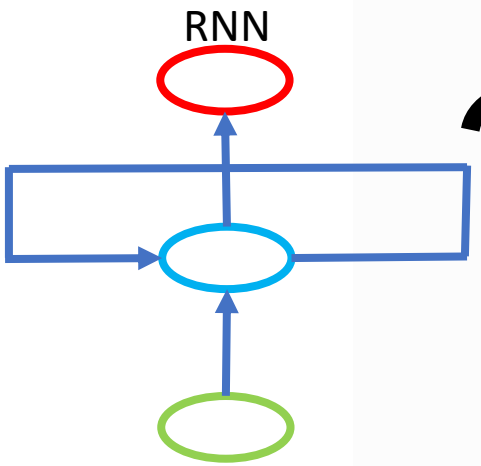
# Time Series (RNN, LSTM, GRU) Deep Learning Models

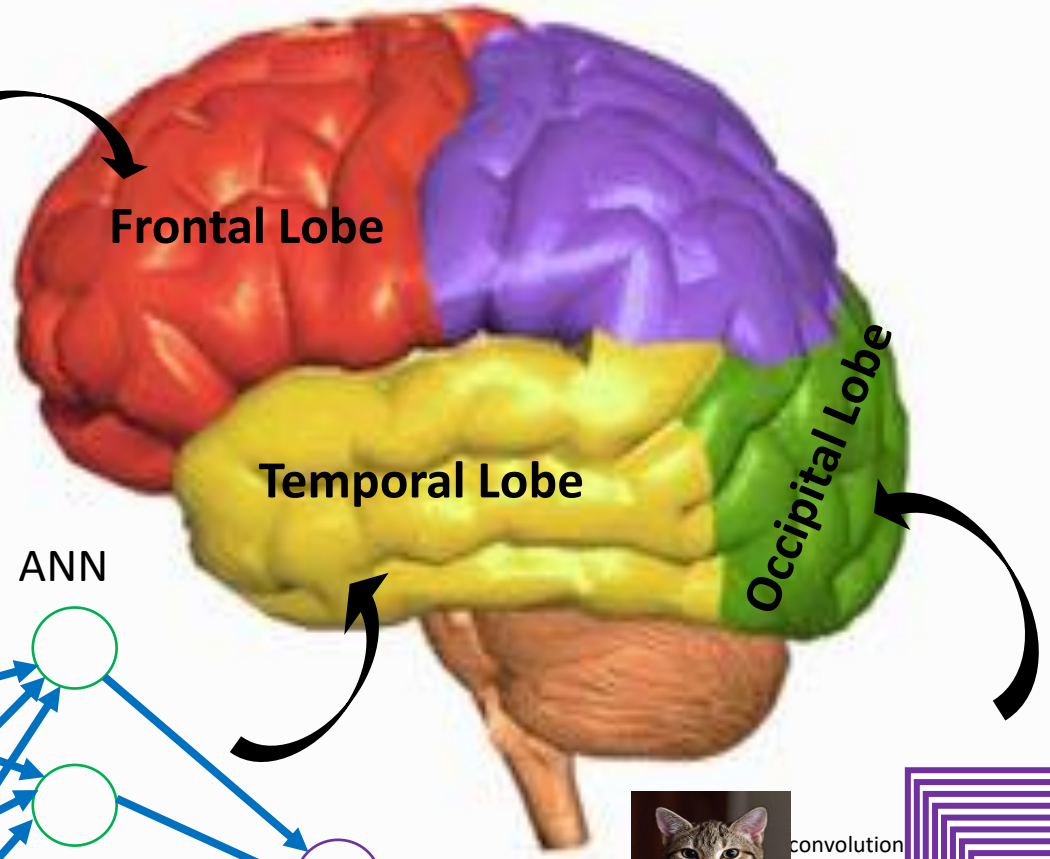
Instructor

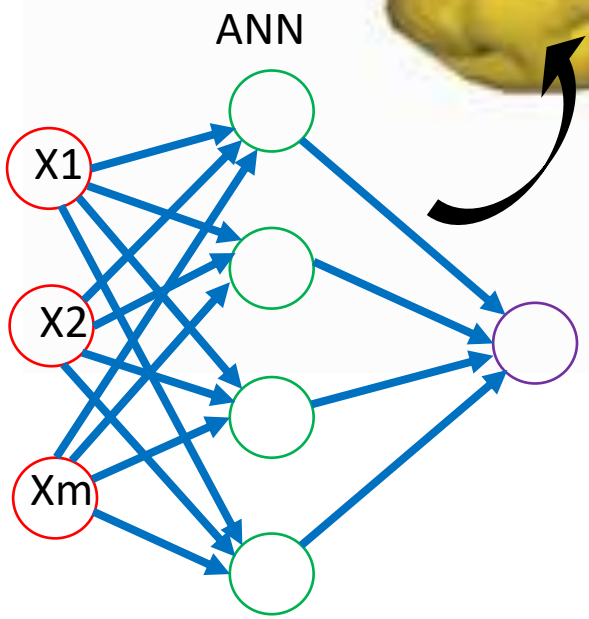
Abdul Qayyum, PhD

Date:20-10-2020

# Brain Analogy with Deep Neural Networks

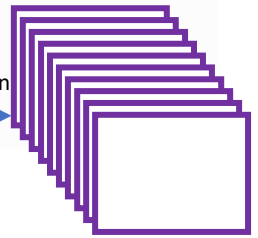


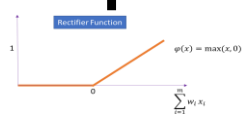




Input Image

convolution



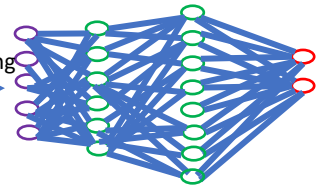
Convolutional Layer




Pooling

Pooling Layer

Flattening



# RNN



## Applications:
## Working with Sequential Data

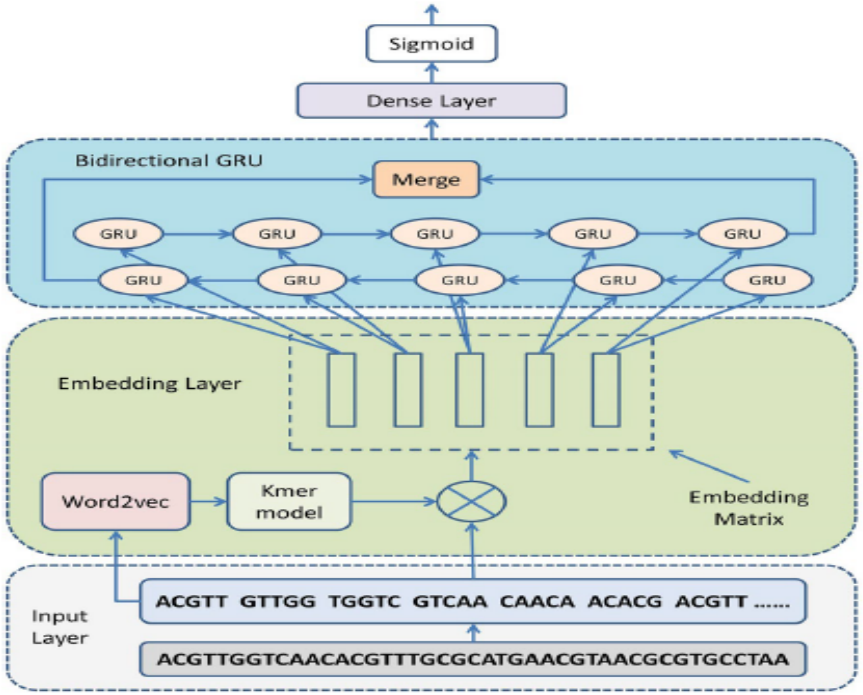
- Text classification
- Speech recognition (acoustic modeling)
- language translation
- ...

Stock market predictions

**Fig 8. Displays the actual data and the predicted data from the four models for each stock index in Year 1 from 2010.10.01 to 2011.09.30.**

https://doi.org/10.1371/journal.pone.0180944.g008

Bao, Wei, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." *PloS one* 12, no. 7 (2017): e0180944.




Shen, Zhen, Wenzheng Bao, and De-Shuang Huang. "Recurrent Neural Network for Predicting Transcription Factor Binding Sites." *Scientific reports* 8, no. 1 (2018): 15270.

DNA or (amino acid/protein) sequence modeling

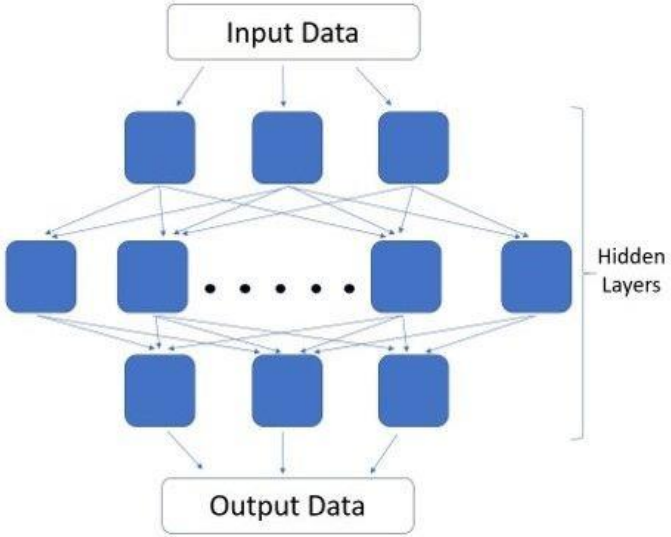
# RNN

https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

Recurrent Neural Networks(RNNs) have been the answer to most problems dealing with **sequential data** and **Natural Language Processing(NLP)** problems for many years

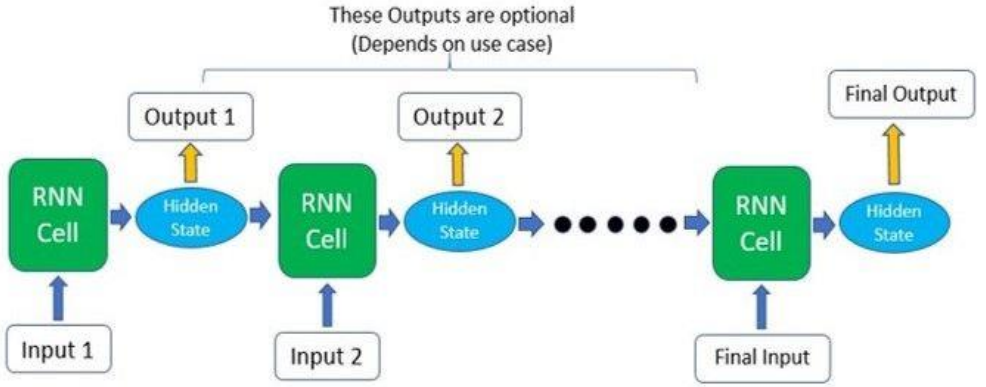
## Comparison Between:

**Traditional Feed-Forward Network**

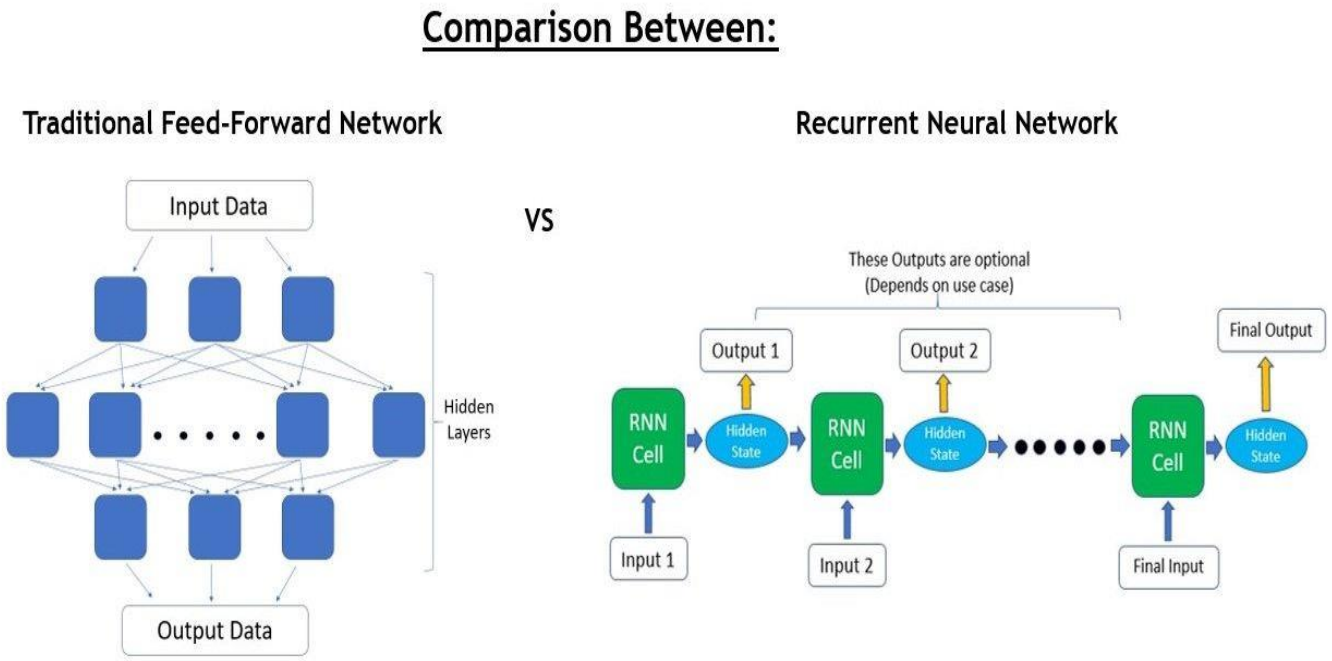



VS

**Recurrent Neural Network**



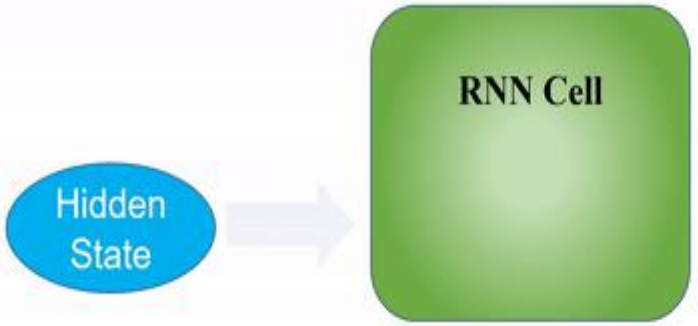
# RNN

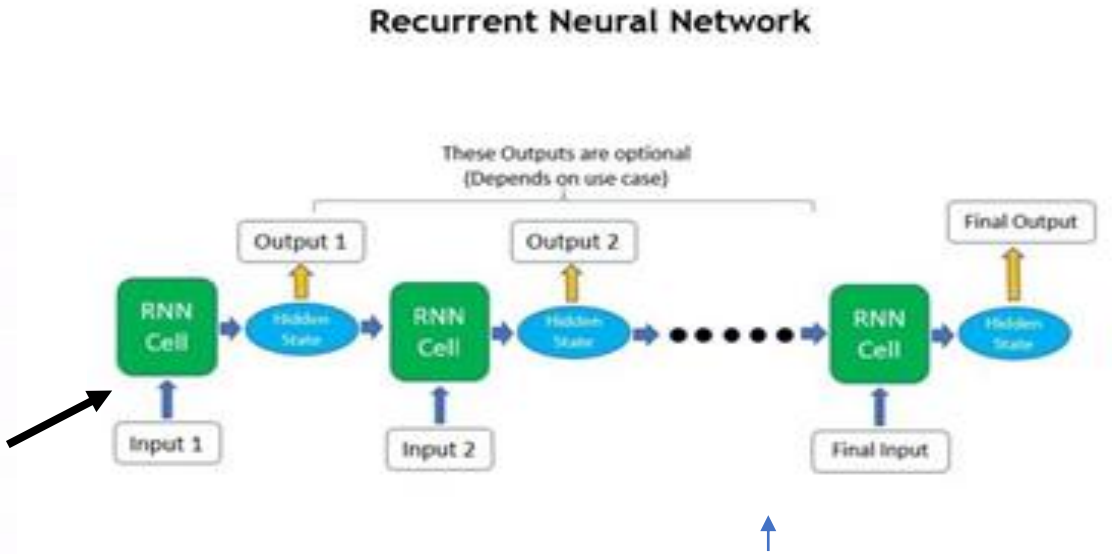



Traditional feed-forward neural networks take in a fixed amount of input data all at the same time and produce a fixed amount of output each time.

On the other hand, RNNs do not consume all the input data at once. Instead, they take them in one at a time and in a sequence. At each step, the RNN does a series of calculations before producing an output. The output, known as the hidden state, is then combined with the next input in the sequence to produce another output. This process continues until the model is programmed to finish or the input sequence ends.

# RNN

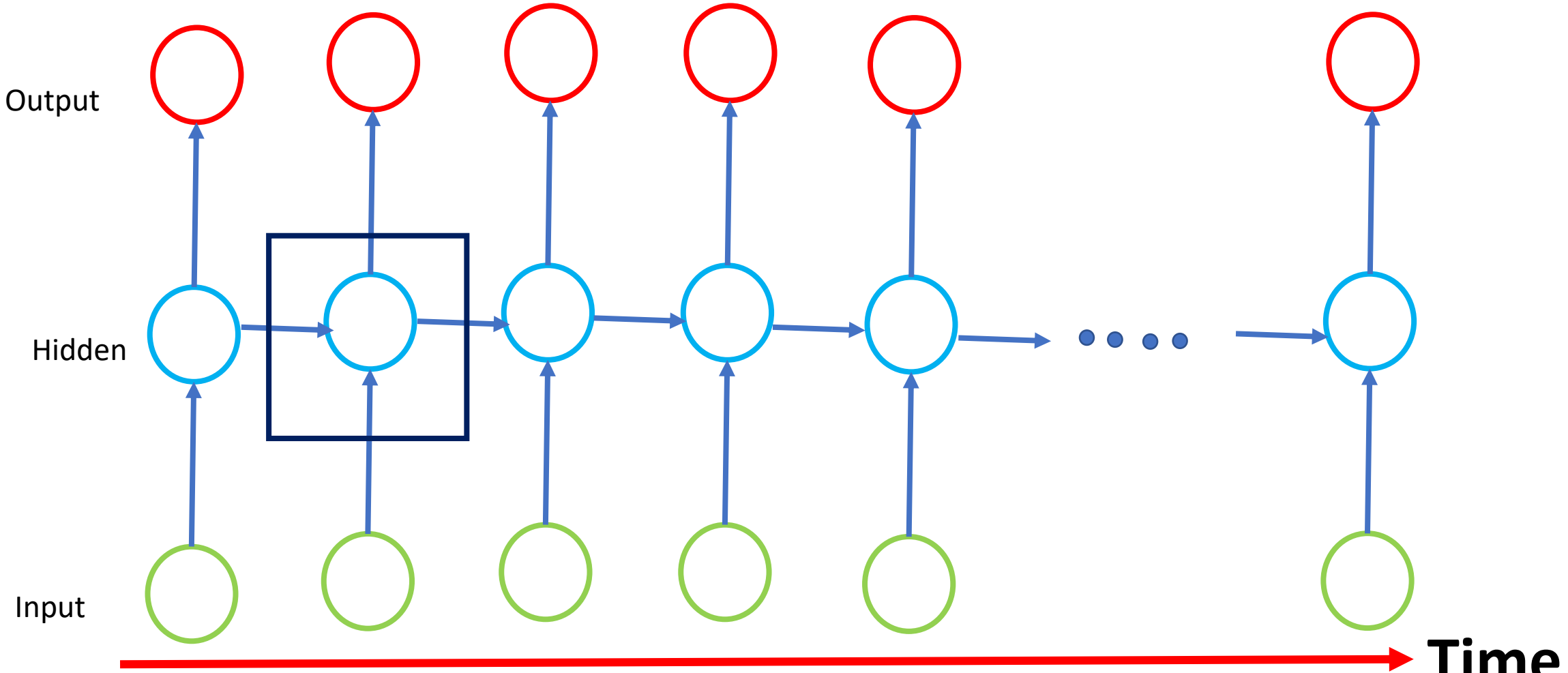


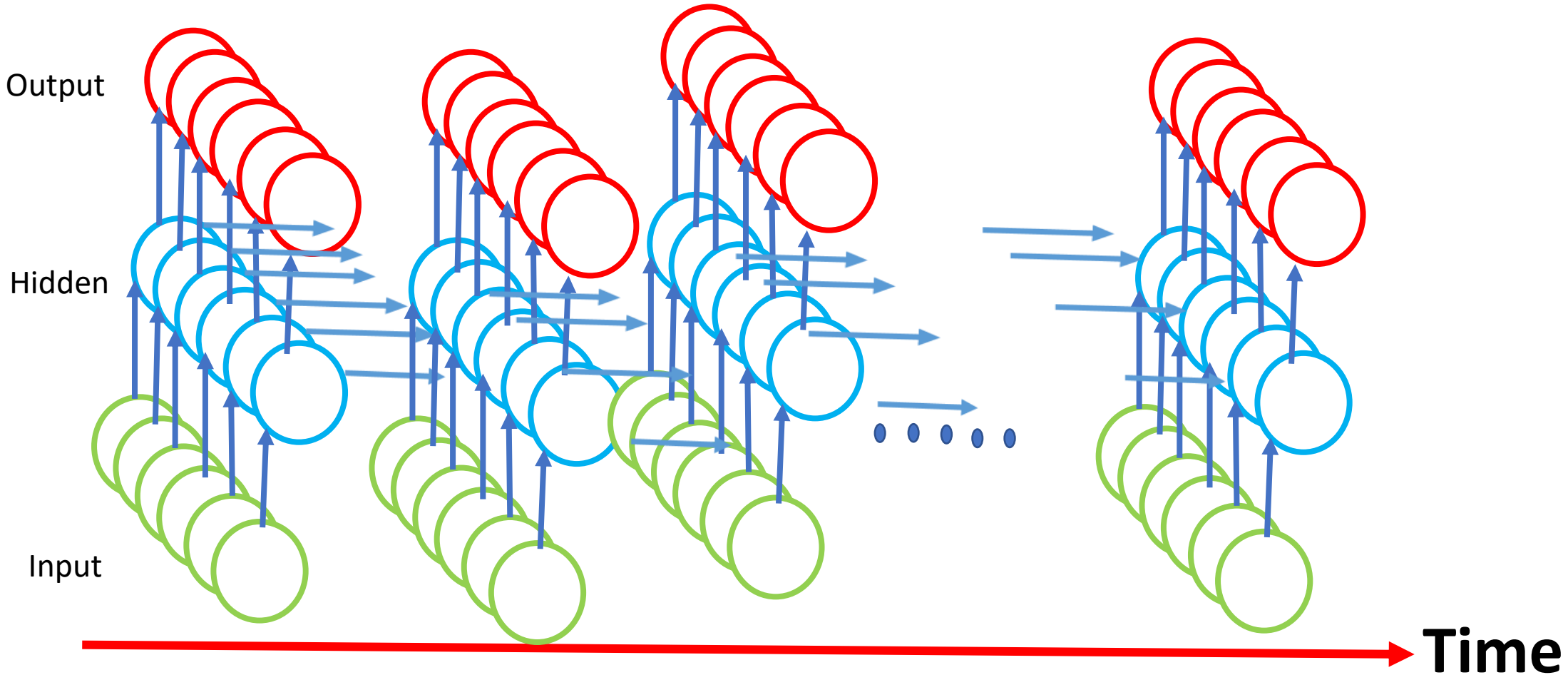

As we can see, the calculations at each time step consider the context of the previous time steps in the form of the hidden state. Being able to use this contextual information from previous inputs is the key essence to RNNs' success in sequential problems.

https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

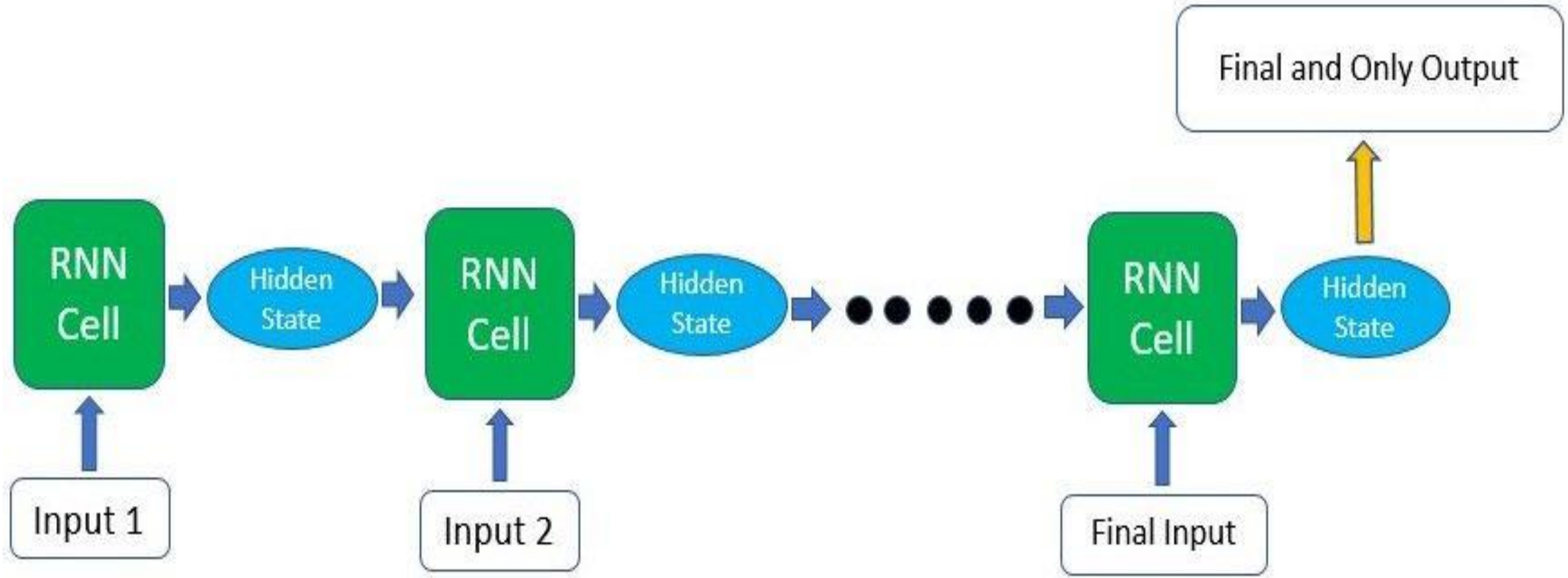
# RNN



# RNN

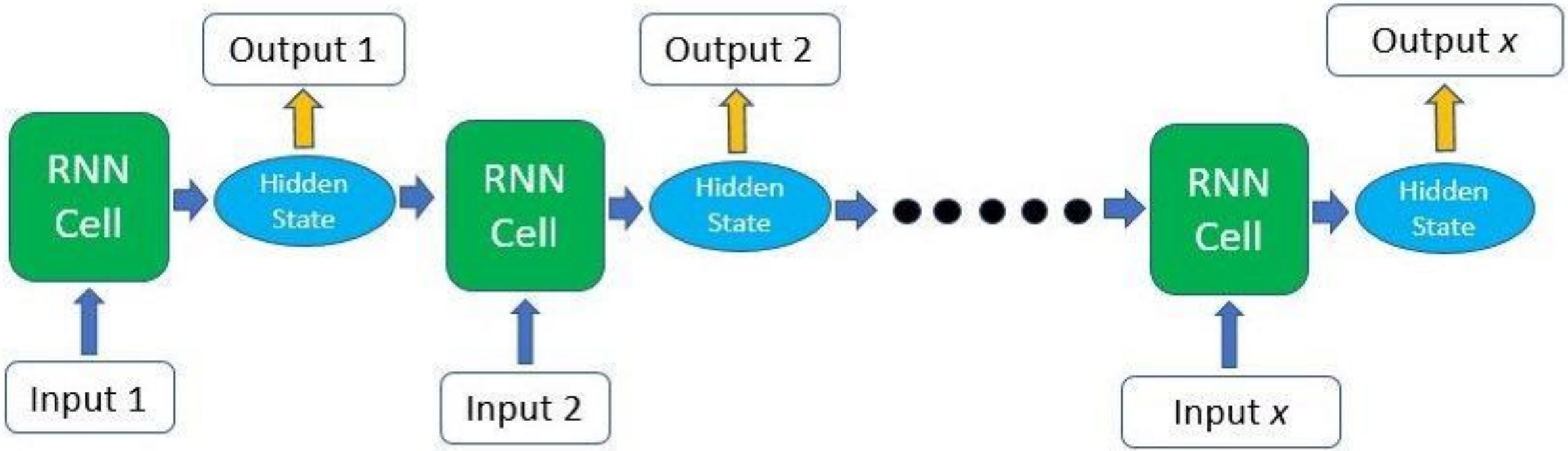


# RNN



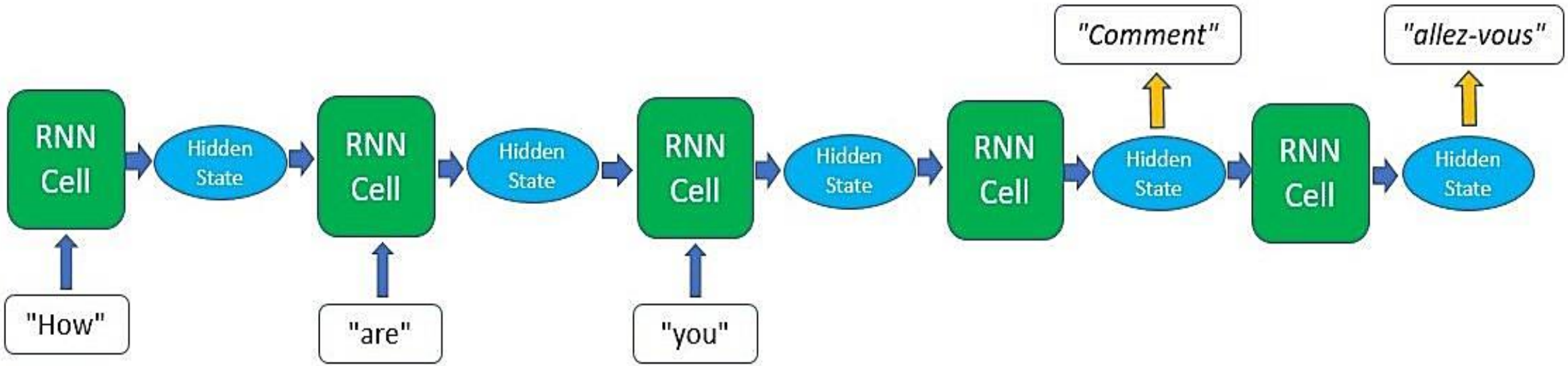

https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

# RNN




https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

# RNN




Structure of a sequence-to-sequence model

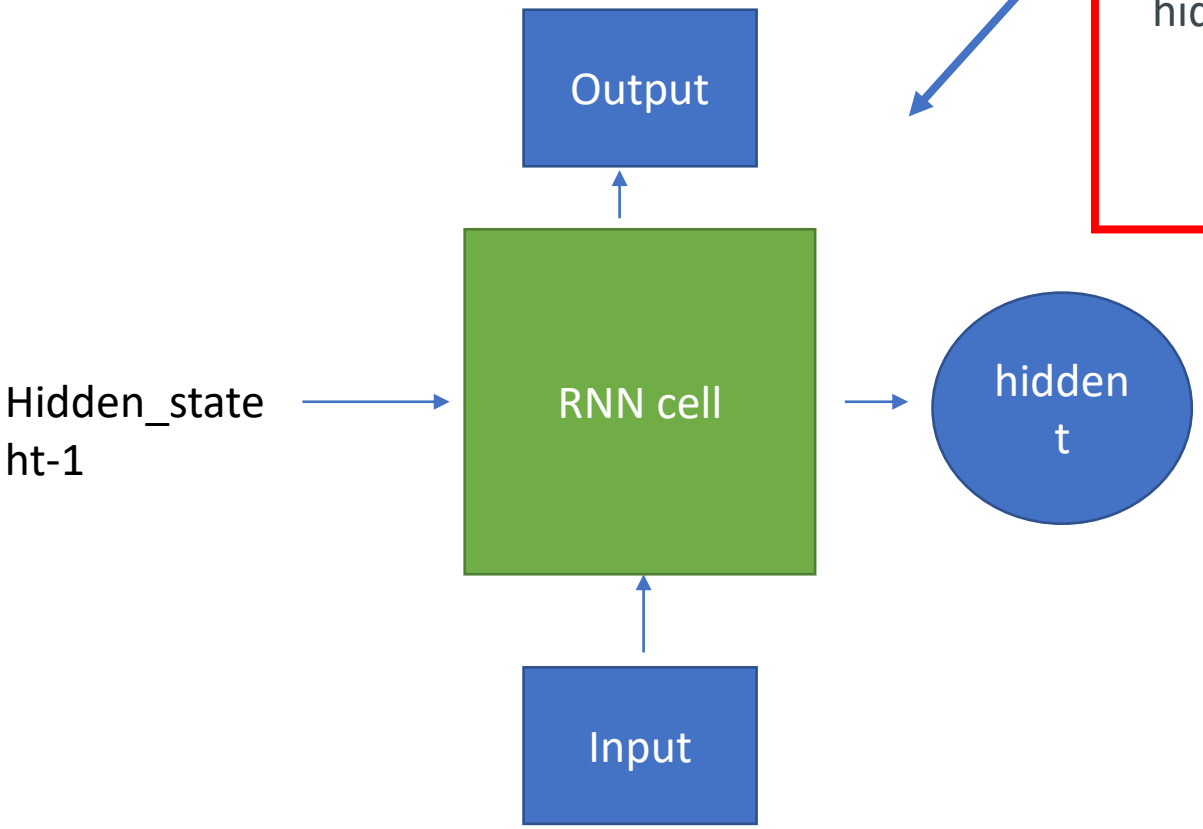
https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

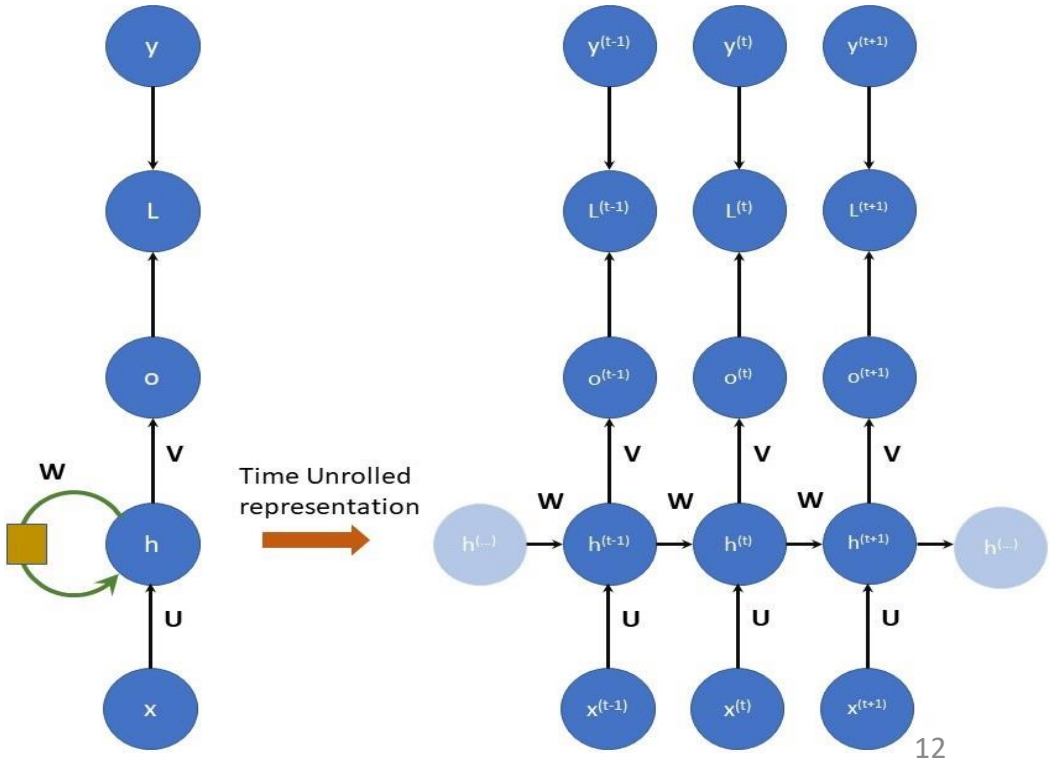
# RNN

$$hidden_t = F(hidden_{t-1}, input_t)$$

$$hidden_t = tanh(weight_{hidden} * hidden_{t-1} + weight_{input} * input_t)$$

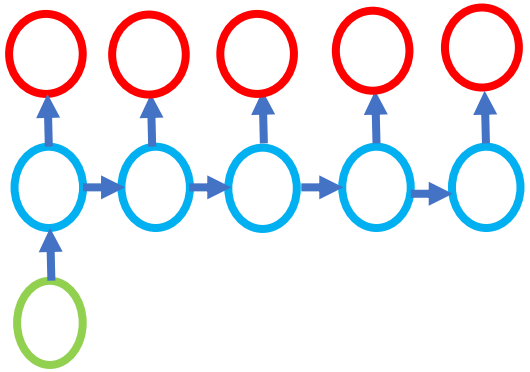
$$output_t = weight_{output} * hidden_t$$



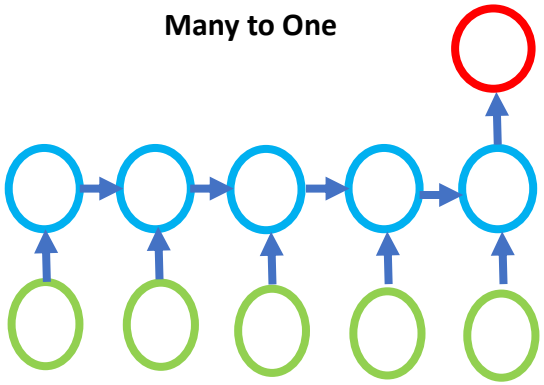
# RNN Models


One to Many



Time

Many to One

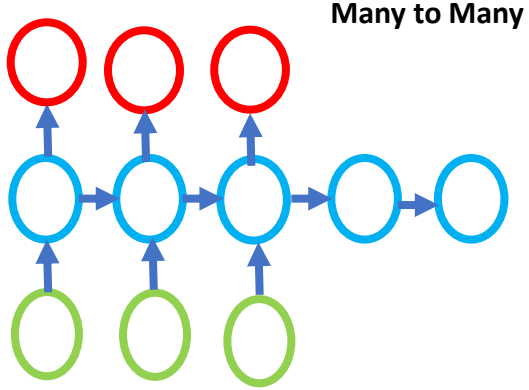


"Thanks for a great party at the weekend, we really enjoyed it!"

sentiment: positive
score: 86%

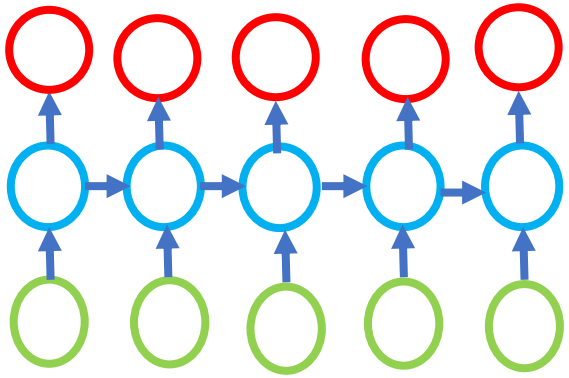
Time

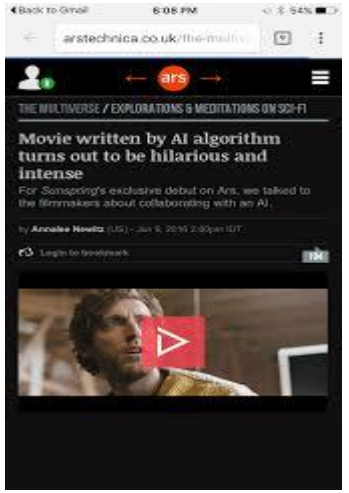

Many to Many



Time

Many to Many






Time

# RNN

RNNs process inputs in a sequential manner, where the context from the previous input is considered when computing the output of the current step.
**This allows the neural network to carry information over different time steps rather than keeping all the inputs independent of each other.**

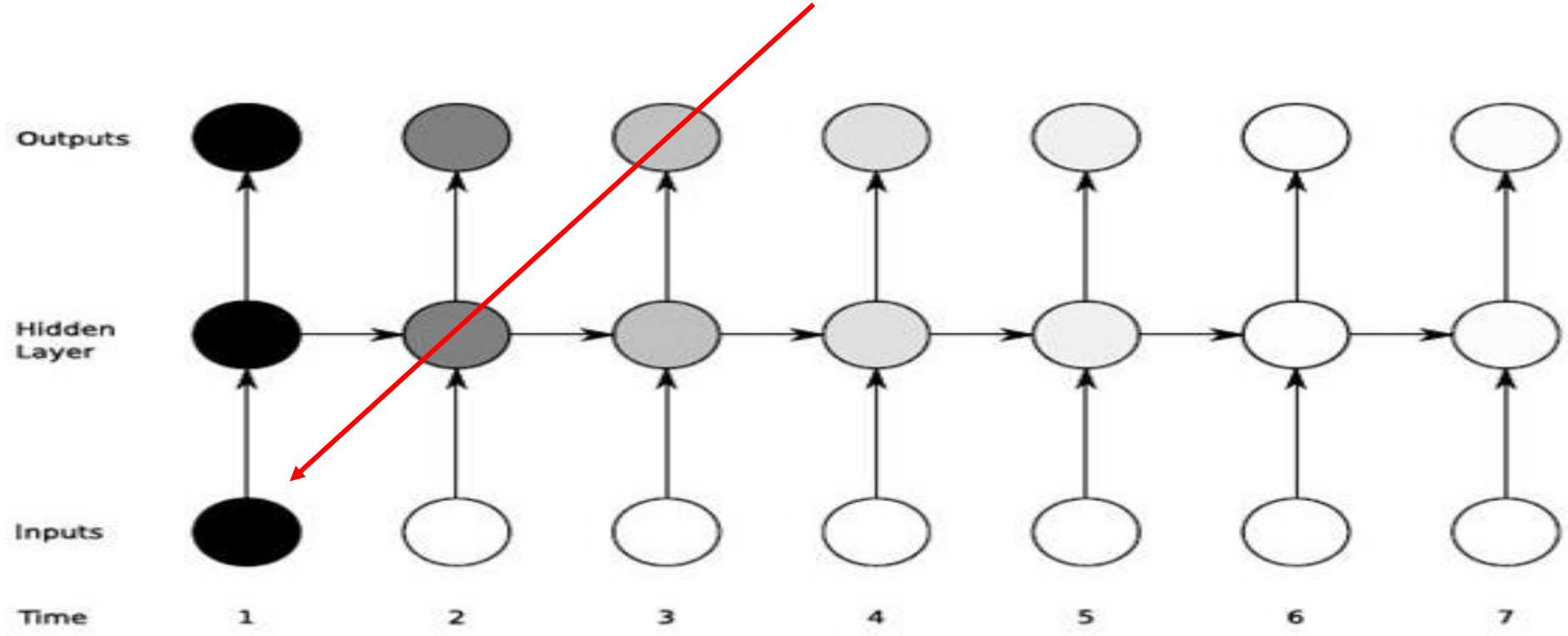
However, a significant shortcoming that plagues the typical RNN is the problem of **vanishing/exploding gradients**. This problem arises when back-propagating through the RNN during training, especially for networks with deeper layers. The gradients have to go through continuous matrix multiplications during the back-propagation process due to the chain rule, causing the gradient to either **shrink exponentially (vanish) or blow up exponentially (explode).** Having a gradient that is too small prevents the weights from updating and learning, whereas extremely large gradients cause the model to be unstable.

Due to these issues, RNNs are unable to work with **longer sequences and hold on to long-term dependencies, making them suffer from "short-term memory"**

https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

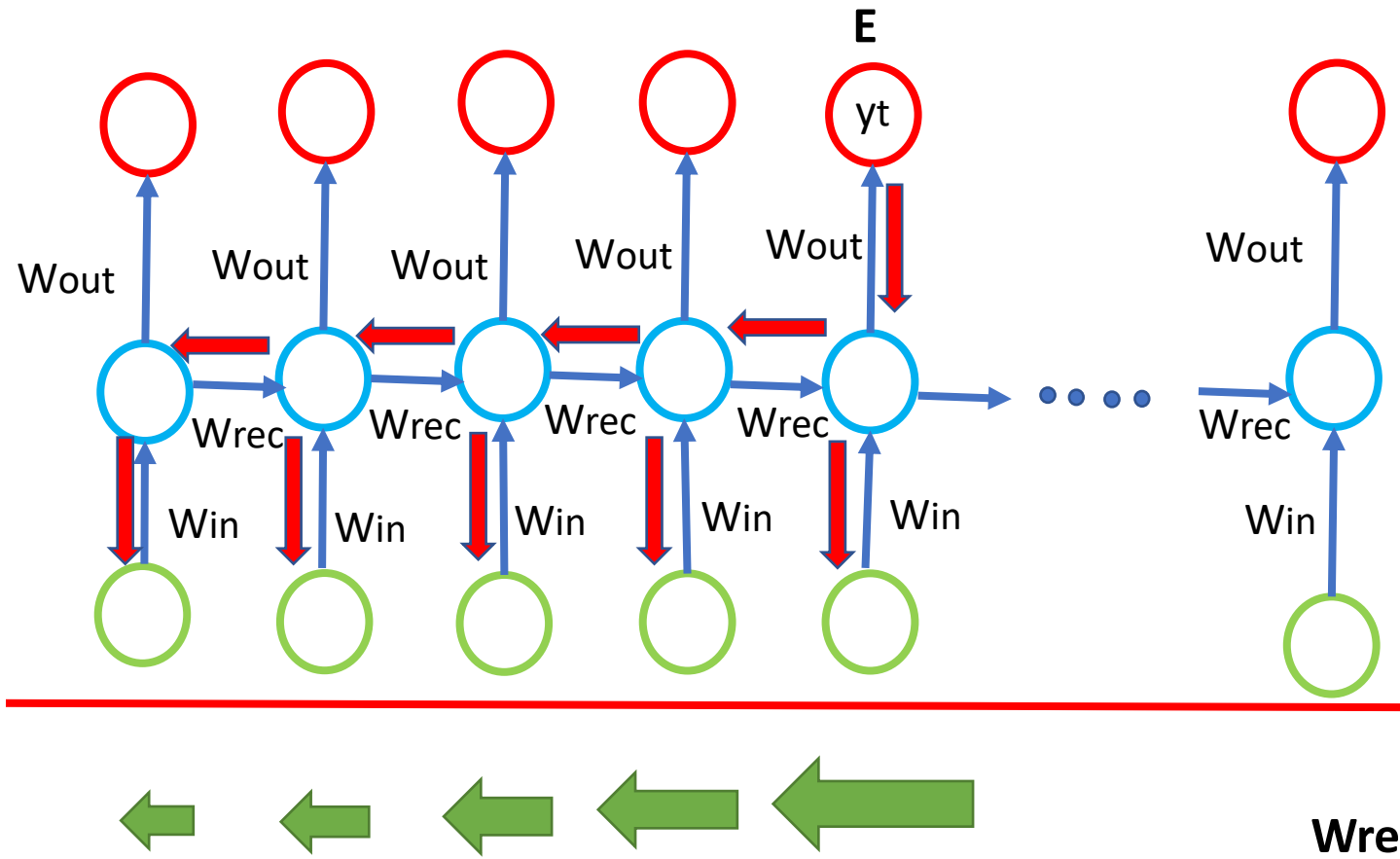
# RNN

**Vanilla RNNs suffer from insensitivity to input for long sequences (sequence length approximately greater than 10 time steps). As sown the figure sensitivity decays as we move from timestep =1 to timestep=7 very fast. The network forgets the first input.**




The figure from Alex Grave's thesis shows the sensitivity of the hidden nodes to the gradient.

# The Vanishing Gradient Problem




**The Vanishing Gradient Problem Solution:**

1. **Exploding**
   - **Truncated Backpropagation**
   - **Penalties**
   - **Grading Clipping**

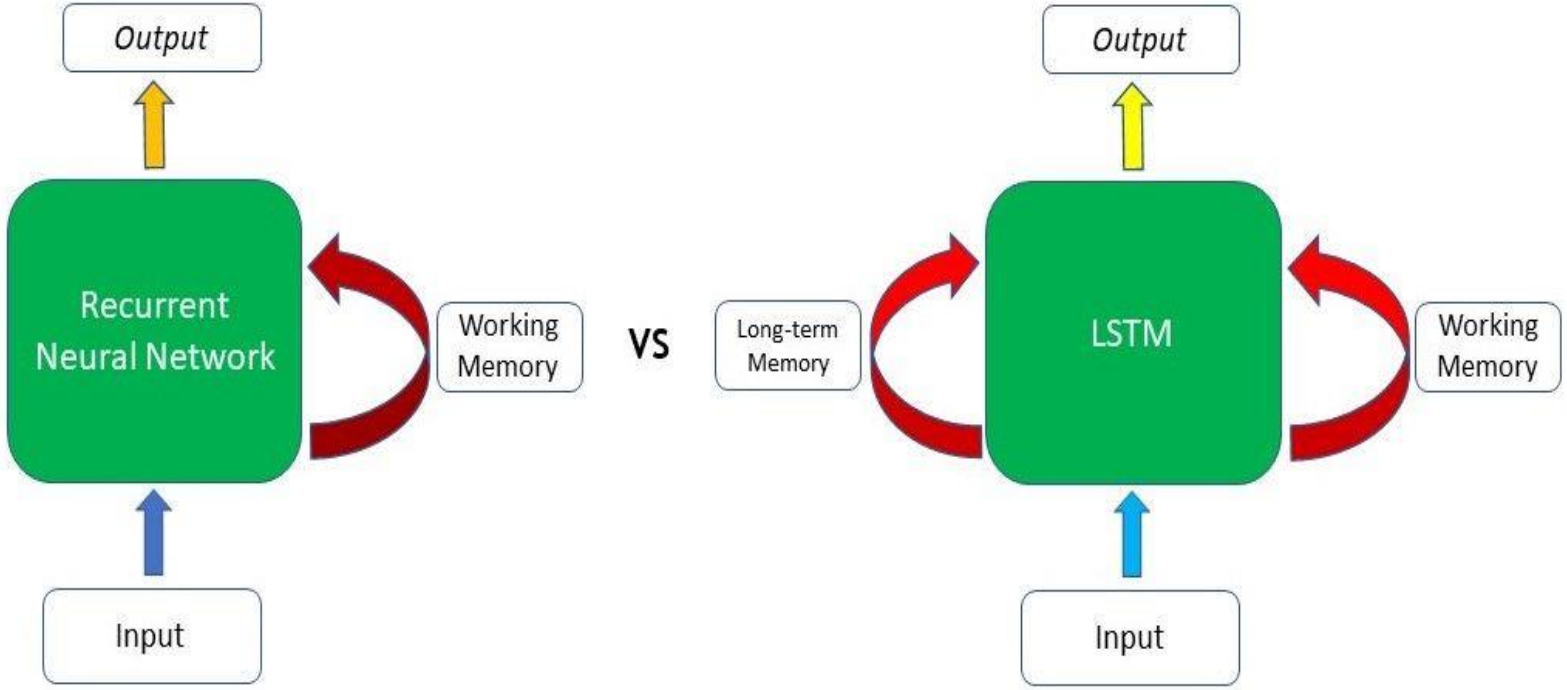
2. **Vanishing Gradient**
   - **Weight initialization**
   - **Echo State Networks**
   - **Long Short-Term Memory Networks (LSTM)**

**Wrec ~ Small ⇒ Vanishing**

**Wrec ~ Large ⇒ Exploding**

# LSTM(Long Short-Term Memory) Networks

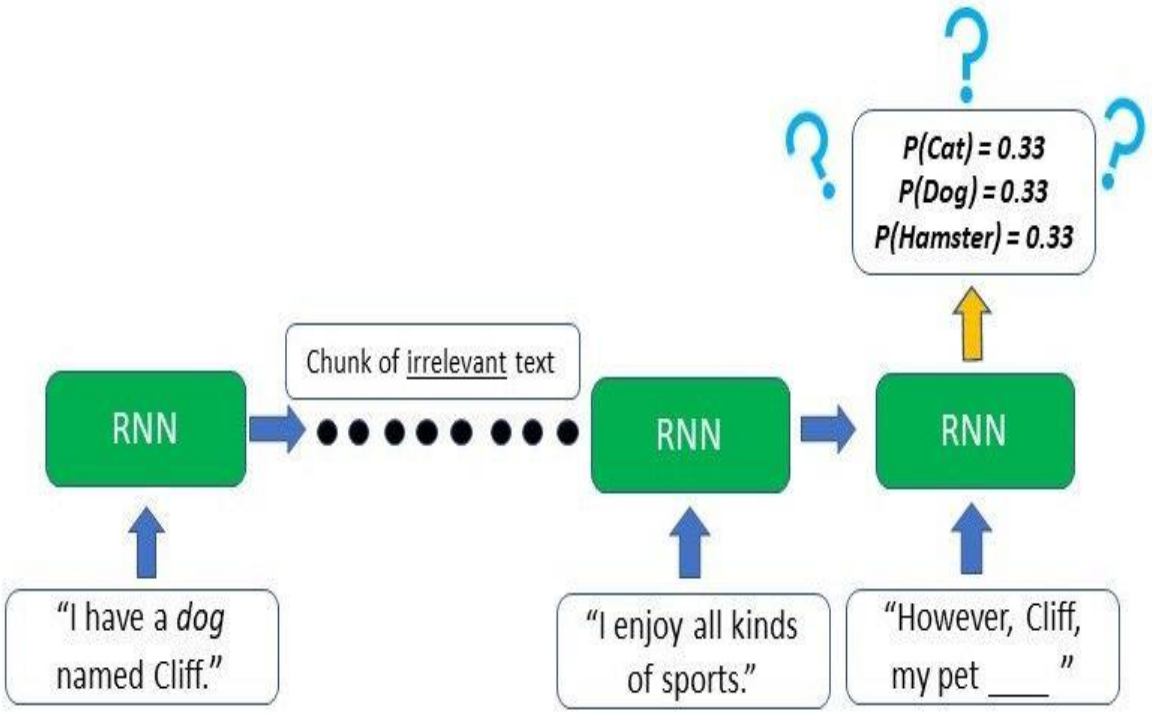
LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs. LSTMs proposed in 1997 remain the most popular solution for overcoming this short coming of the RNNs.**



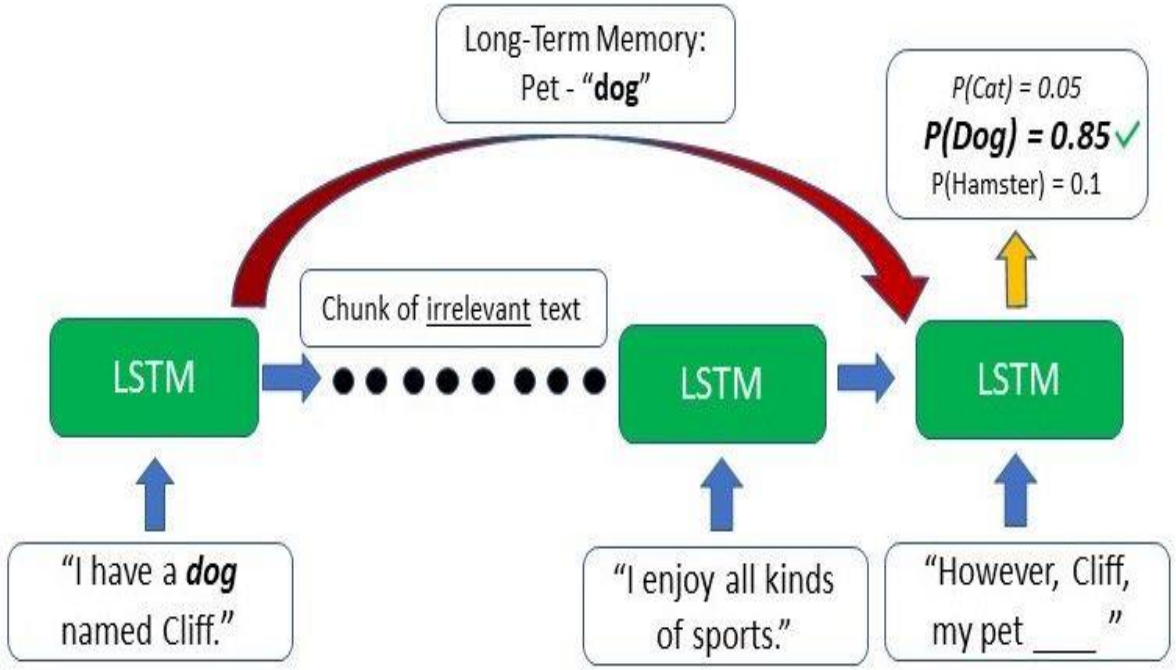

**Vanilla RNN vs LSTM**

# LSTM

LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**



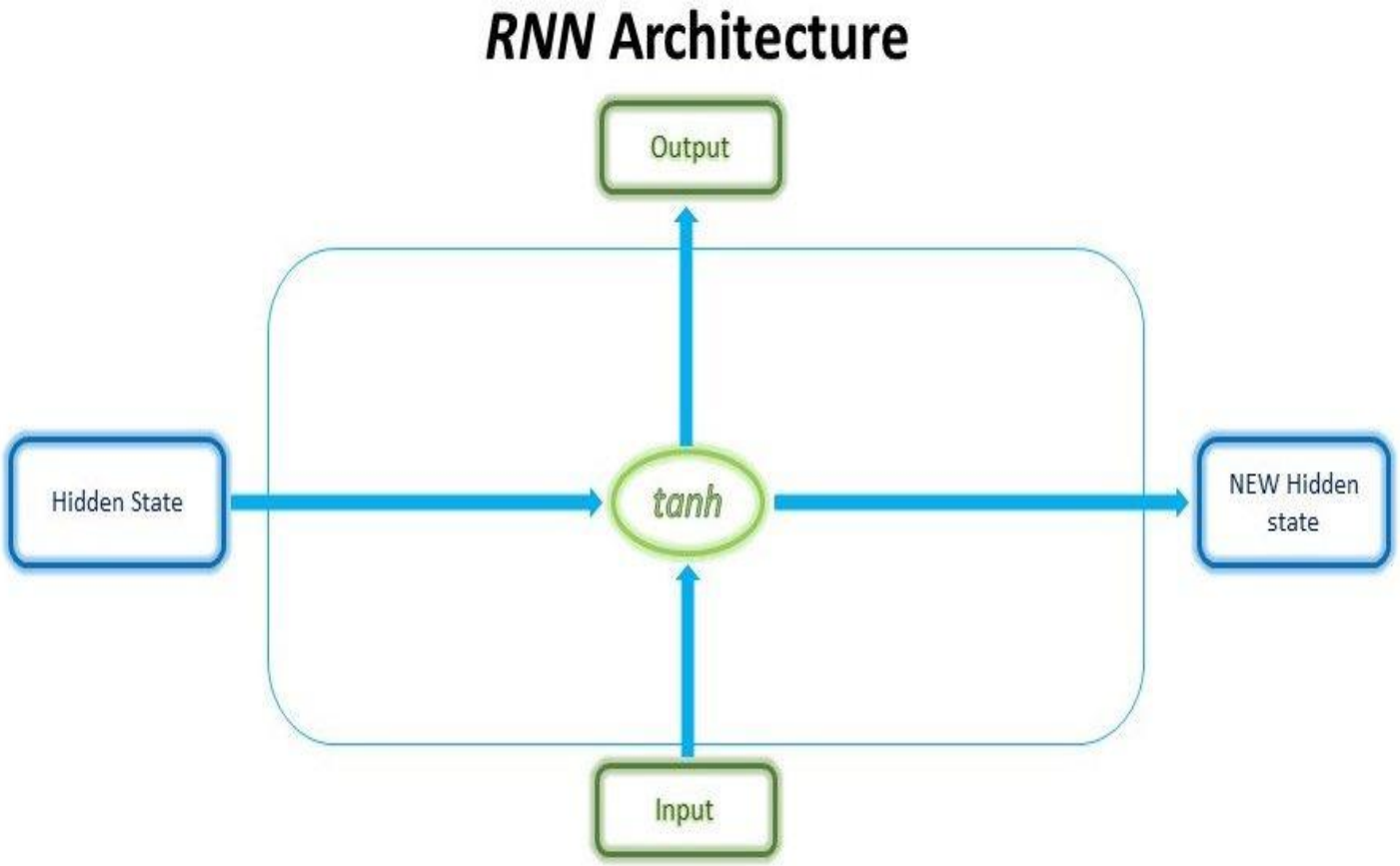

The RNN has no clue as to what animal the pet might be as the relevant information from the start of the text has already been **lost**




the LSTM can retain the earlier information that the author has a pet dog

# LSTM

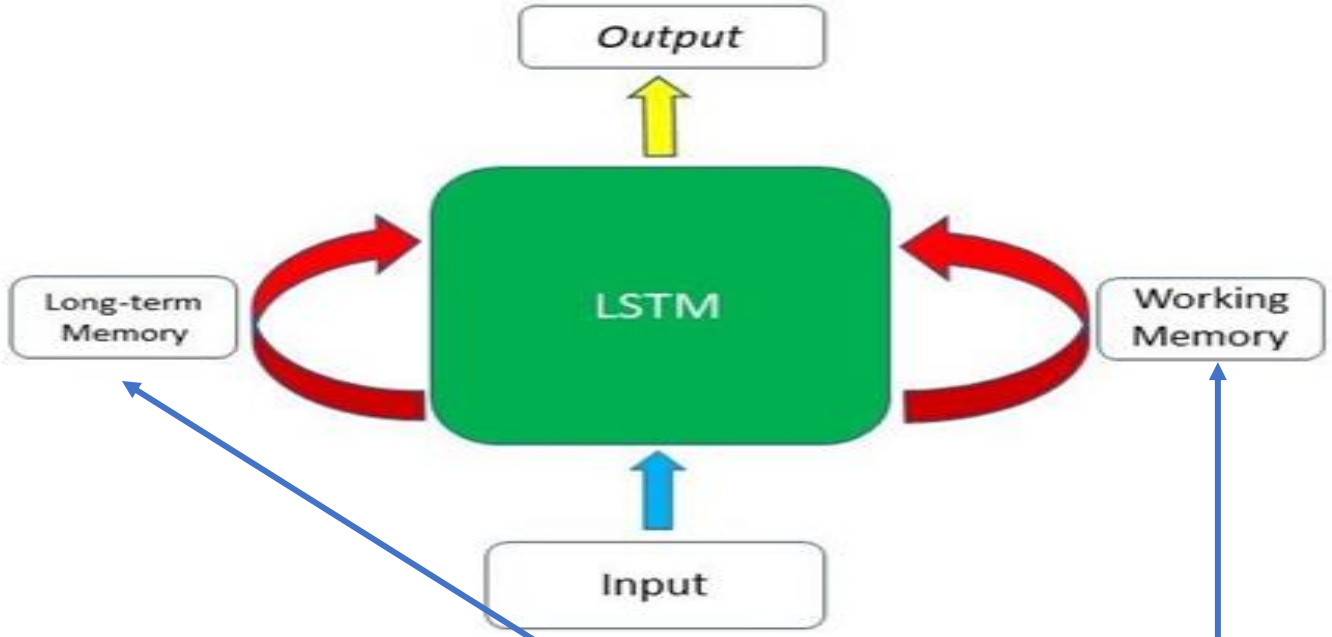
LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**




**Inner workings of an RNN cell**

# LSTM

LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**

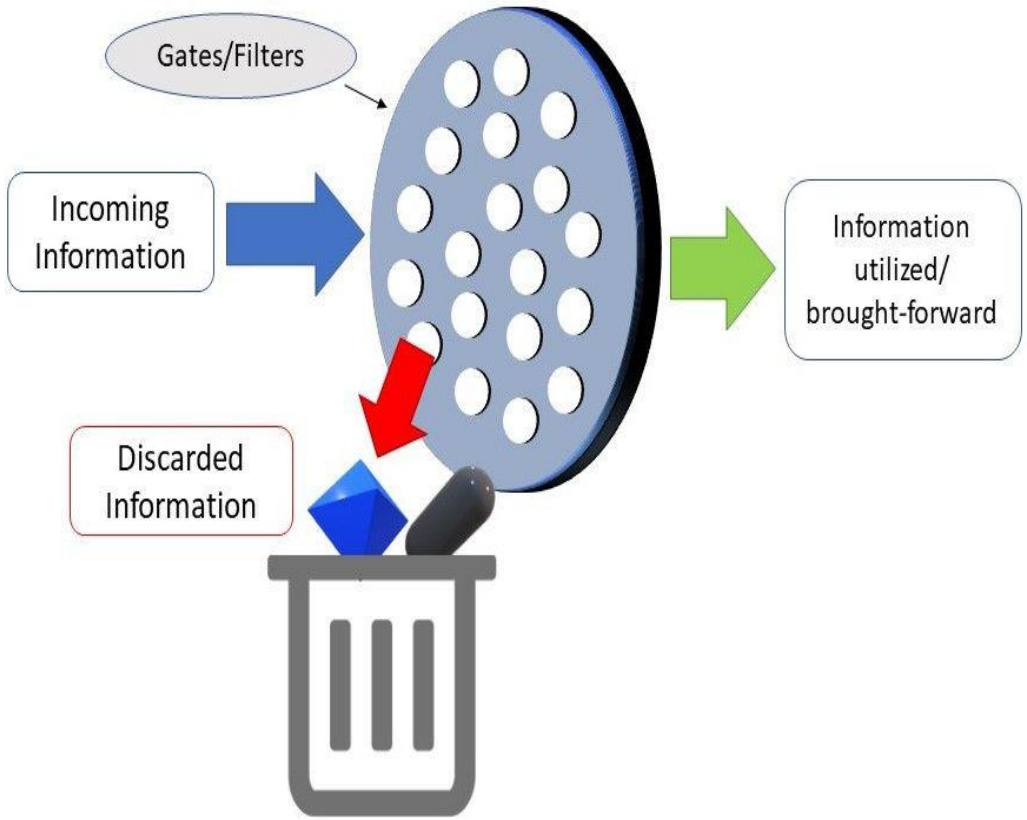



At each time step, the LSTM cell takes in 3 different pieces of information -- the *current input* data, the ***short-term memory*** from the previous cell (**similar to hidden states in RNNs**) and lastly the ***long-term memory*.**

The **short-term memory** is commonly referred to as **the hidden state**,
and the **long-term memory** is usually known as the **cell state**.

# LSTM

LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**



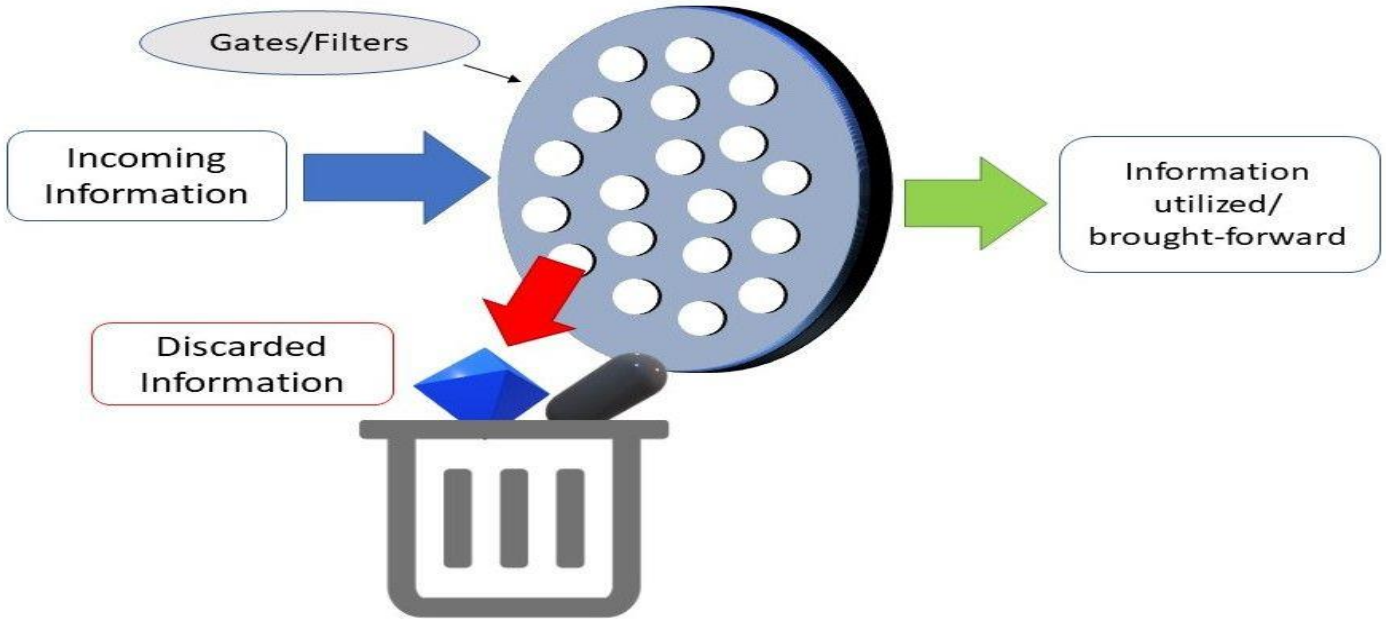

The **cell** then uses **gates** to regulate the information to be **kept or discarded** at each time step before passing on the **long-term and short-term information** to the **next cell**.

Ideally, the role of these gates is supposed to **selectively remove any irrelevant information**

how the **gates** only hold on to the useful information. Of course, these gates need to be **trained** to **accurately filter what is useful and what is not.**

# LSTM

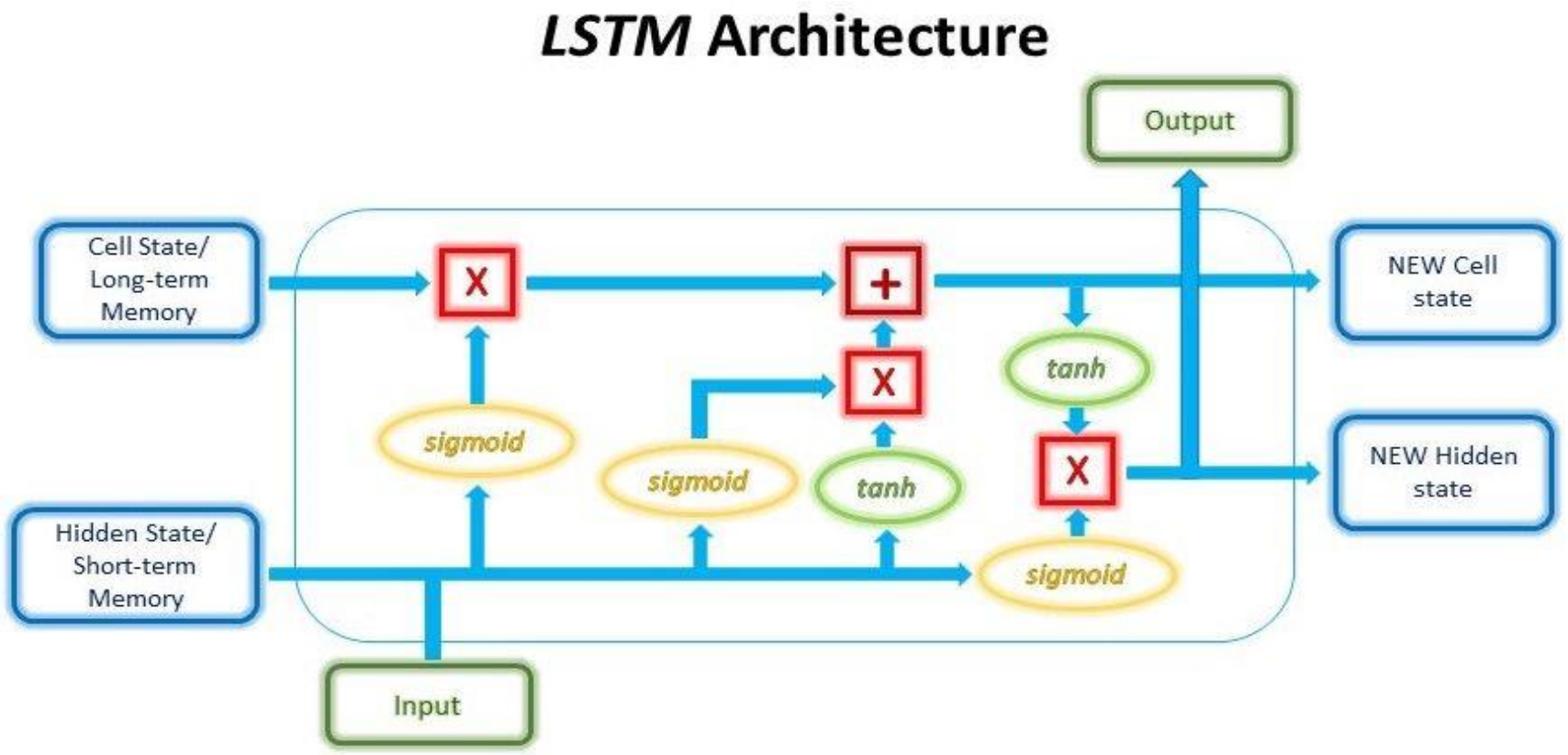
LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**




These gates are called the **Input Gate**, the **Forget Gate**, and the **Output Gate**.

# LSTM

LSTMs are a kind of RNN and function similarly to traditional RNNs, its Gating mechanism is what sets it apart. **This feature addresses the "short-term memory" problem of RNNs.**

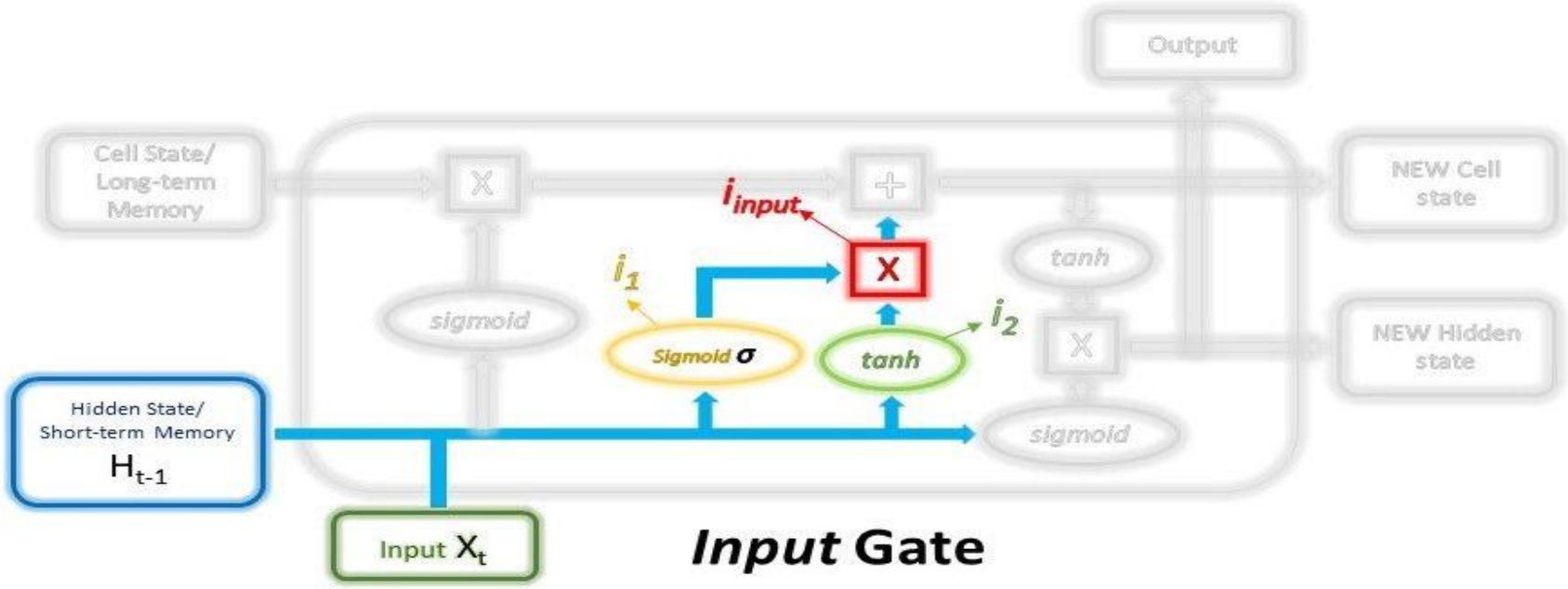



**Inner Workings of the LSTM cell**

# LSTM

**Input Gate**

The input gate decides what new information will be **stored in the long-term memory**. It only works with the information from the **current input and the short-term memory from the previous time step**. Therefore, it has to filter out the information from these variables that are not useful.

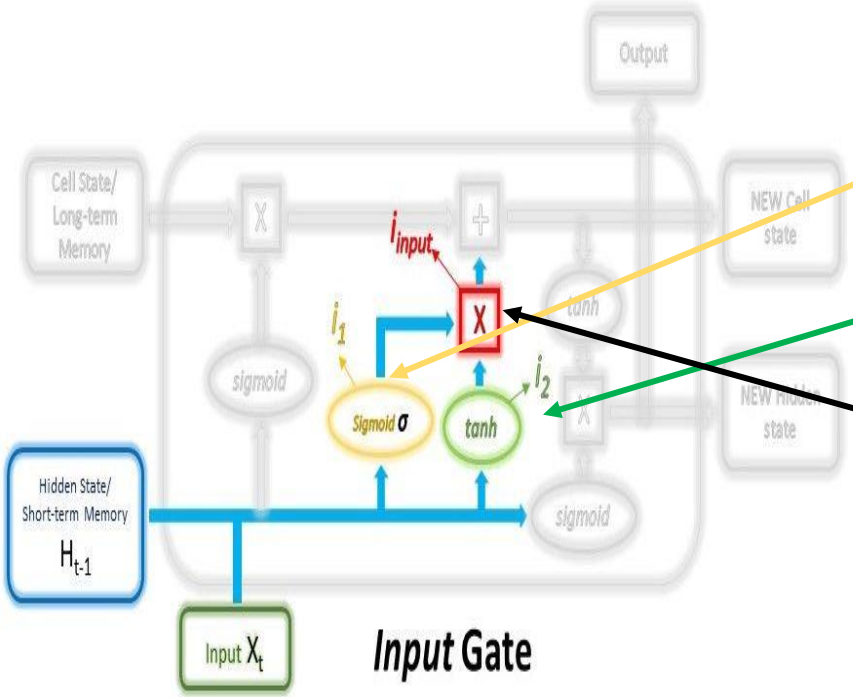



https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

# LSTM

**Input Gate**

The input gate decides what new information will be **stored in the long-term memory**. It only works with the information from the **current input and the short-term memory from the previous time step**. Therefore, it has to filter out the information from these variables that are not useful.




$i1=\sigma(Wi1\cdot(Ht-1,xt)+biasi1)$

$i_2=\tanh(W_{i_2}\cdot(H_{t-1},x_t)+bias_{i_2})$

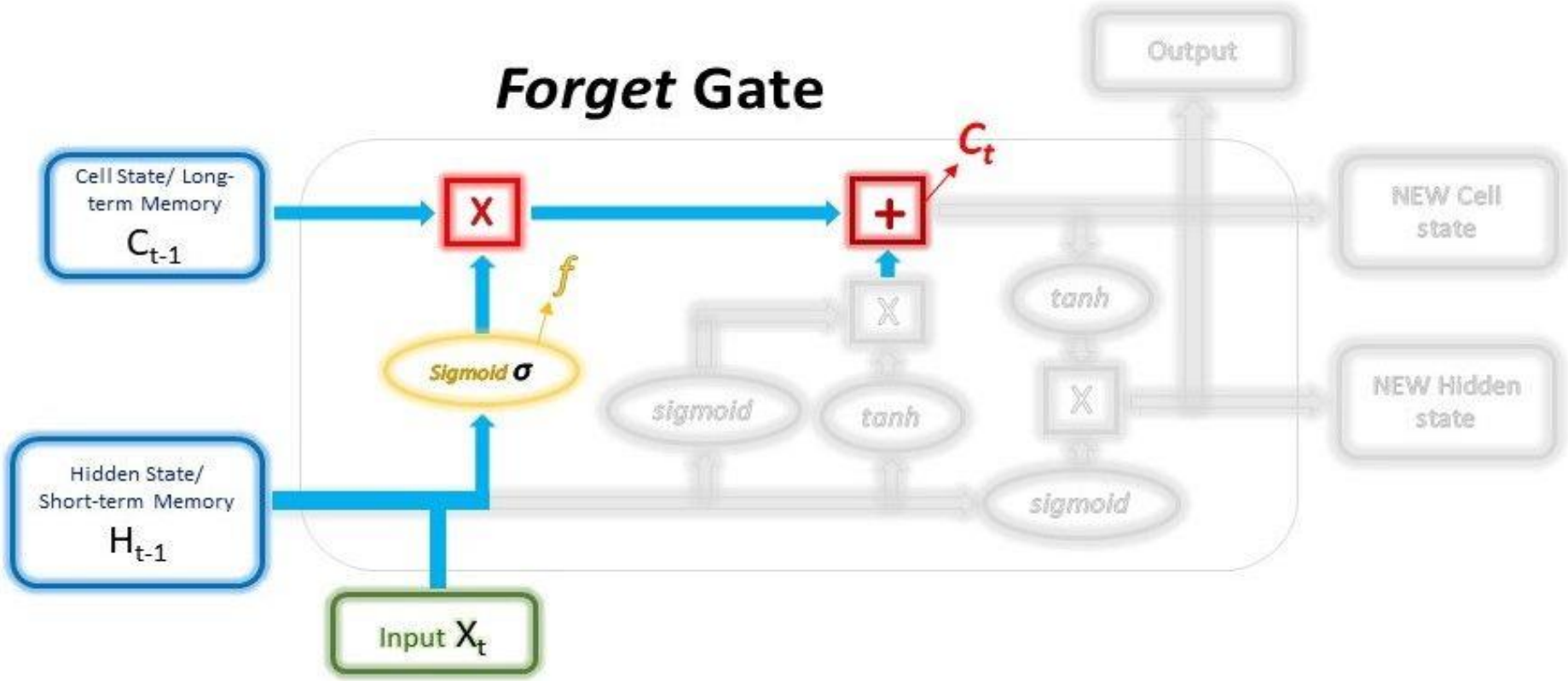
The outputs from these 2 layers are then multiplied, and the final outcome represents the information to be kept in the long-term memory and used as the output.

**input=i1∗i2**

# LSTM

**Forget Gate**

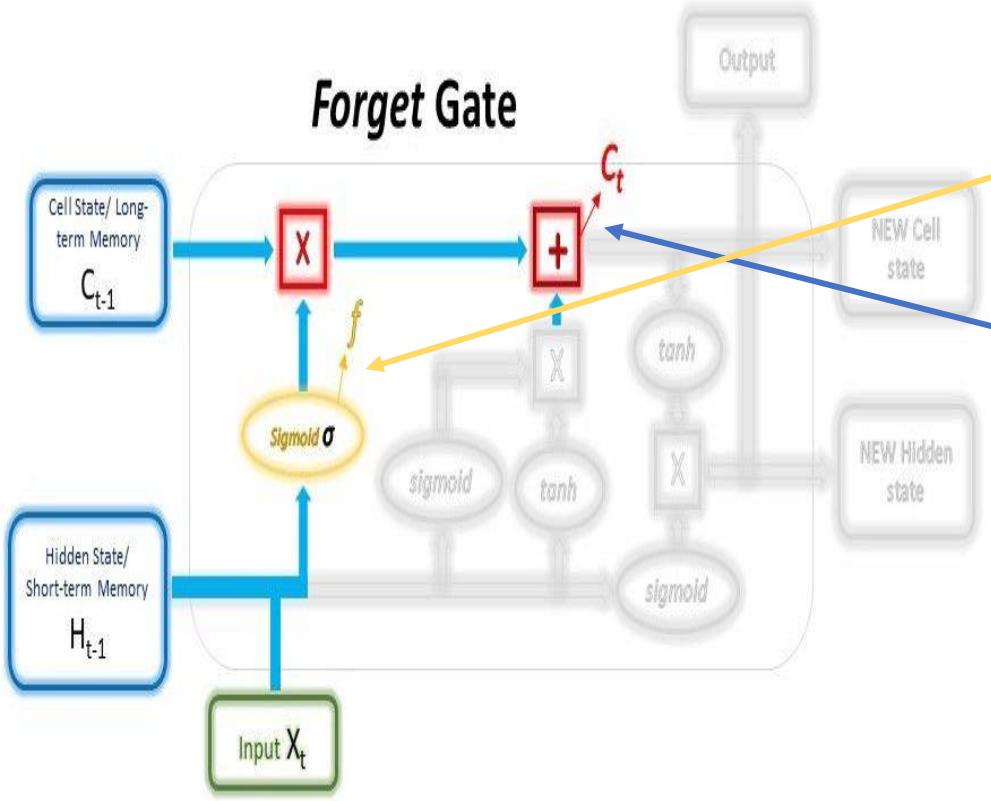
The forget gate decides which information from the **long-term memory should be kept or discarded**. This is done by **multiplying** the incoming **long-term memory by a forget vector generated by the current input and incoming short-term memory.**



# LSTM

**Forget Gate**

The forget gate decides which information from the **long-term memory should be kept or discarded**. This is done by **multiplying** the incoming **long-term memory by a forget vector generated by the current input and incoming short-term memory.**




$$f=\sigma(W_{forget}\cdot(H_{t-1},x_t)+bias_{forget})$$

To obtain the **forget vector**, the short-term memory, and current input is passed through a sigmoid function

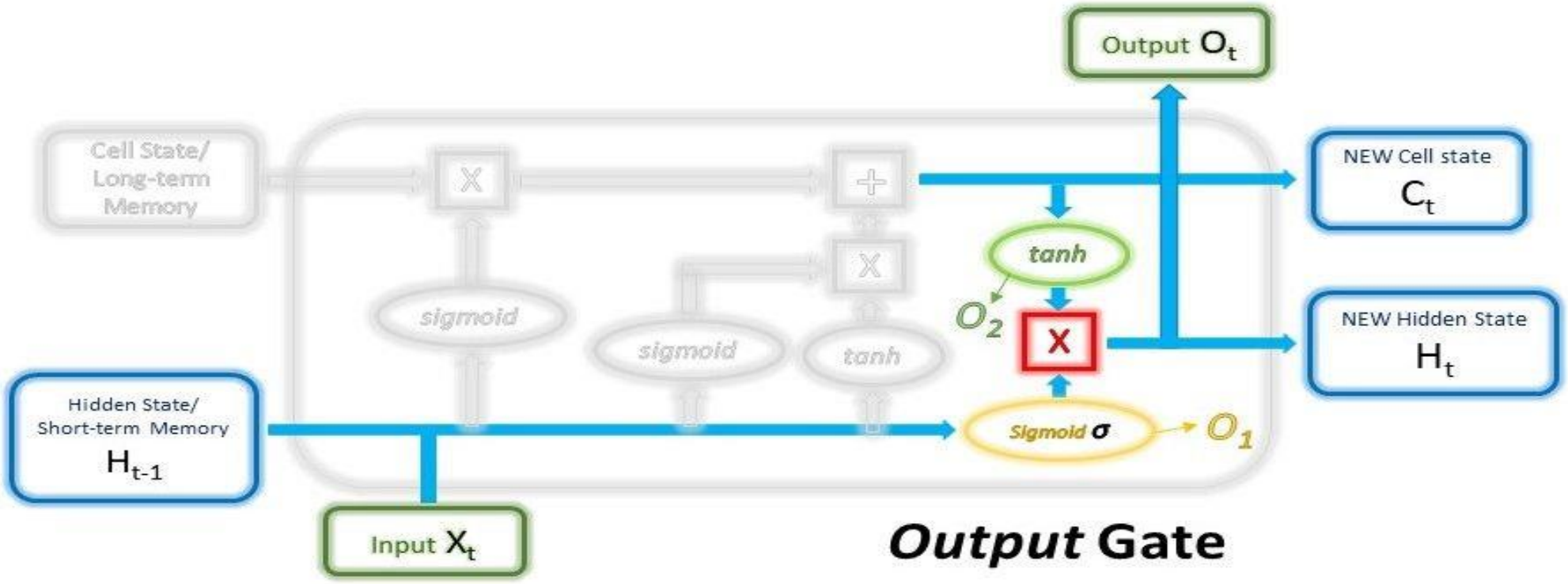
$$Ct=Ct-1*f+iinput$$

The outputs from the **Input gate** and the **Forget gate** will undergo a **pointwise addition** to give a new version of the **long-term memory**, which will be passed on to the **next cell**. This **new long-term memory** will also be used in the final gate, the **Output gate**.

# LSTM

**Output Gate**

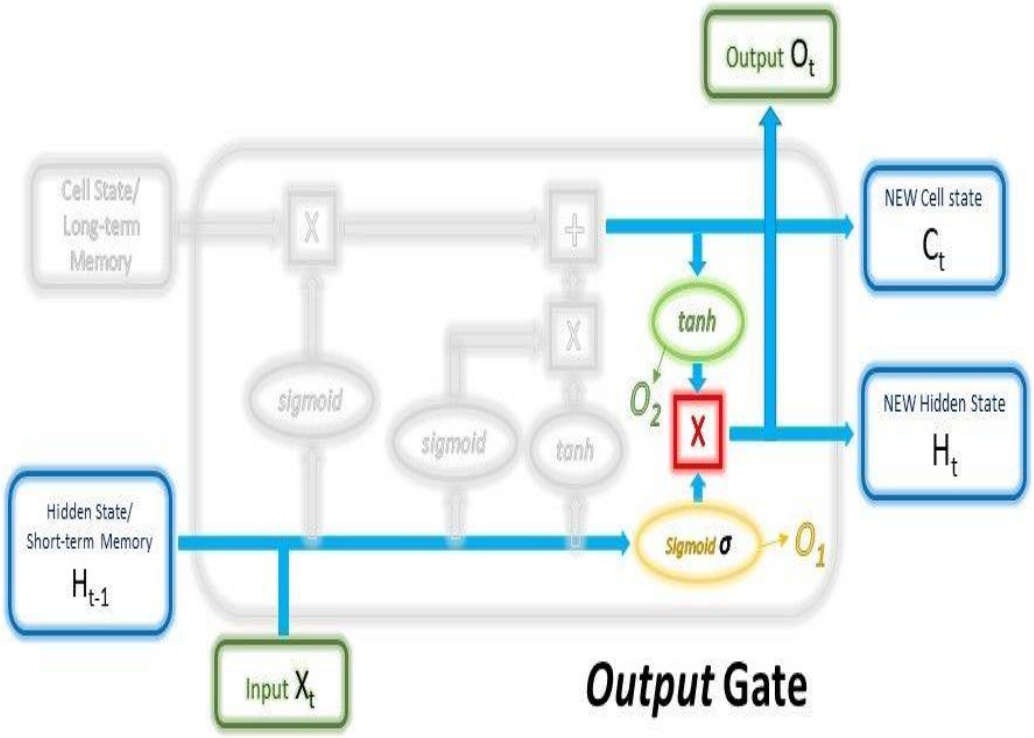
The output gate will take the **current input**, the **previous short-term memory**, and the **newly computed long-term memory** to produce the **new short-term memory/hidden state** which will be passed on to the cell in the next time step. The output of the **current time step** can also be drawn from **this hidden state**.



# LSTM

**Output Gate**

The output gate will take the **current input**, the **previous short-term memory**, and the **newly computed long-term memory** to produce the **new short-term memory/hidden state** which will be passed on to the cell in the next time step. The output of the **current time step** can also be drawn from **this hidden state**.




$$O_1=\sigma(W_{output_1}\cdot(H_{t-1},x_t)+bias_{output_1})$$

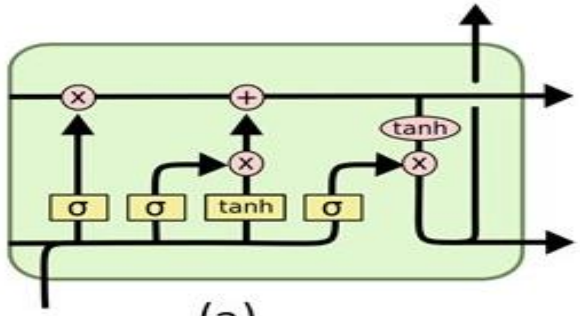
$$O_2=\tanh(W_{output_2}\cdot C_t+bias_{output_2})$$

$$H_t,O_t=O_1*O_2$$

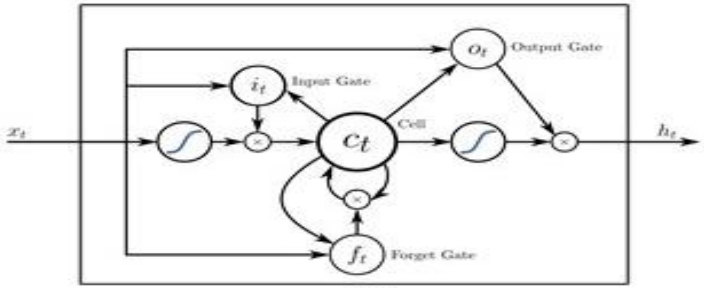
The **short-term and long-term memory produced by these gates** will then be carried over to the next cell for the process to be repeated.

**The output of each time step can be obtained from the short-term memory, also known as the hidden state.**

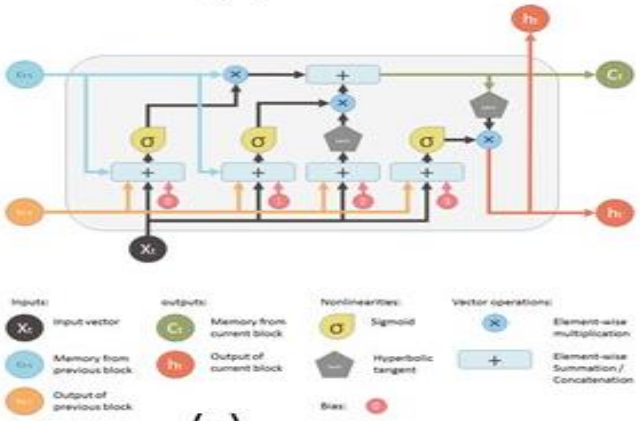
# LSTM



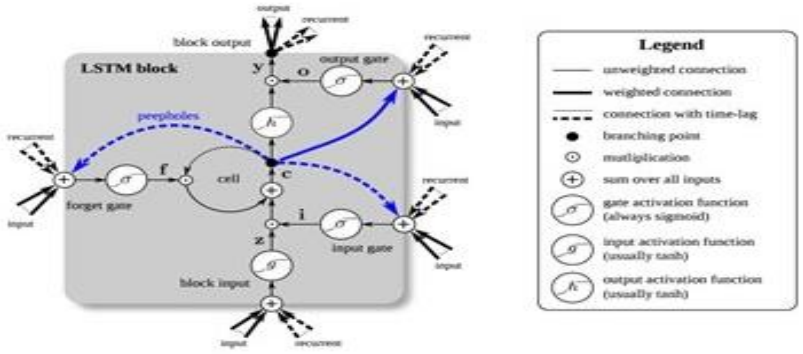

(a)



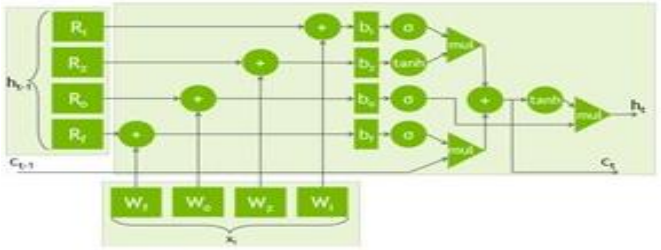

(b)



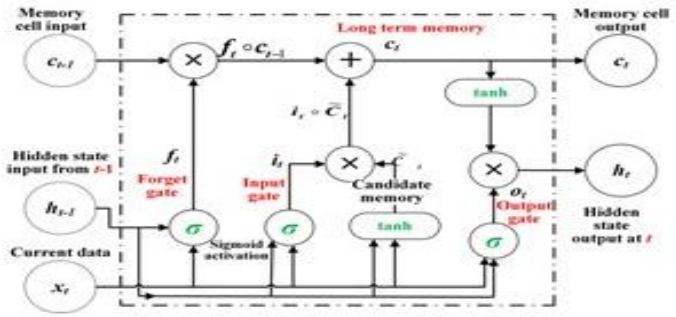

(c)




(d)



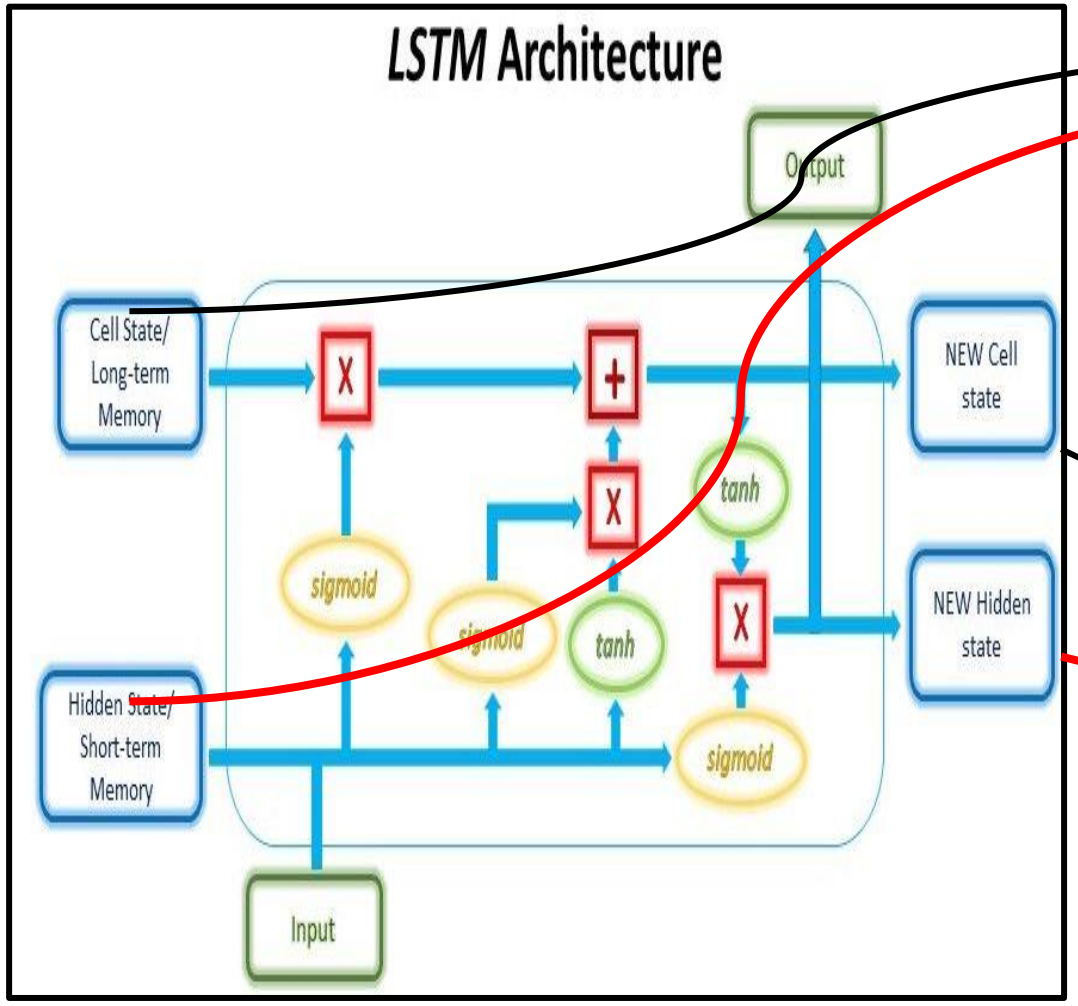
(e)

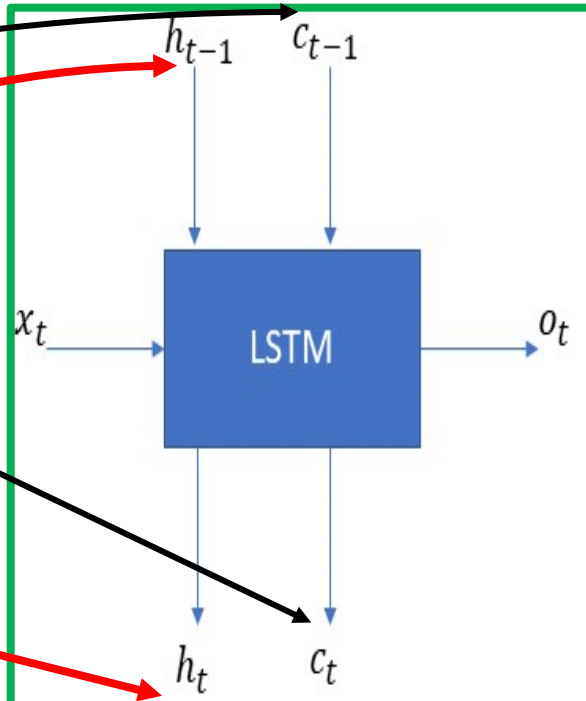



(f)

Different images from popular blogs and paper describing LSTMs.
(a) Christopher Olah's blog.
(b) LSTM from Wikipedia
© Shi Yan's blog on Medium
(d) LSTMs from the deep learning book
(e) Nvidia's blog on accelerating LSTMs
(f) LSTM figure from a Conference paper on human activity detection

# LSTM




$$f_t = \sigma_g\,(W_f \times x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

$\sigma_g$ : sigmoid

$\sigma_c$ : tanh

. : element wise multiplication

$f_t$ is the forget gate

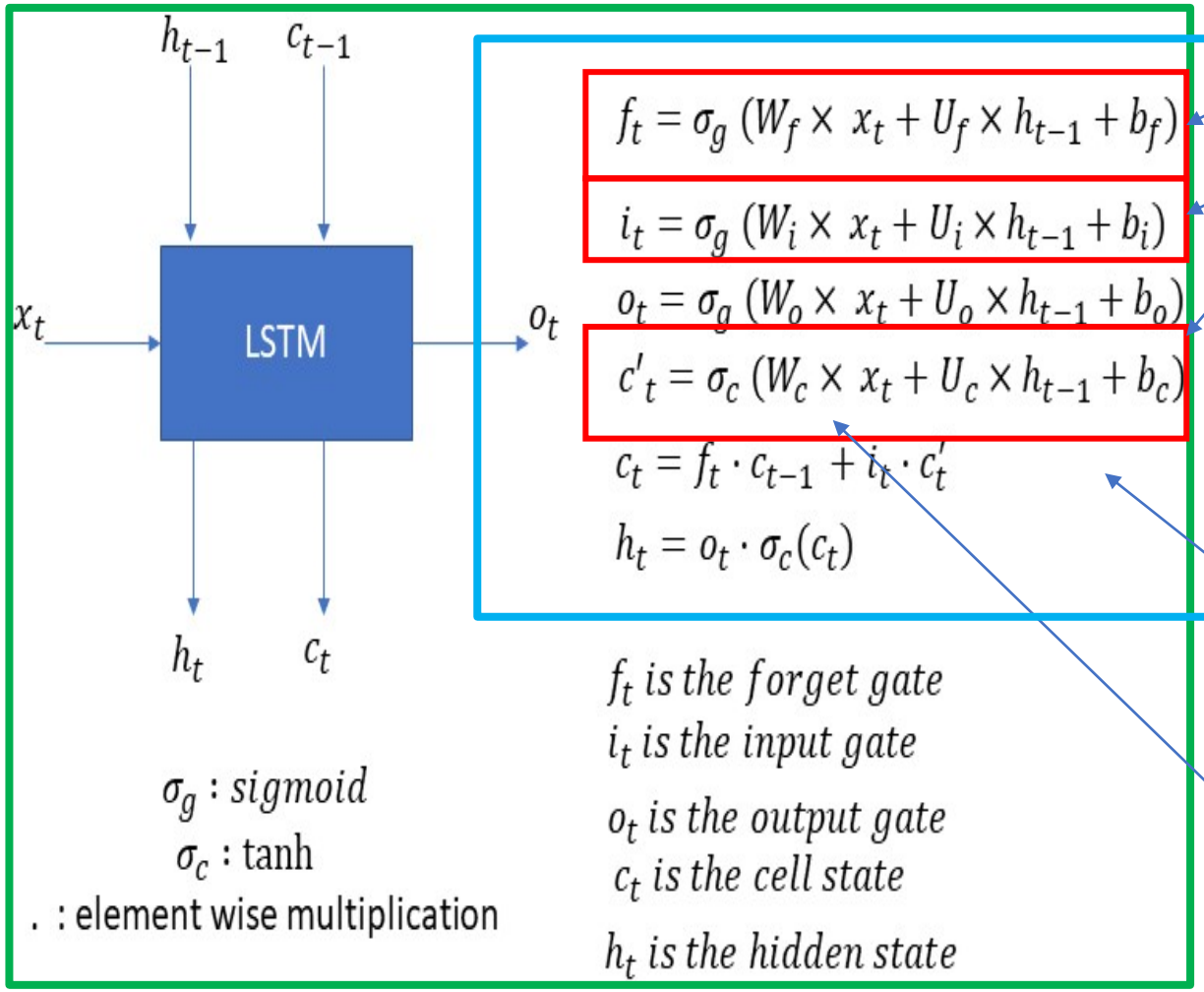
$i_t$ is the input gate

$o_t$ is the output gate

$c_t$ is the cell state

$h_t$ is the hidden state

# LSTM




$$f_t = \sigma_g\,(W_f \times x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

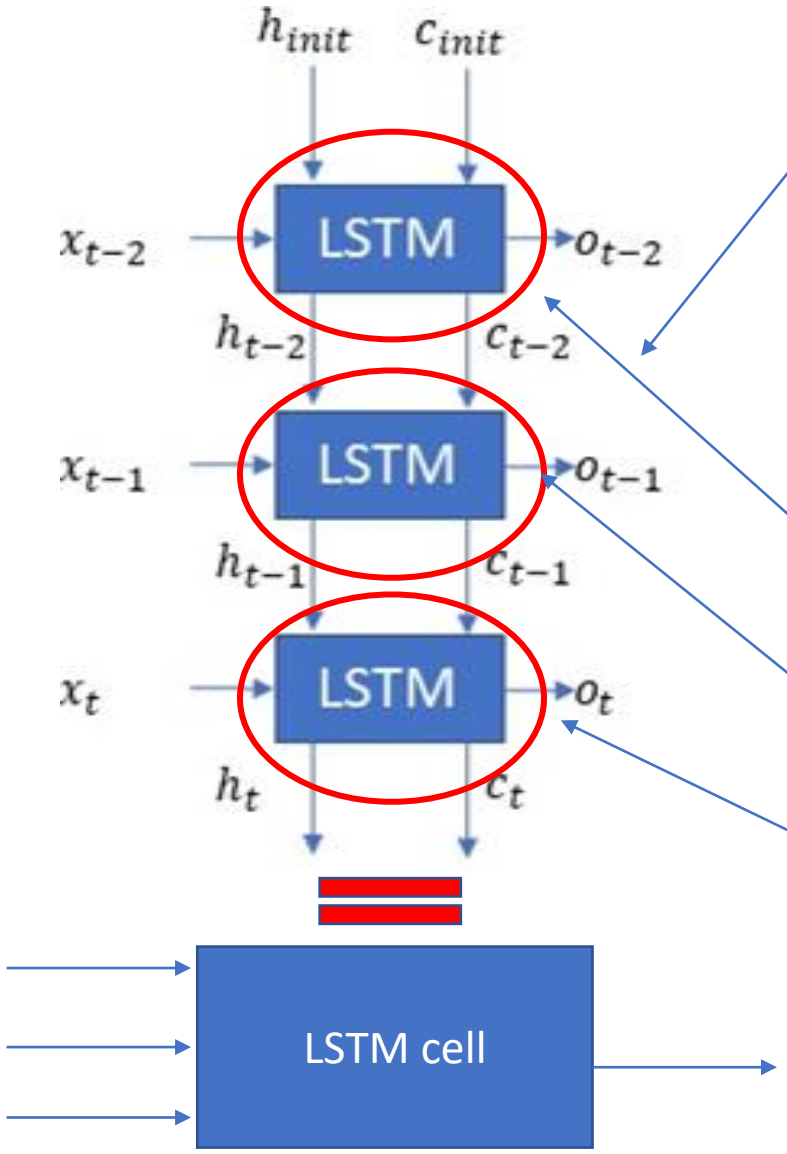
$f_t$ *is the forget gate*
$i_t$ *is the input gate*
$o_t$ *is the output gate*
$c_t$ *is the cell state*
$h_t$ *is the hidden state*

Note that the LSTM equations also generate f(t), i(t), c'(t) these are for internal consumption of the LSTM and are used for generating c(t) and h(t).

1.The above equations are for only a one-time step. This means that these equations will have to be recomputed for the next time step. Thus if we have a sequence of 10 timesteps then the above equations will be computed 10 times for each timestep respectively.

1.The weight matrices (**Wf, Wi, Wo, Wc, Uf, Ui, Uo, Uc**) and biases (**bf, bi, bo, bc**) are not time-dependent. This means that these weight matrices don't change from one time step to another. In other words to calculate the outputs of different timesteps same weight matrices are used.

# LSTM

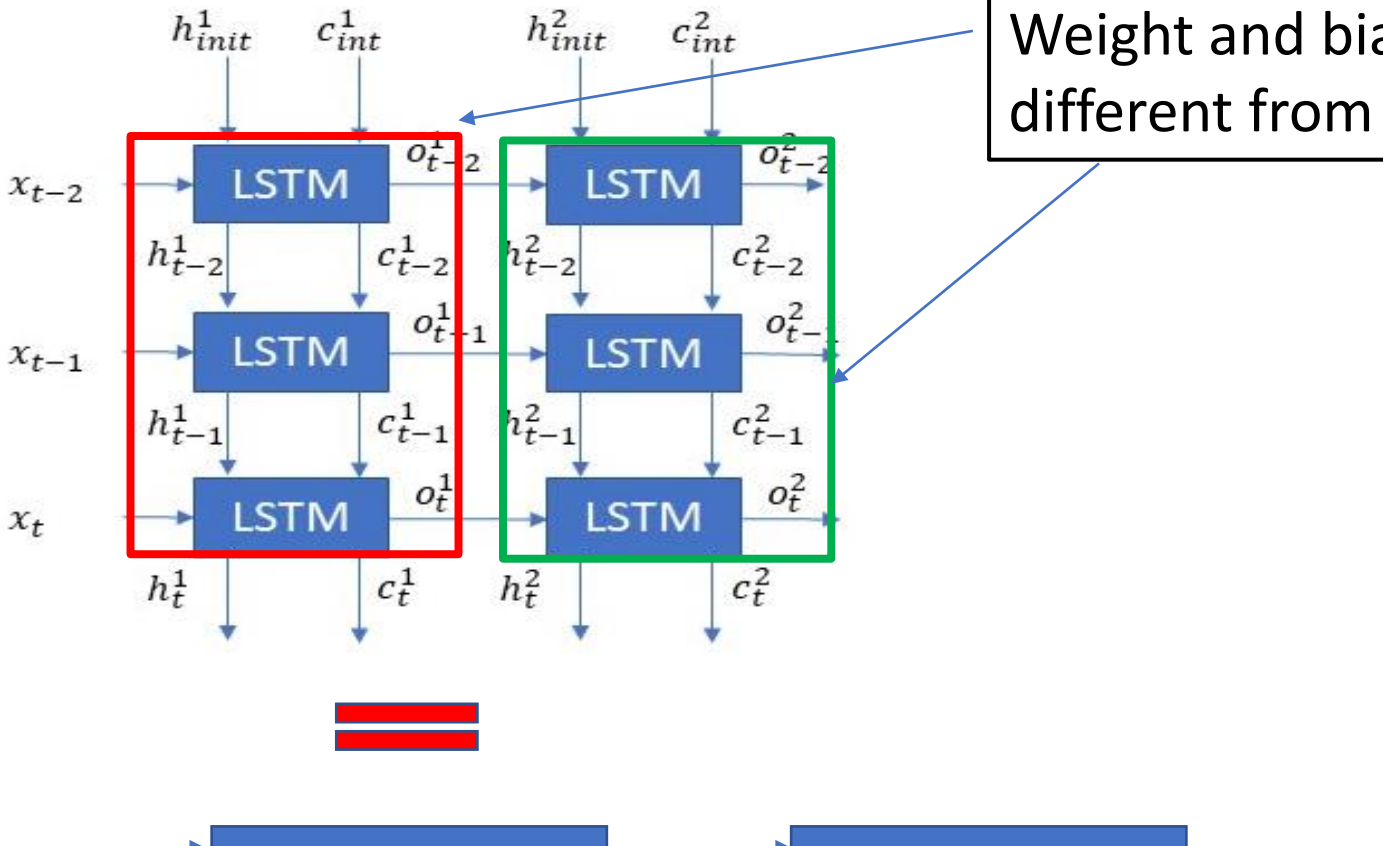



*The weight matrices of an LSTM network do not change from one timestep to another. There are 6 equations that make up an LSTM. If an LSTM is learning a sequence of length 'seq_len'. Then these six equations will be computed a total of 'seq_len'. Essentially for everytime step the equations will be computed.*

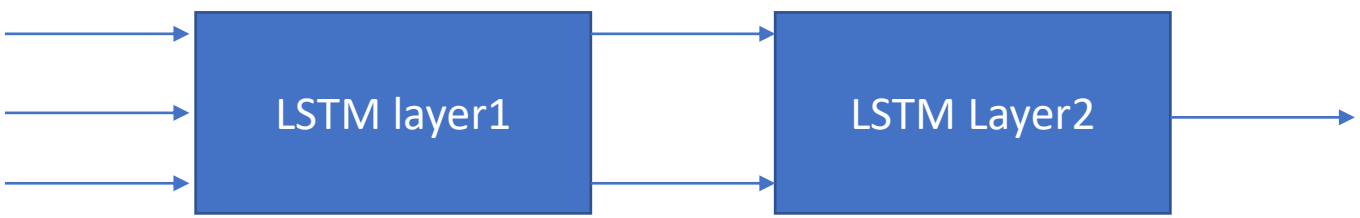
**seq_len=3 in this example**

Weight and biases for all LSTM equations are same for LSTM cells

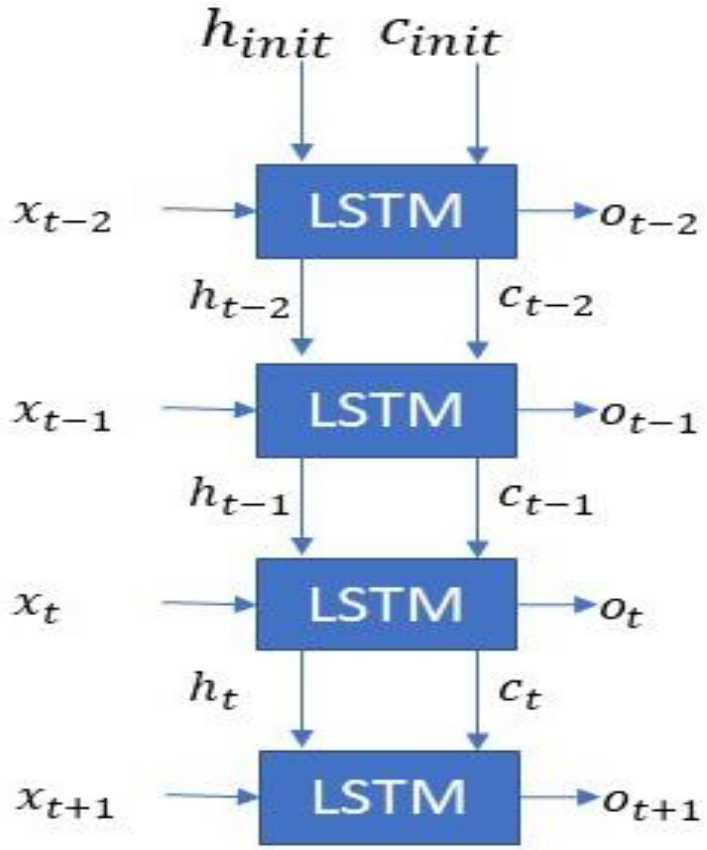
# LSTM




Weight and biases for LSTM layer1 are different from LSTM layer two

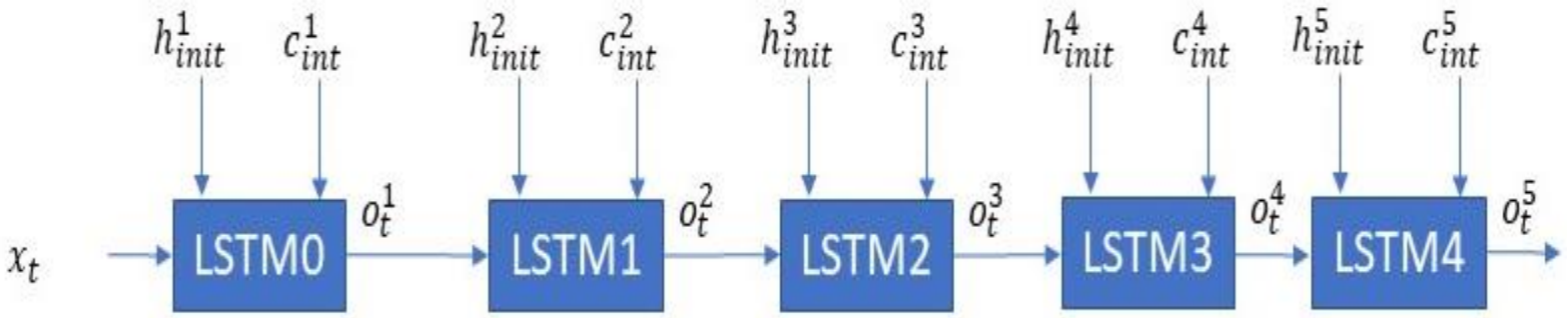


# LSTM



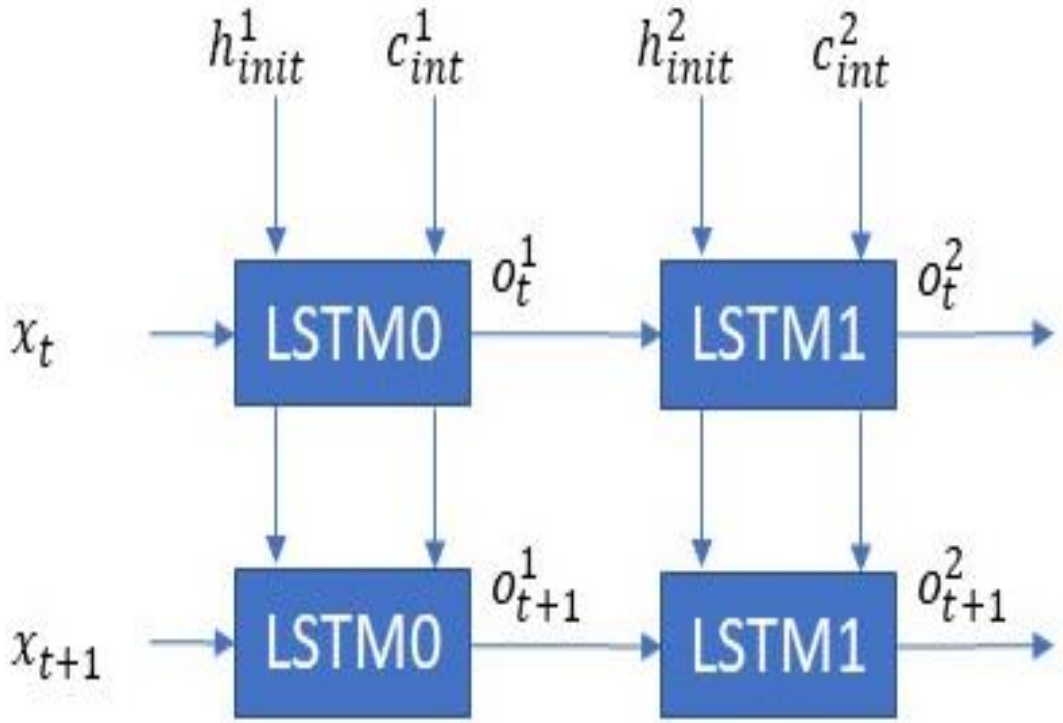

**One layer LSTM and Four timestep sequence**

# LSTM



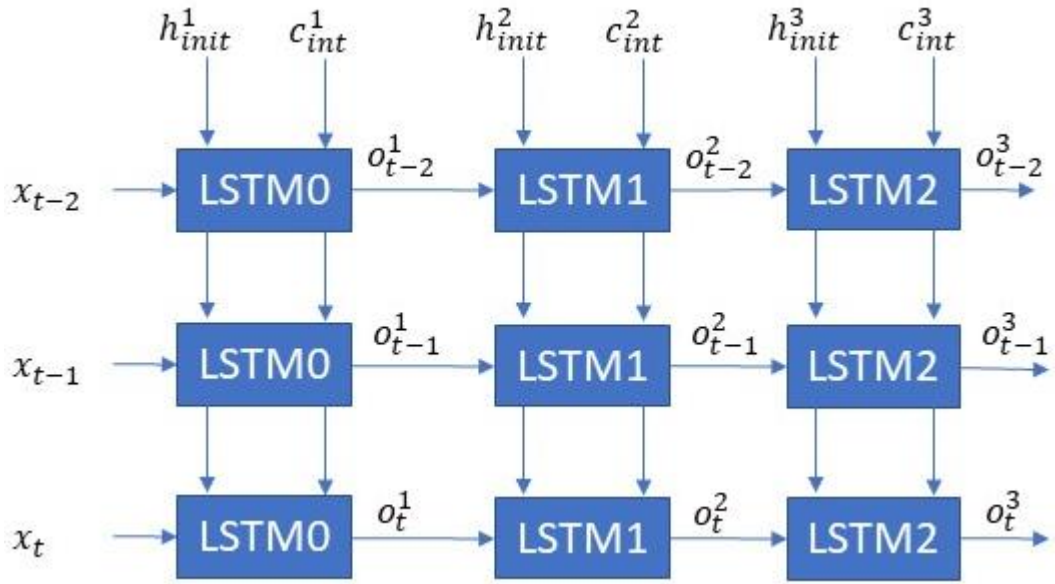

**Five layers LSTM and one timestep sequence**

# LSTM




Two layers LSTM and two timestep sequence

# LSTM




**Three layers LSTM and three timestep sequence**

# LSTM Dimensionalities Computation.

Let take a look at the LSTM equations again in the figure below. As you already know these are the LSTM equations for a single timestep:

$$f_t = \sigma_g\,(W_f \times\ x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times\ x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times\ x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times\ x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

Let's start with an easy one x(t). This is the input signal/feature vector/CNN output. I am assuming that x(t) has an input dimensionality of [80x1]. This implies that Wf has a dimensionality of [Some_Value x 80].

**Input signal or vector or feature matrix=Xt=Features x timestamp**

# LSTM Dimensionalities Computation.

Let take a look at the LSTM equations again in the figure below. As you already know these are the LSTM equations for a single timestep:

$$f_t = \sigma_g\,(W_f \times x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

Feature matrix=samples x features

| Samples | F1 | F2 | F3 | F30 |
|---|---|---|---|---|
| Sample1 | | | | |
| Sample1 | | | | |
| | | | | |
| | | | | |
| | | | | |
| Sample100 | | | | |

Feature matrix=samples x features=100x30

**Feature vector length=30x1**

# LSTM Dimensionalities Computation.

Let take a look at the LSTM equations again in the figure below. As you already know these are the LSTM equations for a single timestep:

$$f_t = \sigma_g\left(W_f \times x_t + U_f \times h_{t-1} + b_f\right)$$

$$i_t = \sigma_g\left(W_i \times x_t + U_i \times h_{t-1} + b_i\right)$$

$$o_t = \sigma_g\left(W_o \times x_t + U_o \times h_{t-1} + b_o\right)$$

$$c'_t = \sigma_c\left(W_c \times x_t + U_c \times h_{t-1} + b_c\right)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

Feature matrix=samples x features

| Samples | F1 | F2 | F3 | F15 |
|---|---|---|---|---|
| Sample 1 | | | | |
| Sample 1 | | | | |
| | | | | |
| | | | | |
| | | | | |
| Sample 100 | | | | |

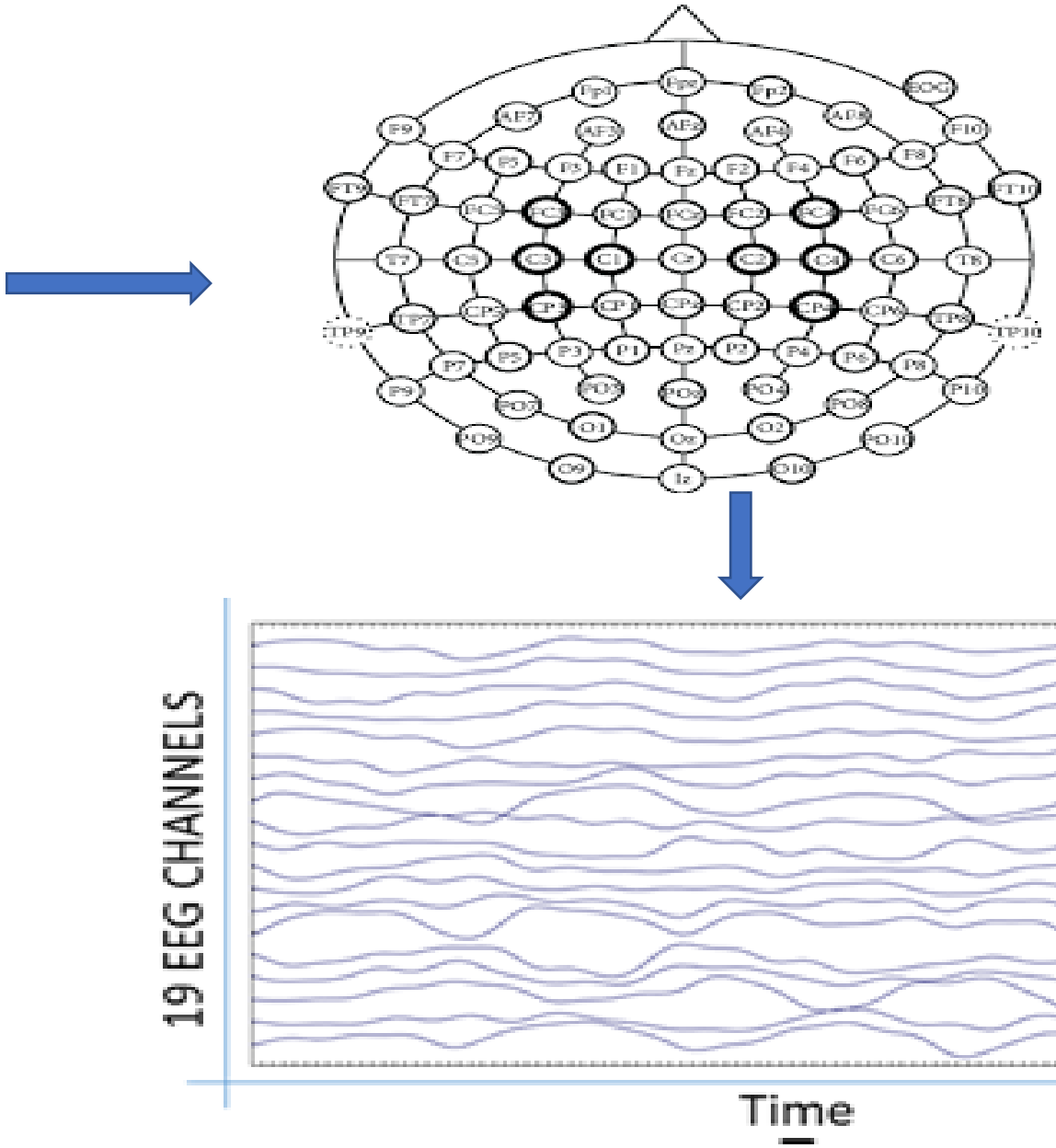
| Samples | F16 | F2 | F3 | F30 |
|---|---|---|---|---|
| Sample 1 | | | | |
| Sample 1 | | | | |
| | | | | |
| | | | | |
| | | | | |
| Sample 100 | | | | |

Reshaping feature vector
For two timestamp, t=0,t=1

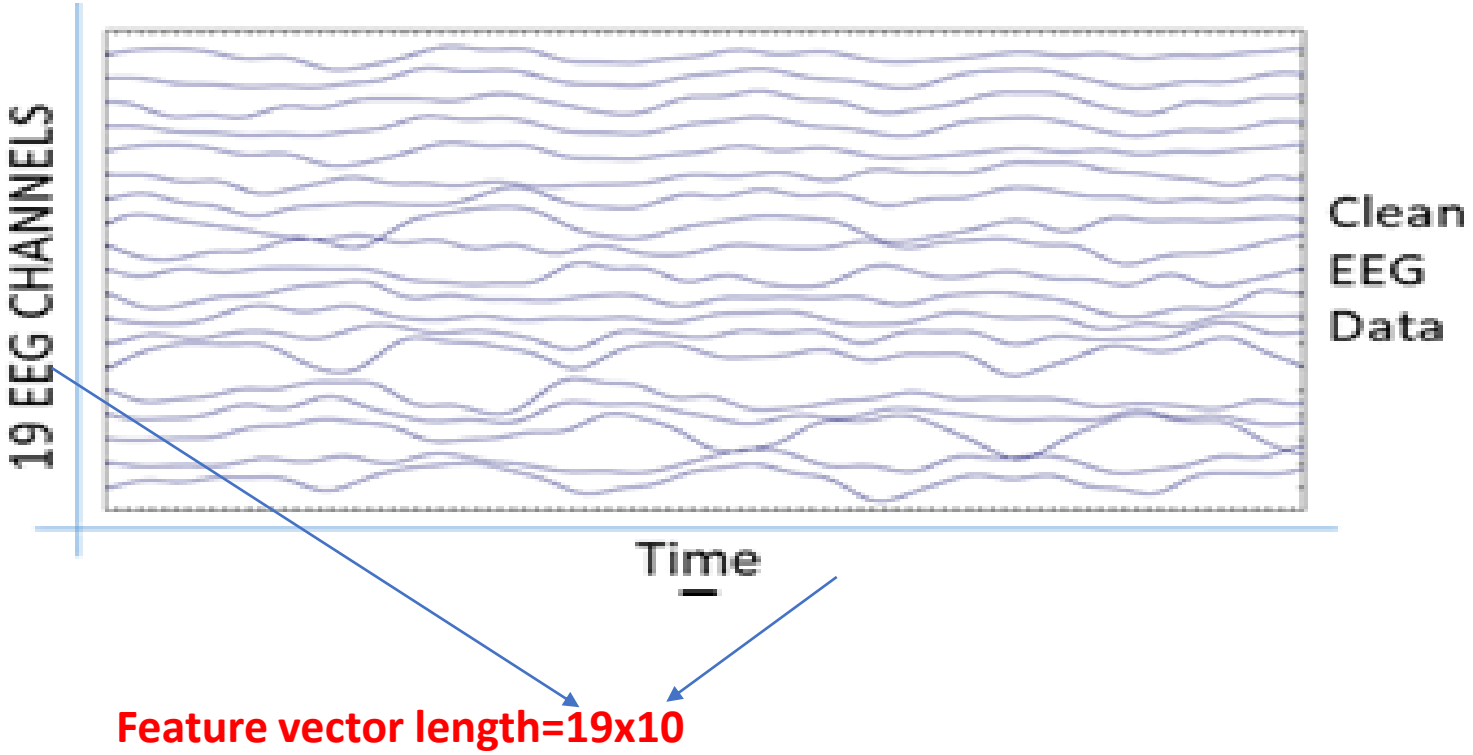
t=0 t=1

**Feature vector length=15x2 (**divide sequence into two timestep **)**
**Feature vector length=30x2 (**you can create the same copy of sequence into two timestep)

# LSTM Dimensionalities Computation.

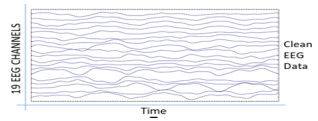
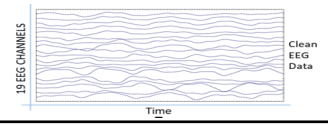
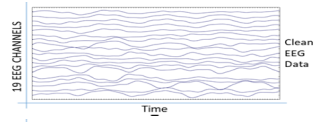
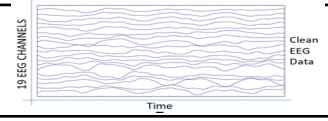
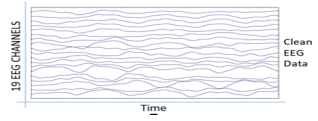
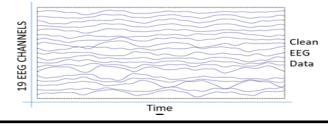



# LSTM Dimensionalities Computation.




Assume that 19 Channels or electrodes has **19 features or signal** and every feature has **10 second time length**

# Basic dimensions of EEG dataset in LSTM models

| No of samples | Classes | Dimension | Labels |
|---|---|---|---|
| 1 | Patient1 | 19x256   | 0 |
| 2 | Patient1 | 19x256   | 0 |
| | Patient1 | ---- | 0 |
| | Patient1 | --- | 0 |
| 50 | Patient1 | 19x256   | 0 |
| 51 | Patient2 | 19x256   | 1 |
| | Patient2 | 19x256   | 1 |
| | Patient2 | ---- | 1 |
| 99 | Patient2 | ---- | 1 |
| 100 | Patient2 | 19x256   | 1 |

Total samples x timestep x features

100 x 256 x 19

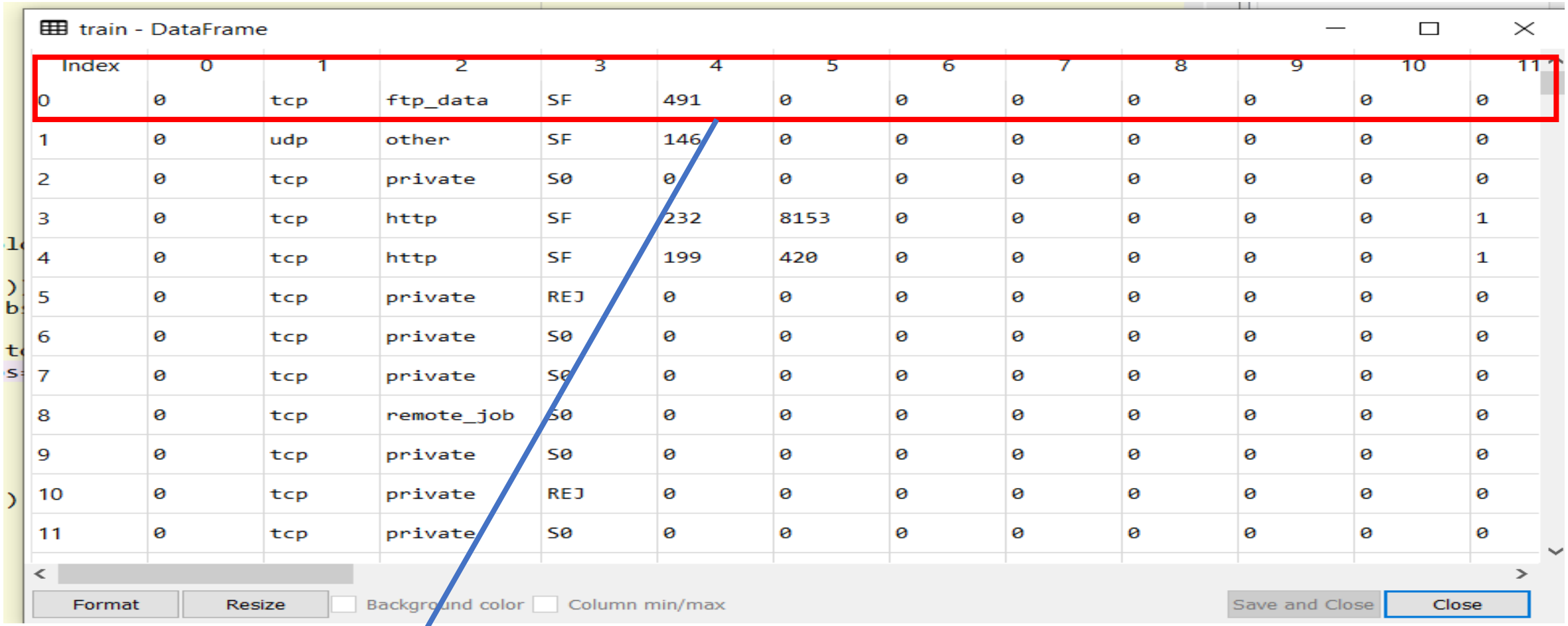
Training samples

80 x 256 x 19

Testing samples

20 x 256 x 19

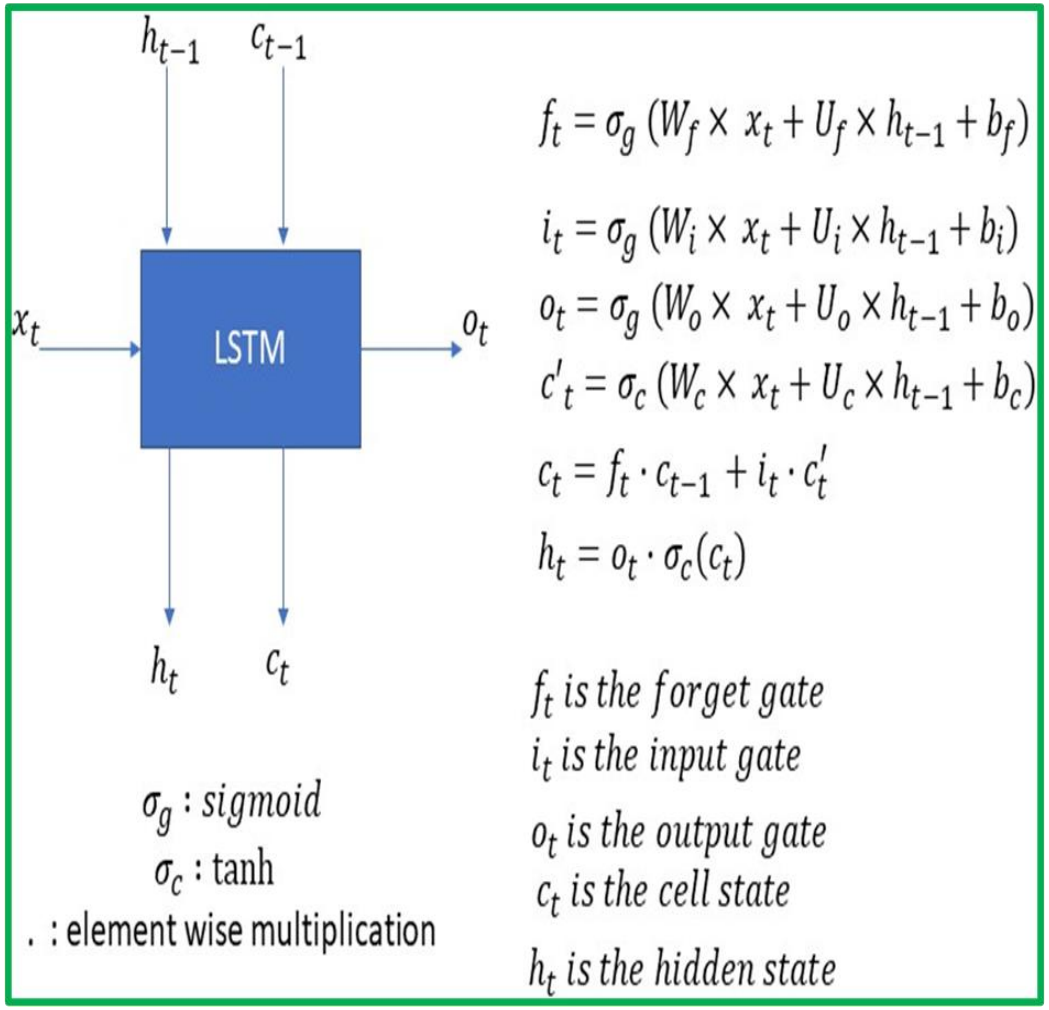
# Basic dimensions of Network Intrusion Dataset in LSTM Models




| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | tcp | ftp_data | SF | 491 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | udp | other | SF | 146 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | tcp | http | SF | 232 | 8153 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | tcp | http | SF | 199 | 420 | 0 | 0 | 0 | 0 | 0 | 1 |
| 5 | 0 | tcp | private | REJ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | tcp | remote_job | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | tcp | private | REJ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | tcp | private | S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Assume that **41 features or signal** and every feature has **1 time length**
**Feature vector length=41x1**

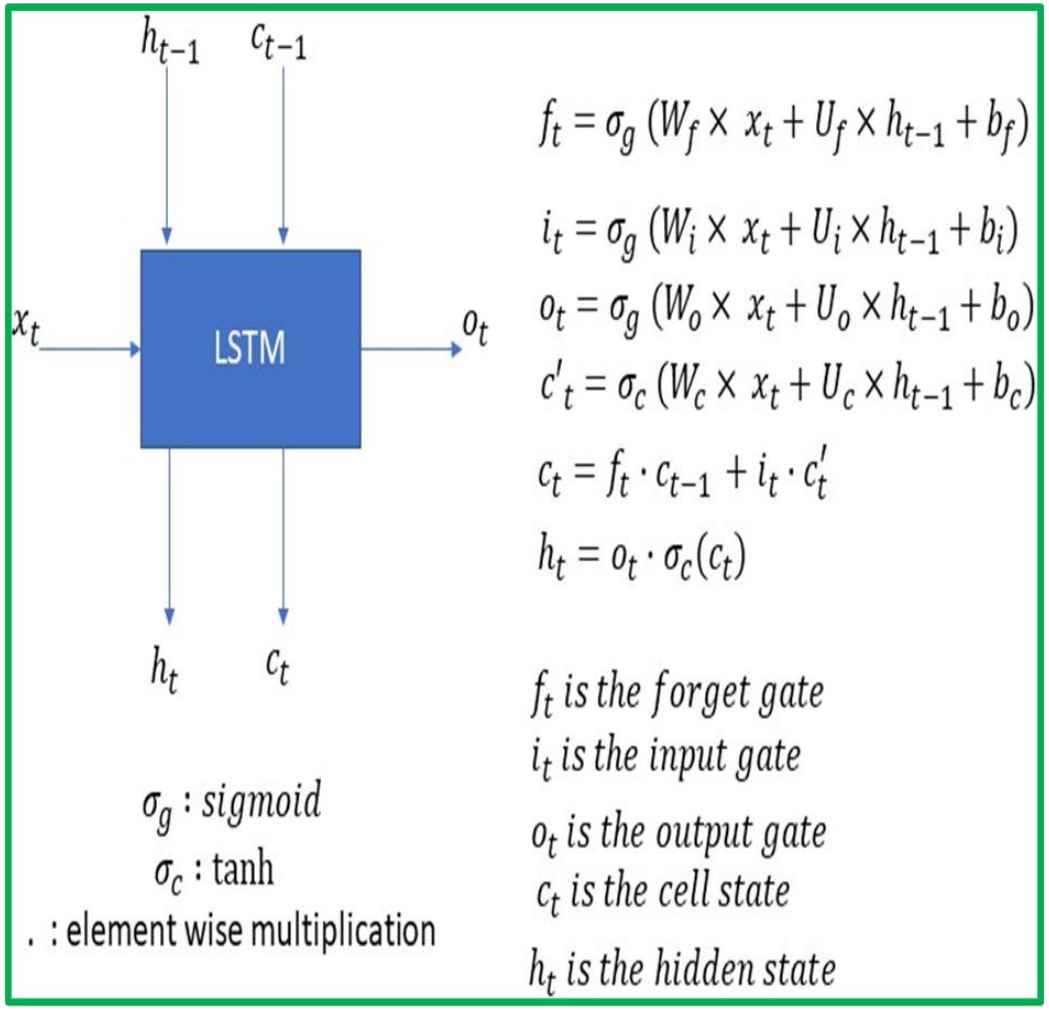
# Basic Dimensions of LSTM Layer




For example:
**x(t) is [80 X 1]** — **input assumption**
Wf is [Some_value X 80 ] — Matrix multiplication laws.

Let's make another assumption the
**output dimensionality or hidden state hidden Units of LSTM or LSTM cell** of the LSTM is **[12x1].**

# Basic Dimensions of LSTM Layer

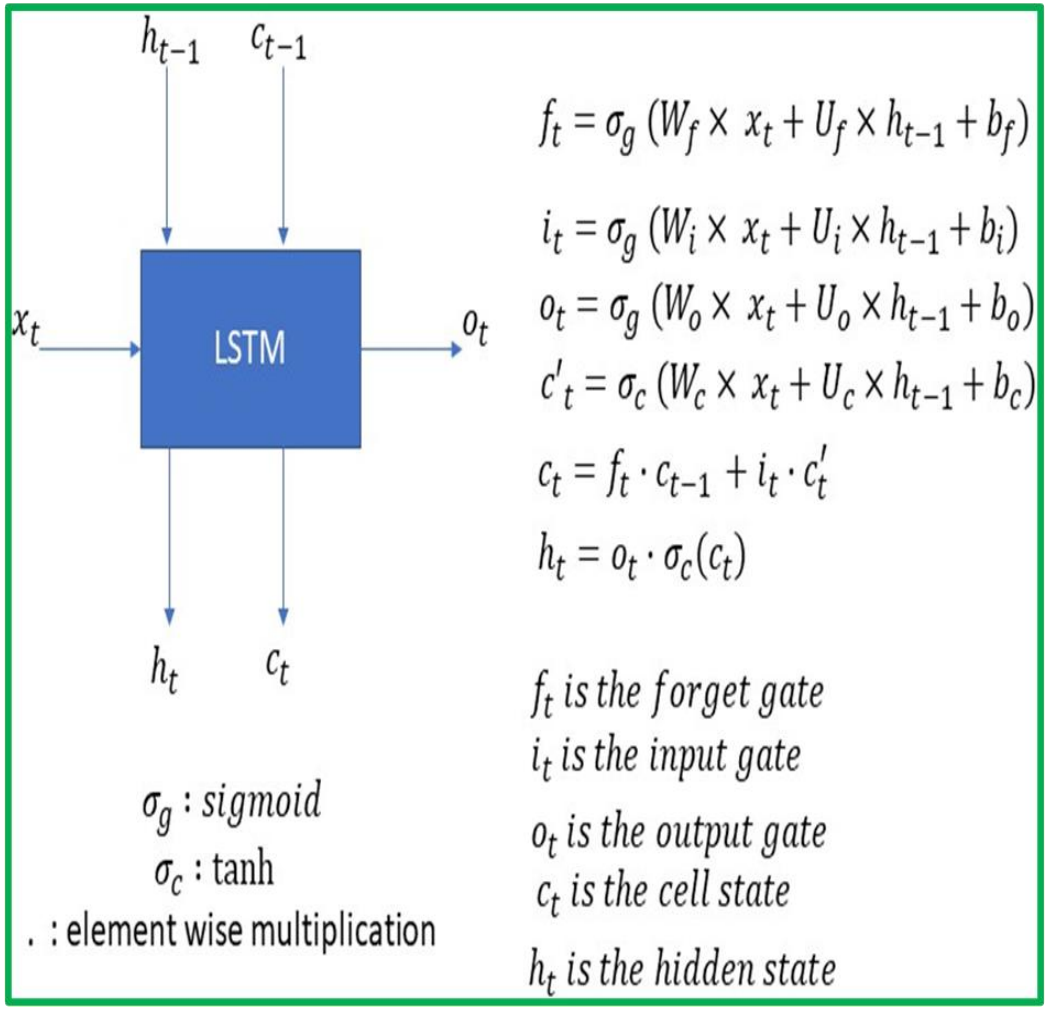



For example:
**x(t) is [80 X 1] — input assumption**
Wf is [Some_value X 80 ] — Matrix multiplication laws.

Let's make another assumption the
**output dimensionality or hidden state hidden Units of LSTM or LSTM cell** of the LSTM is **[12x1].**

**Thus at every timestep, the LSTM generates an output o(t) of size [12 x 1].**

# Basic Dimensions of LSTM Layer



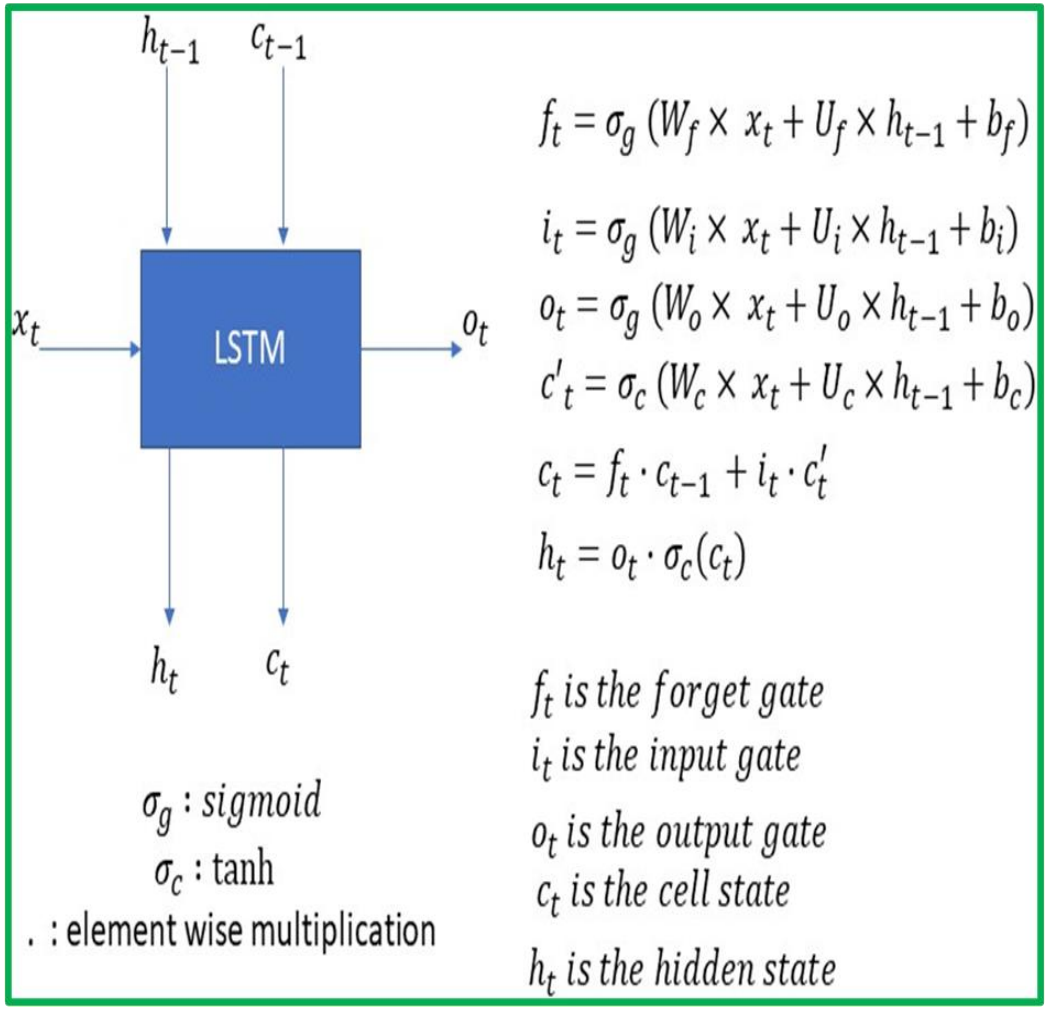

Let's make another assumption the **output dimensionality or hidden state hidden Units of LSTM or LSTM cell** of the LSTM is **[12x1].**

**Thus at every timestep, the LSTM generates an output o(t) of size [12 x 1].**

Now since **o(t)** is [12 x 1] then **h(t)** has to be [12x1] because **h(t)** is calculated by doing an element by element multiplication.
Since **o(t)** is [12x1] then **c(t)** has to be [12x1].

If c(t) is [12x1]
**then f(t), c(t-1), i(t) and c'(t) have to be [12x1].**

# Basic Dimensions of LSTM Layer




Let's make another assumption the **output dimensionality or hidden state hidden Units of LSTM or LSTM cell** of the LSTM is **[12x1].**

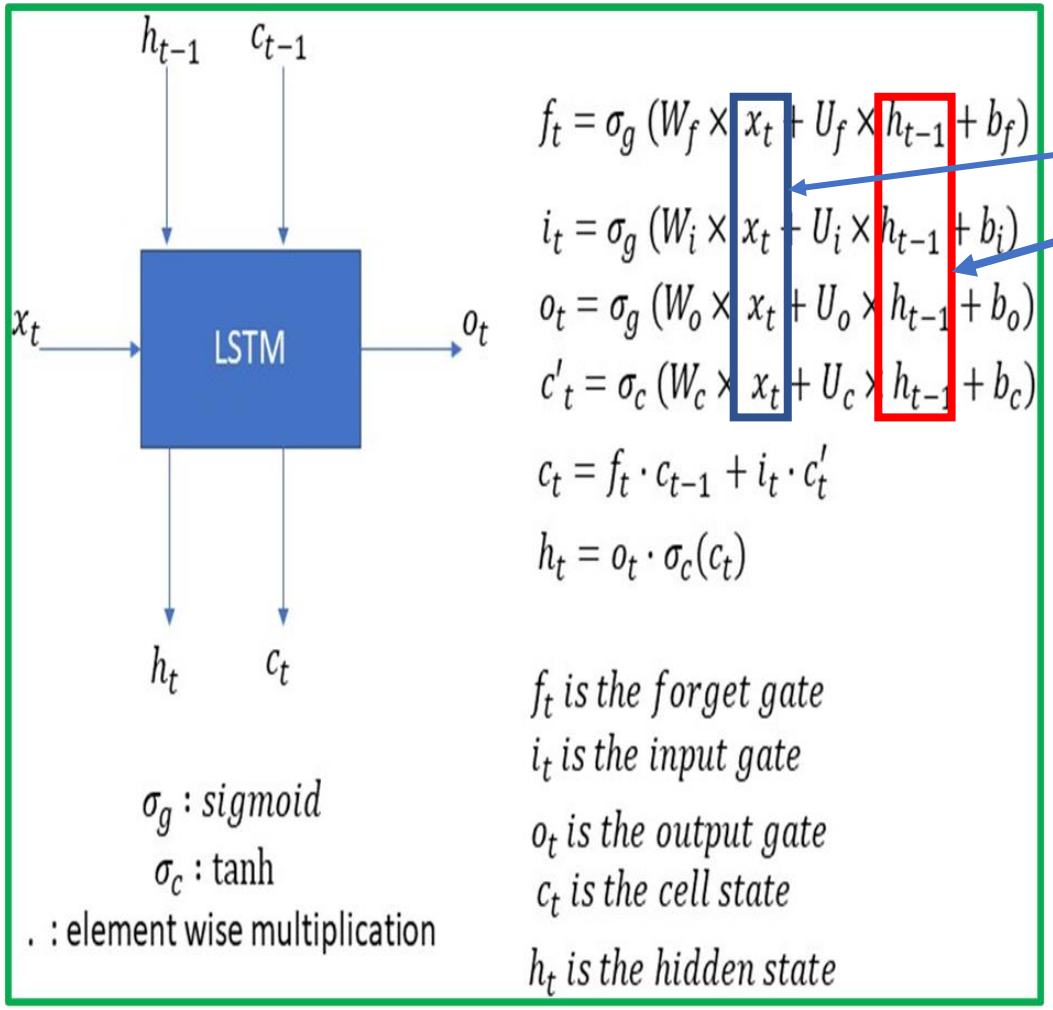
**Initially in first LSTM layer output dimension for cell and hidden state has the same dimension of previous state cell and hidden state.**

**ht=ht+1**

**ct=ct+1**

**ct=ot has same dimension**

# Basic Dimensions of LSTM Layer

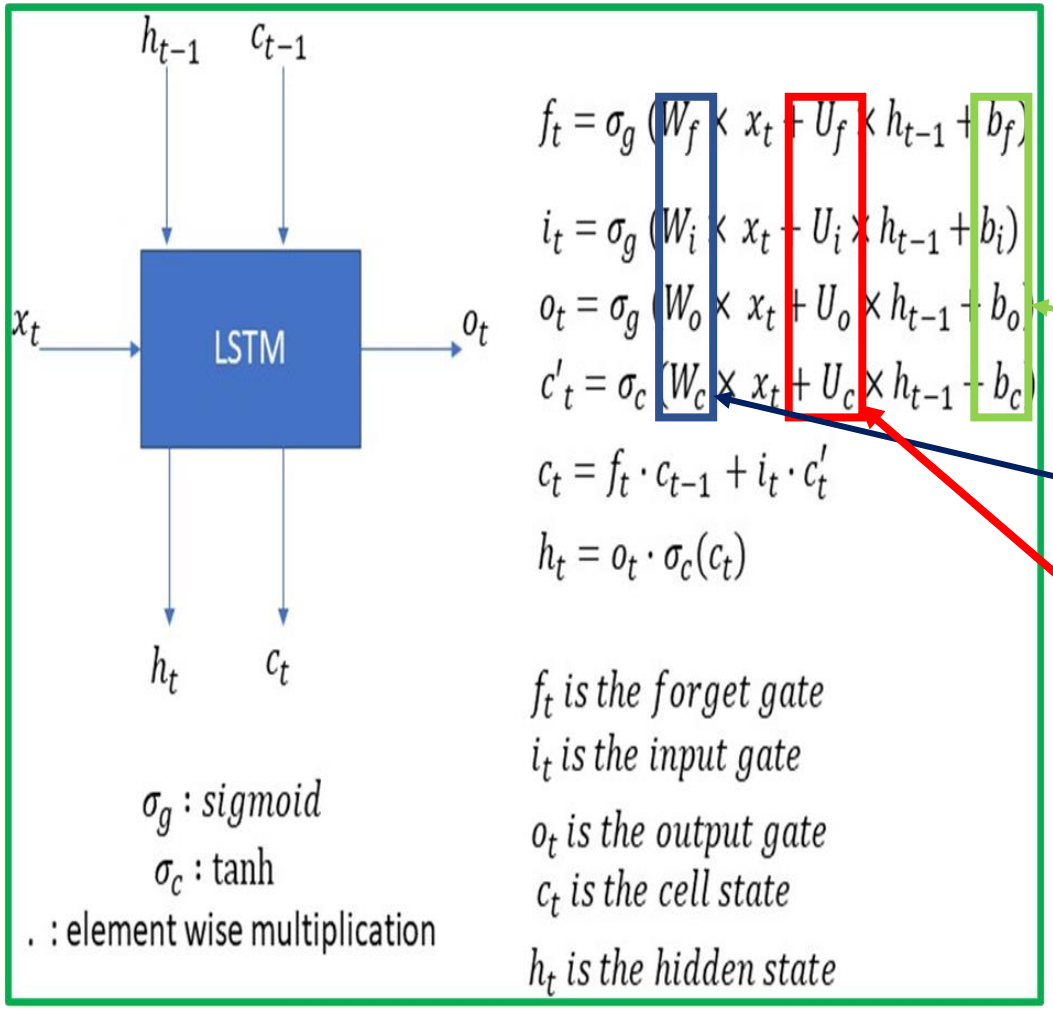



**For example Given:**

**x(t) is [80 X 1] — input assumption**
**h(t) and o(t) and h(t-1) c(t) and c(t-1)-12x1 output dimensionality**

How to compute ?
Weights matrices and biases vectors from input vector dimension

# Basic Dimensions of LSTM Layer




**Since f(t) is of dimension [12x1] then the product of Wf and x(t) has to be [12x1].**

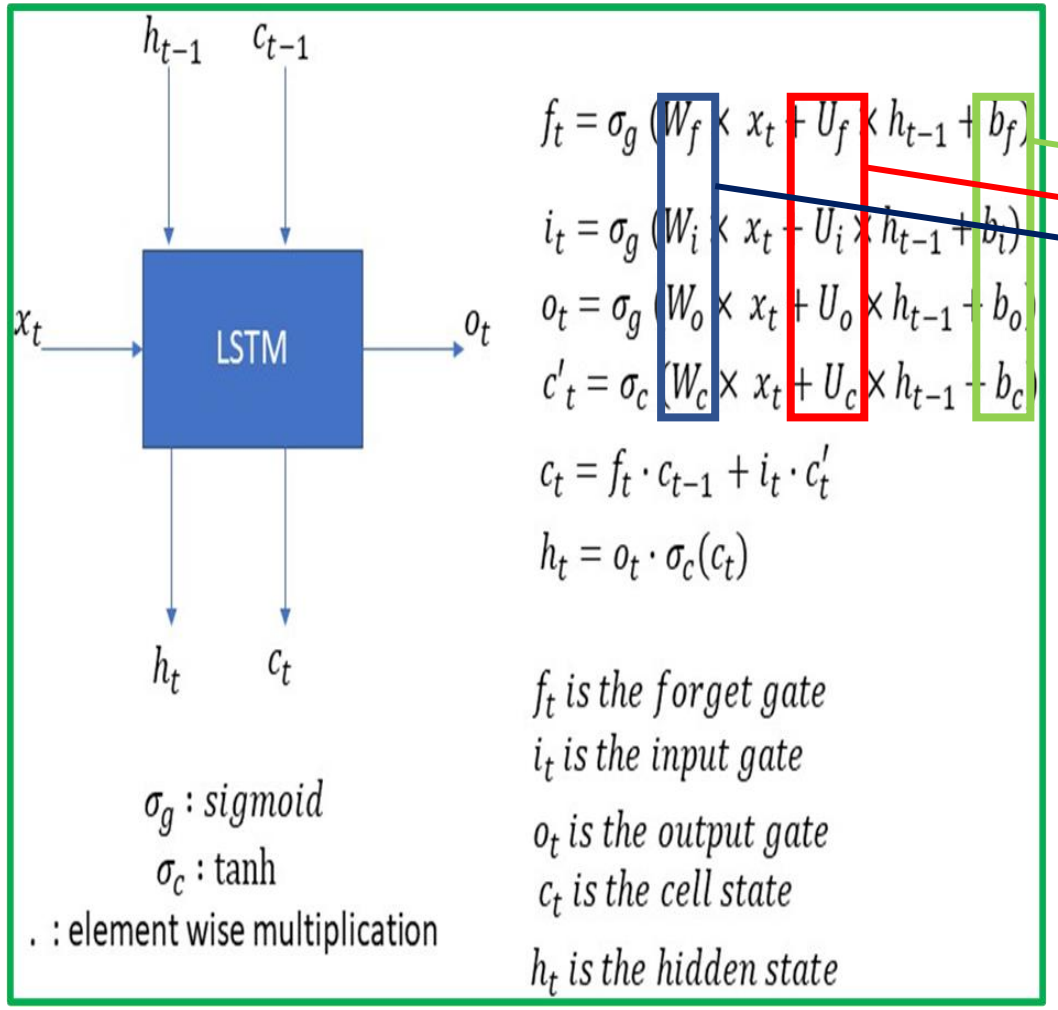
**ft=Wf x x(t)+Ufxh(t-1)=12x80x80x1+12x12x12x1+12x1=12x1**

**Wf=12x80**
**Wi=12x80**
**Wo=12x80**
**Wc=12x80**

***bf, bi, bc, bo each have dimensions of [12x1]***

**Uf=12x12**
**Ui=12x12**
**U0=12x12**
**Uc=12x12**

# Basic Dimensions of LSTM Layer

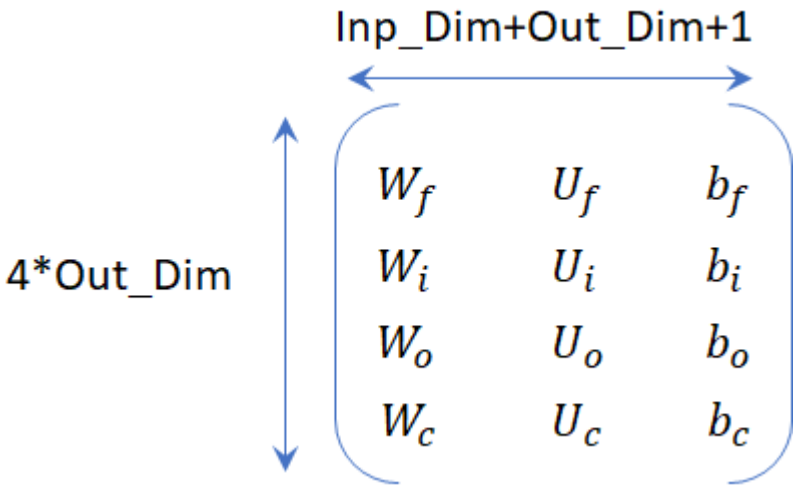



The total weight matrix size of the LSTM is

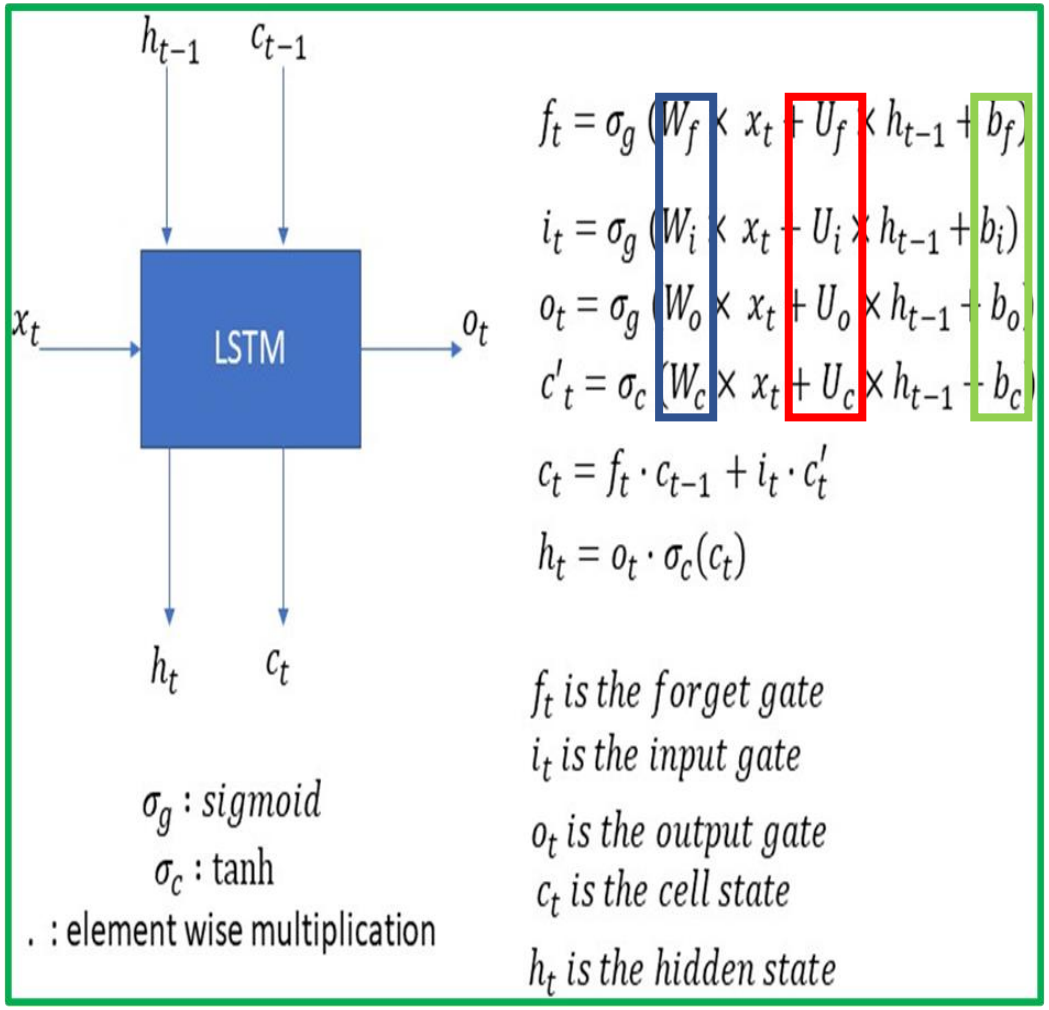
**Weights_LSTM = 4*[12x80] + 4*[12x12] + 4*[12x1]**

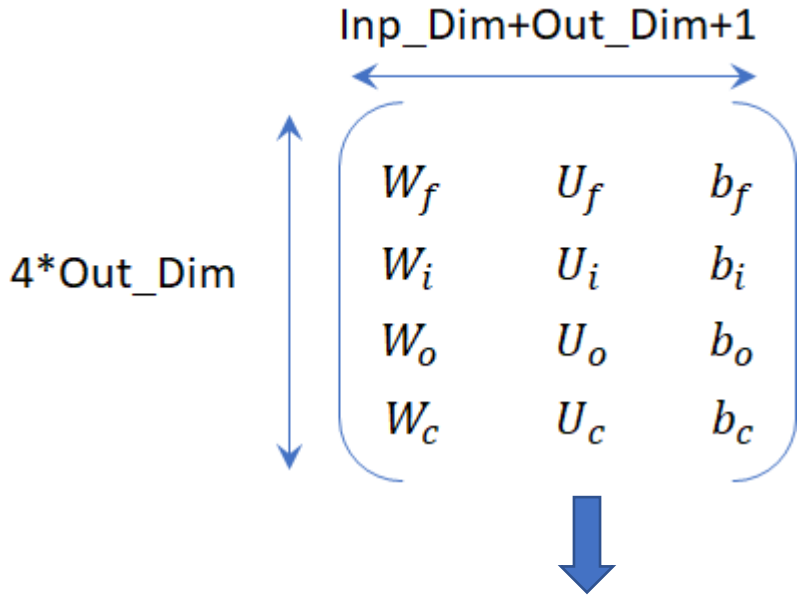
**= 4*[Output_Dim x Input_Dim] + 4*[Output_Dim$^2$] + 4*[Input_Dim]**

**= 4*[960] + 4*[144] + 4*[12] = 3840 + 576+48= 4,464**



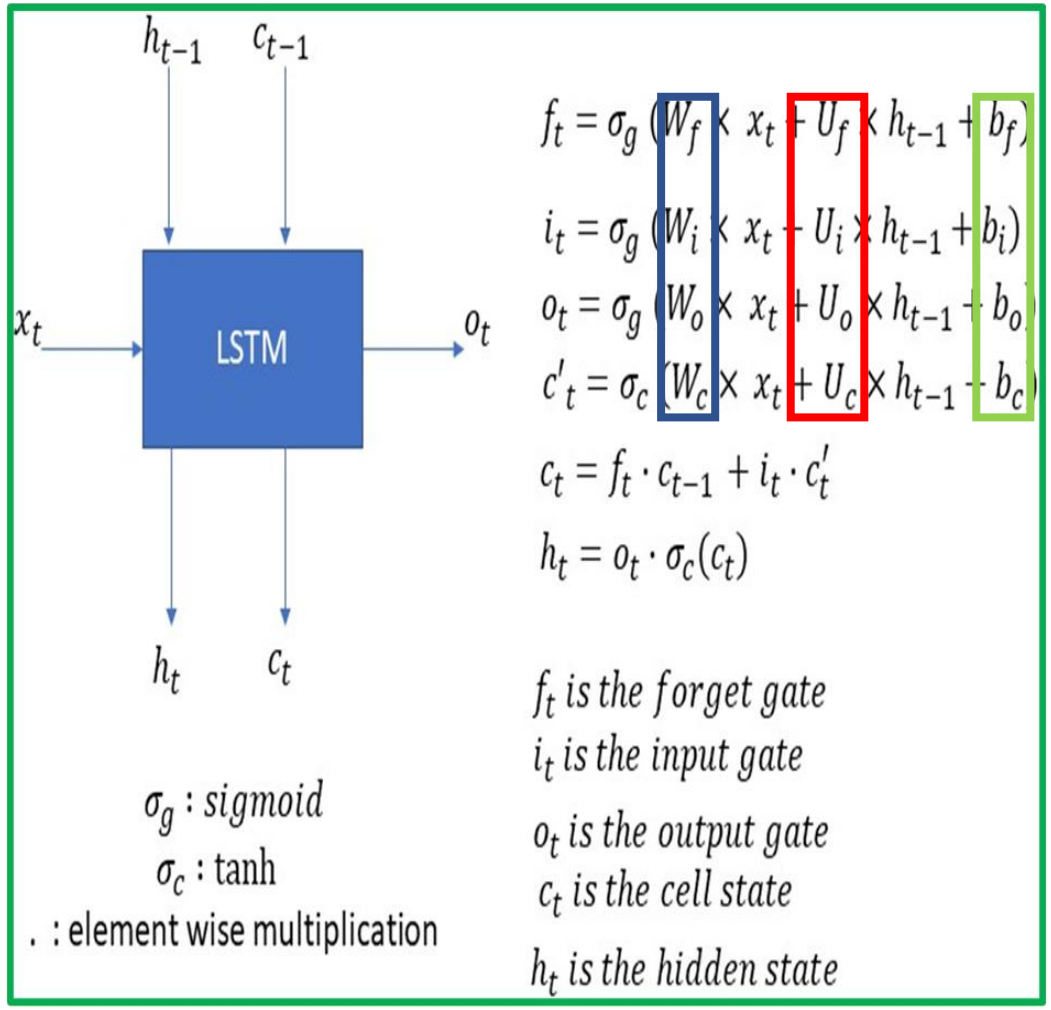
# Basic Dimensions of LSTM Layer




*The weight matrix size is of the size:*
***4*Output_Dim*(Output_Dim + Input_Dim + 1)***

# Basic Dimensions of LSTM Layer

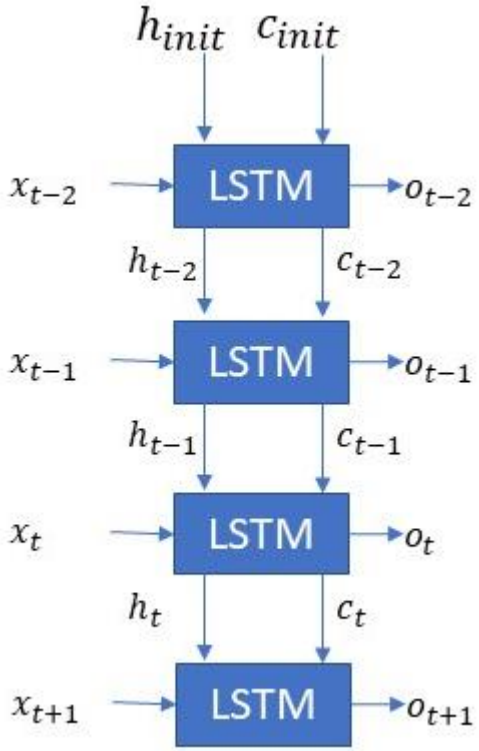



**Manual calculation**

**4,464**

```
import tensorflow as tf
from tensorflow.keras import layers

# Define a sequential model
model = tf.keras.Sequential()

# Input layer takes an input of 1000 and outputs 80
model.add(layers.Embedding(input_dim=1000,
output_dim=80))

# Add an LSTM layer of output dimensionality of 12
model.add(layers.LSTM(12))

# Print the output
print (model.summary())

"""
Output:
Model: "sequential"
_________________________________________________
_________________
Layer (type) Output Shape Param #
=================================================
=======================
embedding (Embedding) (None, None, 80) 80000
_________________________________________________
_________________
lstm (LSTM) (None, 12) 4464
=================================================
=======================
Total params: 84,464
Trainable params: 84,464
Non-trainable params: 0
"""
```

# Basic Dimensions of LSTM Layer




1. How many LSTM layers are there in this network?

— The network has one layer.

Don't get confused with multiple LSTM boxes, they represent the different timesteps, there is only one layer.

2. As shown what is the sequence length or the number of timesteps?

— The number of timesteps is 3. Look at the time indices.

3. If the input dimension i.e. x(t) is [18x1] and o(t) is [19x1] what are the dimensions of h(t), c(t)?

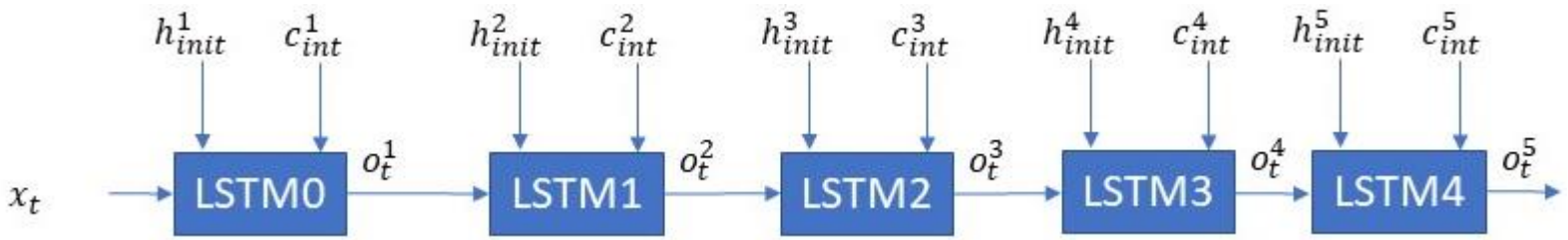
— h(t) and c(t) will be of the dimension [19x1]

4. What are the weights and biases dimensions

Wf=19x18
Wi=19x18
Wo=19x18
Wc=19x18

Uf=19x19
Ui=19x19
U0=19x19
Uc=19x19

# Basic Dimensions of LSTM Layer




**1. How many LSTM layers are there in this network?**

— The total number of LSTM layers is 5.

**2. As shown what is the sequence length or the number of timesteps?**

— This network has a sequence length of 1.

**3. If x(t) is [45x1] and h1(int) is [25x1] what are the dimensions of — c1(int) and o1(t) ?**

— c1(t) and o1(t) will have the same dimensions as h1(t) i.e [25x1]

**4. If x(t) is [4x1], h1(int) is [5x1] and o2(t) is of the size [4x1]. What is the size of the weight matrices for LSTM0 and LSTM1?**

— The weight matrix of an LSTM is [4*output_dim*(input_dim+output_dim+1)]. The input dim for LSTM0 is [4x1], output_dim for LSTM0 is [5x1]. The input to LSTM1 is the output of LSTM0 thus the input dim of LSTM1 is same as the output_dim of LSTM0 i.e. [5x1]. The output dim of LSTM1 is [4x1]. Thus LSTM0 is [4*5*(5+4+1)]=200 and LSTM1 is [4*4*(5+4+1)] = 160.

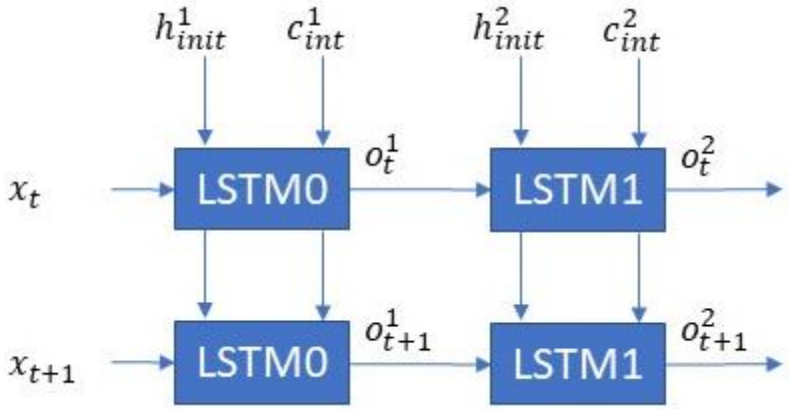
**5. If x(t) is [10x1], h1(int) is [7x1] what is the input dimension of LSTM1?**

— Look at the explanation above. We need to know the output dimensionality of LSTM0 before we can calculate this.

**6. If x(t) is [6x1], h1(int) is [4x1], o2(t) is [3x1], o3(t) is [5x1], o4(t) is [9x1] and o5(t) is [10x1] what is total weight size of the network?**

— The weight matrix of an LSTM for a single timestep is given as [4*output_dim*(input_dim+output_dim+1)]. Work by estimating the input ouput dimensionalities of a single layer.

# Basic Dimensions of LSTM Layer




**1. How many layers are in this network?**

— There are two layers

**2. What is the sequence length as shown in this network?**

— Each layer is unrolled 2 times.

**3. If x(t) is [80x1] and h1(int) is [10x1] what will be the dimensions of o(t), h1(t), c1(t), f(t), i(t). These are not shown in the figure, but you should be able to label this.**

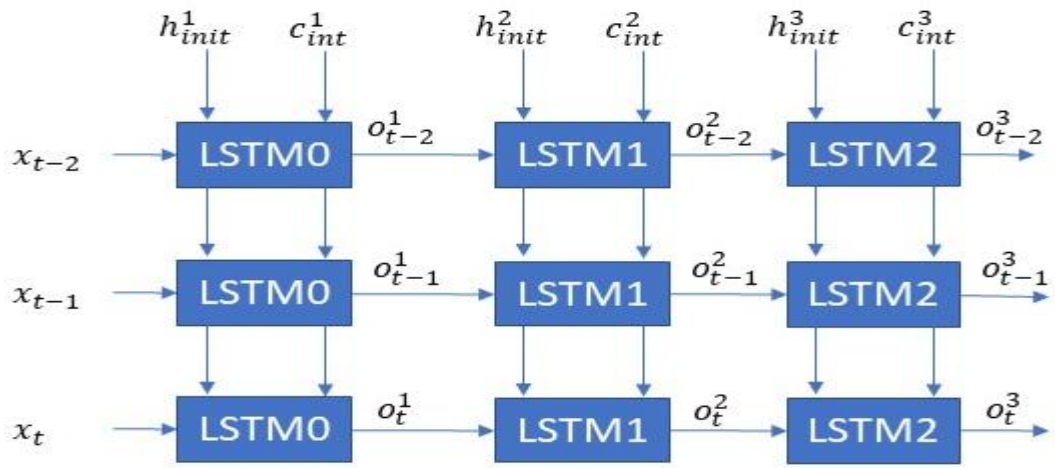
— o(t), h1(t), c1(t), f1(t), i1(t) will have the same dimension as h1(t) i.e. [10x1]

**3. If x(t+1) is [4x1], o1(t+1) is [5x1] and o2(t+1) is [6x1]. What are the size of the weight matrices for LSTM0 and LSTM1?**

— The weight matrix of an LSTM is given by 4*output_dim*(input_dim+output_dim+1). LSTM0 will be 4*5*(4+5+1) i.e. 200. LSTM2 will be 4*6*(5+5+1) = 264.

**4. If x(t+1) is [4x1], o1(t+1) is [5x1] and o2(t+1) is [6x1]. What is the total number of multiply and accumulate operations?**

# Basic Dimensions of LSTM Layer




**1. How many layers are there?**

**— There are 3 layers.**

**2. As shown how many timesteps is this network unrolled?**

**— Each layer is unrolled 3 times.**

**3. How many equations will be executed in all for this network?**

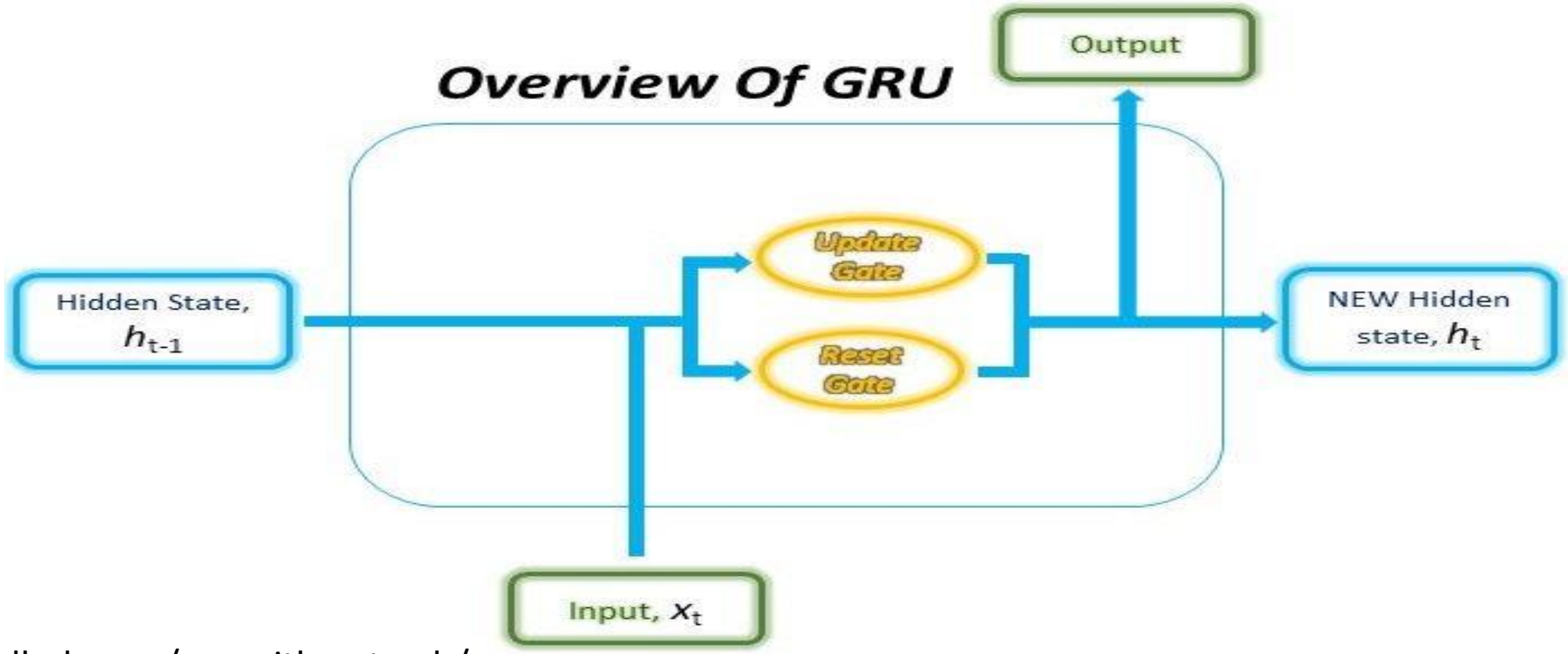
**— Each LSTM requires 6 equations , Then 6 equations/time step/LSTM. Therefore 6 equations * 3 LSTM Layers * 3 timesteps = 54 equations**

**4. If x(t) is [10x1] what other information would you need to estimate the weight matrix of LSTM1? What about LSTM2?**

**— The weight matrix of an LSTM is given by 4*output_dim*(input_dim+out_dim+1). Thus for each LSTM, we need both input_dim and the output_dim. The output of LSTM is the input of LSTM1. We have the input dimension of [10x1] so we need the output dimension or o1(int) dimension and the output dimensions of LSTM1 i.e. o2(t). Similarly for the LSTM2, we need to know, o1(t) and o2(t).**

# GRUs (Gated Recurrent Units)

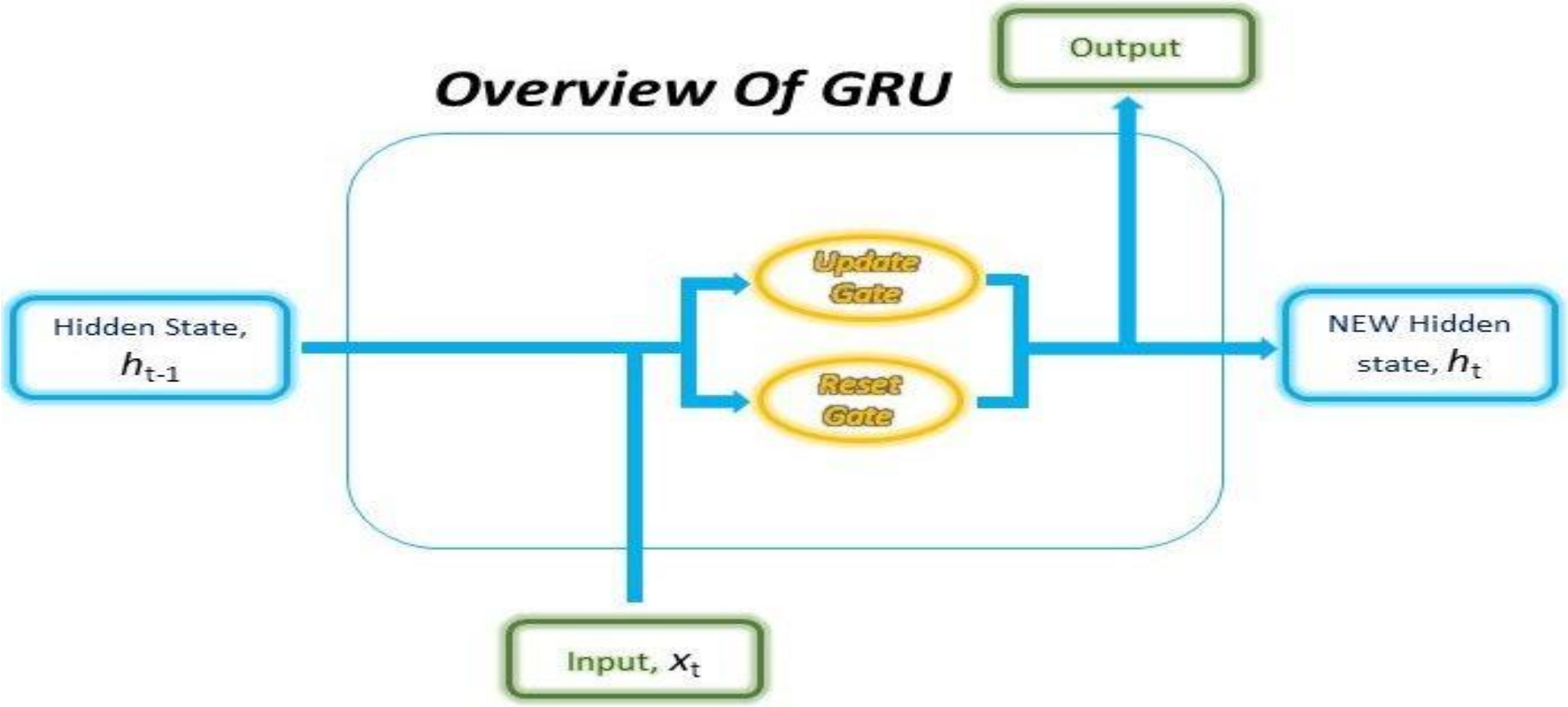
A Gated Recurrent Unit (GRU), as its name suggests, is a variant of the RNN architecture, and **uses gating mechanisms to control and manage the flow of information between cells in the neural network**. GRUs were introduced only in 2014 by Cho, et al. and can be considered a relatively new architecture, especially when compared to the widely-adopted LSTM, which was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber.




https://blog.floydhub.com/gru-with-pytorch/

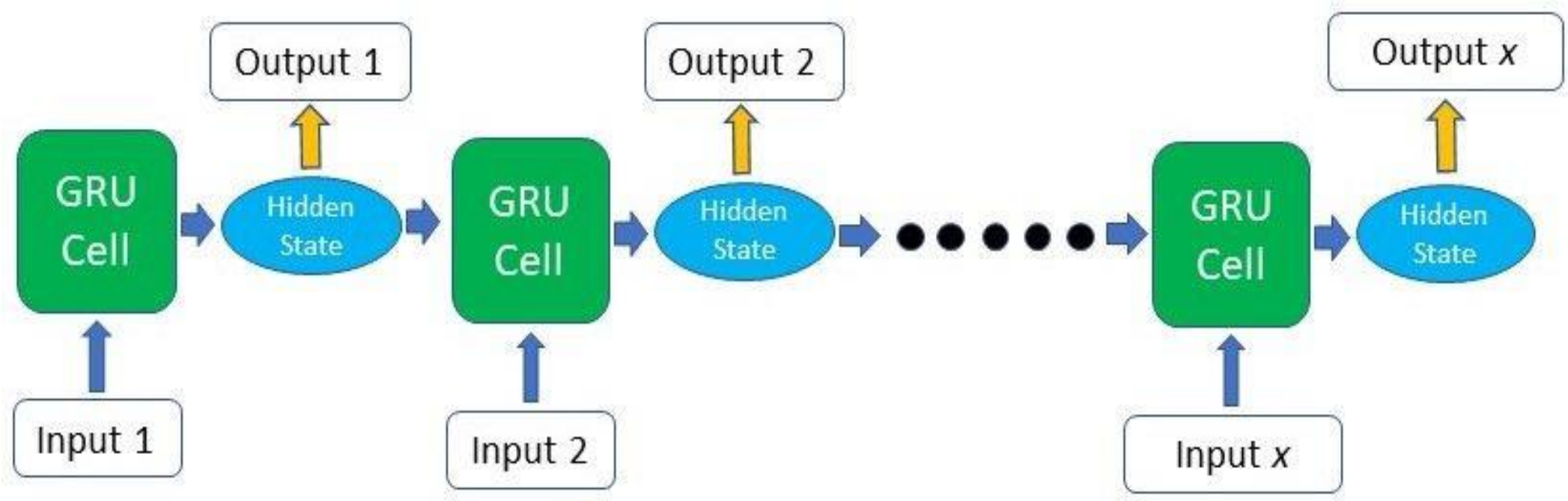
# GRUs

The structure of the GRU allows it to adaptively capture **dependencies from large sequences of data without discarding information from earlier parts of the sequence**. This is achieved through its gating units, similar to the ones in LSTMs, which solve the vanishing/exploding gradient problem of traditional RNNs.



# GRUs

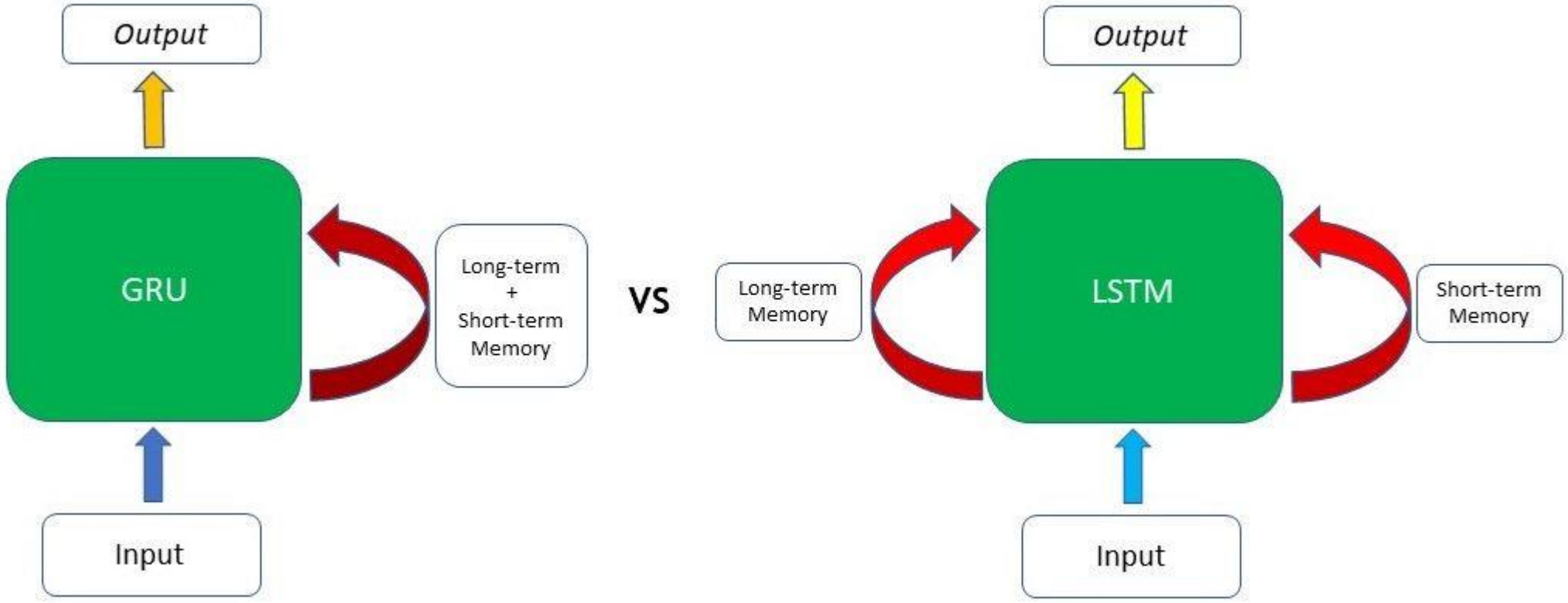
The structure of the GRU allows it to adaptively capture **dependencies from large sequences of data without discarding information from earlier parts of the sequence**. This is achieved through its gating units, similar to the ones in LSTMs, which solve the vanishing/exploding gradient problem of traditional RNNs.




GRUs follow the same flow as the typical RNN

# GRUs

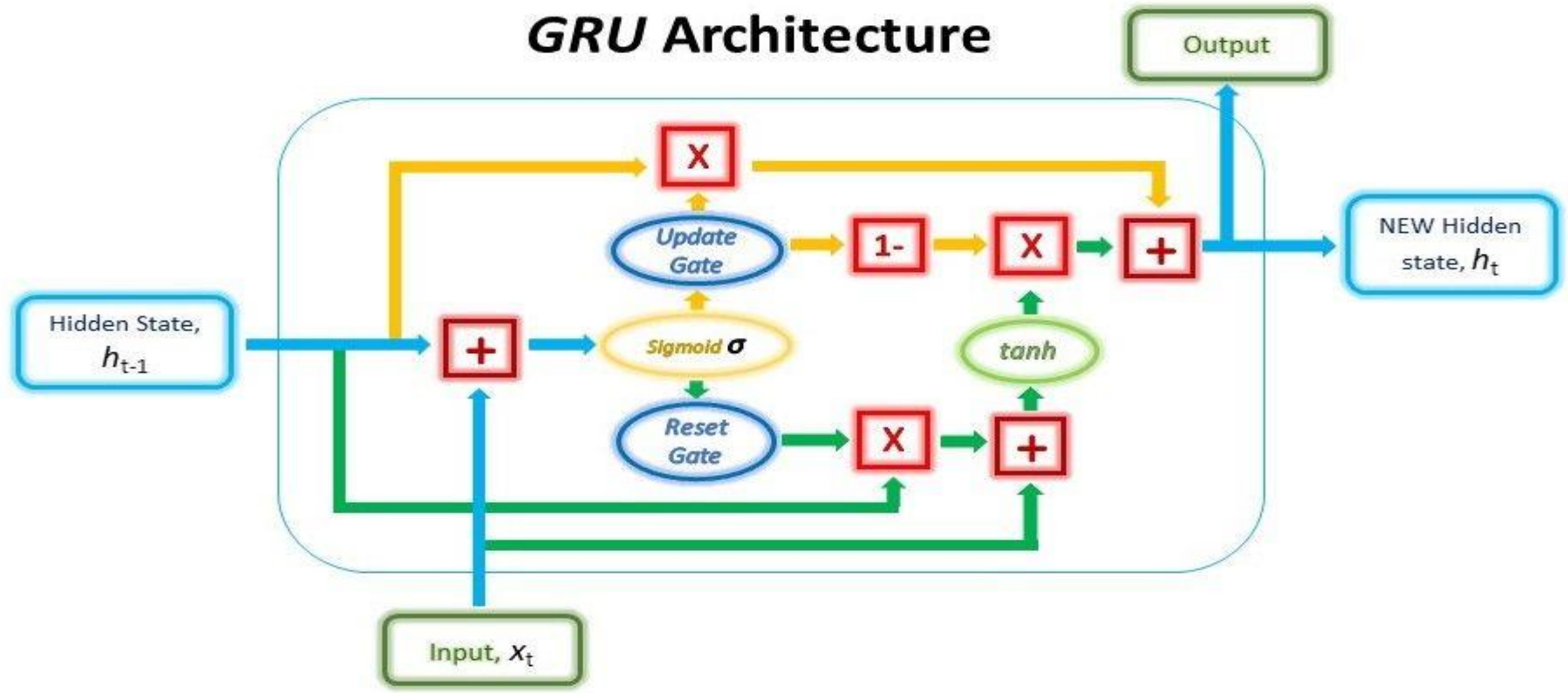
The ability of the GRU to hold on to long-term dependencies or memory stems from the computations **within the GRU cell to produce the hidden state**. While LSTMs have two different states passed between the cells — **the cell state and hidden state**, which carry the long and short-term memory, respectively




GRU vs LSTM

# GRUs

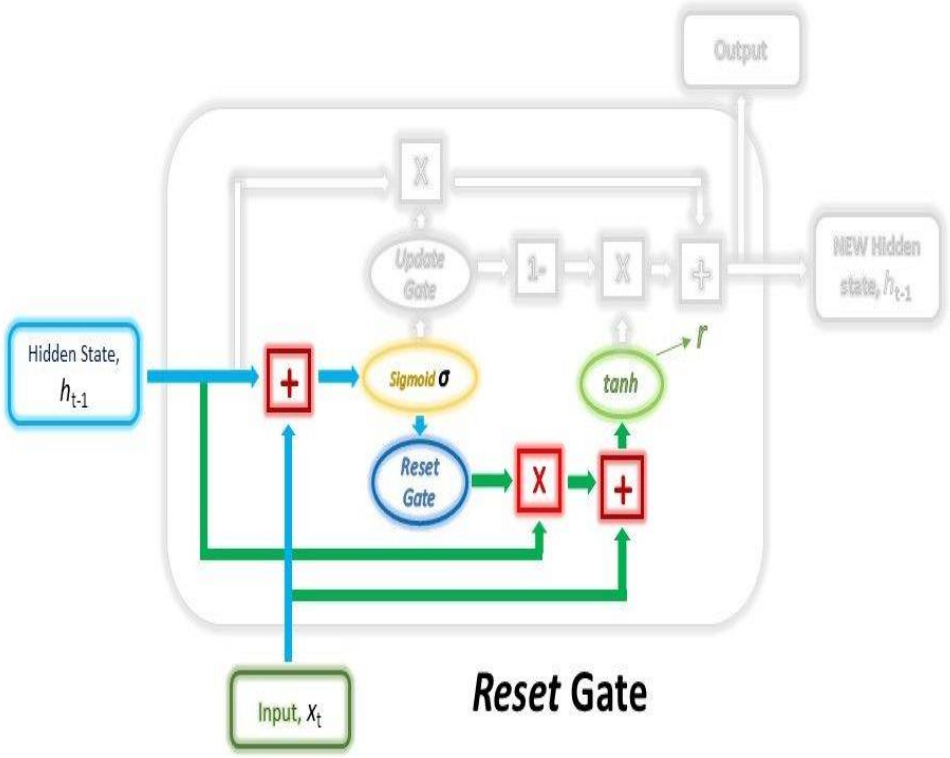
The GRU cell contains only two gates: **the Update gate and the Reset gate**. Just like the gates in LSTMs, these gates in the GRU are **trained** to selectively filter out any **irrelevant information while keeping what's useful**



# GRUs

**Reset Gate**

In the first step, this gate is derived and calculated using both the **hidden state** from the **previous time step** and the input data at the **current time step**.




$$gatereset=\sigma(Winputreset\cdot xt+Whiddenreset\cdot ht-1)$$

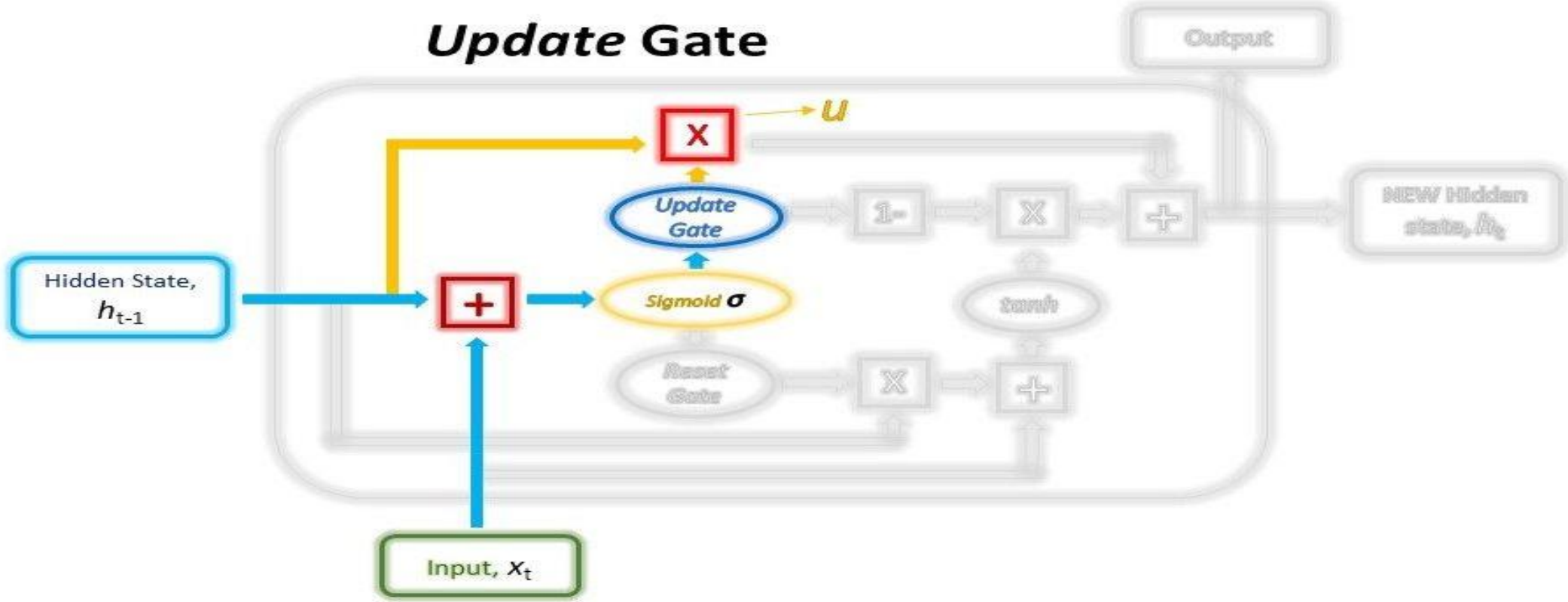
$$r=\tanh(gatereset\odot(Wh1\cdot ht-1)+Wx1\cdot xt)$$

The previous hidden state will first be multiplied by a **trainable weight** and will then undergo an **element-wise multiplication** (Hadamard product) with the **reset vector**. This operation will decide which information is to be kept from the previous time steps together with the new inputs. At the same time, the **current input** will also be multiplied by a **trainable weight** before being summed with the product of the **reset vector** and **previous hidden state** above. Lastly, a non-linear activation tanh function will be applied to the final result to obtain r in the equation above.

# GRUs

**Update Gate**

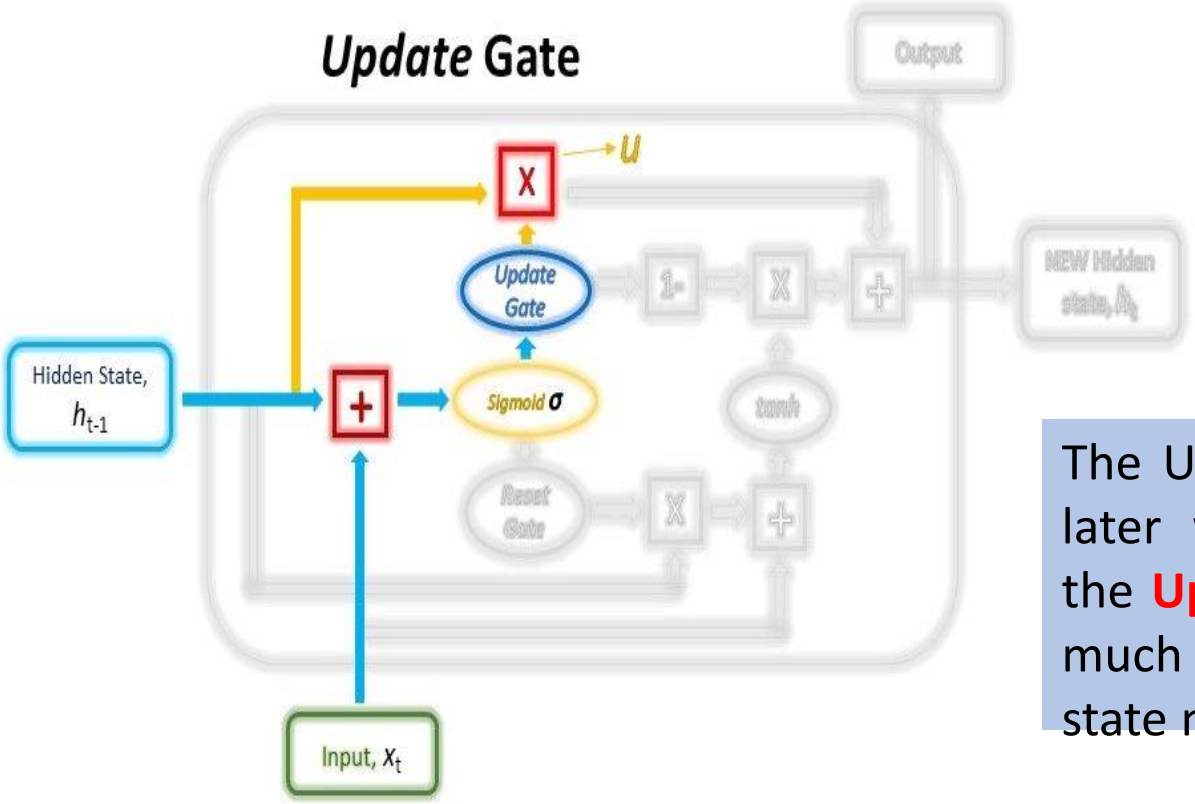
Next, we'll have to create the ***Update*** gate. Just like the***Reset*** gate, the gate is computed using the previous hidden state and current input data.



# GRUs

**Update Gate**

Next, we'll have to create the ***Update*** gate. Just like the***Reset*** gate, the gate is computed using the previous hidden state and current input data.




$$gate_{update} = \sigma(W_{input_{update}} \cdot x_t + W_{hidden_{update}} \cdot h_{t-1})$$

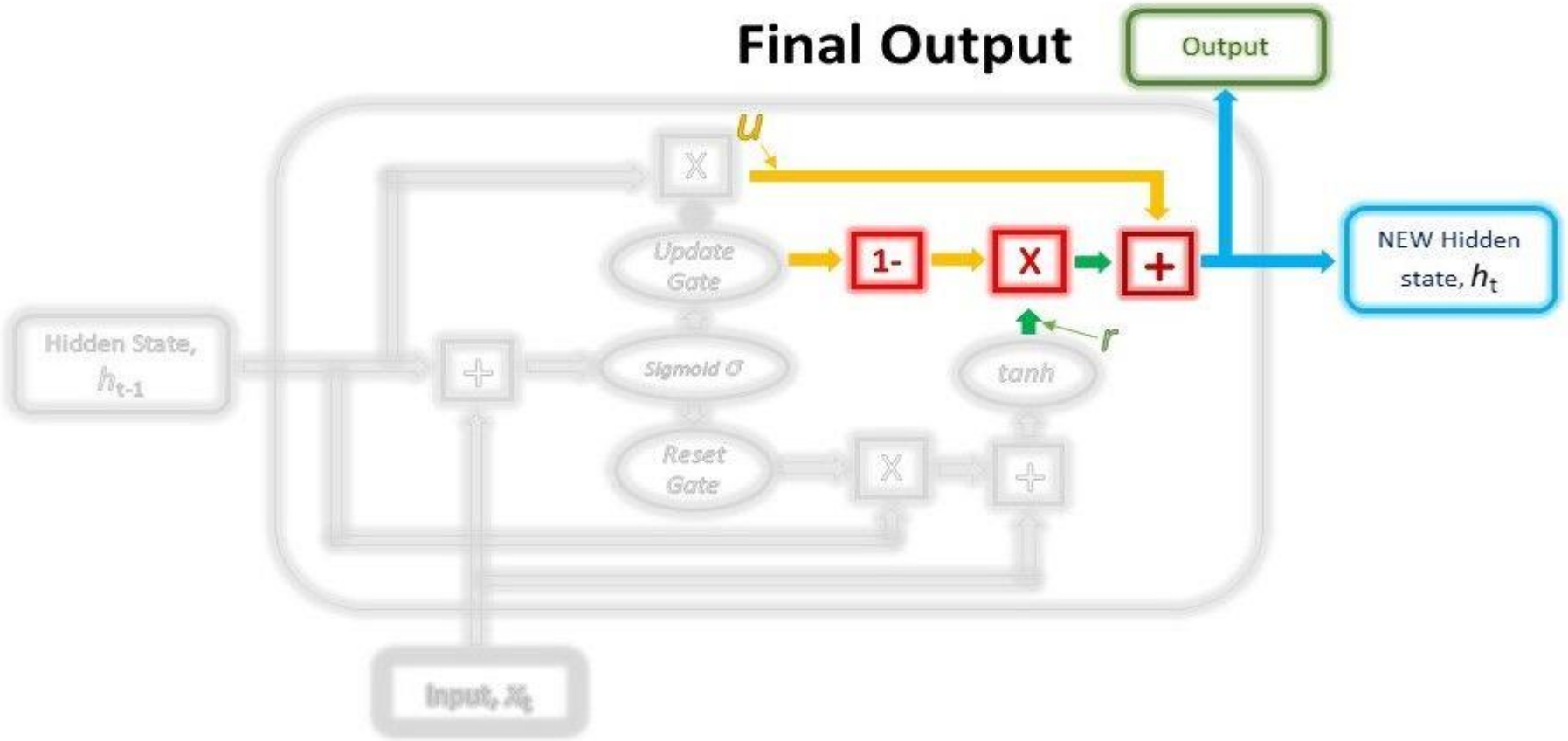
$$gate_{update} = \sigma(W_{input_{update}} \cdot x_t + W_{hidden_{update}} \cdot h_{t-1})$$

The Update vector will also be used in another operation later when obtaining our final output. The purpose of the **Update gate** here is to help the model determine how much of the past information stored in the previous hidden state needs to be retained for the future.

# GRUs

**Combining the outputs**

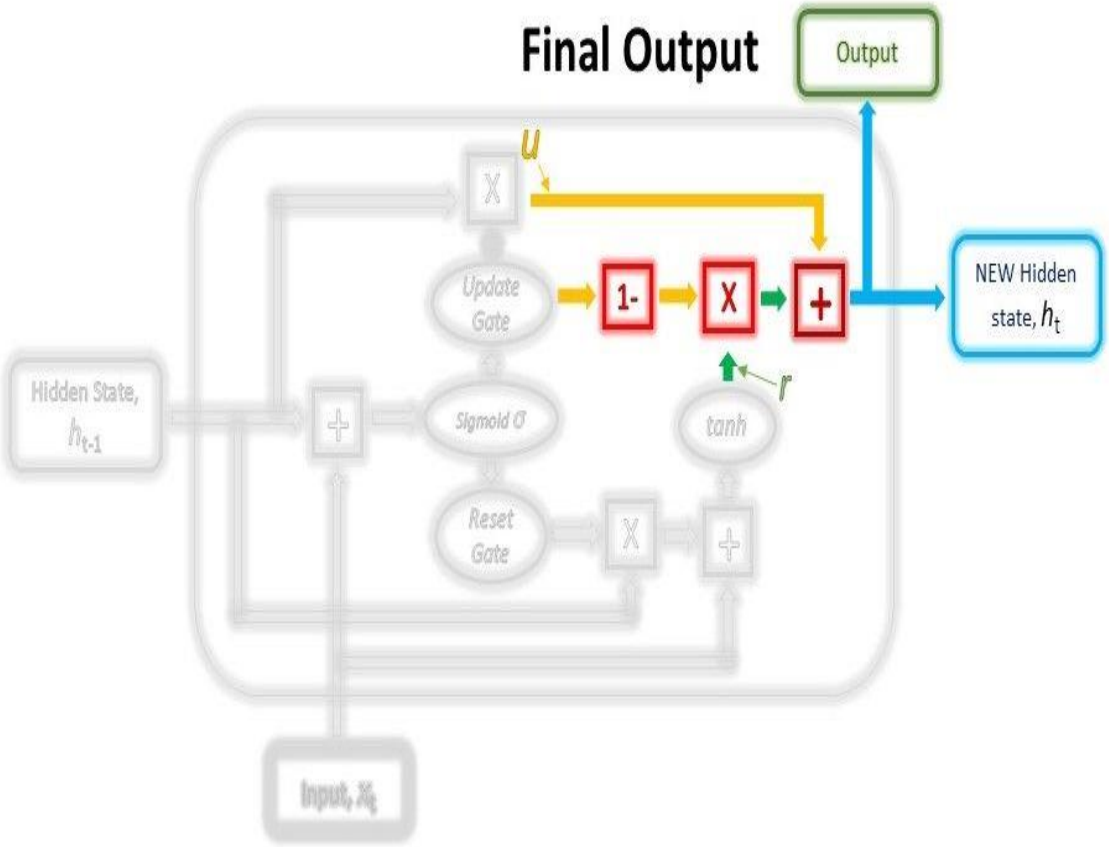
In the last step, we will be reusing the ***Update*** gate and obtaining the **updated hidden state.**



# GRUs

**Combining the outputs**

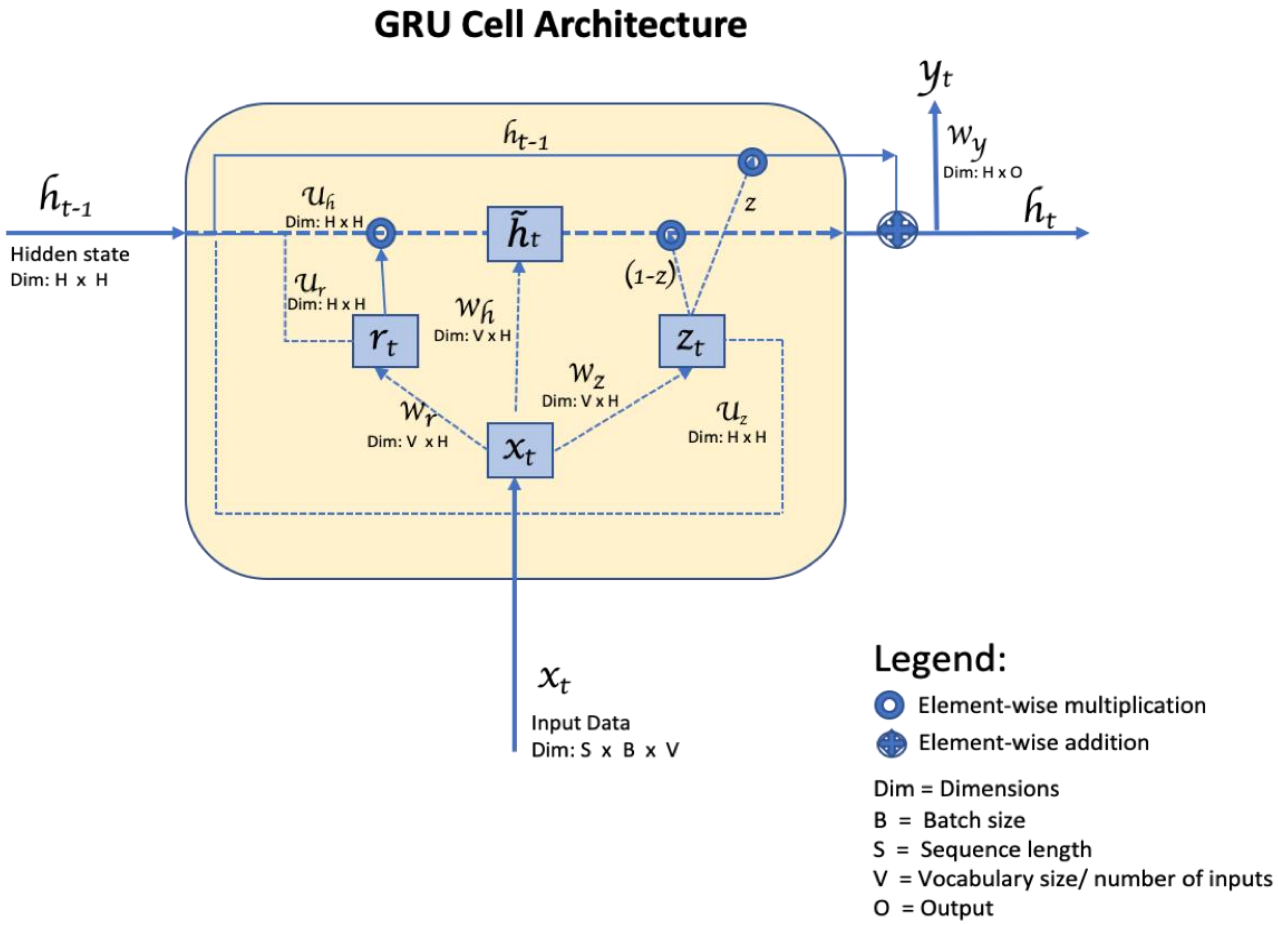
In the last step, we will be reusing the ***Update*** gate and obtaining the **updated hidden state.**




$h_t = r \odot (1 - \text{gate}_{update}) + u$

We can use this **new hidden state** as our output for that time step as well by passing it through a linear activation layer.

# GRUs




$$z = \sigma(W_z \cdot x_t + U_z \cdot h_{(t-1)} + b_z)$$
$$r = \sigma(W_r \cdot x_t + U_r \cdot h_{(t-1)} + b_r)$$
$$\tilde{h} = tanh(W_h \cdot x_t + r * U_h \cdot h_{(t-1)} + b_z)$$
$$h = z * h_{(t-1)} + (1 - z) * \tilde{h}$$

where z and r represent the update and reset gates respectively. While h_tilde and h represent the intermediate memory and output respectively.

https://towardsdatascience.com/gate-recurrent-units-explained-using-matrices-part-1-3c781469fc18

# GRUs

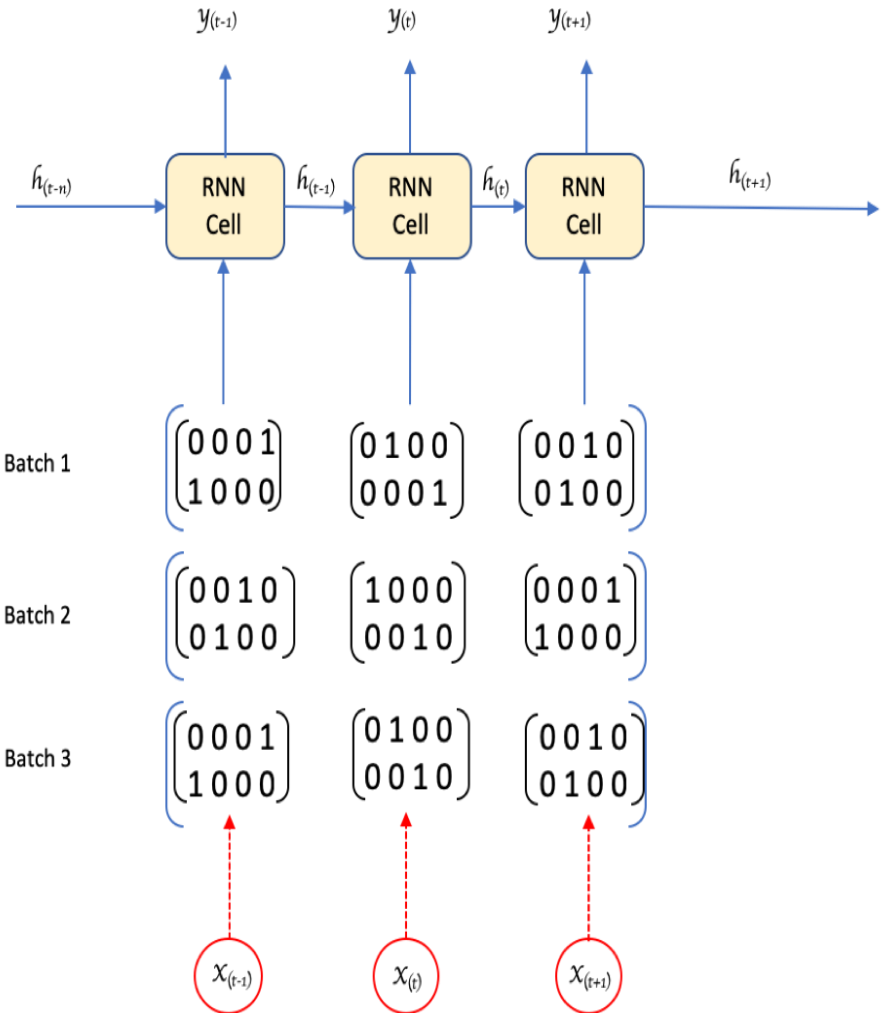
$$z = \sigma(W_z \cdot x_t + U_z \cdot h_{(t-1)} + b_z)$$
$$r = \sigma(W_r \cdot x_t + U_r \cdot h_{(t-1)} + b_r)$$
$$\tilde{h} = tanh(W_h \cdot x_t + r * U_h \cdot h_{(t-1)} + b_z)$$
$$h = z * h_{(t-1)} + (1 - z) * \tilde{h}$$

Let suppose:
x(t)=4x1 and
h(t)=h(t-1)=2x1

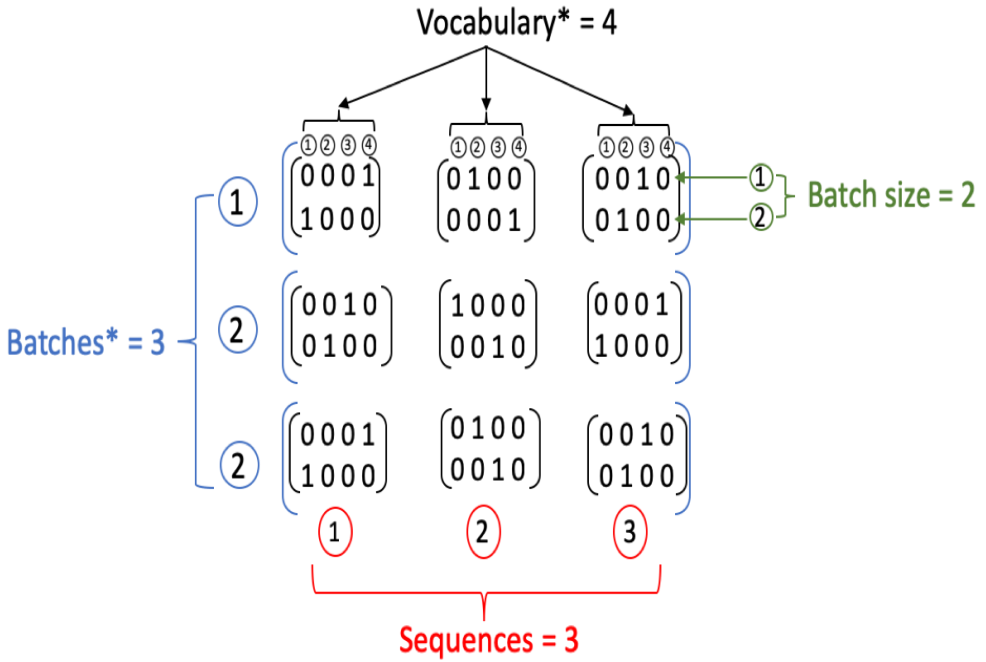
Wz=2x4
Wr=2x4
Wh=2x4
Uz=2x2
Ur=2x2
Uh=2x2
z=2x4x4x1+2x2x2x1+2x1=2x1

r=2x4x4x1+2x2x2x1+2x1=2x1
H_tilde=2x4x4x1+2x1x2x2x2x1+2x1=2x1
h=2x1*2x1+2x1*2x1=2x1
* Is element wise multiplication

# GRUs



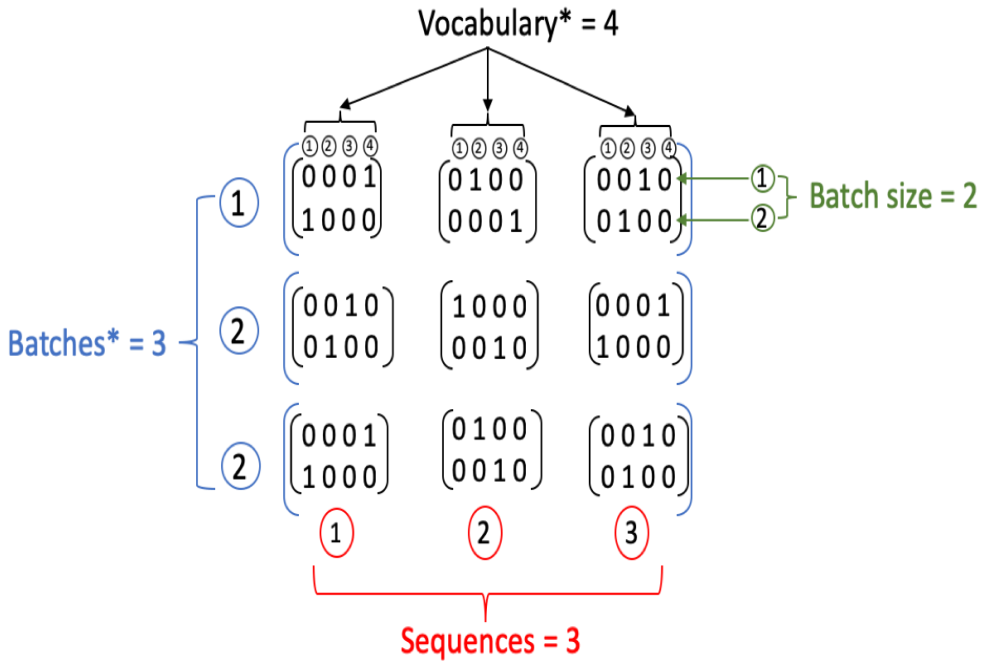

If we check the shape of this batch in python the interpreter will return a 3 x 3 x 2 x 4 dimensional array. Now how is this possible?




* Within each sequence we have batches of two, each having 4 values( essentially all unique characters in our dataset)
* Batches are not to be confused with batch sizes, batches are the number of subsets of sequences created, while the batch size is the number of observations within each sequence

# GRUs

If we check the shape of this batch in python the interpreter will return a 3 x 3 x 2 x 4 dimensional array. Now how is this possible?




Dimensions = (3 x 3 x 2 x 4)

3 = Number of batches

3 = Number of letters in each sequence

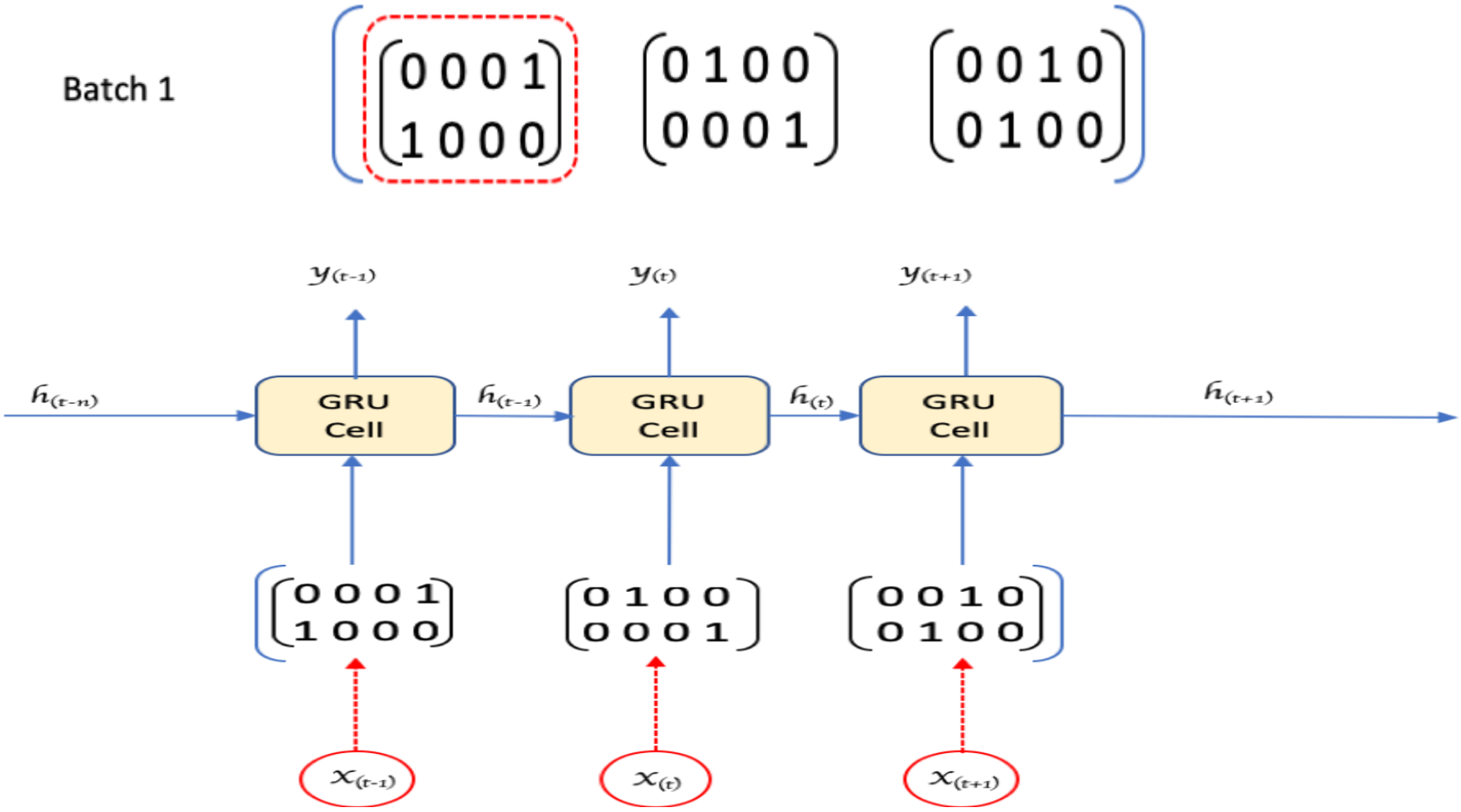
2 = Number of the sequences within each batch

4 = Size of vocabulary ( number of classes)

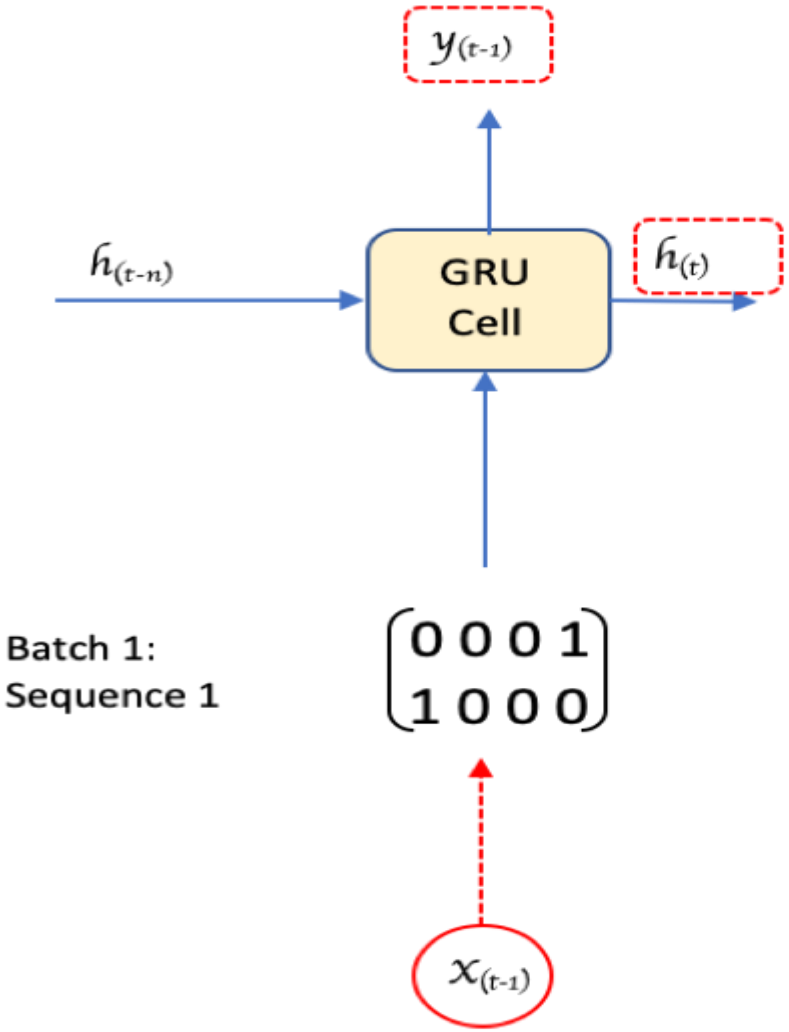
* Within each sequence we have batches of two, each having 4 values( essentially all unique characters in our dataset)

* Batches are not to be confused with batch sizes, batches are the number of subsets of sequences created, while the batch size is the number of observations within each sequence

# GRUs


Batch 1
0 0 0 1
1 0 0 0
0 1 0 0
0 0 0 1
0 0 1 0
0 1 0 0
$y_{(t-1)}$
$y_{(t)}$
$y_{(t+1)}$
$h_{(t-n)}$
GRU Cell
$h_{(t-1)}$
GRU Cell
$h_{(t)}$
GRU Cell
$h_{(t+1)}$
$x_{(t-1)}$
$x_{(t)}$
$x_{(t+1)}$

# GRUs




$$z_t = \sigma(\overset{①}{W_z x_t} + \overset{②}{U_z h_{t-1}} + \overset{③}{b_z})$$

Hidden size : 2

Vocabulary size : 4

| | |
|---|---|
| 0.6614 | 0.2669 |
| 0.0617 | 0.6213 |
| 0.4519 | -0.1661 |
| -1.5228. | 0.3817 |

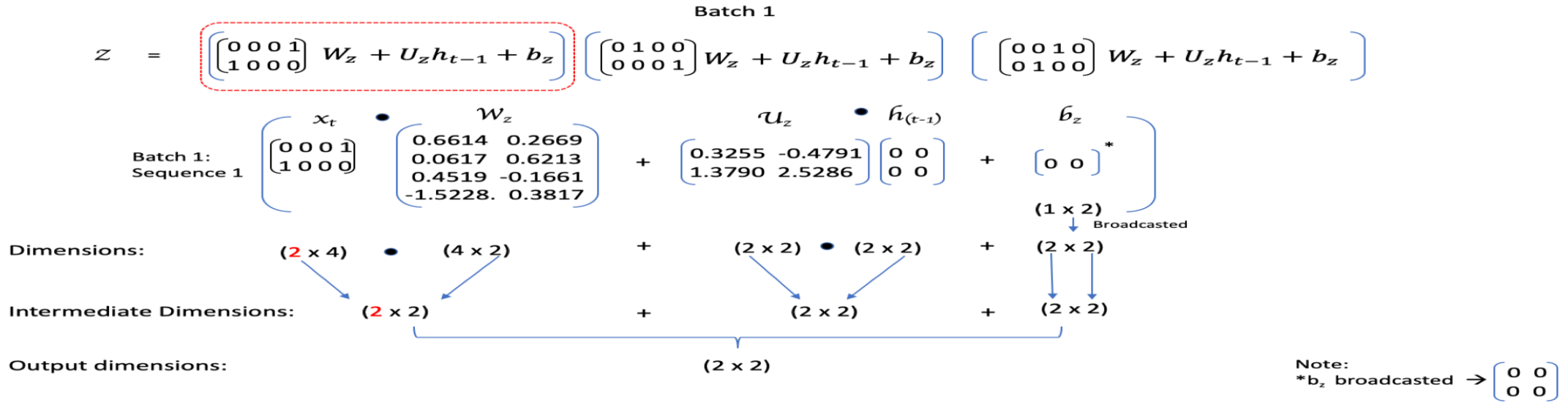
$W_z$

Calculated:

# GRUs

Batch 1

$$z = \left( \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} W_z + U_z h_{t-1} + b_z \right) \left( \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} W_z + U_z h_{t-1} + b_z \right) \left( \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} W_z + U_z h_{t-1} + b_z \right)$$



# GRUs

## Step1 Weights applied to the inputs

$$z_t = \sigma(\textcircled{1} W_z x_t + \textcircled{2} U_z h_{t-1} + \textcircled{3} b_z)$$

$$W_z x_i = \begin{pmatrix} 0001 \\ 1000 \end{pmatrix} \begin{matrix} 0.6614 & 0.2669 \\ 0.0617 & 0.6213 \\ 0.4519 & -0.1661 \\ -1.5228 & 0.3817 \end{matrix} \Rightarrow \begin{pmatrix} 0 \times 0.6614 + 0 \times 0.0617 + 0 \times 0.4519 + 1 \times -1.5228 & 0 \times 0.2699 + 0 \times 0.6213 + 0 \times 0.1661 + 1 \times 0.3817 \\ 1 \times 0.6614 + 0 \times 0.0617 + 0 \times 0.4519 + 0 \times -1.5228 & 1 \times 0.2699 + 0 \times 0.6213 + 0 \times 0.1661 + 1 \times 0.3817 \end{pmatrix} = \begin{matrix} -1.5228. & 0.3817 \\ 0.6614 & 0.2699 \end{matrix}$$

Non-zero terms:

## Step2 Hidden Weights

$$z_t = \sigma(\textcircled{1} W_z x_t + \textcircled{2} U_z h_{t-1} + \textcircled{3} b_z)$$

$$U_z h_{(t-1)} = \begin{matrix} 0.3255 & -0.4791 \\ 1.3790 & 2.5286 \end{matrix} \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \Rightarrow \begin{pmatrix} 0 \times 0.3255 + 0 \times -0.4791 & 0 \times 0.3255 + 0 \times -0.4791 \\ 0 \times 1.3790 + 0 \times 0.0617 & 0 \times 1.3790 + 0 \times 0.0617 \end{pmatrix} = \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

## Step3 Bias Vector

$$z_t = \sigma(\textcircled{1} W_z x_t + \textcircled{2} U_z h_{t-1} + \textcircled{3} b_z)$$

Broadcasted

$$b_z = \begin{matrix} 0 & 0 \end{matrix} \Rightarrow \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

# GRUs

$$W_z x_t \quad + \quad U_z h_{t-1} \quad + \quad b_z$$

$$z_{inner} = \begin{pmatrix} -1.5228 & 0.3817 \\ 0.6614 & 0.2699 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} -1.5228. & 0.3817 \\ 0.6614 & 0.2699 \end{pmatrix}$$

$$\sigma = \frac{1}{1+exp^{-input}}$$

# GRUs

**The reset gate: r**

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$r_t = \begin{pmatrix} 0.6041 & 0.5664 \\ 0.2635 & 0.3628 \end{pmatrix}$$

# GRUs

**Intermediate Memory: h_tilde**

$$\tilde{h} = tanh(\underset{①}{W_h x_i} + \underset{②}{r_h} * \underset{②}{U_h h_{t-1}} + \underset{③}{b_h})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$r_t = \begin{pmatrix} 0.6041 & 0.5664 \\ 0.2635 & 0.3628 \end{pmatrix}$$

$$\tilde{h} = tanh(\underset{①}{W_h x_t} + \underset{②}{r_t * U_h h_{t-1}} + \underset{③}{b_h})$$

$$r_t * U_r h_{t-1i} = \begin{pmatrix} 0.6041 & 0.5664 \\ 0.2635 & 0.3628 \end{pmatrix} * \begin{pmatrix} 0.4107 & -0.9880 \\ -0.9081 & 0.5423 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0.6041 & 0.5664 \\ 0.2635 & 0.3628 \end{pmatrix} * \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \times 0.6041 & 0 \times 0.5664 \\ 0 \times 0.2635 & 0 \times 0.3628 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$W_h x_t + r_t * U_h h_{t-1} + b_h$$

$$\tilde{h}_{inner} = \begin{pmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{pmatrix}$$

# GRUs

**Intermediate Memory: h_tilde**

$$\tilde{h} = tanh(\overset{①}{W_h x_i} + \overset{②}{r_h} * \overset{③}{U_h h_{t-1}} + b_h)$$

$$W_h x_t \quad + \quad r_t * U_h h_{t-1} \quad + \quad b_h$$

$$\tilde{h}_{inner} = \begin{pmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{pmatrix}$$

$$tanh = \frac{exp^{input} - exp^{-input}}{exp^{input} + exp^{-input}}$$

# GRUs

**Intermediate Memory: h_tilde**

$$W_h x_t \quad + \quad r_t \ * \ U_h h_{t-1} \quad + \quad b_h$$

$$\tilde{h}_{inner} = \begin{bmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{bmatrix}$$

$$tanh = \frac{exp^{input} - exp^{-input}}{exp^{input} + exp^{-input}}$$

$$\tilde{h} = tanh \begin{bmatrix} 1.5987 & -1.2770 \\ -0.4212 & -0.5107 \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{exp^{1.5987} - exp^{-1.5987}}{exp^{1.5987} + exp^{-1.5987}} & \frac{exp^{-(-1.2770)} - exp^{-(-1.2770}}{exp^{-(-1.2770)} + exp^{-(-1.2770)}} \\ \frac{exp^{-0.4212} - exp^{-(-0.4212)}}{exp^{(-0.4212)} + exp^{-(-0.4212)}} & \frac{exp^{-0.5107} - exp^{-(-0.5107)}}{exp^{(-0.5107)} + exp^{-(-0.5107)}} \end{bmatrix} = \begin{bmatrix} 0.9215 & -0.8557 \\ -0.3979 & -0.4705 \end{bmatrix}$$

# GRUs

**Output hidden layer at time step t: h_(t-1)**

$$h_t = \overset{①}{z_t} * h_{(t-1)} + (1 - \overset{②}{z_t}) * \tilde{h}$$

$$h_t = \begin{matrix} z_t \\ \begin{pmatrix} 0.1791 & 0.5943 \\ 0.6596 & 0.5663 \end{pmatrix} \end{matrix} * \begin{matrix} h_{(t-1)} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{matrix} + \begin{matrix} (1 - z_t). \\ \left(1^{*} - \begin{pmatrix} 0.1791 & 0.5943 \\ 0.6596 & 0.5663 \end{pmatrix}\right) \end{matrix} * \begin{matrix} \tilde{h} \\ \begin{pmatrix} 0.9215 & -0.8557 \\ -0.3979 & -0.4705 \end{pmatrix} \end{matrix}$$

* Broadcasted

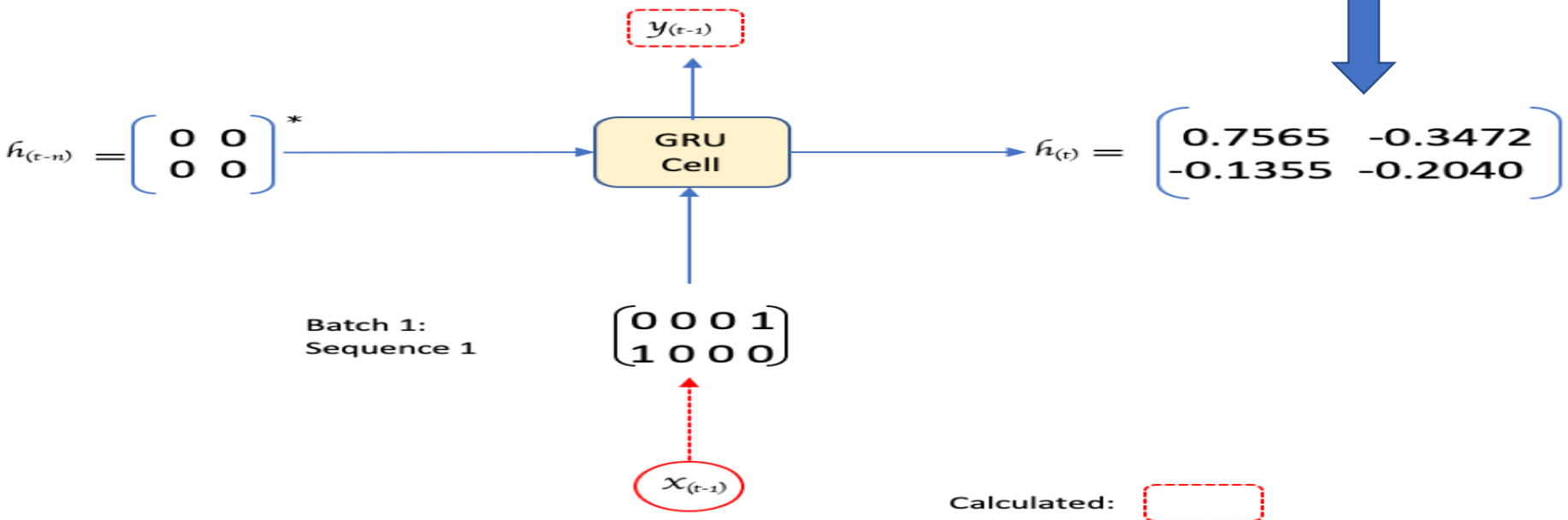
$$z_t * h_{(t-1)} + (1 - z_t) * \tilde{h}$$

$$h_t = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0.7565 & -0.3472 \\ -0.1355 & -0.2040 \end{pmatrix} = \begin{pmatrix} 0.7565 & -0.3472 \\ -0.1355 & -0.2040 \end{pmatrix}$$

# GRUs

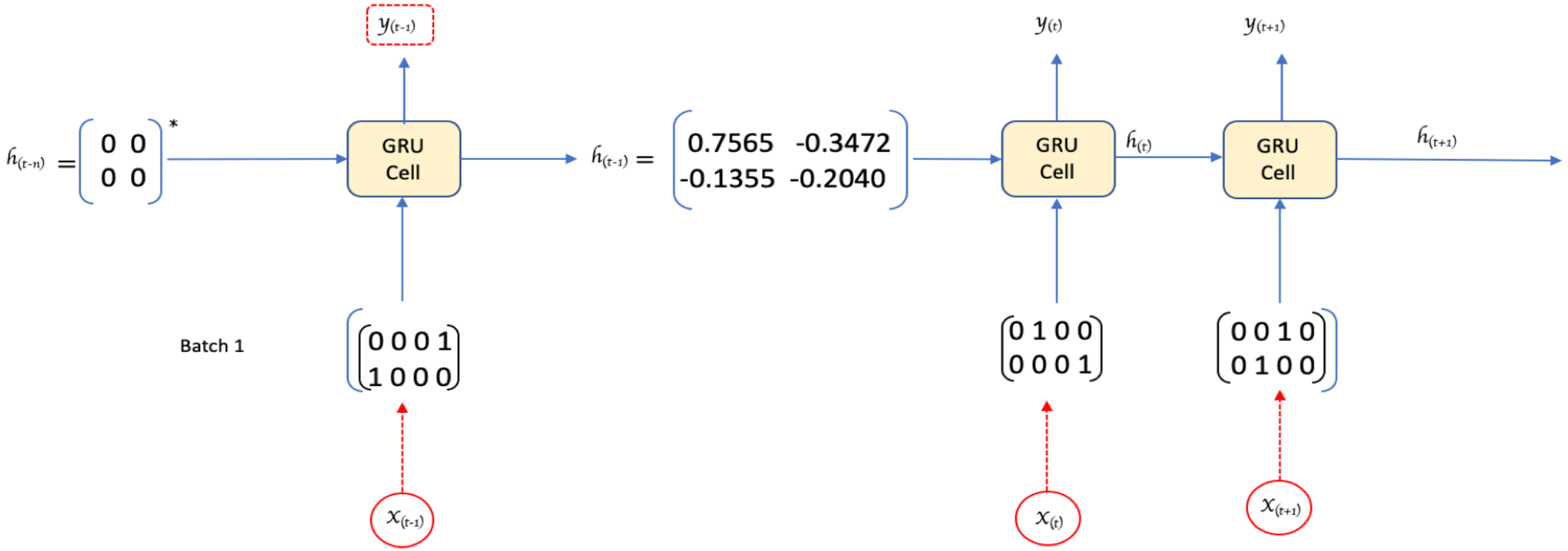
$$z_t * h_{(t-1)} + (1 - z_t) * \tilde{h}$$

$$h_t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0.7565 & -0.3472 \\ -0.1355 & -0.2040 \end{bmatrix} = \begin{bmatrix} 0.7565 & -0.3472 \\ -0.1355 & -0.2040 \end{bmatrix}$$




* Intialized to zeros to begin the training

# GRUs

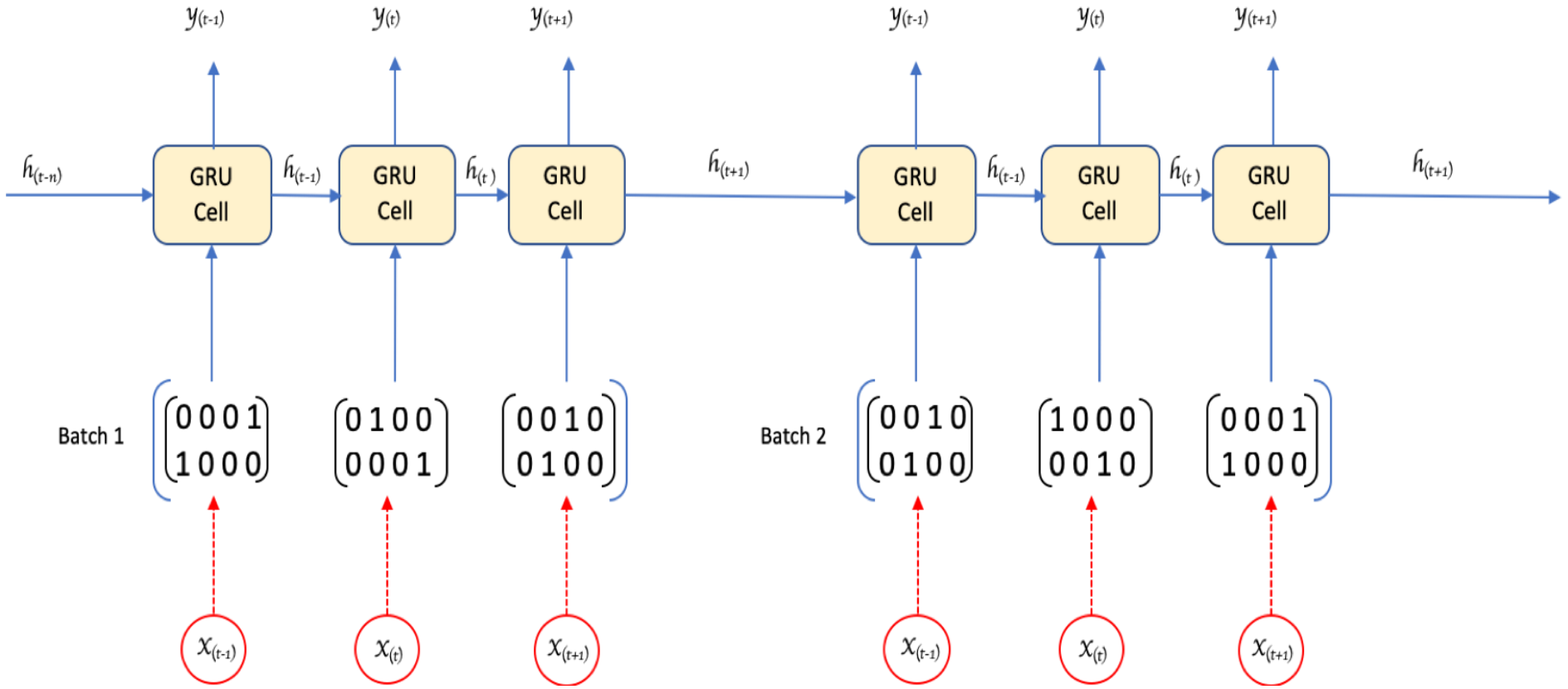



* Intialized to zeros to begin the training

To be Calculated:

# GRUs

**What will be the hidden state for the second batch?**



# GRUs

**Step 3: Calculate the out predictions for each time step**

$$\text{Linear} = W_y h_{(t-1)} + b_y$$

$$\text{Linear} = \begin{pmatrix} 0.7565 & -0.3472 \\ -0.1355 & -0.2040 \end{pmatrix} \begin{pmatrix} 0.8310 & -0.2477 & -0.8029 & 0.2366 \\ 0.2857 & 0.6898 & -0.6331 & 0.8795 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^{*} = \begin{pmatrix} 0.5295 & -0.4269 & -0.3876 & -0.1264 \\ -0.1709 & -0.1072 & 0.2379 & -0.2115 \end{pmatrix}$$

Note:
*$b_z$ broadcasted from $\begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \Longrightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

# GRUs

**Step 3: Calculate the out predictions for each time step**

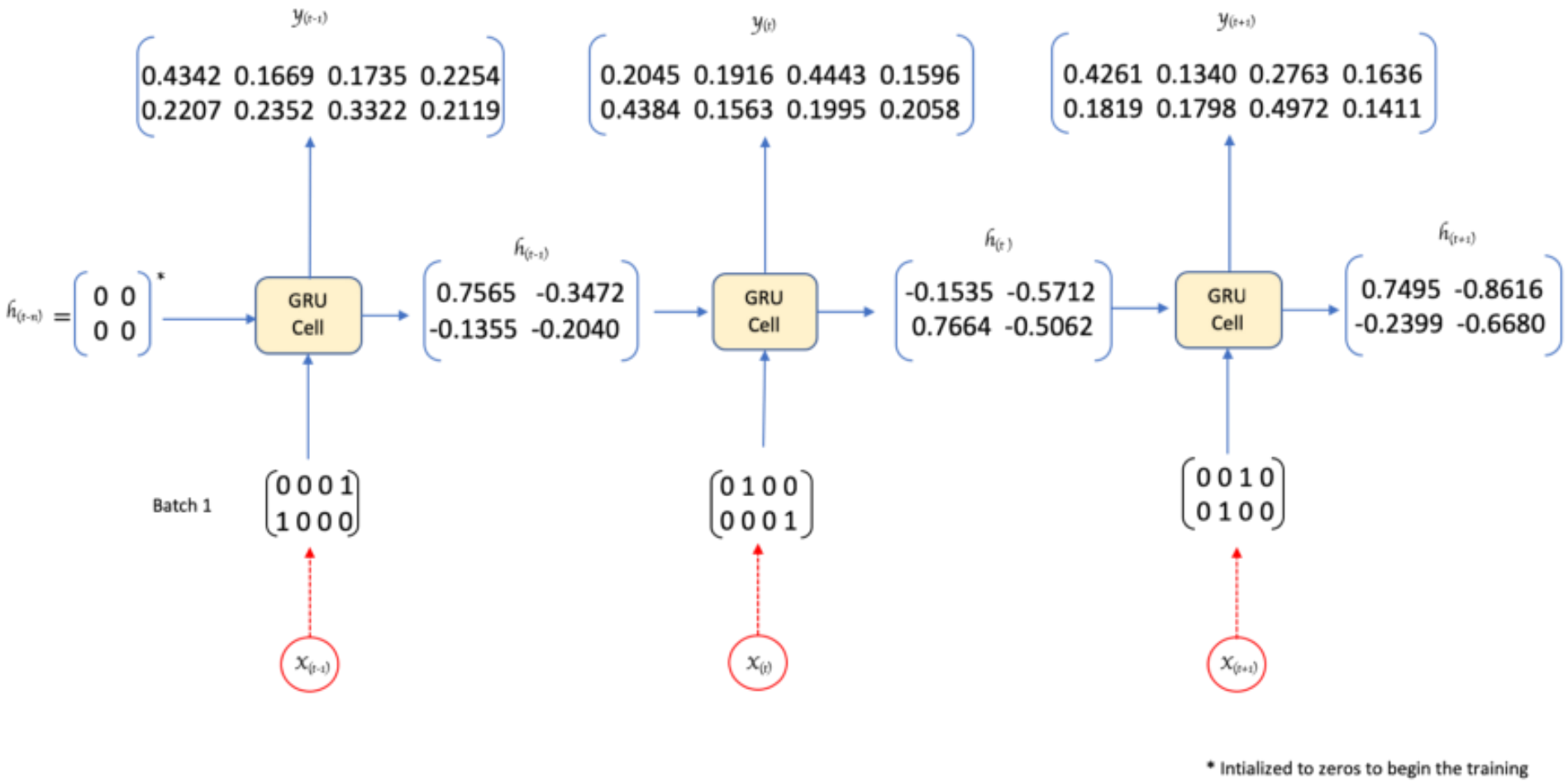
$$\exp(y_{linear} - y_{linear\_max}) = \exp\begin{pmatrix} 0.5295 & -0.4269 & -0.3876 & -0.1264 \\ -0.1709 & -0.1072 & 0.2379 & -0.2115 \end{pmatrix} - \boxed{0.9021} = \exp\begin{pmatrix} 0.5295 - \boxed{0.9021} & -0.4269 - \boxed{0.9021} & -0.3876 - \boxed{0.9021} & -0.1264 - \boxed{0.9021} \\ -0.1709 - \boxed{0.9021} & -0.1072 - \boxed{0.9021} & 0.2379 - \boxed{0.9021} & -0.2115 - \boxed{0.9021} \end{pmatrix}$$

$$= \begin{pmatrix} exp^{-0.3727} & exp^{-1.3290} & exp^{-1.2898} & exp^{-1.0285} \\ exp^{-1.0730} & exp^{-1.0093} & exp^{-0.6642} & exp^{-1.1136} \end{pmatrix}$$

$$= \begin{pmatrix} 0.6889 & 0.2647 & 0.2753 & 0.3575 \\ 0.3420 & 0.3645 & 0.5147 & 0.3284 \end{pmatrix}$$

$$\text{Softmax} = \begin{pmatrix} \frac{0.6889}{1.5865} & \frac{0.2647}{1.5865} & \frac{0.2753}{1.5865} & \frac{0.3575}{1.5865} \\ \frac{0.3420}{1.5495} & \frac{0.3645}{1.5495} & \frac{0.5147}{1.5495} & \frac{0.3284}{1.5495} \end{pmatrix} = \begin{pmatrix} 0.4342 & 0.1669 & 0.1735 & 0.2254 \\ 0.2207 & 0.2352 & 0.3322 & 0.2119 \end{pmatrix}$$

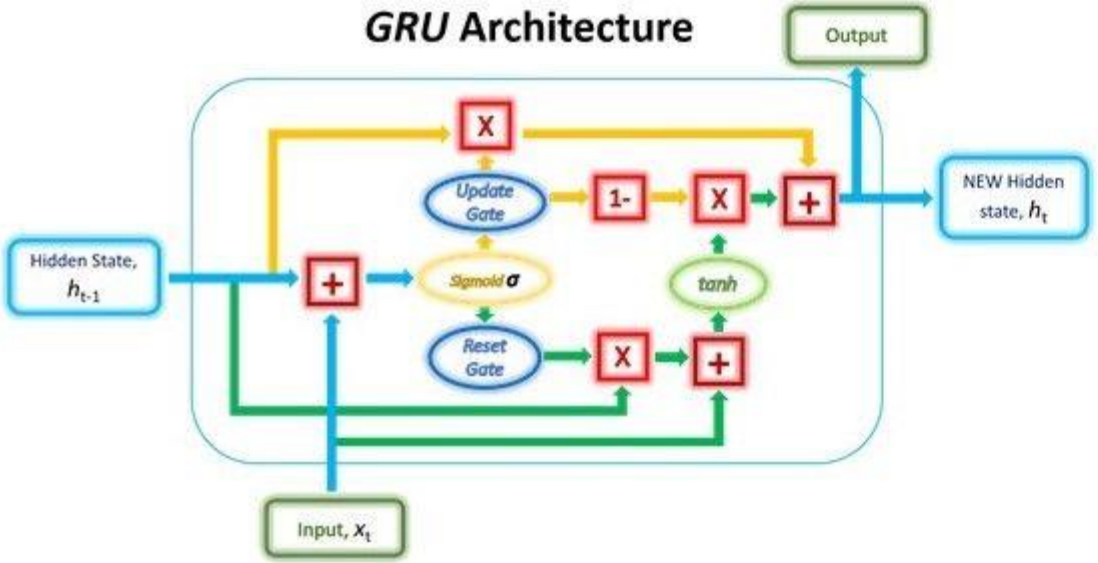
# GRUs

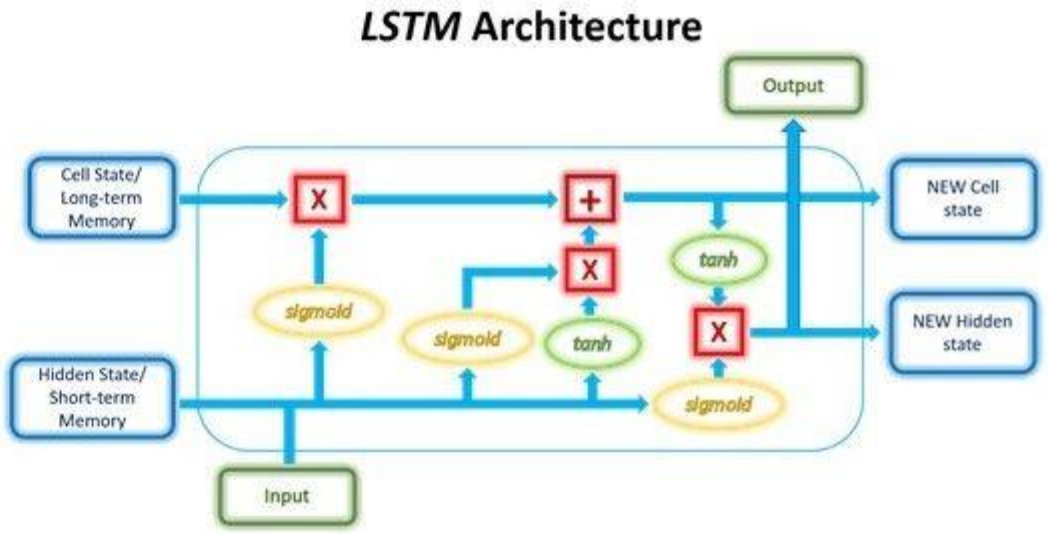



* Intialized to zeros to begin the training

# GRUs

While both GRUs and LSTMs contain gates, the main difference between these two structures lies in the number of **gates and their specific roles**. The role of the **Update gate** in the **GRU** is very similar to the **Input and Forget gates** in the **LSTM**. However, **the control of new memory content added to the network differs between these two.**

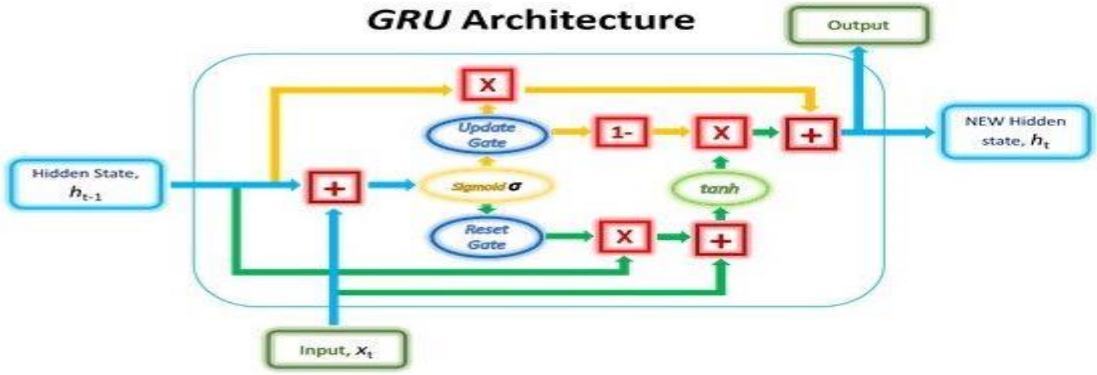


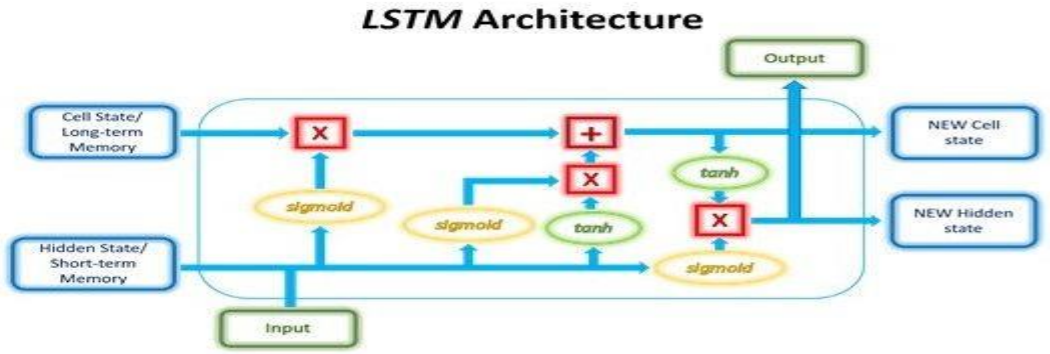



# GRUs

In the LSTM, while the **Forget gate** determines which part of the previous cell state to retain, the **Input gate** determines the amount of **new memory to be added**. **These two gates are independent of each other**, meaning that the amount of new information added through the Input gate is completely independent of the information retained through the Forget gate.

As for the GRU, the **Update gate** is responsible for determining which information from the previous **memory to retain** and is also responsible for controlling the **new memory to be added**. This means that the **retention of previous memory and addition of new information to the memory** in the GRU is NOT independent.

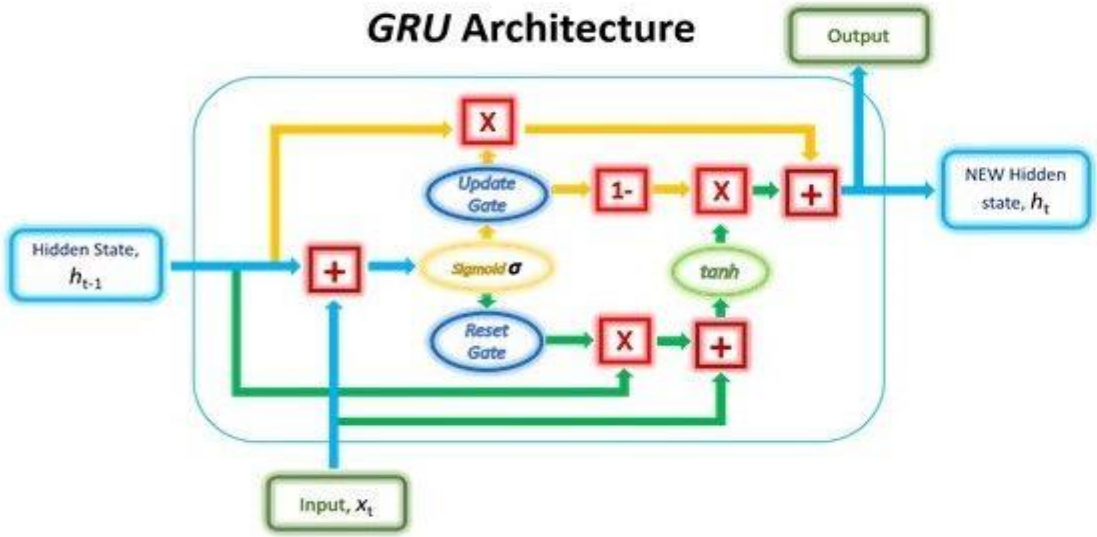


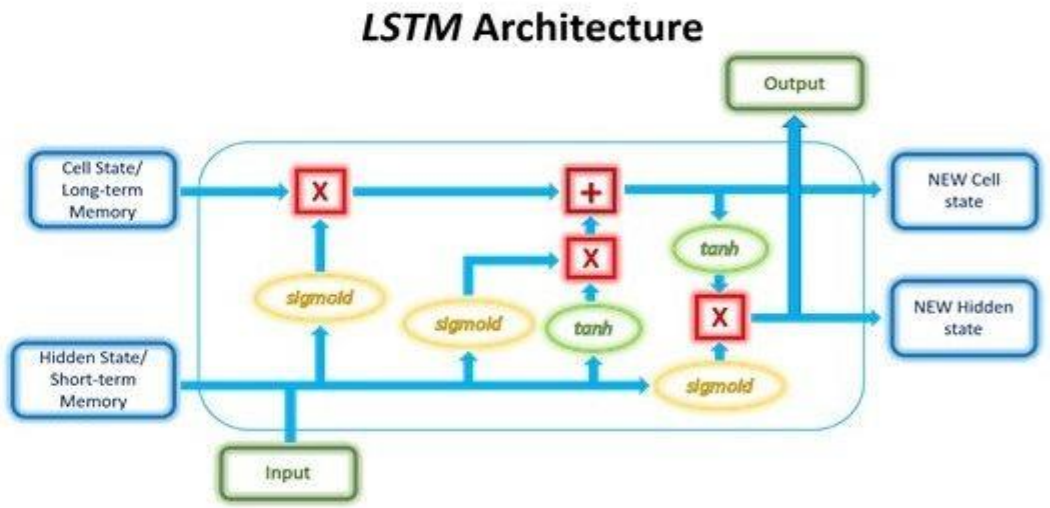



# GRUs

GRUs are faster to train as compared to LSTMs due to the **fewer number of weights and parameters to update during training**. This can be attributed to the **fewer number of gates** in the **GRU cell (two gates)** as compared to the **LSTM's three gates**.

The accuracy of a model, whether it is measured by the **margin of error or proportion of correct classifications**, is usually the main factor when deciding which type of model to use for a task



# LSTM used in EEG Applications



## Affective EEG-Based Person Identification Using the Deep Learning Approach

Theerawit Wilaiprasitporn, *Member, IEEE,* Apiwat Ditthapron, Karis Matchaparn, Tanaboon Tongbuasirilai, Nannapas Banluesombatkul and Ekapol Chuangsuwanich

**Abstract**—Electroencephalography (EEG) is another mode for performing Person Identification (PI). Due to the nature of the EEG signals, EEG-based PI is typically done while the person is performing some kind of mental task, such as motor control. However, few works have considered EEG-based PI while the person is in different mental states (affective EEG). The aim of this paper is to improve the performance of affective EEG-based PI using a deep learning approach. A cascade of deep learning architectures is proposed, using a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs are used to handle the spatial information from the EEG while RNNs extract the temporal information. Two kinds RNNs are evaluated, namely Long Short-Term Memory (CNN-LSTM) and Gated Recurrent Unit (CNN-GRU). The proposed method is evaluated on the state-of-the-art affective dataset DEAP. The results indicate that CNN-GRU and CNN-LSTM can perform PI from different affective states and reach up to 99.90–100% mean Correct Recognition Rate *CRR*, significantly outperforming a support vector machine (SVM) baseline system that uses power spectral density (PSD) features. Notably, the 100% mean *CRR* comes from only 40 subjects in DEAP dataset. To reduce the number of EEG electrodes from thirty-two to five for more practical applications, the frontal region gives the best results reaching up to 99.17% mean *CRR* (from CNN-GRU). Amongst the two deep learning models, we find CNN-GRU to slightly outperform CNN-LSTM, while having faster training time.

**Index Terms**—Electroencephalography, Personal identification, Biometrics, Deep learning, Affective computing, Convolutional neural networks, Long short-term memory, Recurrent neural networks

# LSTM used in EEG Applications

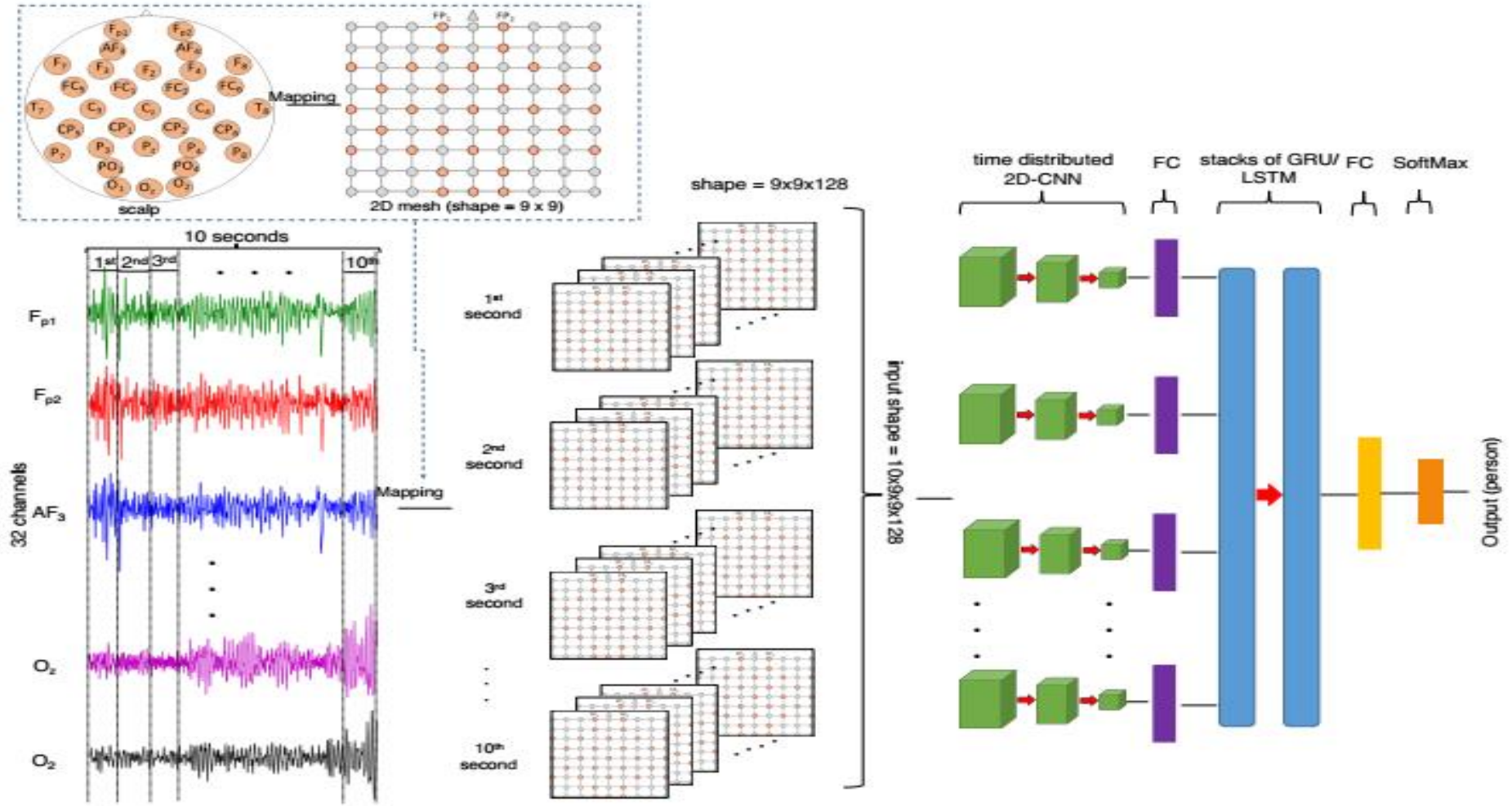



Fig. 2. Implementation of the cascade CNN-GRU/LSTM model according to EEG data. Meshing is the first step in converting multi-channel EEG signals into sequences of 2D images. The 2D mesh time series is passed through the cascade of CNN and recurrent layers for training, validation, and testing.

# LSTM used in EEG Applications

Computers in Biology and Medicine 99 (2018) 24–37

Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: www.elsevier.com/locate/compbiomed

## A Long Short-Term Memory deep learning network for the prediction of epileptic seizures using EEG signals

Kostas M. Tsiouris [a,b], Vasileios C. Pezoulas [b], Michalis Zervakis [c], Spiros Konitsiotis [d], Dimitrios D. Koutsouris [a], Dimitrios I. Fotiadis [b,e,*]

[a] Biomedical Engineering Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, GR15773, Athens, Greece
[b] Unit of Medical Technology and Intelligent Information Systems, Dept. of Material Science and Engineering, University of Ioannina, GR45110, Ioannina, Greece
[c] Digital Image and Signal Processing Laboratory, School of Electrical and Computer Engineering Technical University of Crete, Chania, Greece
[d] Dept. of Neurology, Medical School, University of Ioannina, GR45110, Ioannina, Greece
[e] Dept. of Biomedical Research, Institute of Molecular Biology and Biotechnology, FORTH, GR45110, Ioannina, Greece

ARTICLE INFO

*Keywords:*
EEG
Epilepsy
Seizure prediction
LSTM model
Deep learning

ABSTRACT

The electroencephalogram (EEG) is the most prominent means to study epilepsy and capture changes in electrical brain activity that could declare an imminent seizure. In this work, Long Short-Term Memory (LSTM) networks are introduced in epileptic seizure prediction using EEG signals, expanding the use of deep learning algorithms with convolutional neural networks (CNN). A pre-analysis is initially performed to find the optimal architecture of the LSTM network by testing several modules and layers of memory units. Based on these results, a two-layer LSTM network is selected to evaluate seizure prediction performance using four different lengths of preictal windows, ranging from 15 min to 2 h. The LSTM model exploits a wide range of features extracted prior to classification, including time and frequency domain features, between EEG channels cross-correlation and graph theoretic features. The evaluation is performed using long-term EEG recordings from the open CHB-MIT Scalp EEG database, suggest that the proposed methodology is able to predict all 185 seizures, providing high rates of seizure prediction sensitivity and low false prediction rates (FPR) of 0.11–0.02 false alarms per hour, depending on the duration of the preictal window. The proposed LSTM-based methodology delivers a significant increase in seizure prediction performance compared to both traditional machine learning techniques and convolutional neural networks that have been previously evaluated in the literature.

# LSTM used in EEG Applications

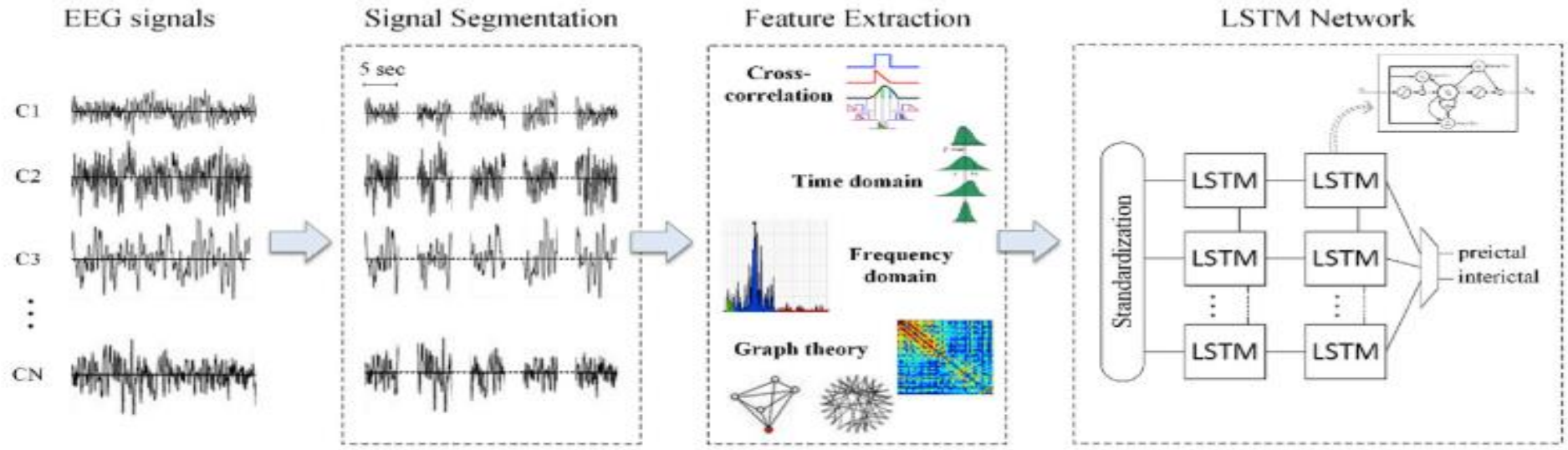



**Fig. 1.** Schematic representation of the proposed seizure prediction methodology. Features are extracted from 5-s long EEG segments and are fed into the LSTM network for classification.

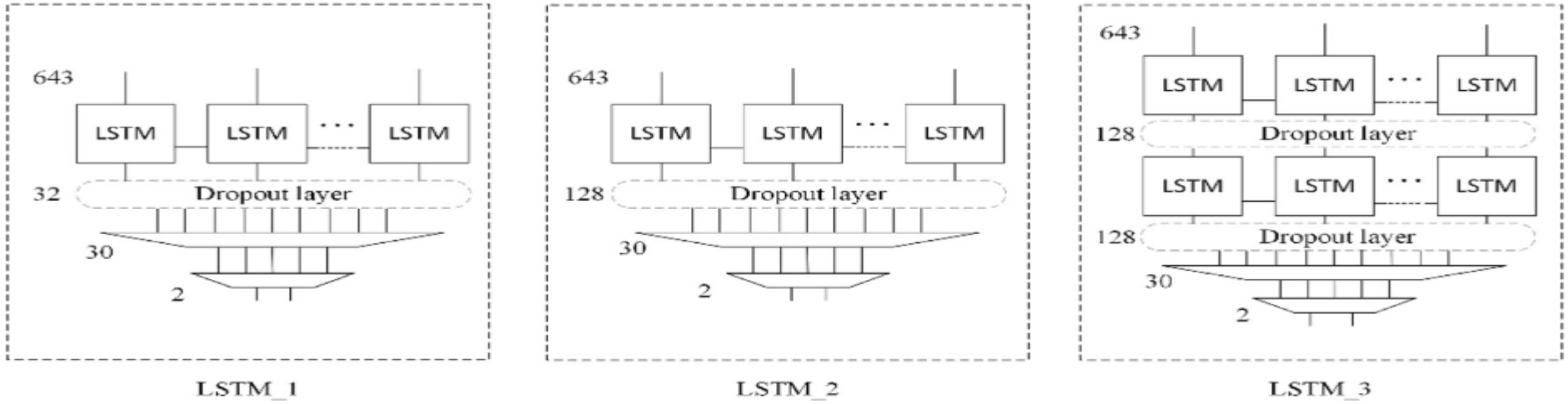



**Fig. 2.** The architecture of the three LSTM networks. The dropout layers are presented in dot lines as the evaluation is performed with and without including them.

# LSTM used in Tumor Growth Prediction

Transactions on Medical Imaging



## Spatio-Temporal Convolutional LSTMs for Tumor Growth Prediction by Learning 4D Longitudinal Patient Data

Ling Zhang, Le Lu, *Senior Member, IEEE*, Xiaosong Wang, Robert M. Zhu, Mohammadhadi Bagheri, Ronald M. Summers, and Jianhua Yao

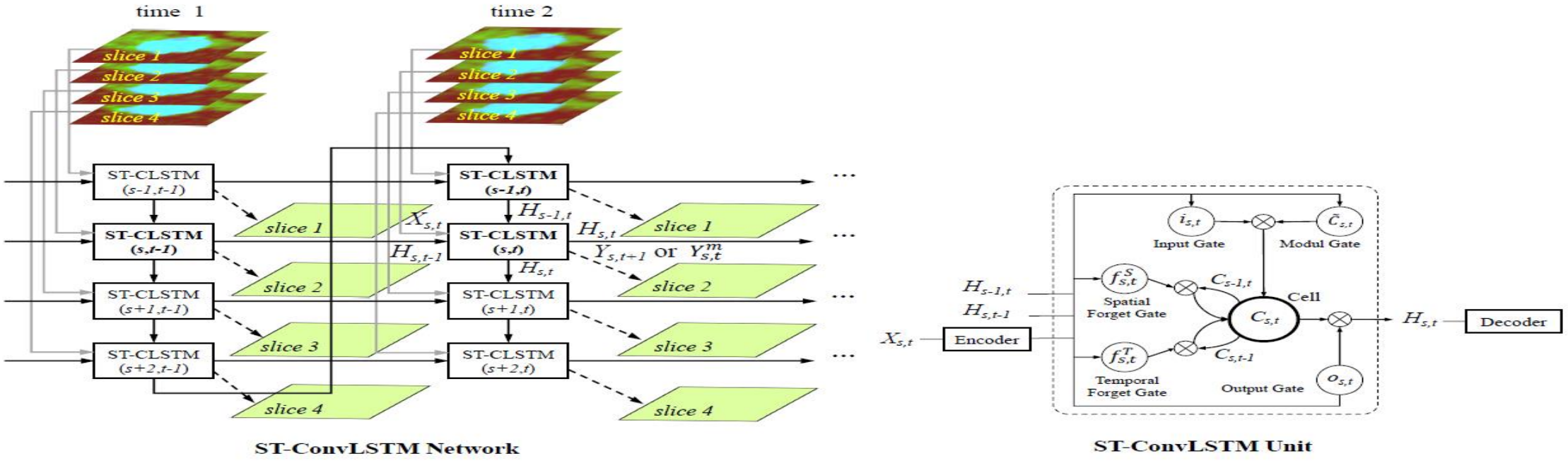
**Prognostic tumor growth modeling via volumetric medical imaging observations can potentially lead to better outcomes of tumor treatment management and surgical planning. Recent advances of convolutional networks (ConvNets) have demonstrated higher accuracy than traditional mathematical models can be achieved in predicting future tumor volumes. This indicates that deep learning based data-driven techniques may have great potentials on addressing such problem. However, current 2D image patch based modeling approaches can not make full use of the spatio-temporal imaging context of the tumor's longitudinal 4D (3D + time) patient data. Moreover, they are incapable to predict clinically-relevant tumor properties, other than the tumor volumes. In this paper, we exploit to formulate the tumor growth process through convolutional Long Short-Term Memory (ConvLSTM) that extract tumor's static imaging appearances and simultaneously capture its temporal dynamic changes within a single network. We extend ConvLSTM into the spatio-temporal domain (ST-ConvLSTM) by jointly learning the inter-slice 3D contexts and the longitudinal or temporal dynamics from multiple patient studies. Our approach can incorporate other non-imaging patient information in an end-to-end trainable manner. Experiments are conducted on the largest 4D longitudinal tumor dataset of 33 patients to date. Results validate that the proposed ST-ConvLSTM model produces a Dice score of 83.2%±5.1% and a RVD of 11.2%±10.8%, both statistically significantly outperforming ($p$ <0.05) other compared methods of traditional linear model, ConvLSTM, and generative adversarial network (GAN) under the metric of predicting future tumor volumes. Additionally, our new method enables the prediction of both cell density and CT intensity numbers. Last, we demonstrate the generalizability of ST-ConvLSTM by employing it in 4D medical image segmentation task, which achieves an averaged Dice score of 86.3%±1.2% for left-ventricle segmentation in 4D ultrasound with 3 seconds per patient case.**

### I. Introduction

Tumor growth modeling using medical images of longitudinal studies is a challenging yet important problem in precision and predictive medicine, because it may potentially lead to better tumor treatment management and surgical planning for patients. For example, treatments of pancreatic neuroendocrine tumor (PanNET or PNET) include active surveillance, surgical intervention, and medical treatment. Active surveillance is undertaken if a PanNET does not reach 3 cm in diameter or a tumor-doubling time <500 days; otherwise the corresponding PanNET should be resected due to the high risk of metastatic disease [1]. Medical treatment (e.g., everolimus) is for the intermediate-grade (PanNETs with radiologic documents of progression within the previous 12 months), advanced or metastatic disease [2]. Therefore the patient-specific prediction of PanNET's growth pattern at earlier stages is highly desirable, since it will assist decision making on different treatment strategies to better manage the undergoing treatment or surgical planning.

Conventionally, this task has been well exploited through complex and sophisticated mathematical modeling [3]–[9], which accounts for both cell invasion and mass-effect using reaction-diffusion equations and bio-mechanical models. From there the actual tumor growth can be predicted by personalizing the established model based on clinical imaging derived tumor physiological parameters, such as morphology, metabolic rate, and cell density. While these methods yield

# LSTM used in Tumor Growth Prediction



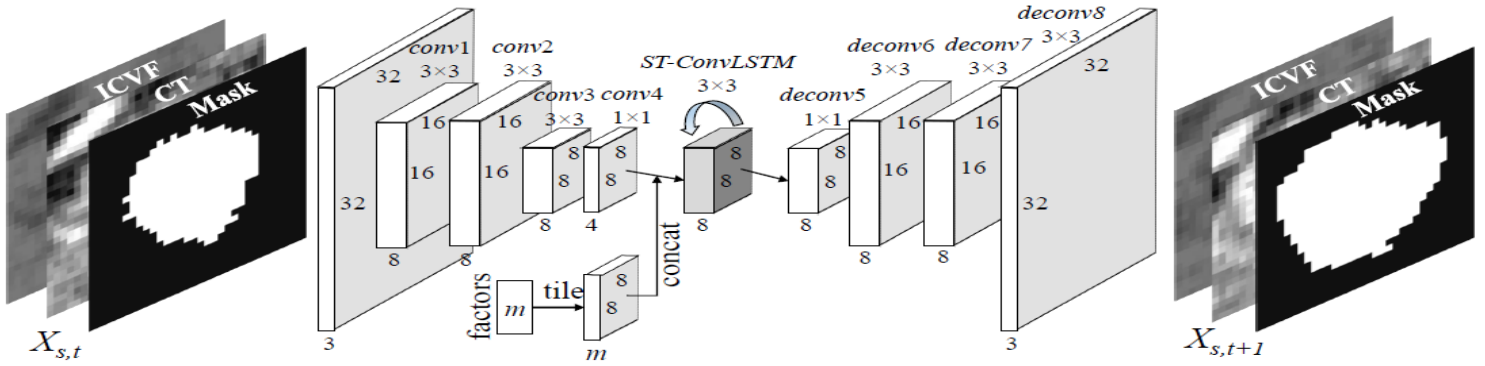




Fig. 3. The end-to-end network architecture of our proposed encoder-ST-ConvLSTM-decoder for tumor growth prediction. For 4D segmentation task, the input is replaced with the raw (e.g., ultrasound) image, the output is its mask, no "factor" branch for other clinical properties, and network model channels are set to 1-8-16-32-64-64-64-32-16-1.

# LSTM used in Tumor Growth Prediction

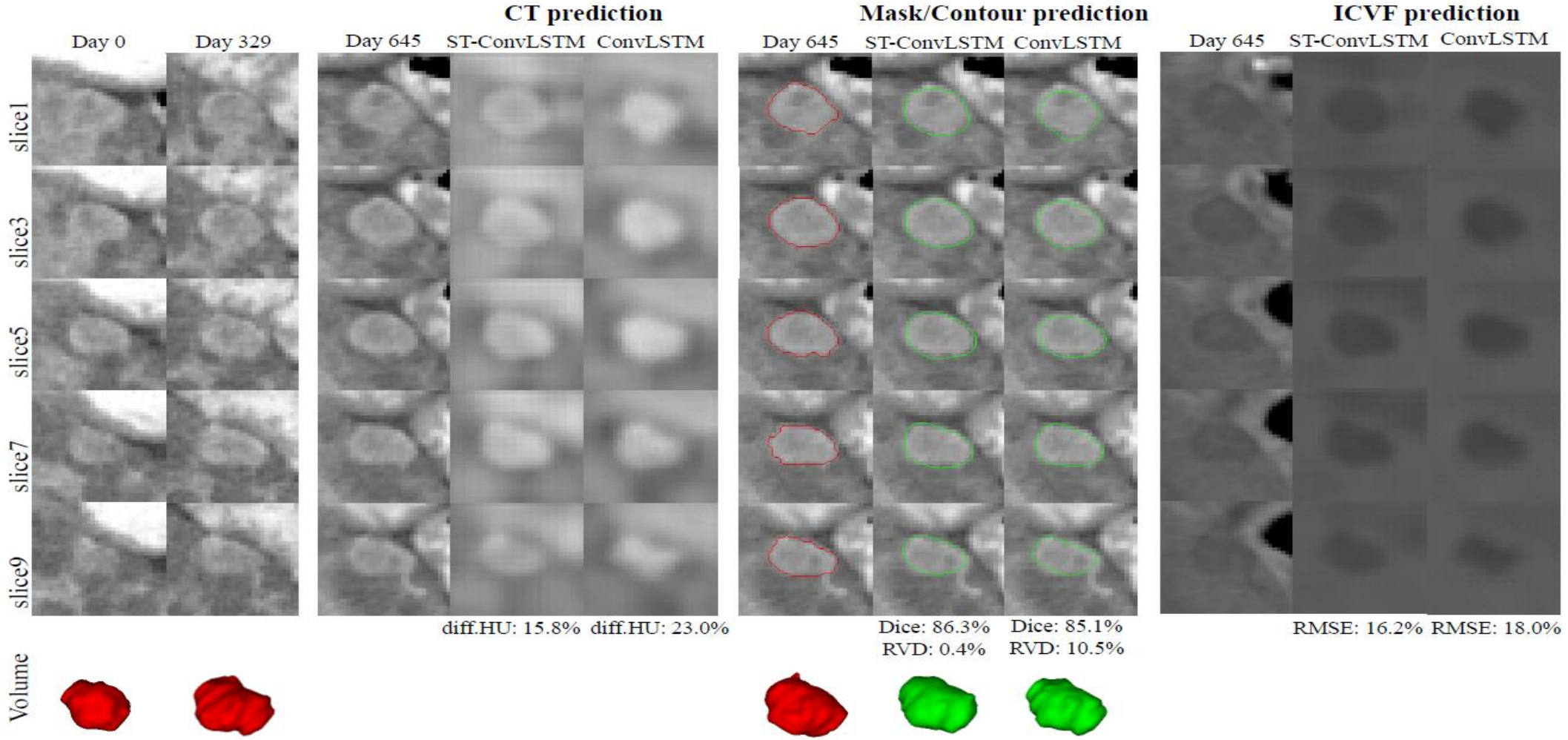



Fig. 5. An illustrated example shows the prediction results of CT, mask/volume, and ICVF of a tumor by ST-ConvLSTM and ConvLSTM. Note that the tumor contours are superimposed on the ground truth CT images at time 3. Red: ground truth boundaries; Green: predicted tumor boundaries. In this example, consecutive image slices with the spatial interval of two slices are shown for better visualization of the spatial changes/differences.