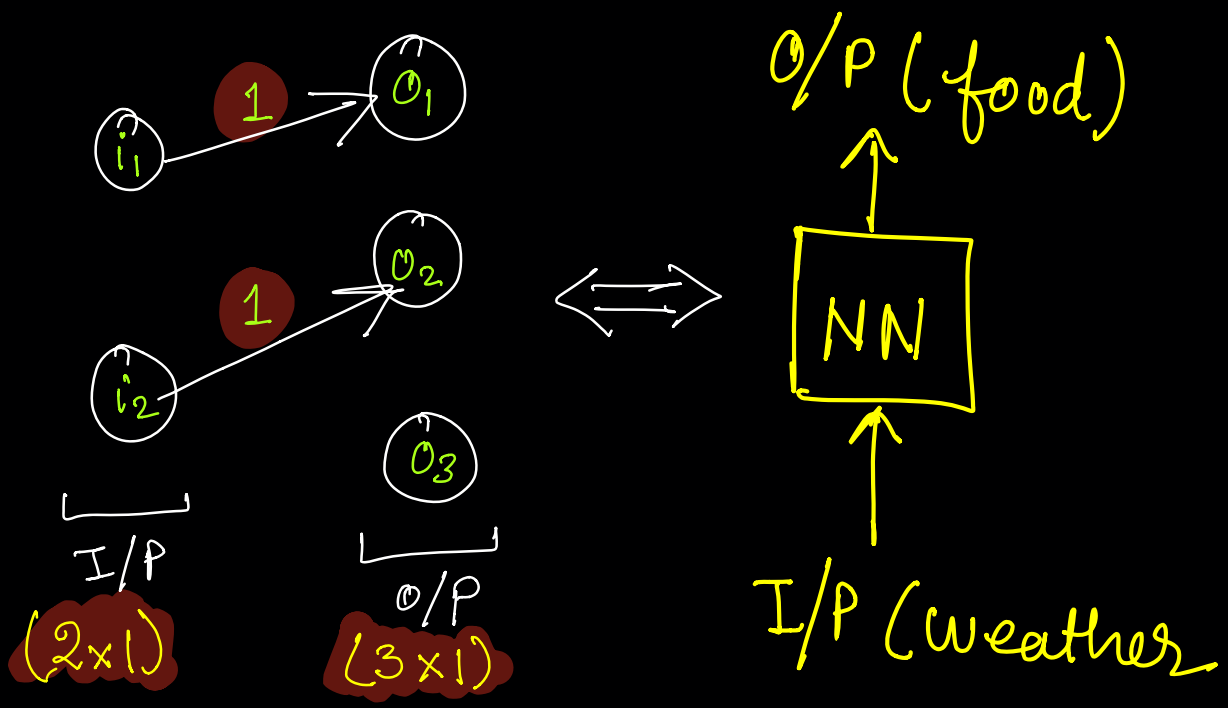


Problem - 01 (Fixed rule)

(*) Weather = Sunny ⟹ Apple pie

(*) Weather = Rainy ⟹ Burger





Problem-02 (Round Robin Schedule)

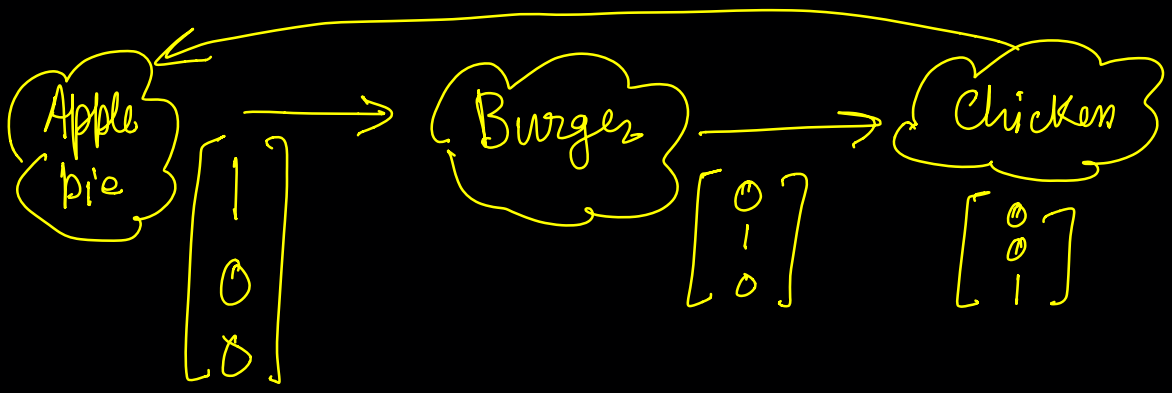
Applepie → Burger → chicken

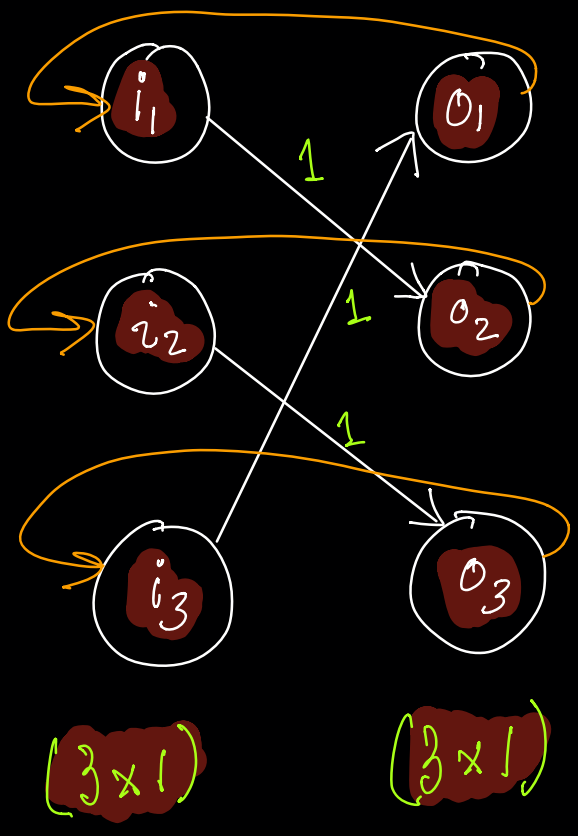
- O/P has to be fed back as i/p
- Network need to remember last food as cook accordingly.

Permutation matrix

$$\begin{matrix} o_1 \\ o_2 \\ o_3 \end{matrix} \begin{matrix} i_1 & i_2 & i_3 \end{matrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} c \\ a \\ b \end{bmatrix}$$

Above permutation matrix can be used as a model to realize the RR scheduling which takes a food as an (i/p) and gives next day food.



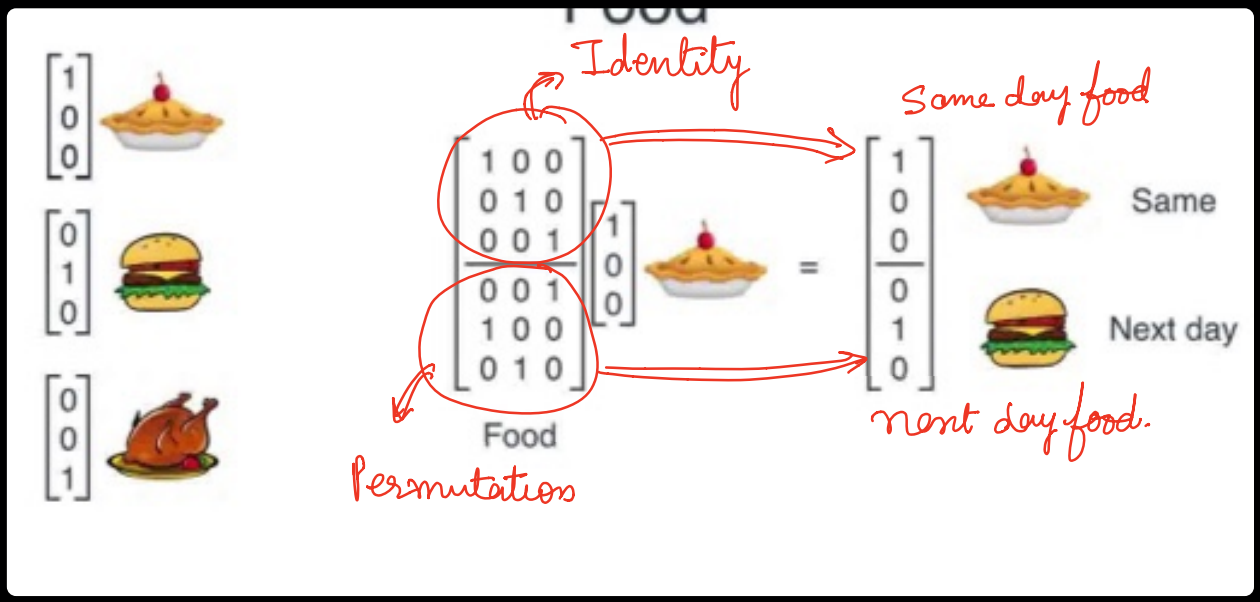


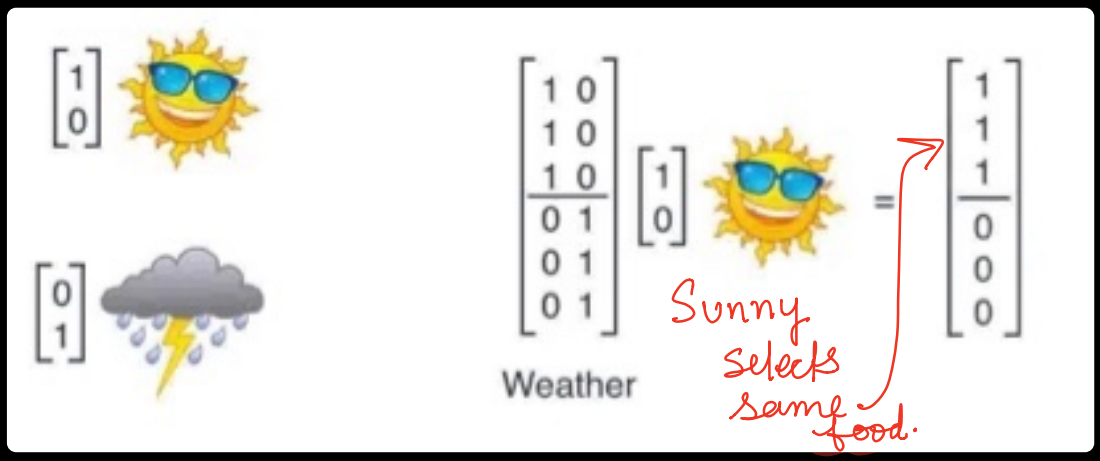
(Previous day food has to be remembered as well as the RR schedule.)

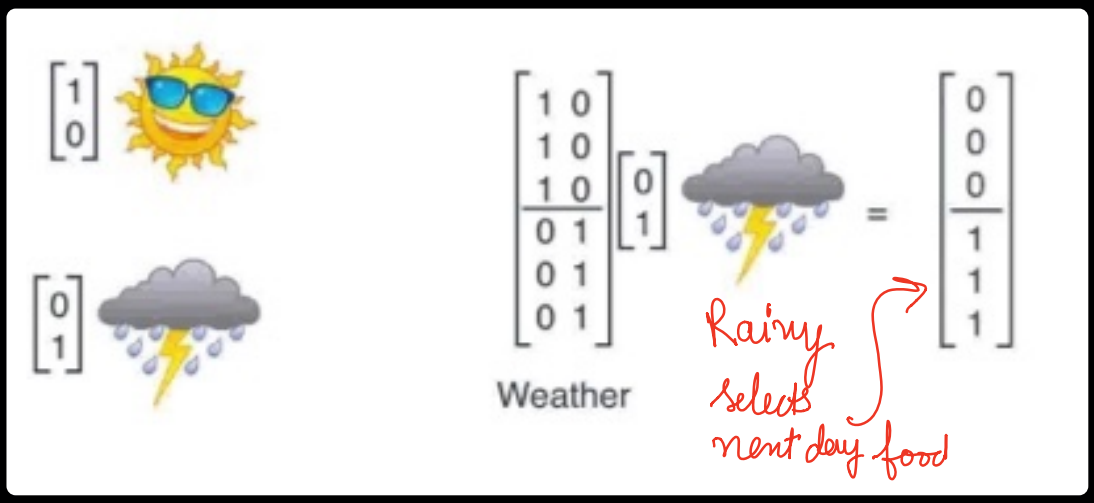
Problem-03 : ⊛ If Sunny, do not cook and just repeat the last day left over food.

⊛ If Rainy, Cook the next day's food according to the RR scheduling.

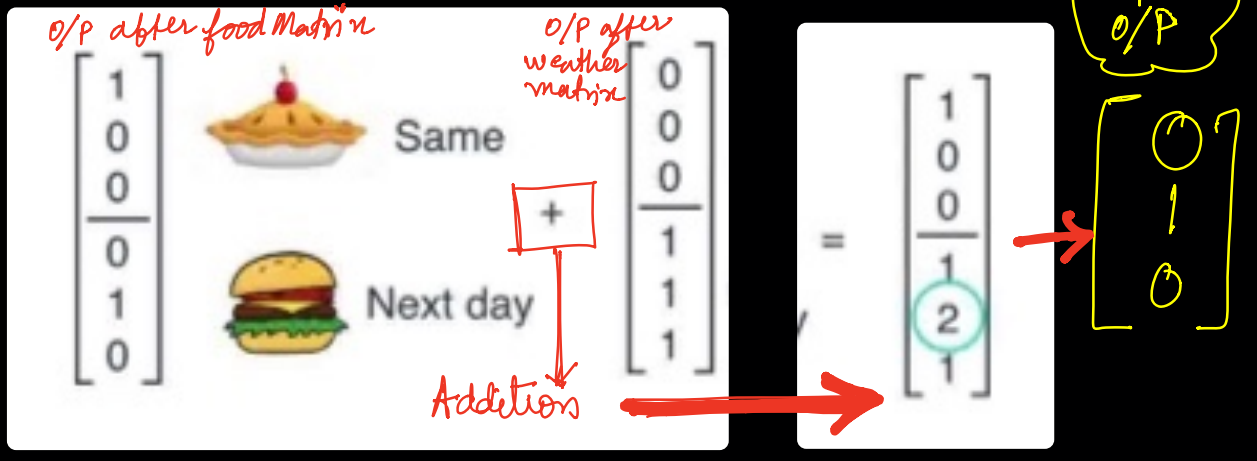
Now along with the previous day food (for RR) one should have to check the weather. Depending upon the weather choose yesterday's food or today's food. Along with yesterday's food today's weather is also an i/p

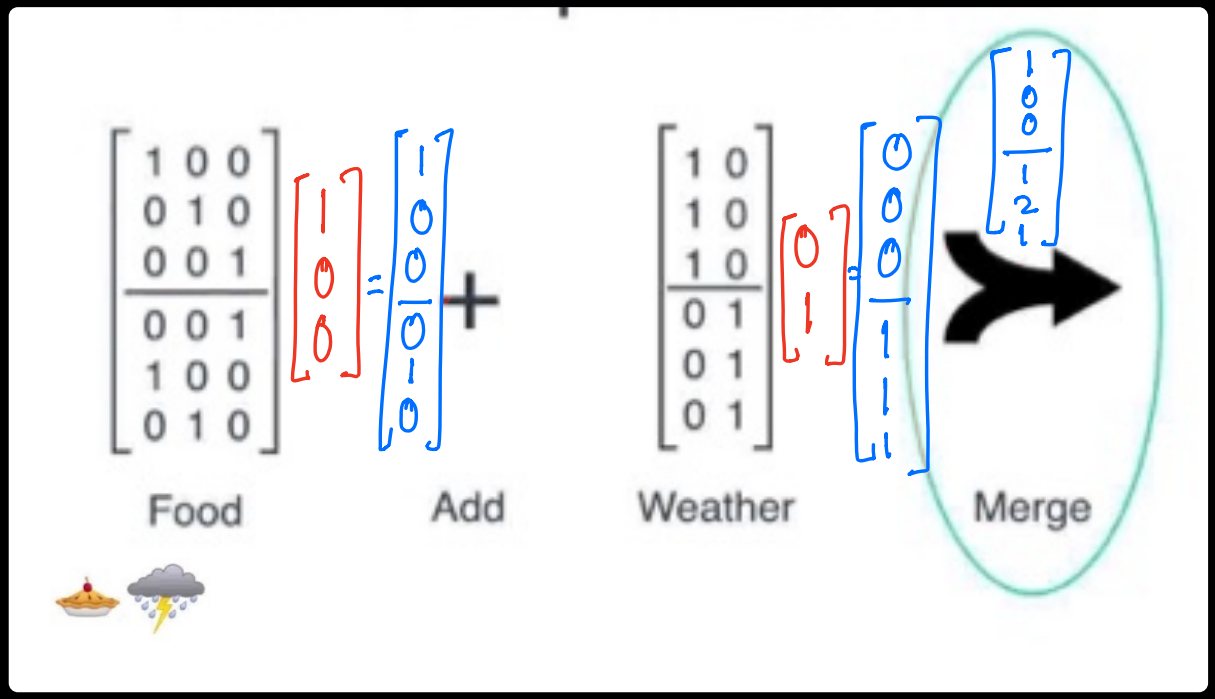


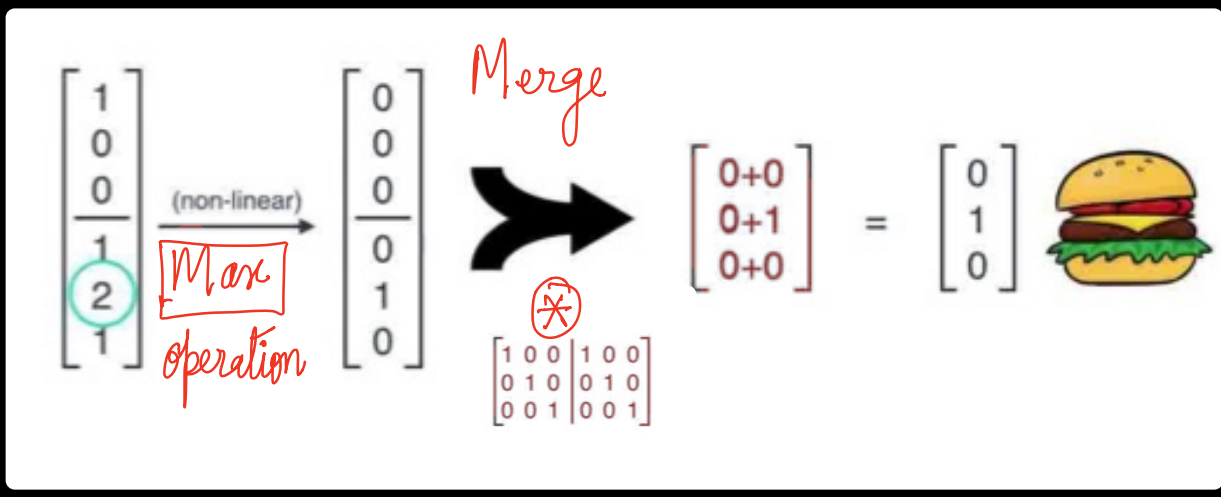




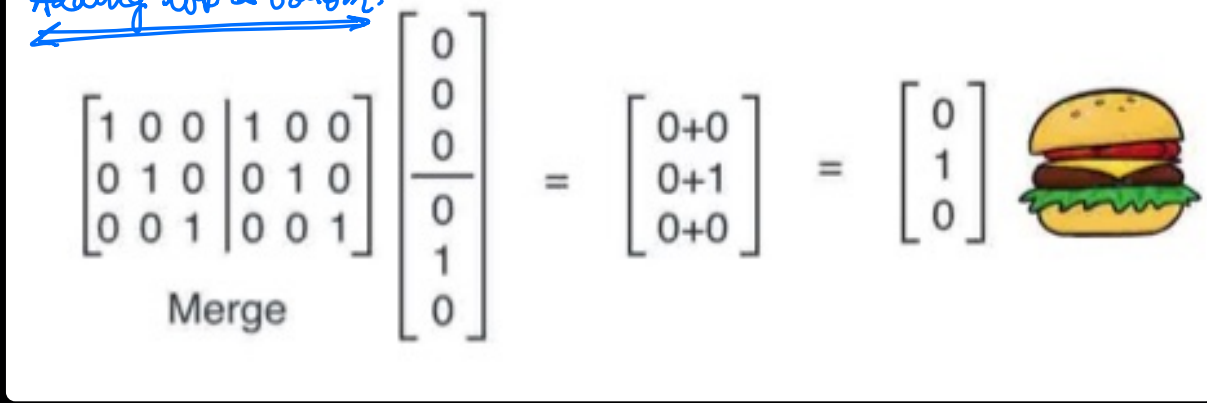
If this is the i/p

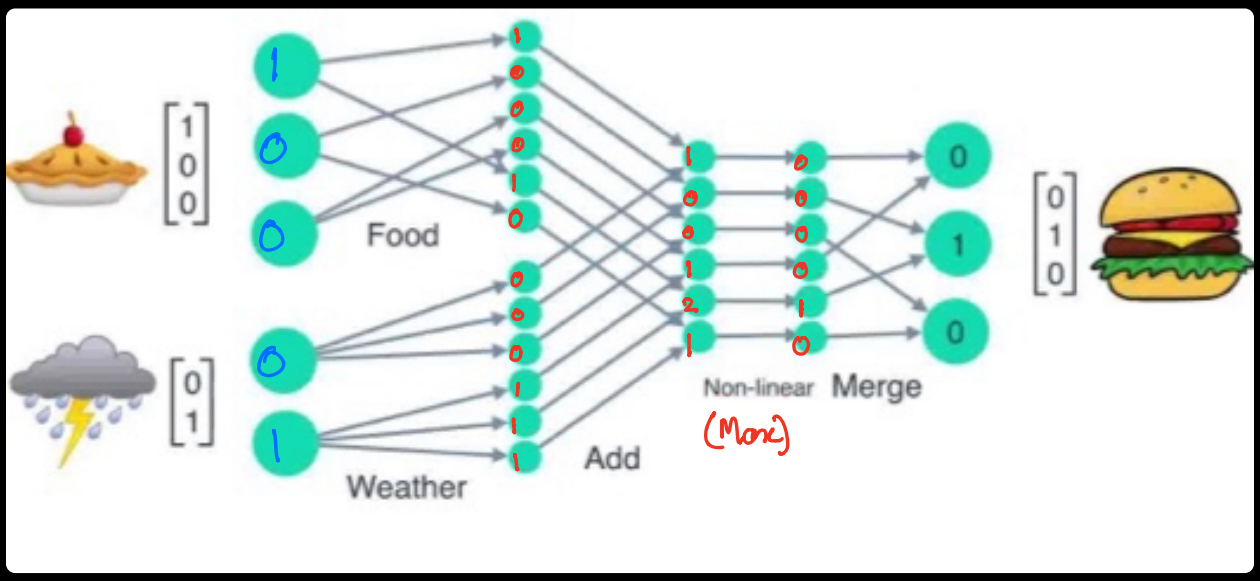


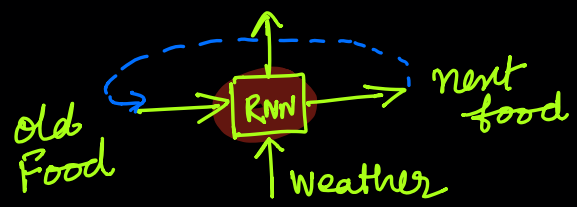


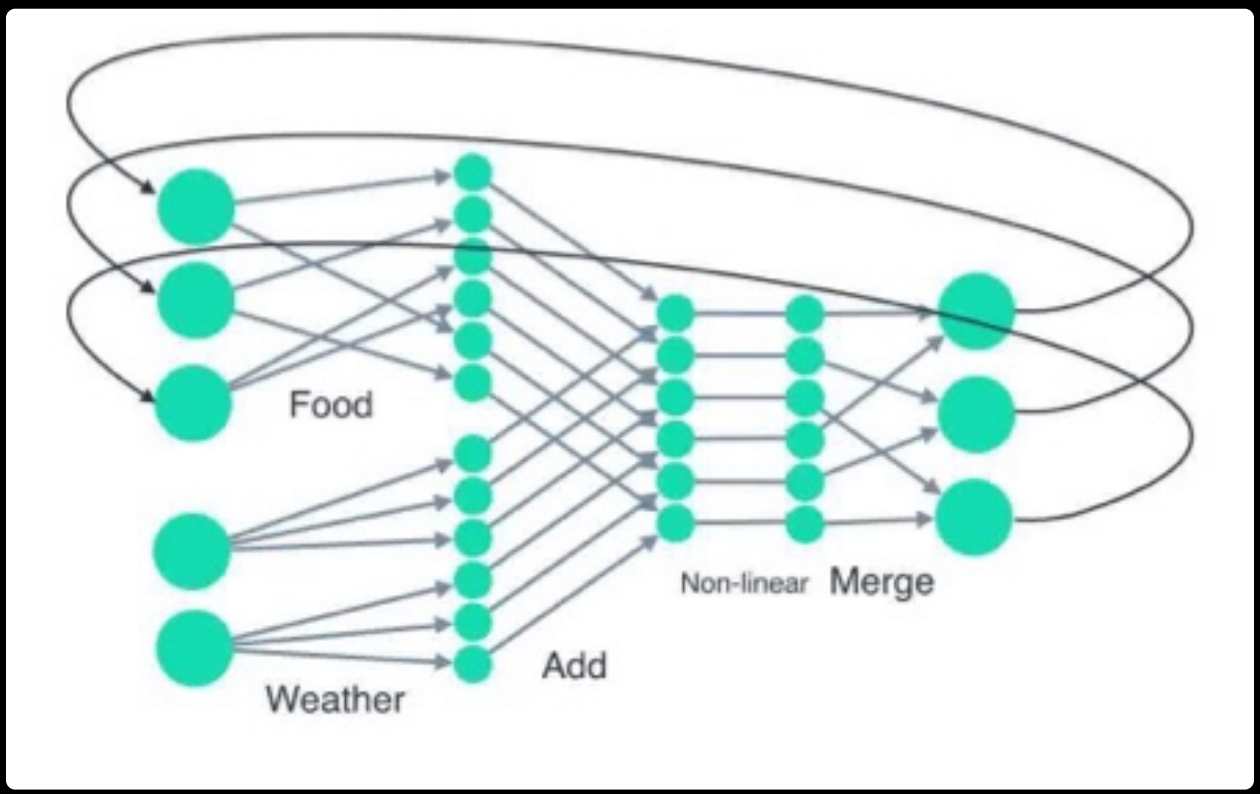


Adding top & bottom.




Food
Weather
Add
Non-linear
Merge
(Max)


old Food
RNN
next food
weather


Food
Weather
Add
Non-linear
Merge

PART-B: Mathematical formulation of RNN

Name recognition in the text modules

useful for indexing people in news article

(A) Given any sentence, find/lable out the names. (Text labeling)

Say:

$X^{<1>}$ $X^{<2>}$ ---- $X^{<t>}$ -- . $X^{<T_x>}$

X : Sachim played Shoaib very well.

(1st word) (2nd word) - - - - - - - - - .

(i/p) (first I/P sentence)

(o/p) Y : 1 0 1 0 0

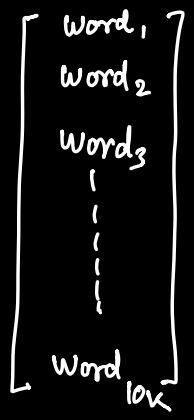
$Y^{<1>}$ $Y^{<2>}$ --- $Y^{<t>}$ $Y^{<T_y>}$

Here $T_x = T_y$ (Total length of (X) and (Y) Seq) (may or may not be same)

$X^{i<t>} \Longrightarrow$ (i^{th}) i/p training example.

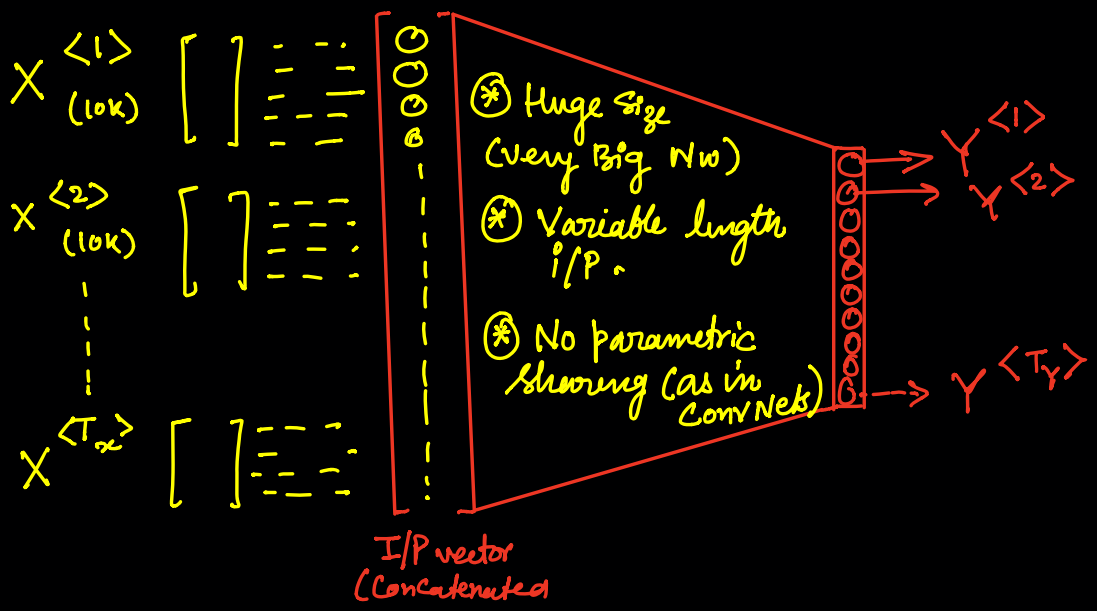
T_x^i and $T_y^i \Longrightarrow$ Corresponding lengths.

(B) Word representations: (1 Hot encoding)

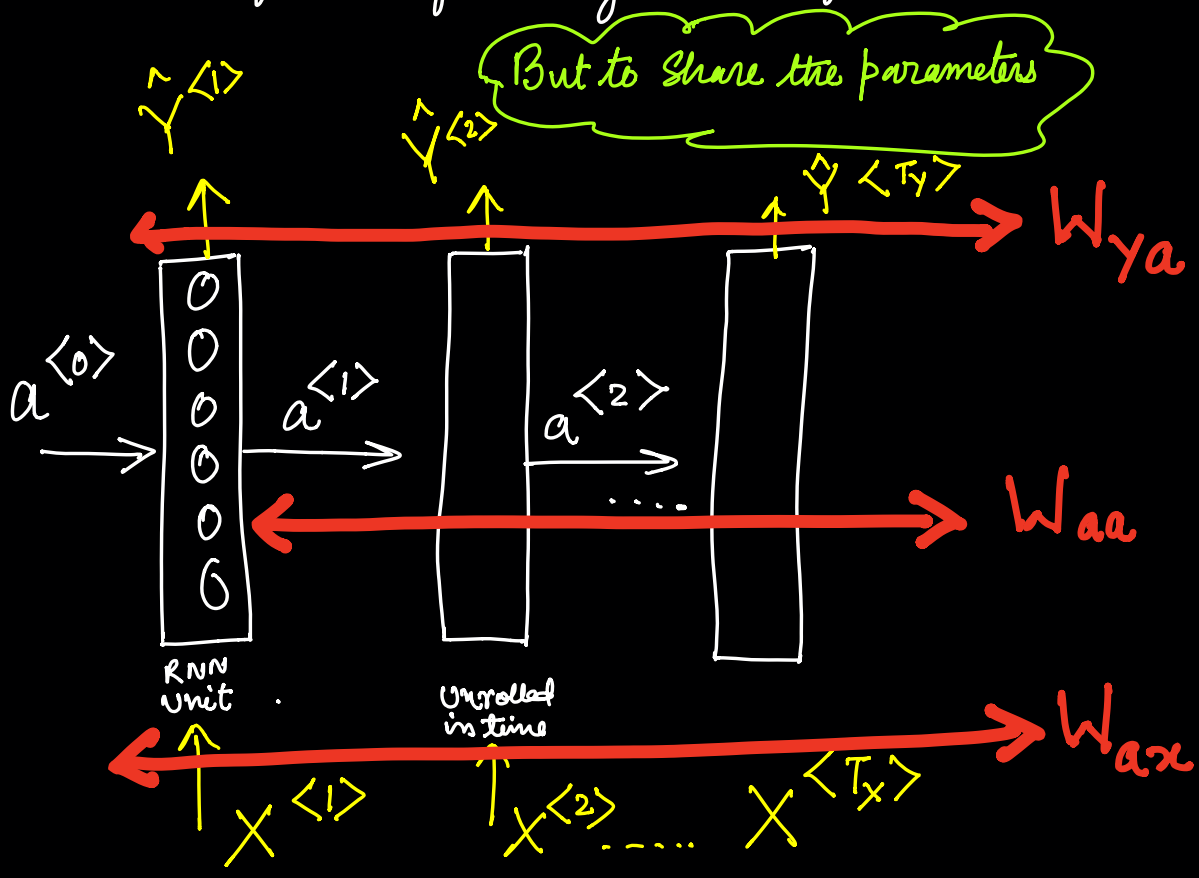


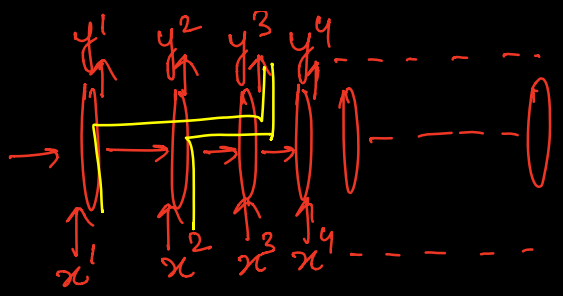
* make a dictionary of unique words in your traing data. (say 10K)
* Each word can be represented as a 10K dimensional 0/1 (binary vector) ⇒ 1-Hot encoding.
* Our goal to get a Label (Y) for given (X).

Learning a mapping from (X) ⟹ (Y)



* We require our network to read the sentence word by word from left to right (Seeing data as a Time Seq)



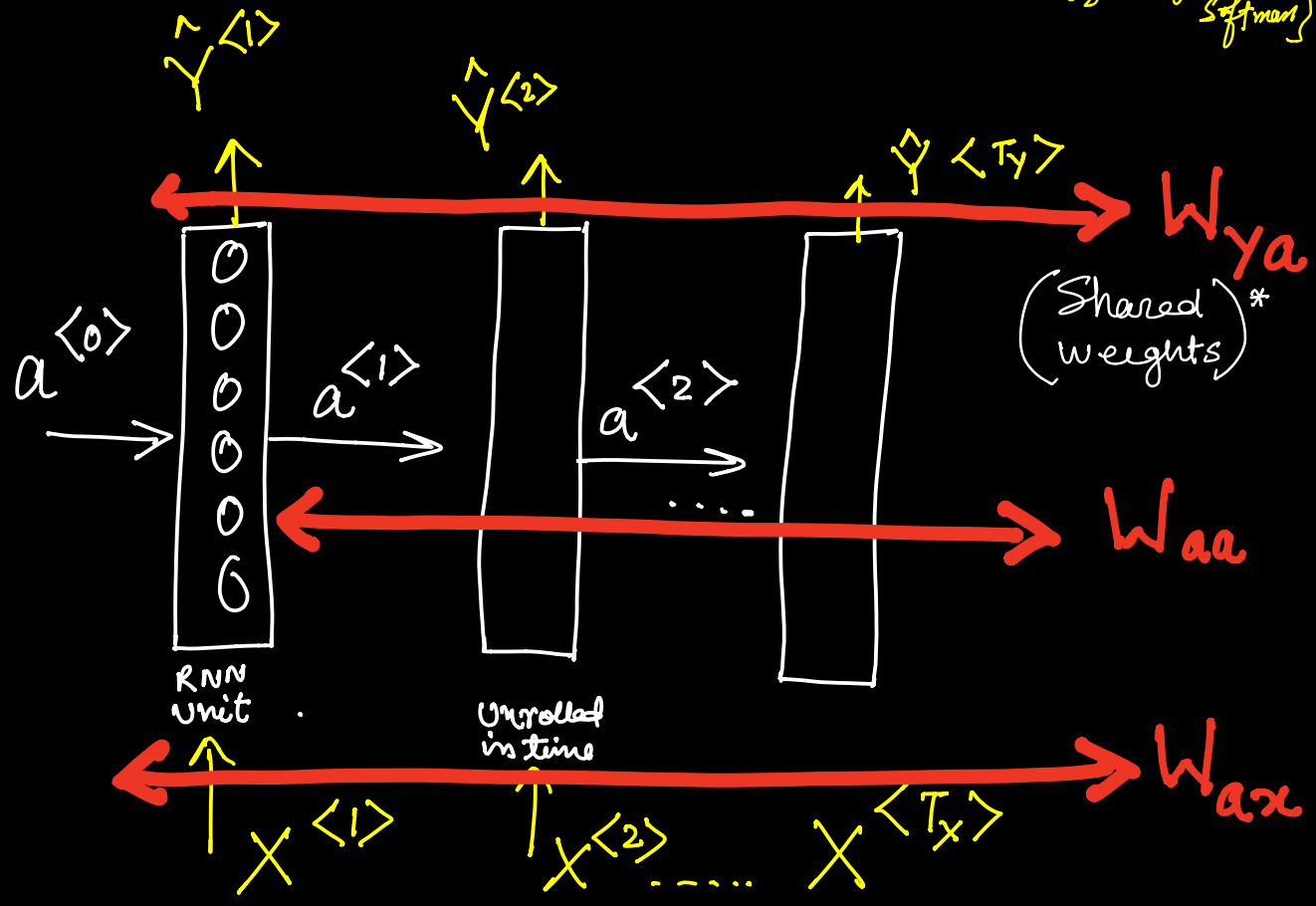


In order to predict $\hat{y}^{<3>}$ from $x^{<3>}$ $x^{<1>}$ & $x^{<2>}$ they are also utilized.

① $a^{<0>} = \vec{0}$ (initialized)

② $a^{<t>} = g_1(W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>} + b_a)$

③ $\hat{y}^{<t>} = g_2(W_{ya}\, a^{<t>} + b_y)$ $\left\{ g_1 \equiv \tanh/\text{Relu},\ g_2 \equiv \text{Sigmoid}/\text{Softmax} \right\}$



$$a^{<t>} = g\left(W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>} + b_a\right)$$

(100×100)

2D matrix

Take a 100-dim vector and convert it into another vector to encode the previous history

(100×1)

100-dim Hidden representation

$(100 \times 10{,}000)$

2D-matrix

Takes an input vector $x^{<t>}$ of 10,000 dimensions and convert it into augumented history encoding

$(10{,}000 \times 1)$

Input vector $x^{<t>}$ of 10,000 dimensions

Matrix Augumentation :

(weight Matrix)

(a) $\left[\underset{100\times100}{W_{aa}} \;\middle|\; \underset{100\times10{,}000}{W_{ax}}\right] = \underset{100\times10100}{W_a}$

[Horizontal Concat]

(b) $\left[\begin{matrix} a^{<t-1>} \\ \hline x^{<t>} \end{matrix}\right] = \left[a^{<t-1>}, x^{<t>}\right]$

$a^{<t-1>}$: 100×1; $x^{<t>}$: $10{,}000\times1$; 10100×1

[Vertical Concat]

$\therefore\; W_a\left[a^{<t-1>}, x^{<t>}\right] \Rightarrow [100 \times 1]$

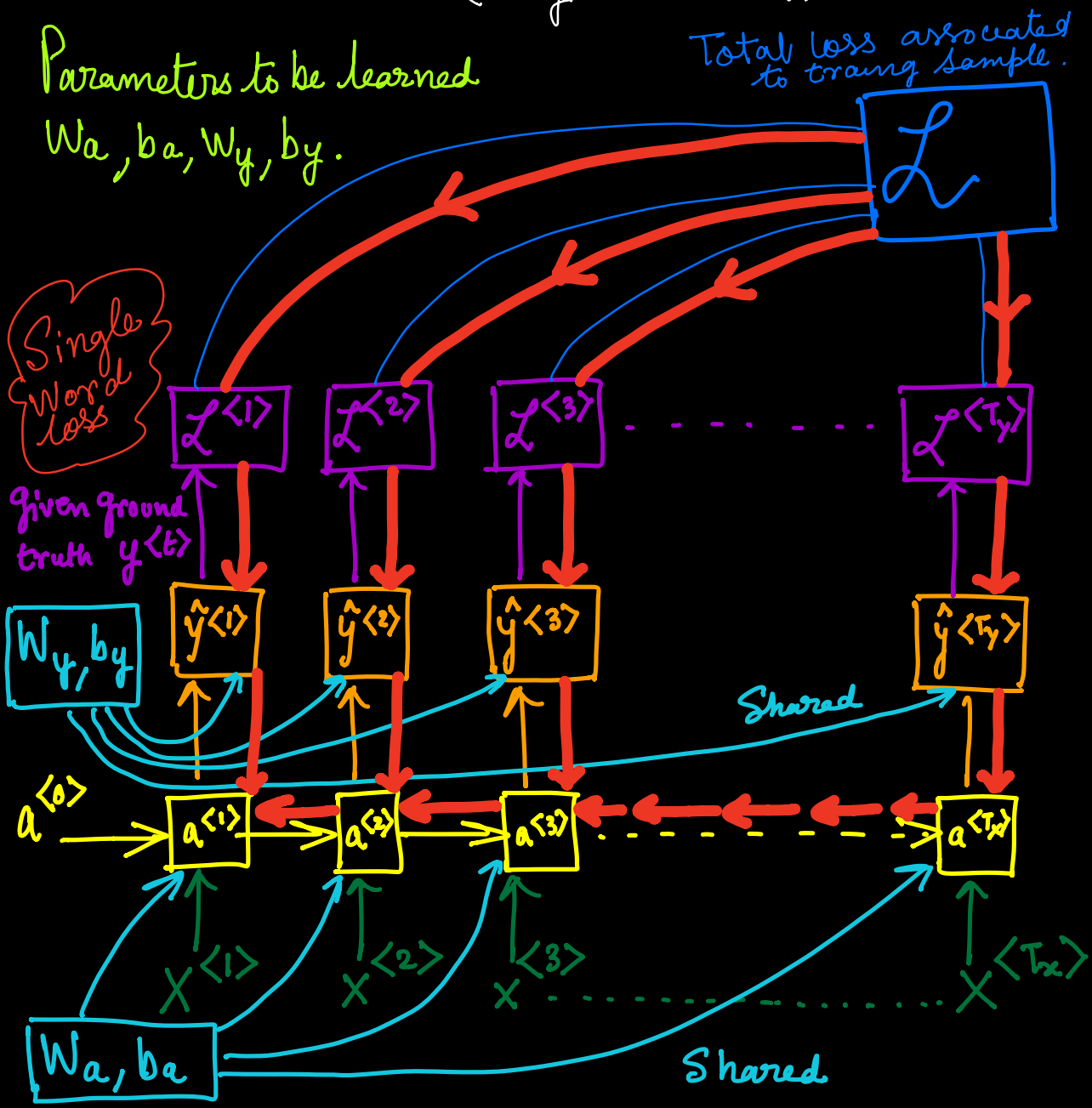
$[I \mid I]\begin{bmatrix} a \\ b \end{bmatrix} = [a+b]$

Therefore our equations :

$$a^{<t>} = g_1\left(W_a\left[a^{<t-1>}, x^{<t>}\right] + b_a\right)$$

$$\hat{y}^{<t>} = g_2\left(W_y a^{<t>} + b_y\right)$$

Parameters to be learned
W_a, b_a, W_y, b_y.

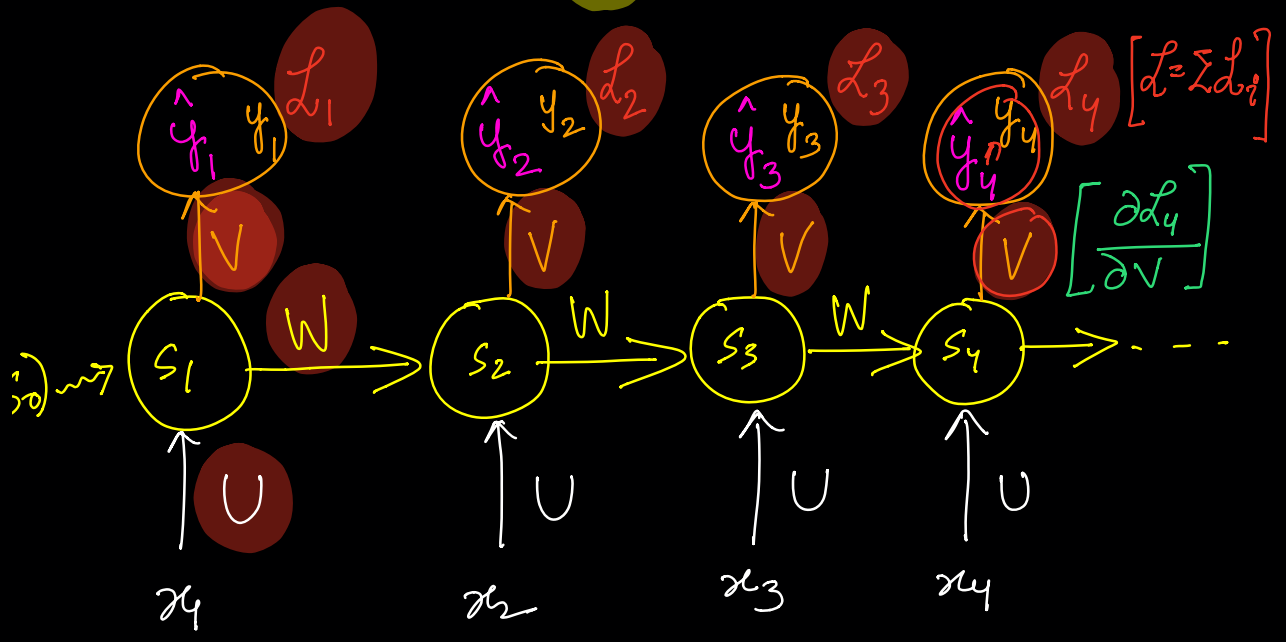


① Word level loss

$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1-y^{<t>}) \log(1-\hat{y}^{<t>})$$

② Sentence level loss

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$



Part C : Vanishing Gradient (RNN backprop)

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3 + \mathcal{L}_4 \qquad \text{where} \left[\mathcal{L}_i = -\log\left(\hat{y}_i^{tc}\right)\right]$$

Predicted probability of true character at $(t=i)$

$$\left(\frac{\partial \mathcal{L}}{\partial V} \,\middle|\, \frac{\partial \mathcal{L}}{\partial W} \,\middle|\, \frac{\partial \mathcal{L}}{\partial U}\right) \left(\frac{\partial \mathcal{L}}{\partial V/W/U} = \sum_{K=1}^{4} \frac{\mathcal{L}_K}{\partial V/W/U}\right)$$

(need to be computed)

as $\dfrac{\partial \mathcal{L}_4}{\partial V} = \dfrac{\partial f(y_4, \hat{y}_4)}{\partial V} = \dfrac{\partial f(g(V), \hat{y}_3)}{\partial V}$ [direct Computation]

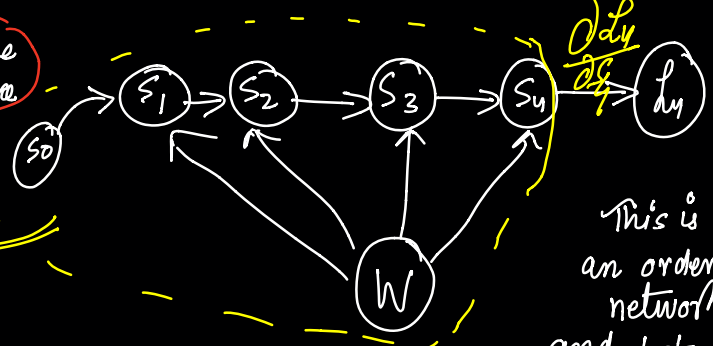
See Computation Graph for (W)

$$\frac{\partial \mathcal{L}_4}{\partial W} = \frac{\partial \mathcal{L}_4}{\partial s_4} * \frac{\partial s_4}{\partial W}$$

(But) s_4 is recursively dependent on $(s_3 \ldots s_1)$
Simply treating (s_3) Const and differentiate don't work.

Direct dependence

Recursive dependence

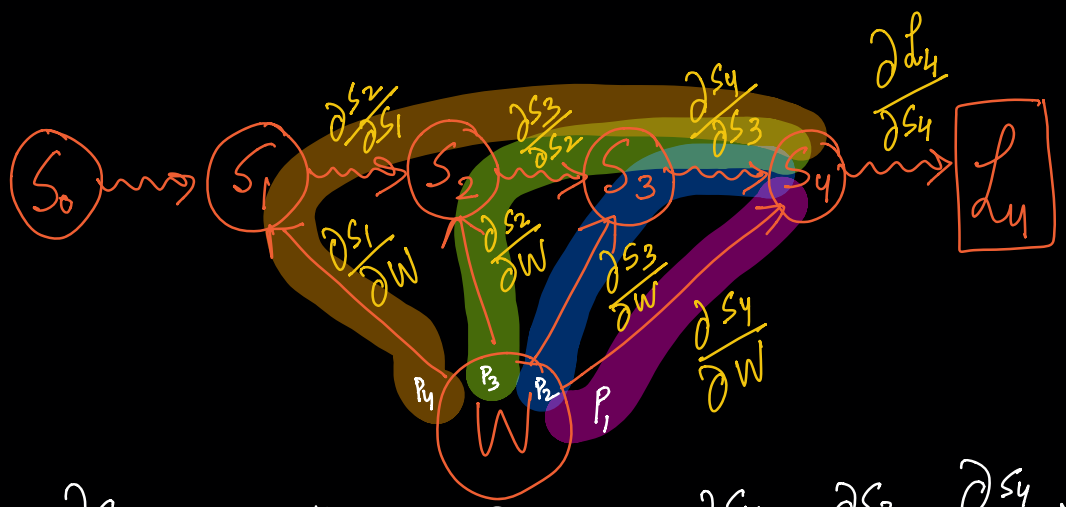


All backward paths one can take to reach (W) from (s_4)

$\left(\dfrac{\partial s_4}{\partial W}\right)$

This is an ordered network and total derivate need to computated after adding all the possible paths.

To Compute $\left[\dfrac{\partial s_4}{\partial W}\right]$ ⇓



$$\therefore \frac{\partial S_4}{\partial W} = \text{Adding all Paths derivatio} = \frac{\partial S_4}{\partial W} + \frac{\partial S_4}{\partial S_3} * \frac{\partial S_3}{\partial W} + \frac{\partial S_4}{\partial S_3} * \frac{\partial S_3}{\partial S_2} * \frac{\partial S_2}{\partial W}$$

$\hookrightarrow \frac{\partial S_4}{\partial S_2}$

$\frac{\partial S_4}{\partial S_4}$

$$+ \frac{\partial S_4}{\partial S_3} * \frac{\partial S_3}{\partial S_2} * \frac{\partial S_2}{\partial S_1} * \frac{\partial S_1}{\partial W}$$

$\hookrightarrow \frac{\partial S_4}{\partial S_1}$

$$\frac{\partial S_4}{\partial W} = \sum_{K=1}^{4} \frac{\partial S_4}{\partial S_K} * \frac{\partial S_K}{\partial W}$$

$$\therefore \frac{\partial \mathcal{L}_4}{\partial W} = \frac{\partial \mathcal{L}_4}{\partial S_4} * \frac{\partial S_4}{\partial W} = \frac{\partial \mathcal{L}_4}{\partial S_4} * \left[\sum_{K=1}^{4} \frac{\partial S_4}{\partial S_K} * \frac{\partial S_K}{\partial W} \right]$$

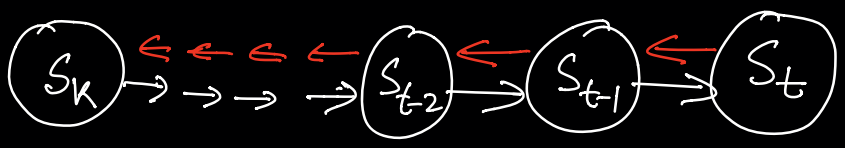
$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \frac{\partial \mathcal{L}_2}{\partial W} + \frac{\partial \mathcal{L}_3}{\partial W} + \frac{\partial \mathcal{L}_4}{\partial W}$$

[This is called BPTT]

Bounded

$S_i = f(W, S_{i-1})$
$\therefore$ related to $[W]$

This algorithm back propagation is times need to Compute a chained differentiation which requires recursive and repetative multiplication. $\left(\frac{\partial S_t}{\partial S_k}\right)$



$$\frac{\partial S_t}{\partial S_k} = \frac{\partial S_t}{\partial S_{t-1}} * \frac{\partial S_{t-1}}{\partial S_{t-2}} \times \cdots \frac{\partial S_{k+1}}{\partial S_k} = \prod_{j=k}^{t-1} \frac{\partial S_{j+1}}{\partial S_j}$$

Since these (S_i's) are activated o/p hence are bounded therefore there derivatis are also bounded

$$(\text{Sigmoid})' \leq 1/4$$

$$(\tanh)' \leq 1$$

$\therefore$ $\frac{\partial L}{\partial W}$ (may explode/Vanish)

① Word level loss

$$\mathcal{L}^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -y^{\langle t \rangle} \log \hat{y}^{\langle t \rangle} - (1 - y^{\langle t \rangle}) \log(1 - \hat{y}^{\langle t \rangle})$$

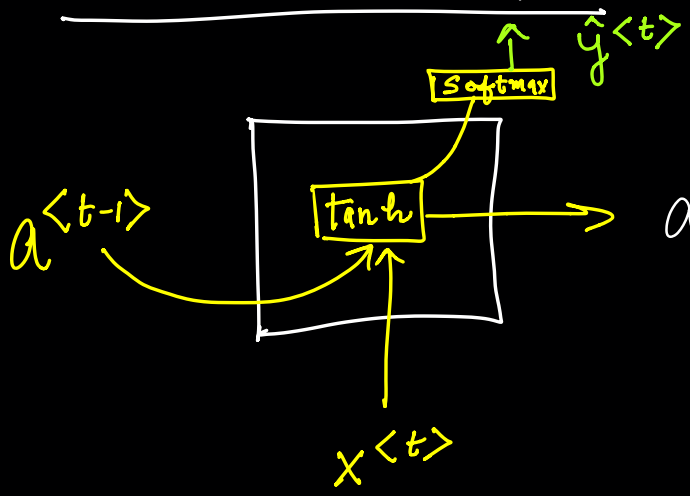
② Sentence level loss

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$$

Major Problem RNN faces is Vanishing gradients and as the backpropagation is in time it fails to update well with respect to some old time.

↳ This causes RNN to forget about long term events and their effect.

Vanilla RNN Block diagram



$$a^{\langle t \rangle} = \tanh\left[W_a\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_a\right]$$

PART-D : Gated Recurrent Unit (GRU)

(*) Virat is a good player. He is the Captain ----..

dependency

(*) Three persons are going with us and all of them are ---

long term dependency.

Problem is that how to remember that subject is

Singular / Plural // Male / female

was / were He / She

⇒ we consider our hidden state memory cells to understand such dependencies and tract / keep on memorizing them till the point they are required or something more important need to be remembered.

$\Gamma_u = 1$ $\Gamma_u = 0$ ---------- $\Gamma_u = 0$ $\Gamma_u = 0$ ⇒ (When to toggle)

Keep on remember

Virat is a good player. He is the Captain ----..

dependency

Now you may update

Remember he is MALE (what to keep)

$c^t = 1$ $c^t = 1$ ------ $c^t = 1$

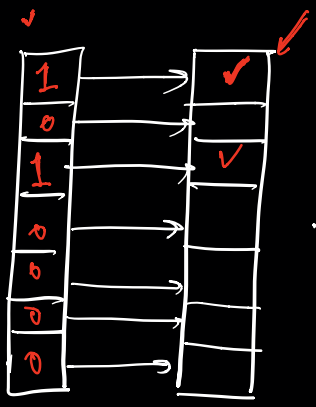
(some bit value got set)

We need to remember Subjects Gender or it's Singular or plural.

Gender → mal, female

(*) Here c^{t} ⇒ It can be seen as the cell value at time (t).

Γ_u Gamma u ⇒ It can see as the gate [allows/block] Some information



Element wise multiplication will help to produce the effect of gating.

(Γ_u^{t}) (c^{t})

Our I/p's are at (t)

$c^{<t-1>}$ and $x^{<t>}$

New Cell State needs to be Computed

(a) Candidate update value

$$\tilde{c}^{<t>} = \tanh\left(W_c\left[c^{<t-1>}, x^{<t>}\right) + b_c\right)$$

Neural network parameterized by (W_c, b_c) with input as $c^{<t-1>}$ and $x^{<t>}$ [what to update]

(b) Update gate Γ_u (when to update)

$$\Gamma_u = \sigma\left(W_u\left[c^{<t-1>}, x^{<t>}\right] + b_u\right)$$

Neural network parameterized by (W_u, b_u) with input as $c^{<t-1>}$ and $x^{<t>}$

This network tries to learn wheather we need to update the current Cell State or not.

Γ_u allows flexible Cell updation as it is activated using Sigmoid fn which is always in b/w $\{0, 1\}$

(c) Cell update :

Simplified GRU

$$c^{<t>} = \underbrace{\Gamma_u \odot \tilde{c}^{<t>}}_{\text{What needs to be updated}} + \underbrace{(1 - \Gamma_u)\, c^{<t-1>}}_{\text{How much to retain}}$$

Γ_u is 1 $\Rightarrow$ Signifies that to update the cell Content by $\tilde{c}^{<t>}$ Else keep the same.

But update governed only by one variable Γ_u

$c^{\langle t \rangle}$, $\tilde{c}^{\langle t \rangle}$, Γ_u all are of same dimensions say (100×1) then Cell update equation is element wise multiplication

(Γ_u) understands few key cell states bits behaviour (what they are learning) and just try to gate their (O/P) effectively so as reduce the loss.

Full GRU :

How relevant any bit of $c^{\langle t-1 \rangle}$ can be — Another neural network for gating

① $\tilde{c}^{\langle t \rangle} = \tanh\left(W_c \left[\Gamma_r \odot c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$

② $\Gamma_u = \sigma\left(W_u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$

③ $\Gamma_r = \sigma\left(W_r \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$

Relevance gate, another neural network parameterized by (W_r, b_r) with input as $(c^{\langle t-1 \rangle}, x^{\langle t \rangle})$

④ $c^{\langle t \rangle} = \Gamma_u \odot \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) \odot c^{\langle t-1 \rangle}$

after several updates handle longer range dependencies, vanishing gradient and better convergence. $[a^{\langle t \rangle} = c^{\langle t \rangle}]$

PART-E : GRU to LSTM

GRU

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c\left[\Gamma_r \odot c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

relevance gate

LSTM

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

In LSTM (no) relevance gate

Update gate (GRU)

$$\Gamma_u = \sigma\left(W_u\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$$

Signifies which bits require updation remaining bits are not changed/modified.

$\tilde{c}^{\langle t \rangle}$ and $c^{\langle t-1 \rangle}$ are weighted by Γ_u.

Update gate (LSTM)

$$\Gamma_u = \sigma\left(W_u\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$$

This signifies which bits need to be update.

$\tilde{c}^{\langle t \rangle}$ will only be weighted

Relevance gate (GRU) absent in LSTM

$$\Gamma_r = \sigma\left(W_r\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$$

Forget gate (LSTM) ~~absent in LSTM~~ GRU

$$\Gamma_f = \sigma\left(W_f\left[a^{<t-1>}, x^{<t>}\right] + b_f\right)$$

LSTM will learn a forget gate seperately. Basicallly how we need to forget from $c^{<t-1>}$ (Previous State)

Output gate (LSTM)

$$\Gamma_o = \sigma\left(W_o\left[a^{<t-1>}, x^{<t>}\right] + b_o\right)$$

absent in GRU ($a^{<t>} = c^{<t>}$)

$c^{<t>}$ gated by Γ_o is the final o/p

Updation (GRU)

$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + (1 - \Gamma_u) \odot c^{<t-1>}$$

Updation (LSTM)

$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

---

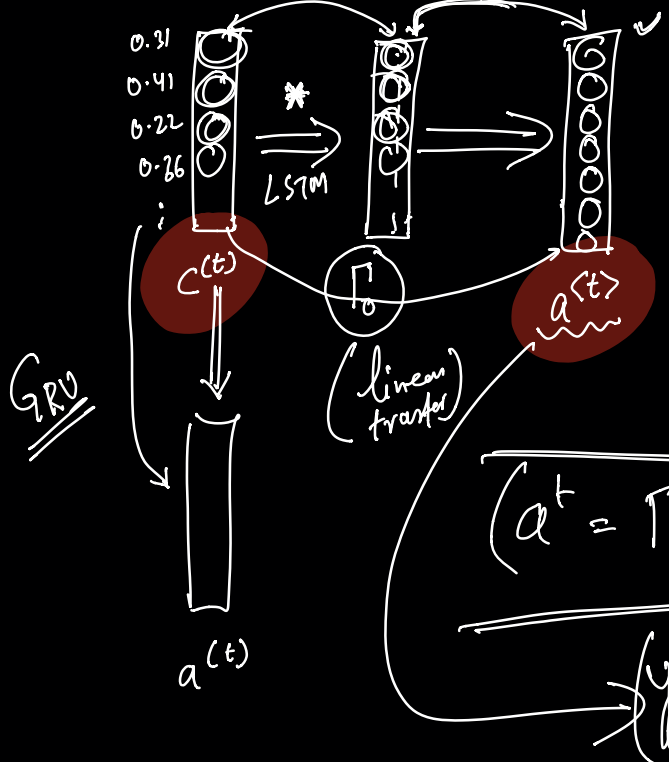
final o/p GRU

$$a^{\langle t \rangle} = c^{\langle t \rangle}$$

---

final o/p LSTM

$$a^{\langle t \rangle} = \Gamma_o \odot c^{\langle t \rangle}$$



GRU (Γ_u) $(1-\Gamma_u)$

LSTM $(\Gamma_u \quad \Gamma_f)$

Γ_o (O/P is missing)

$a^t = \Gamma_o * c^{(t)}$

$y = W a^{\langle t \rangle}$

Sequence Prediction Problems

PART F : Review

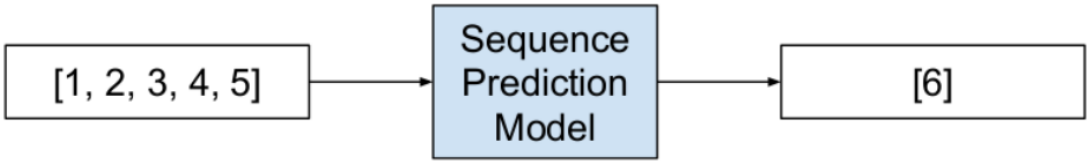
- Sequence Prediction.
- Sequence Classification.
- Sequence Generation.
- Sequence-to-Sequence Prediction.

Sequence Prediction.

Sequence prediction involves predicting the next value for a given input sequence. For example:

```
Input Sequence: 1, 2, 3, 4, 5
Output Sequence: 6
```

Listing 1.1: Example of a sequence prediction problem.



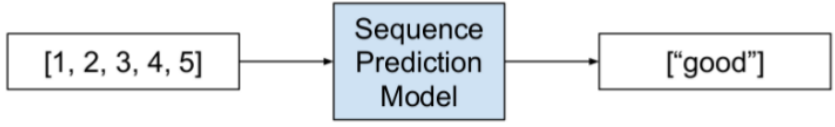
- Weather Forecasting. Given a sequence of observations about the weather over time, predict the expected weather tomorrow.
- Stock Market Prediction. Given a sequence of movements of a security over time, predict the next movement of the security.
- Product Recommendation. Given a sequence of past purchases for a customer, predict the next purchase for a customer.

Sequence Classification

Sequence classification involves predicting a class label for a given input sequence. For example:

```
Input Sequence: 1, 2, 3, 4, 5
Output Sequence: "good"
```

Listing 1.2: Example of a sequence classification problem.



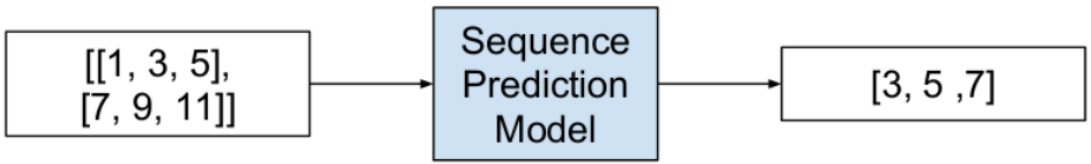
- DNA Sequence Classification. Given a DNA sequence of A, C, G, and T values, predict whether the sequence is for a coding or non-coding region.
- Anomaly Detection. Given a sequence of observations, predict whether the sequence is anomalous or not.
- Sentiment Analysis. Given a sequence of text such as a review or a tweet, predict whether the sentiment of the text is positive or negative.

Sequence Generation.

Sequence generation involves generating a new output sequence that has the same general characteristics as other sequences in the corpus. For example:

```
Input Sequence: [1, 3, 5], [7, 9, 11]
Output Sequence: [3, 5 ,7]
```

Listing 1.3: Example of a sequence generation problem.



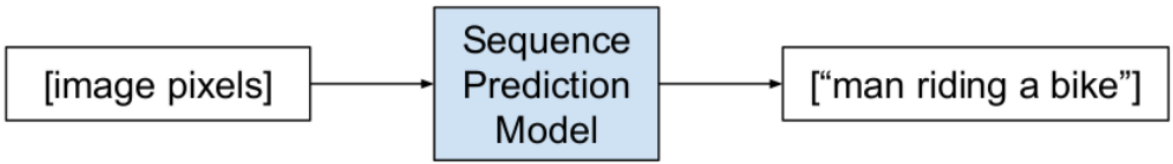
- Text Generation. Given a corpus of text, such as the works of Shakespeare, generate new sentences or paragraphs of text that read they could have been drawn from the corpus.
- Handwriting Prediction. Given a corpus of handwriting examples, generate handwrit- ing for new phrases that has the properties of handwriting in the corpus.
- Image Caption Generation. Given an image as input, generate a sequence of words that describe an image.

Sequence-to-Sequence Prediction

For example:

```
Input Sequence: [image pixels]
Output Sequence: ["man riding a bike"]
```

Listing 1.4: Example of a sequence generation problem.



- Multi-Step Time Series Forecasting. Given a time series of observations, predict a sequence of observations for a range of future time steps.
- Text Summarization. Given a document of text, predict a shorter sequence of text that describes the salient parts of the source document.
- Program Execution. Given the textual description program or mathematical equation predict the sequence of characters that describes the correct output.

Limitations of Multilayer Perceptrons

- Stateless. MLPs learn a fixed function approximation. Any outputs that are conditional on the context of the input sequence must be generalized and frozen into the network weights.
- Unaware of Temporal Structure. Time steps are modeled as input features, meaning that network has no explicit handling or understanding of the temporal structure or order between observations.
- Messy Scaling. For problems that require modeling multiple parallel input sequences, the number of input features increases as a factor of the size of the sliding window without any explicit separation of time steps of series.
- Fixed Sized Inputs. The size of the sliding window is fixed and must be imposed on all inputs to the network.
- Fixed Sized Outputs. The size of the output is also fixed and any outputs that do not conform must be forced.

Promise of Recurrent Neural Networks

- Recurrent Neural Networks, or RNNs for short, are a special type of neural network designed for sequence problems.
- Given a standard feedforward MLP network, an RNN can be thought of as the addition of loops to the architecture.
- For example, in a given layer, each neuron may pass its signal laterally (sideways) in addition to forward to the next layer.
- The output of the network may feedback as an input to the network with the next input vector. And so on.
- The recurrent connections add state or memory to the network and allow it to learn and harness the ordered nature of observations within input sequences.

Introduction to LSTMs

- The Long Short-Term Memory, or LSTM, network is a type of Recurrent Neural Network.
- Long Short-Term Memory (LSTM) is able to solve many time series tasks unsolvable by feedforward networks using fixed size time windows.
- The addition of sequence is a new dimension to the function being approximated.
- Instead of mapping inputs to outputs alone, the network is capable of learning a mapping function for the inputs over time to an output.
- The internal memory can mean outputs are conditional on the recent context in the input sequence, not what has just been presented as input to the network.
- In a sense, this capability unlocks sequence prediction for neural networks.

RNN vs LSTM

- Like RNNs, the LSTMs have recurrent connections so that the state from previous activations of the neuron from the previous time step is used as context for formulating an output.
- But unlike other RNNs, the LSTM has a unique formulation that allows it to avoid the problems that prevent the training and scaling of other RNNs.
- The key technical historical challenge faced with RNNs is how to train them effectively.
- Experiments show how difficult this was where the weight update procedure resulted in weight changes that quickly became so small as to have no effect (vanishing gradients) or so large as to result in very large changes or even overflow (exploding gradients).
- LSTMs overcome this challenge by design.

LSTM architecture

- The computational unit of the LSTM network is called the memory cell, memory block, or just cell for short.
- The term neuron as the computational unit is so ingrained when describing MLPs that it too is often used to refer to the LSTM memory cell.
- LSTM cells are comprised of weights and gates.

LSTM Weights

A memory cell has weight parameters for the input, output, as well as an internal state that is built up through exposure to input time steps.

- Input Weights. Used to weight input for the current time step.
- Output Weights. Used to weight the output from the last time step.
- Internal State. Internal state used in the calculation of the output for this time step.

LSTM Gates

The key to the memory cell are the gates. These too are weighted functions that further govern the information flow in the cell. There are three gates:

- Forget Gate: Decides what information to discard from the cell.
- Input Gate: Decides which values from the input to update the memory state.
- Output Gate: Decides what to output based on input and the memory of the cell.

The forget gate and input gate are used in the updating of the internal state. The output gate is a final limiter on what the cell actually outputs.