Mehrdad Baradaran

Assignment 5

RNN-LSTM-GRU

Text Prediction

99222020

Exercise 1:

In machine learning, it is always assumed that the training samples are independent and uniform. But if the sequence values have time dependence between them, it is rejected like time series data.

So sequential modeling algorithms can have two architectures.

1) stateless

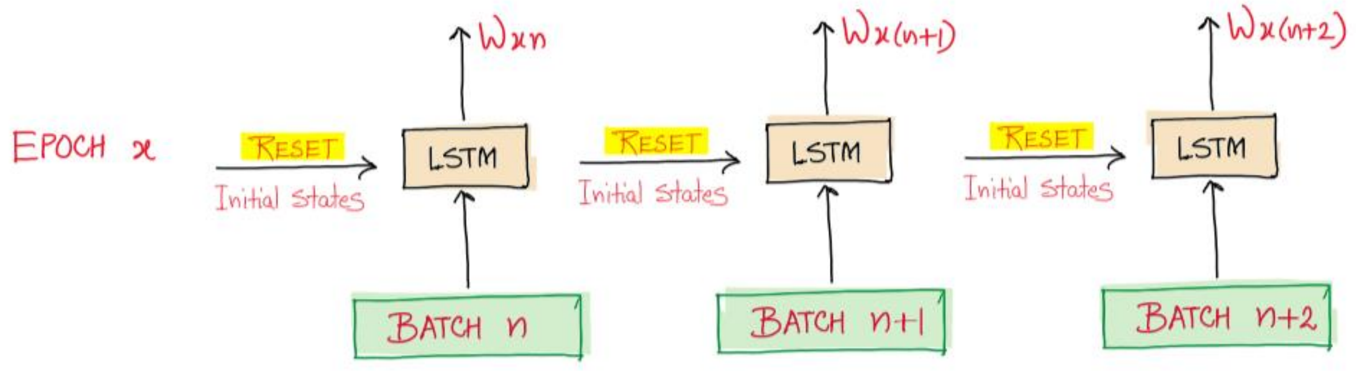
2) stateful

For better understanding, we will check further with LSTM, it will be the same with other models like RNN and GRU.

Stateless architecture is used when the assumption of data independence is established.

This means that there is no interrelationship between categories and each category is independent of each other

A typical training process in a stateless LSTM architecture is shown below:



But in the stateful states (cellular and hidden states) of the model (related to each category)

As the training process progresses from one batch to another batch, the value is added.

Of course, you should not be confused with changing weights and updating them during training.

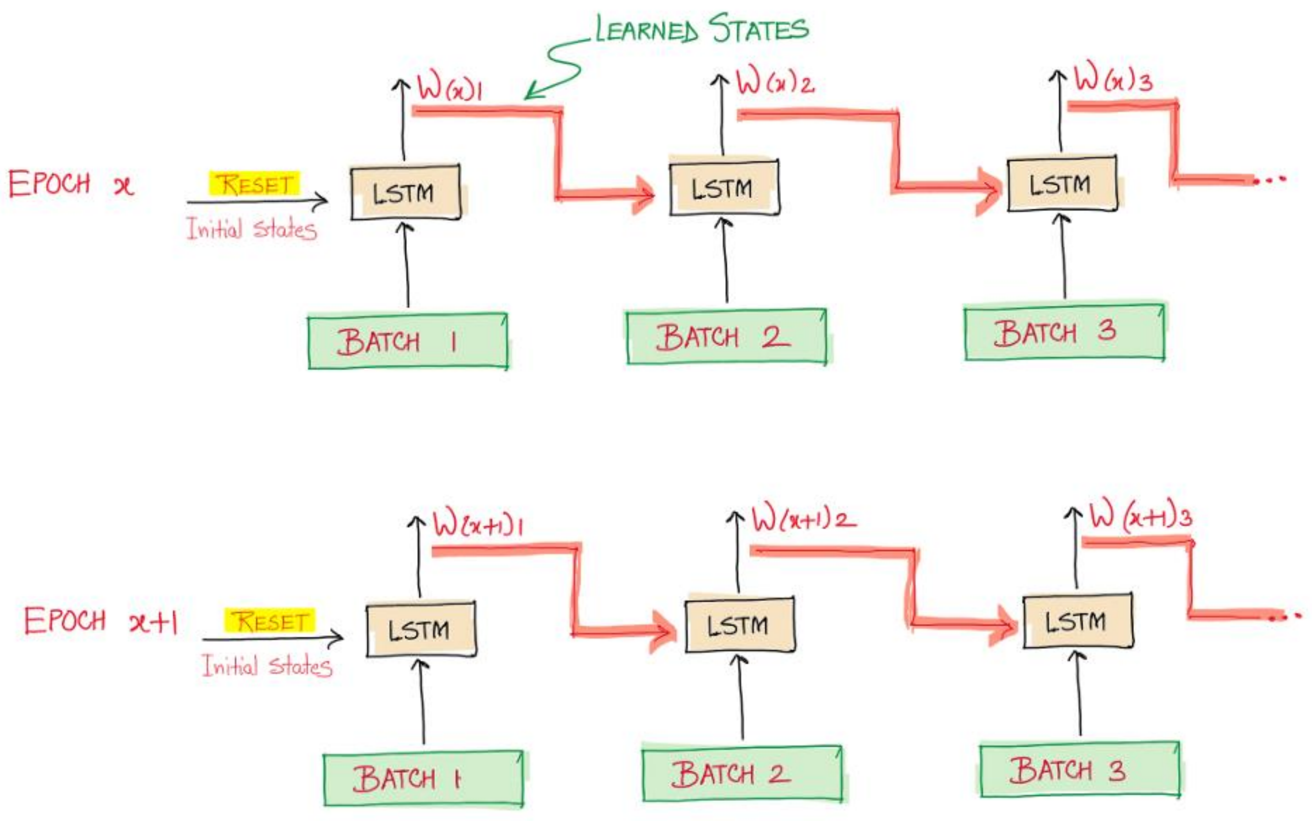
In the stateless method, by changing each input bacth and processing, the initial states of LSTM are reset to zero.

After that, the learned knowledge will not be used and will not be published.

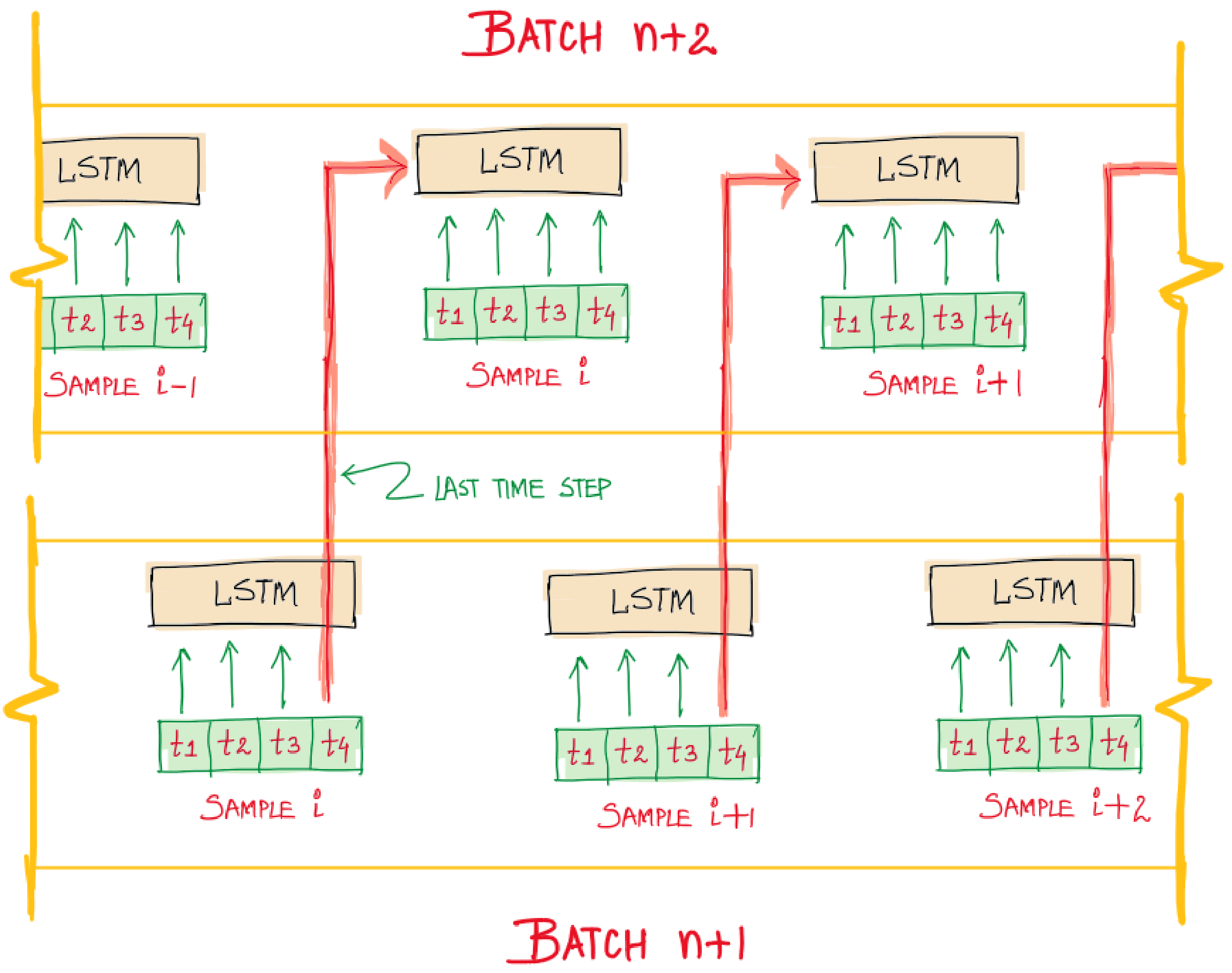
Sequential data, like time series, are dependent and not uniformly distributed, so the propagation of learned states in subsequent batches is intuitive, so that the model captures temporal dependence not only within each sample sequence, but also across batches.

But the textual data consisting of sentences is assumed to be independent of each other and the stateless method can be used in them.

Below is what the Stateful LSTM architecture looks like:



The cells and hidden states of LSTM are initialized for each input batch using the learning from the previous batch, so the model learns the dependencies between batches.



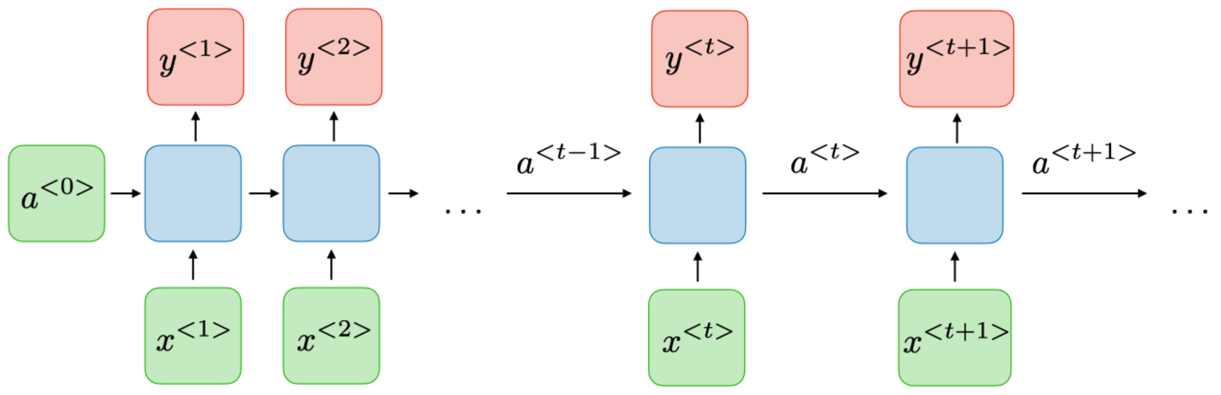
As a result, sample i will be used in the calculation of sample X[i + b s] in the next batch.

Actually, the last state for sample i will be the initial state for sample i in the new batch

In each sample, the time step is divided into 4 parts, and the values of LSTM states at time step t=4 are used for initialization in the next batch.

Exercise 2:

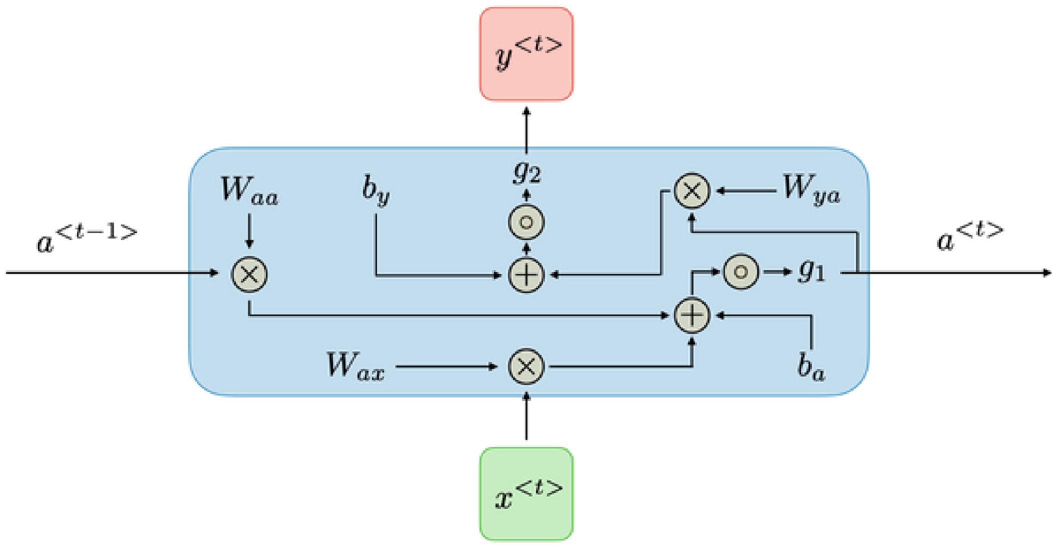
RNN neural networks, or recurrent neural networks, allow previous outputs to be used as inputs while having hidden states.



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

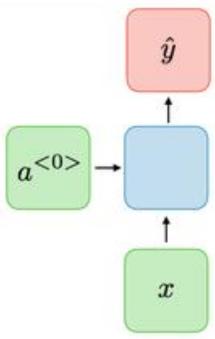
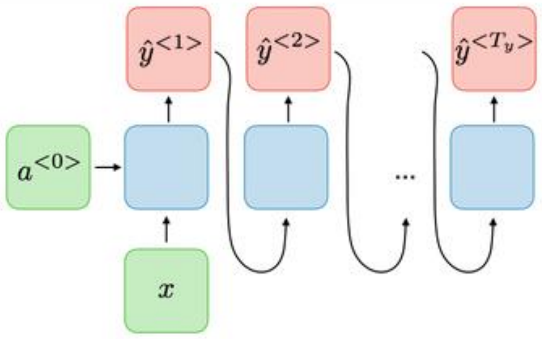
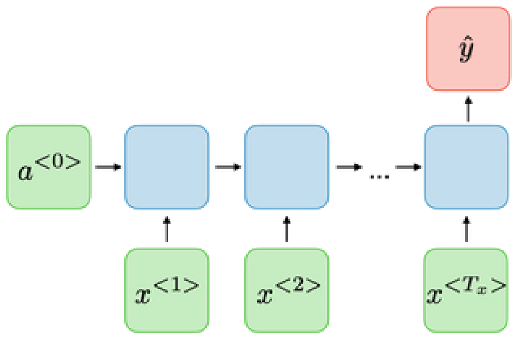
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.



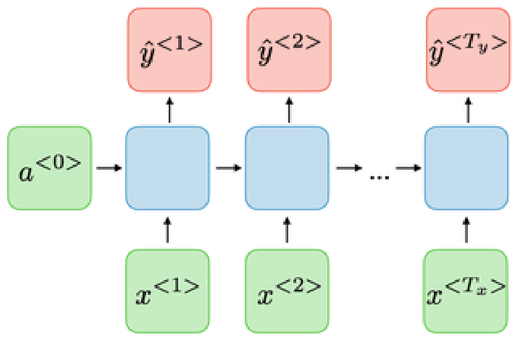
Some of the advantages and disadvantages of RNN neural networks are as follows:

Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

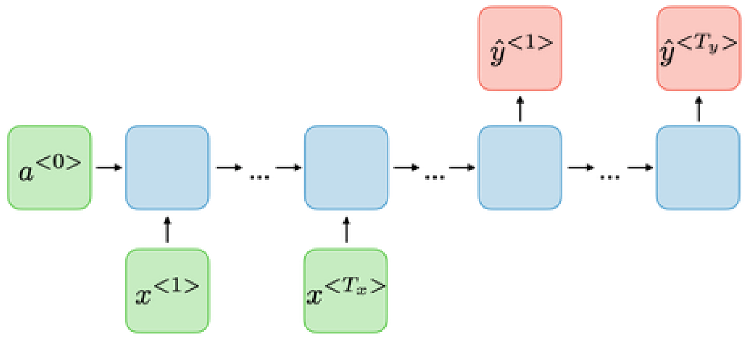
There are different architectures for this neural network, the last two models need more investigation.

One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$	 $\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<T_y>}$ $a^{<0>}$ ... x	Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

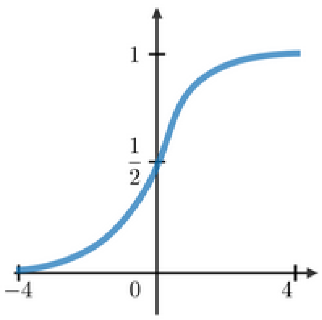
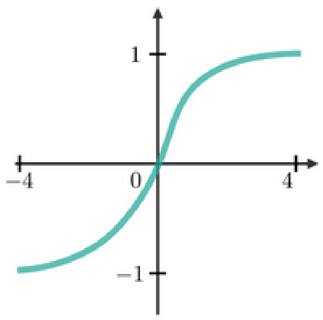
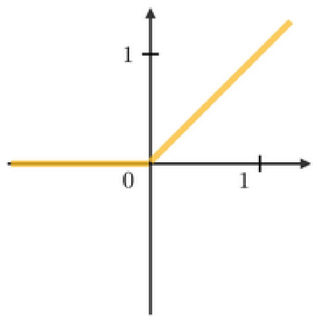
This many-to-many model, which is called entity recognition, is more useful for text predictions because without the need for other inputs, it creates the corresponding output by receiving the corresponding input, which is what is needed for text prediction.

Many-to-many $T_x = T_y$	 $\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<T_y>}$ $a^{<0>}$ ... $x^{<1>}$ $x^{<2>}$ $x^{<T_x>}$	Name entity recognition

But this model that we will show at the end is a model known as machine translation. And they are known as a type of RNN Encoder/Decoder model.

Many-to-many $T_x \neq T_y$	 $\hat{y}^{<1>}$ $\hat{y}^{<T_y>}$ $a^{<0>}$ ... ... $x^{<1>}$ $x^{<T_x>}$	Machine translation

The most common activation functions used in RNN modules are described below:

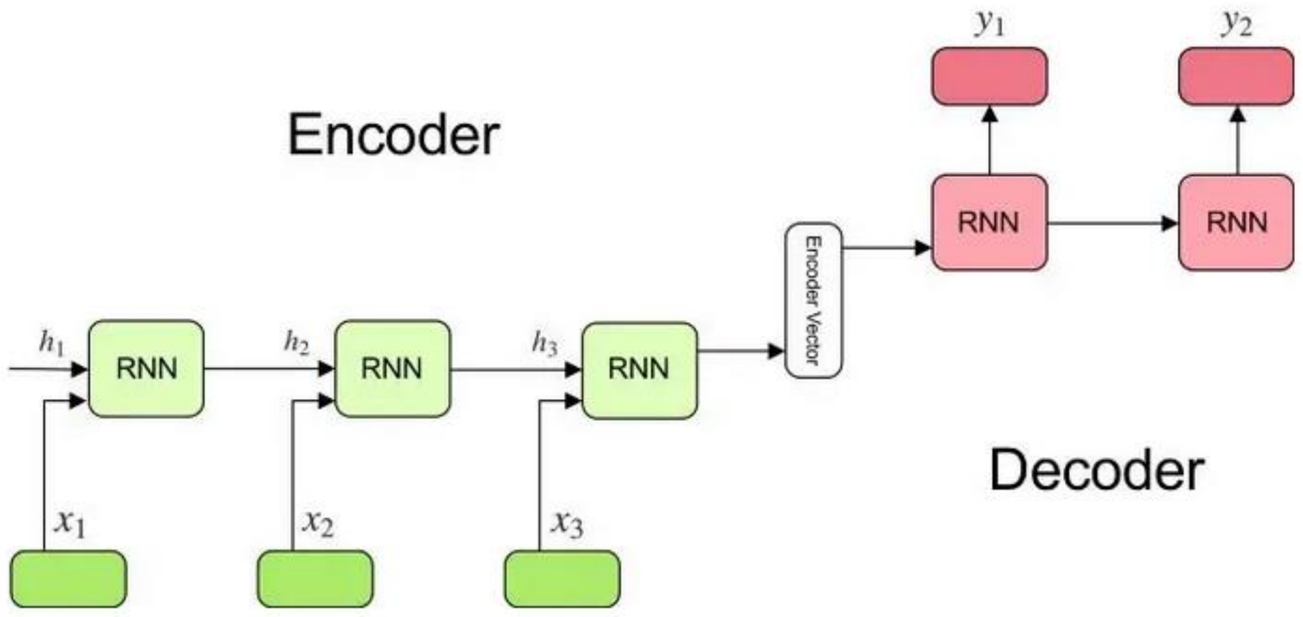
Sigmoid	**Tanh**	**RELU**
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

This model can be used as a solution to any sequence-based problem

These models are more useful for times when our inputs and outputs have different sizes and also there are several different categories.

The goal of a sequence-to-sequence model is to map a fixed-length input to a fixed-length output, where the length of the input and output can be the same or different.

For example, translating "What are you doing today?" From English to Chinese, the input is 5 words and the output is 7 symbols (今天你的做什麼？).



Encoder-decoder sequence to sequence model

Encoder

This section is a stack of layers such as LSTM, GRU, etc., which receives the input elements one by one and then collects information and moves to the next layer and stage.

The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

We just apply appropriate weights to the previously hidden state h_(t-1) and the input vector x_t.

Encoder Vector

The final hidden state is obtained from the above formula in the Encoder section

This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions

Decoder

A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.

Each Decoder section receives one of the previous hidden modes and creates its own hidden mode after processing.

Any hidden state *h_i* is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

The output *y_t* at time step *t* is computed using the formula:

$$y_t = softmax(W^S h_t)$$

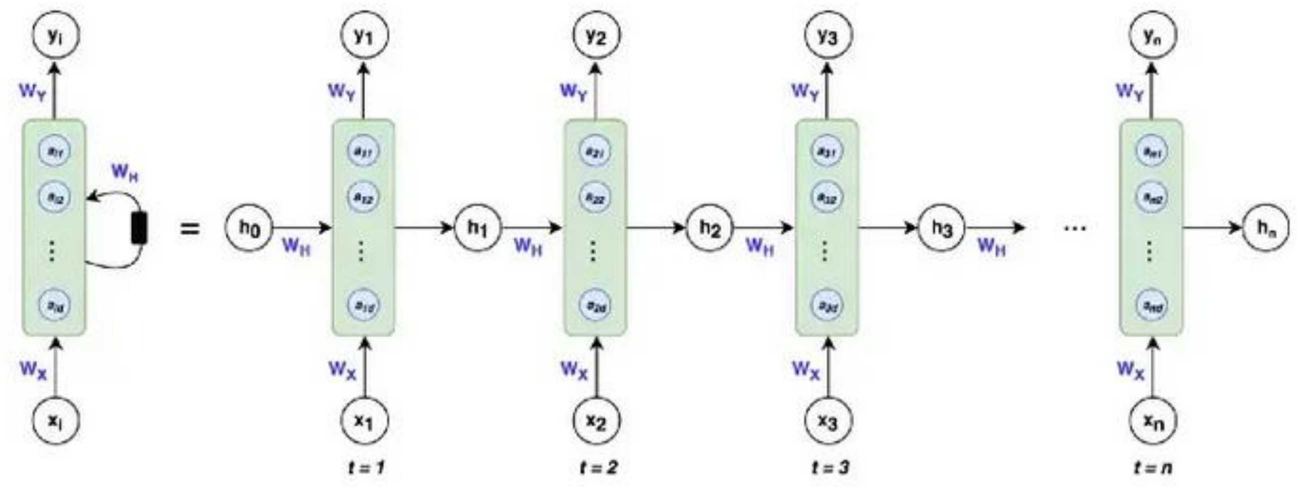
The power of this model lies in the fact that it can map sequences of different lengths to each other.

As we have seen, the fit of inputs and outputs is not necessarily the same, and this can create a wider range that can help solve more problems.

Exercise 3:

RNN :

The architecture of an RNN



Equations is:

$$a_t = \boxed{W_H h_{t-1} + W_X X_t}$$ **Hidden Nodes**

Activation Function

Output From Hidden State $\longrightarrow$ $$h_t = \underbrace{\boxed{tanh(\boxed{a_t})}}_{\textbf{Hidden State}}$$

Prediction at time t $\longrightarrow$ $$y_t = softmax(W_Y h_t)$$

For simplest calculations we consider that :

$W_h = 1$, $W_x = 1$, $W_y = 1$

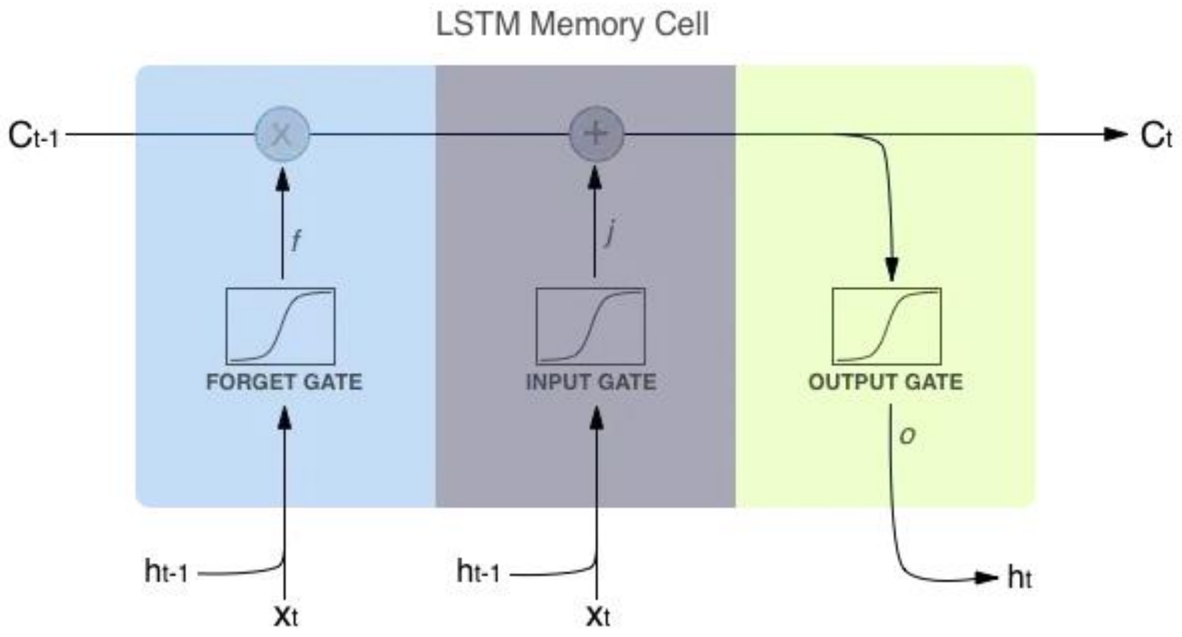
Replace tanh with linear

$A = h_{t-1} + x_t$

$H_t = linear(A) \Rightarrow h_t = h_{t-1} + x_t$

$Y_t = h_t \Rightarrow y_t = h_{t-1} + x_t$

LSTM :



Equations :

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

$$h_t = o_t * tanh(c^t)$$

For convenience, consider the following assumptions:

$I_t = 1$, $f_t = 0$, $o_t = 1$

$W_c = 1$, $b_c = 0$

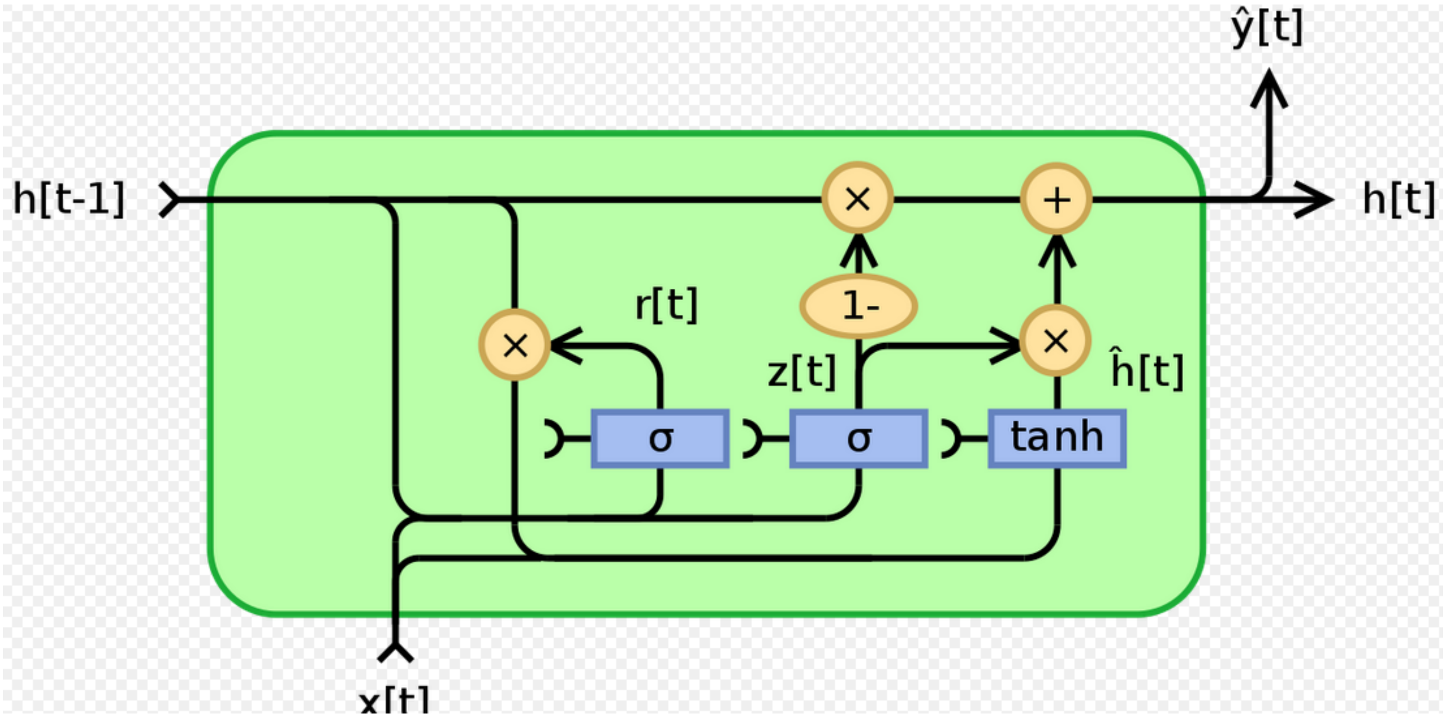
Replace tanh with linear

$C^{\wedge}_t = h_{t-1} + x_t$

$C_t = C^{\wedge}_t$

$H_t = \text{linear}(C_t) \Rightarrow h_t = h_{t-1} + x_t$

GRU :



Equations :

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

For convenience, consider the following assumptions:

Replace tanh with linear

$Z_t = 1$

$R_t = 1$

$W = 1$

$H^{\wedge}_t = \text{linear}(h_{t-1} + x_t) \Rightarrow h_{t-1} + x_t$

$H_t = h^{\wedge}_t \Rightarrow h_{t-1} + x_t$