Deep Learning for NLP

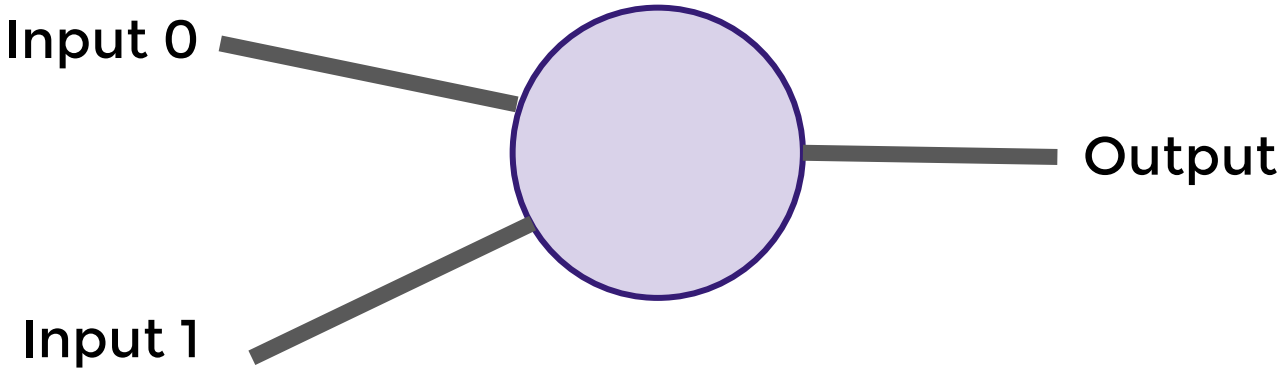
RNN and LSTM

Natural Language Processing Bootcamp

- Section Goals
  - Review basic overview of Deep Learning
  - Understand basics of LSTM and RNN
  - Use LSTM to generate text from source corpus
  - Create QA Chat Bots with Python
  - Transformers for NLP - BERT / GPT

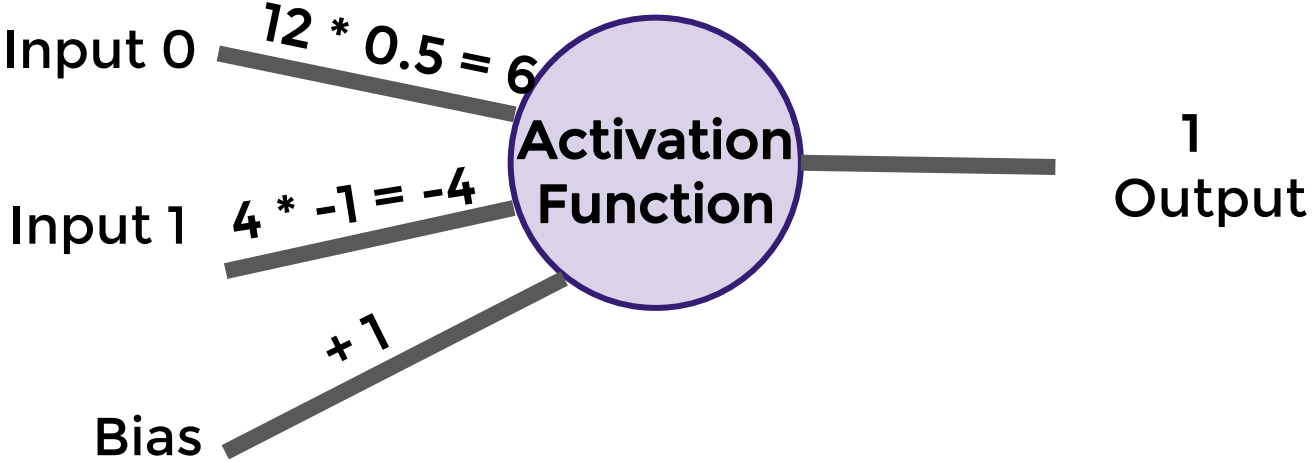
Artificial Neuron

- The artificial neuron also has inputs and outputs.
  Words Features => NUMBERS !



Artificial Neuron

- So what does this look like mathematically?

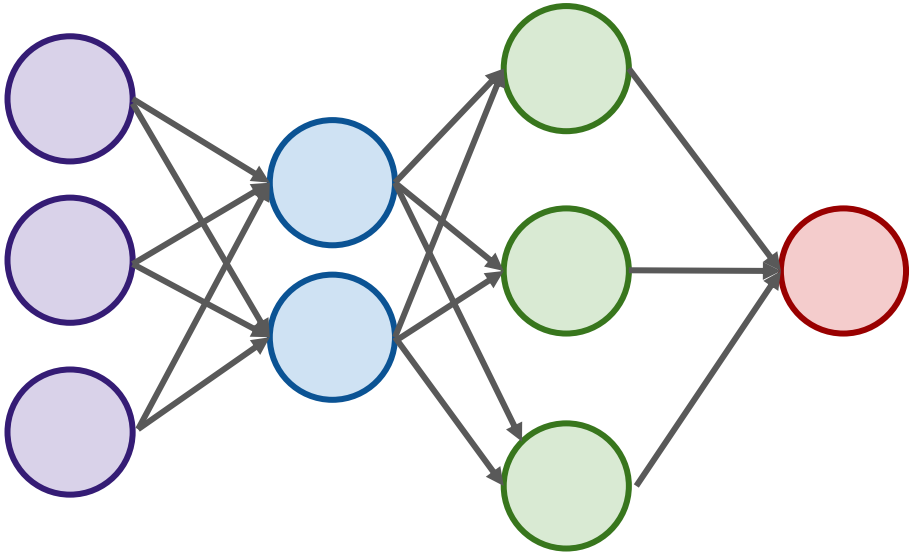


From Single Perceptron to ANN

- We've seen how a single perceptron behaves, now let's expand this concept to the idea of a neural network!
- Let's see how to connect many perceptrons together and then how to represent this mathematically!

Multiple Perceptrons Network

Input Layer + 2 hidden layers + Output Layer



Layers

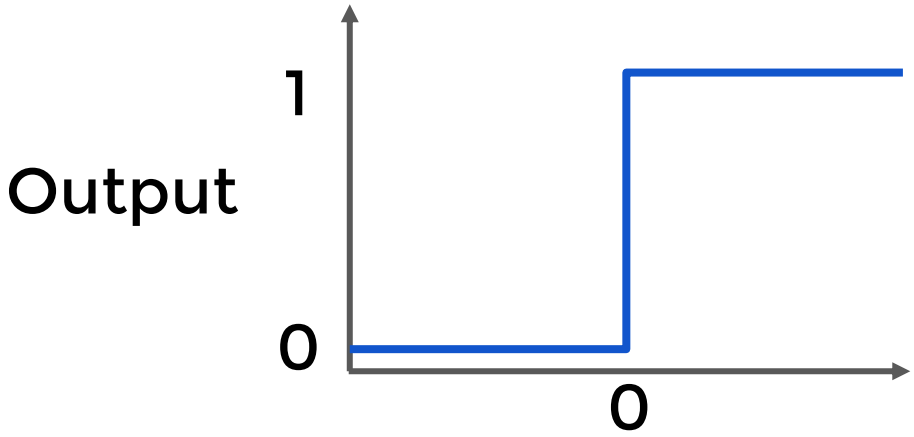
- Input Layers
  - Real values from the data
- Hidden Layers
  - Layers in between input and output
  - 3 or more layers is “**deep network**”
- Output Layer
  - Final estimate of the output

Layers of NN

- As you go forwards through more layers, the level of abstraction increases.
- Use of the activation function

Activation Function

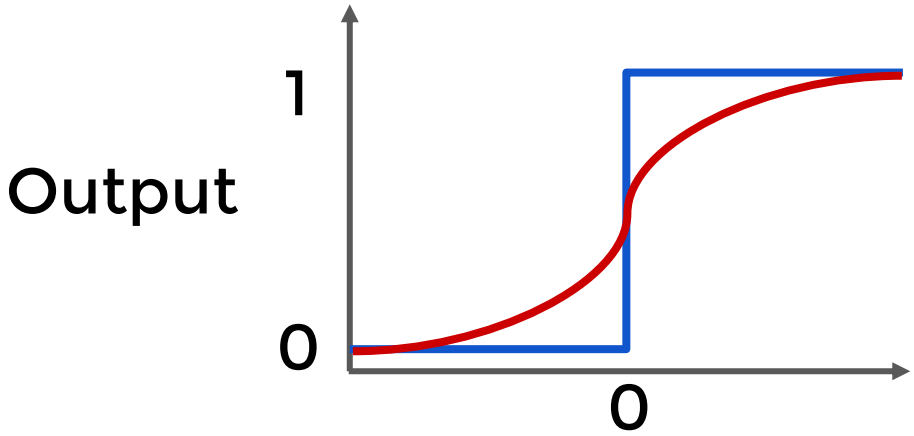
- A simple activation function may output 0 or 1.
- This is a pretty dramatic as small changes aren't reflected.



$$z = wx + b$$

Activation Function

- It would be nice if we could have a more dynamic function, for example the red line!



$$z = wx + b$$

Deep Learning Libraries

- Deep Learning libraries have built in activation functions for us, so we don’t need to worry about having to implement them manually!

Keras Next

Now that we understand the basics of neural network theory, let's move on to implementing and building our own neural network models with Keras!

Keras Basics

Natural Language Processing Bootcamp

- Let’s learn how to create a very simple neural network for classifying the famous Iris data set!
- The iris data set contains measurements of flower petals and sepals and has corresponding labels to one of three classes (3 flower species).

Recurrent Neural Networks Theory

Deep Learning

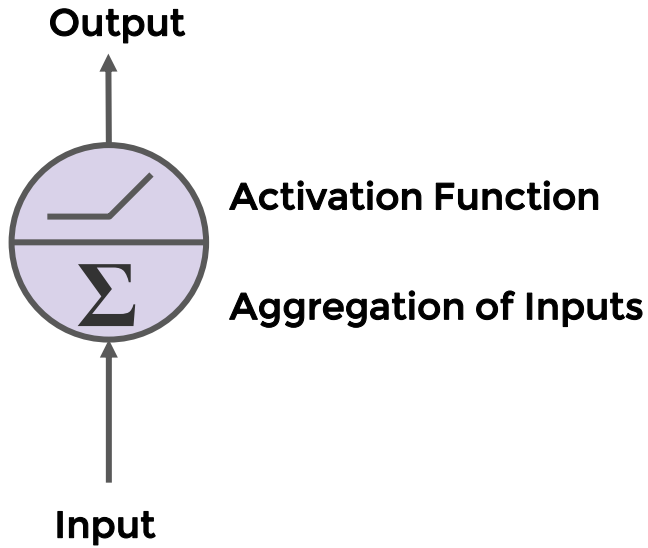
- Examples of Sequences
  - Time Series Data (Sales)
  - Sentences
  - Audio
  - Car Trajectories
  - Music

Deep Learning

- Let’s imagine a sequence:
  - [1,2,3,4,5,6]
  - Would you be able to predict a similar sequence shifted one time step into the future?
    - [2,3,4,5,6,7]

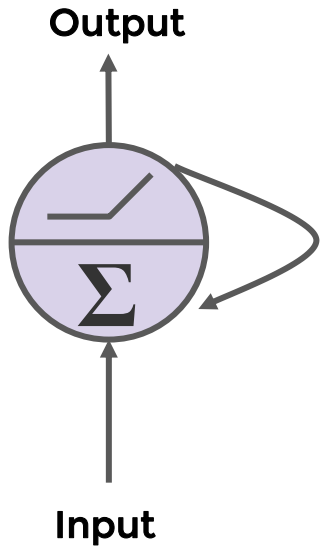
Deep Learning

- Normal Neuron in Feed Forward Network



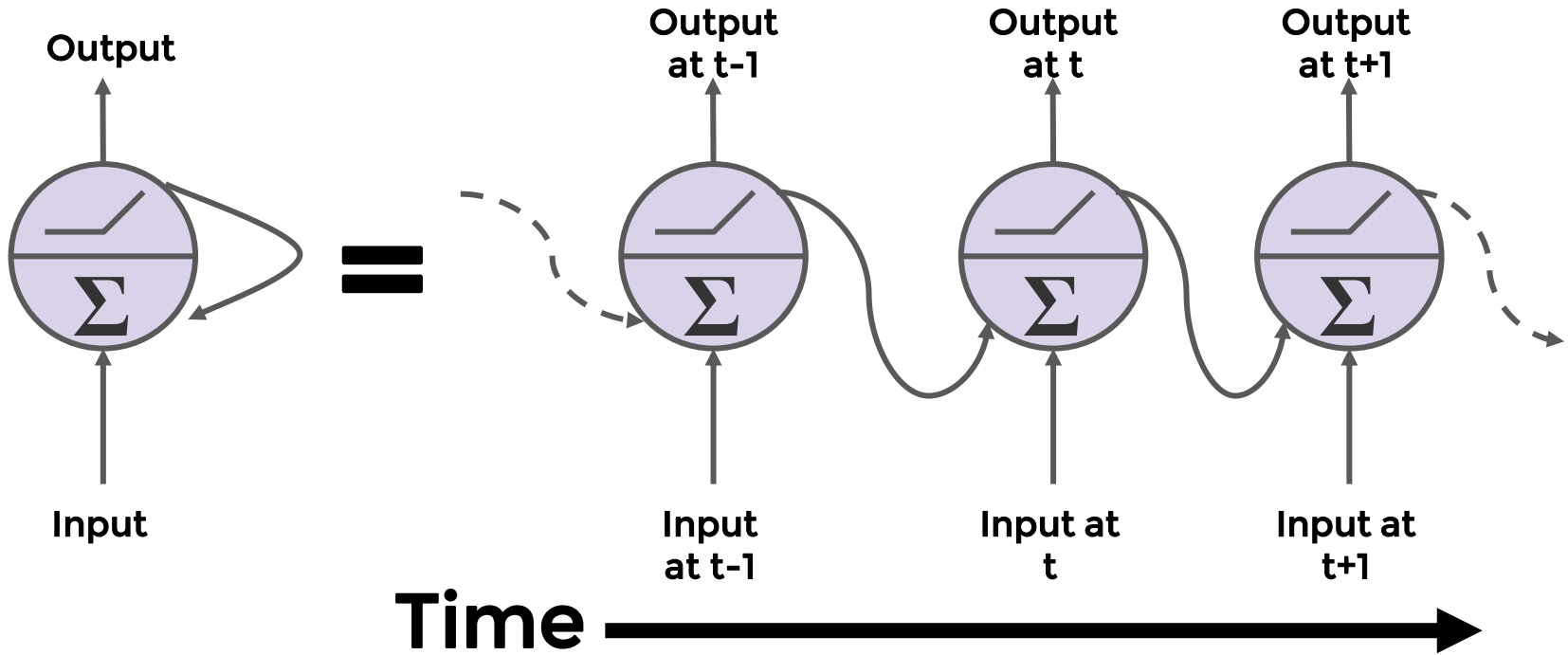
Deep Learning

- Recurrent Neuron - Sends output back to itself!
  - Let's see what this looks like over time!



Deep Learning

- Recurrent Neuron



Deep Learning

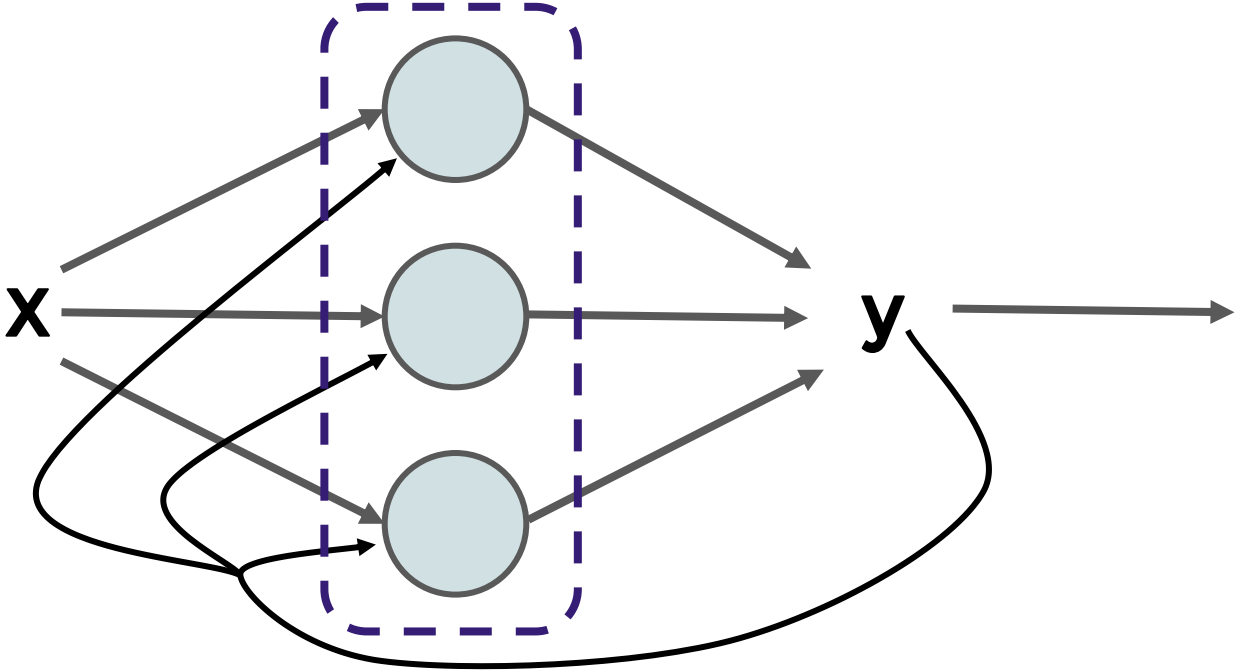
- Cells that are a function of inputs from previous time steps are also known as *memory cells.*
- RNN are also flexible in their inputs and outputs, for both sequences and single vector values.

Deep Learning

- We can also create entire layers of Recurrent Neurons...

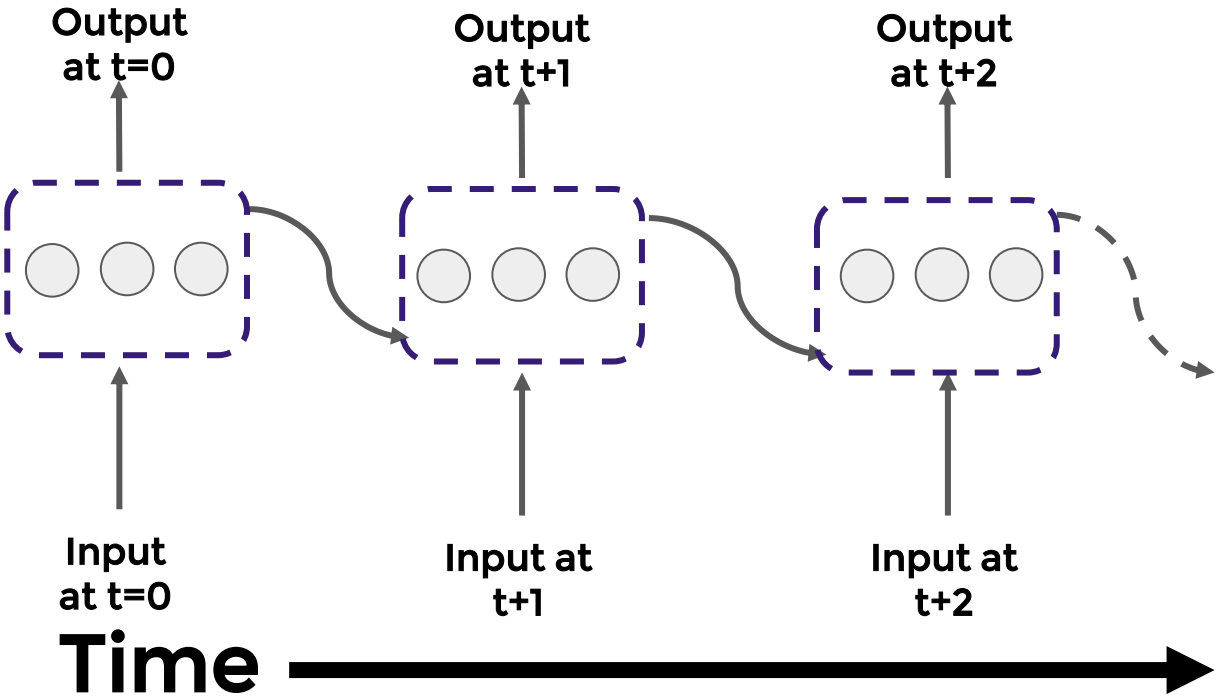
Deep Learning

- RNN Layer with 3 Neurons:



Deep Learning

- “Unrolled” layer.

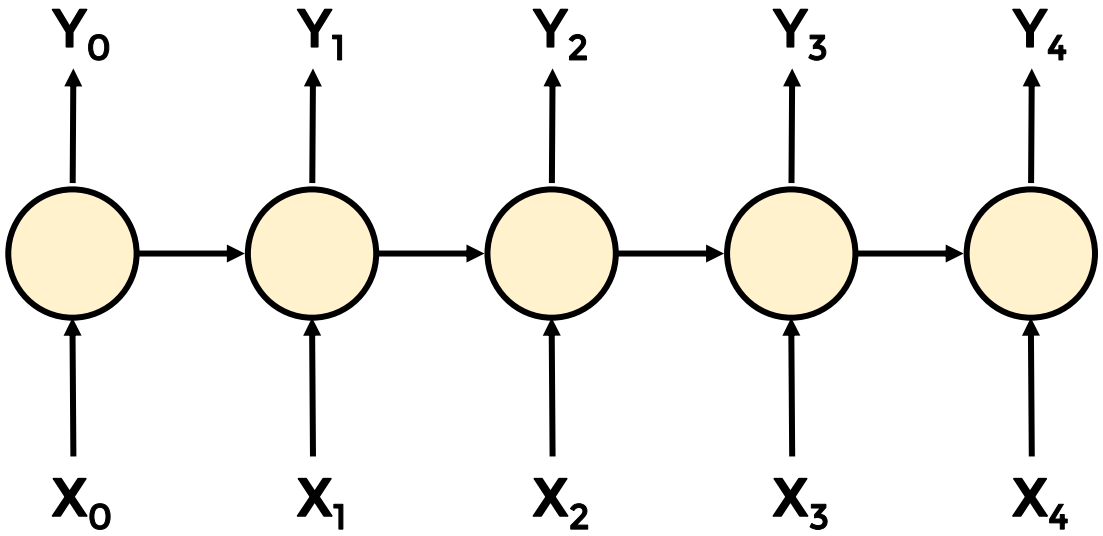


Deep Learning

- Let’s see a few examples.

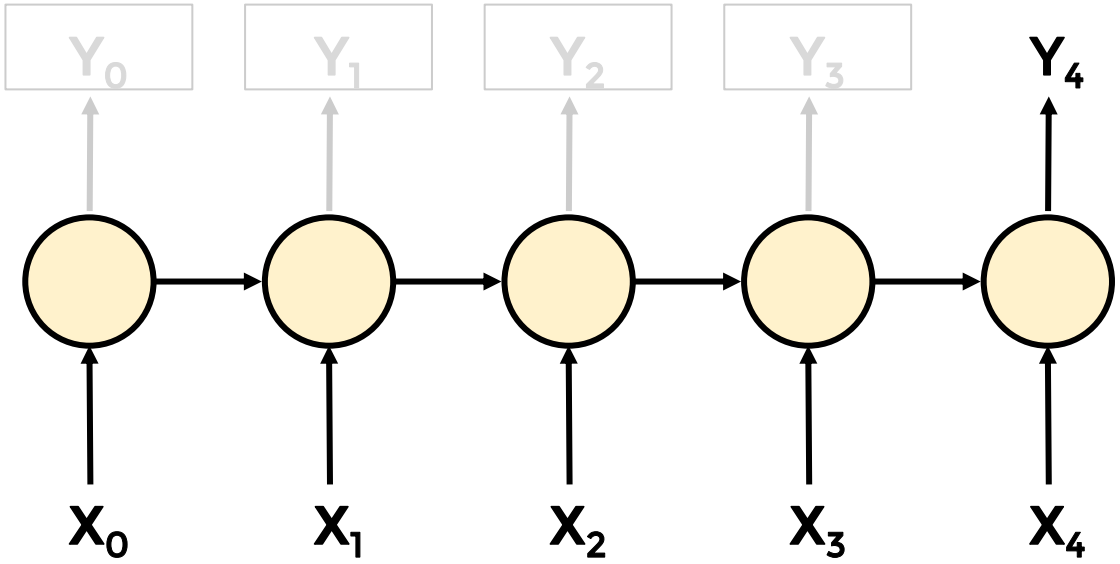
Deep Learning

- Sequence to Sequence



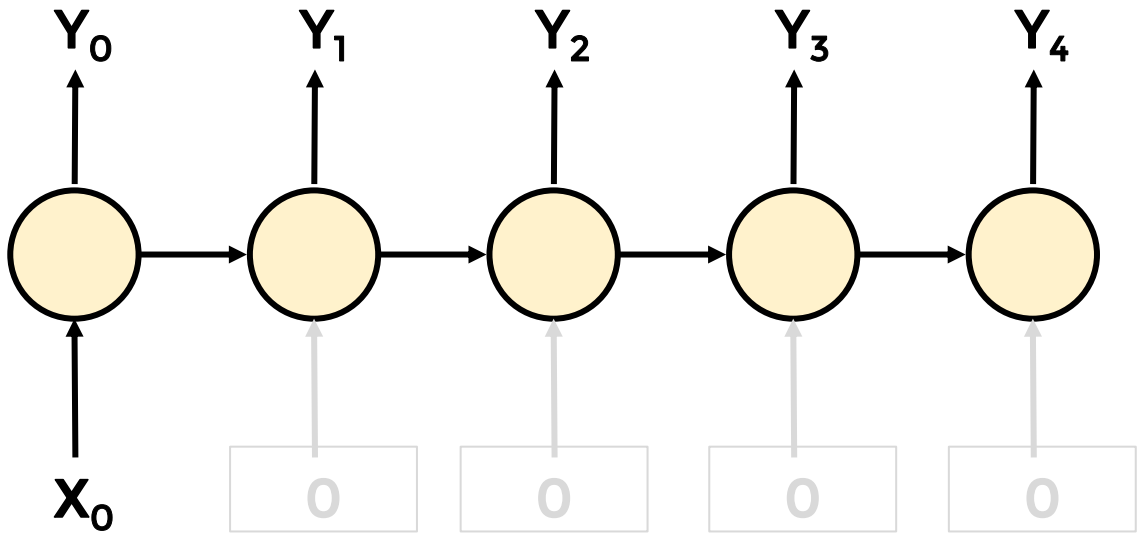
Deep Learning

- Sequence to Vector



Deep Learning

- Vector to Sequence



Deep Learning

- Now that we understand basic RNNs we'll move on to understanding a particular cell structure known as LSTM (Long Short Term Memory Units).

LSTM
and Variants GRU

Deep Learning

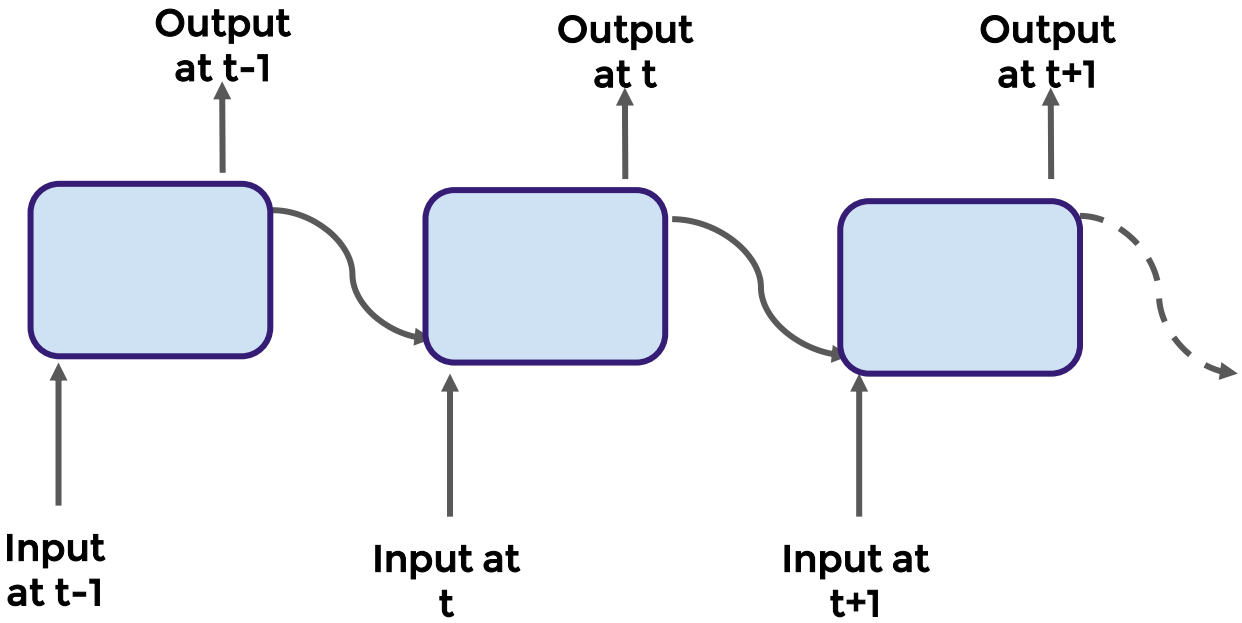
- An issue RNN face is that after awhile the network will begin to “forget” the first inputs, as information is lost at each step going through the RNN.
- We need some sort of “long-term memory” for our networks.

Deep Learning

- The LSTM (Long Short-Term Memory) cell was created to help address these RNN issues.
- Let’s go through how an LSTM cell works!
- Keep in mind, there will be a lot of Math here!

Deep Learning

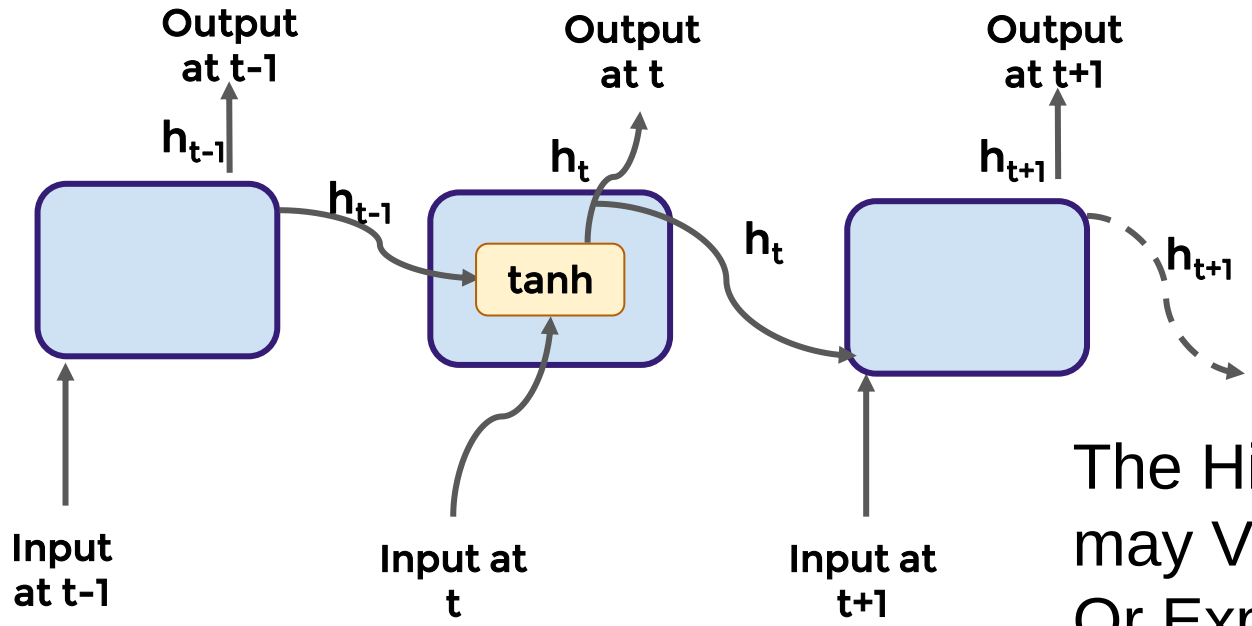
- A typical RNN



Deep Learning

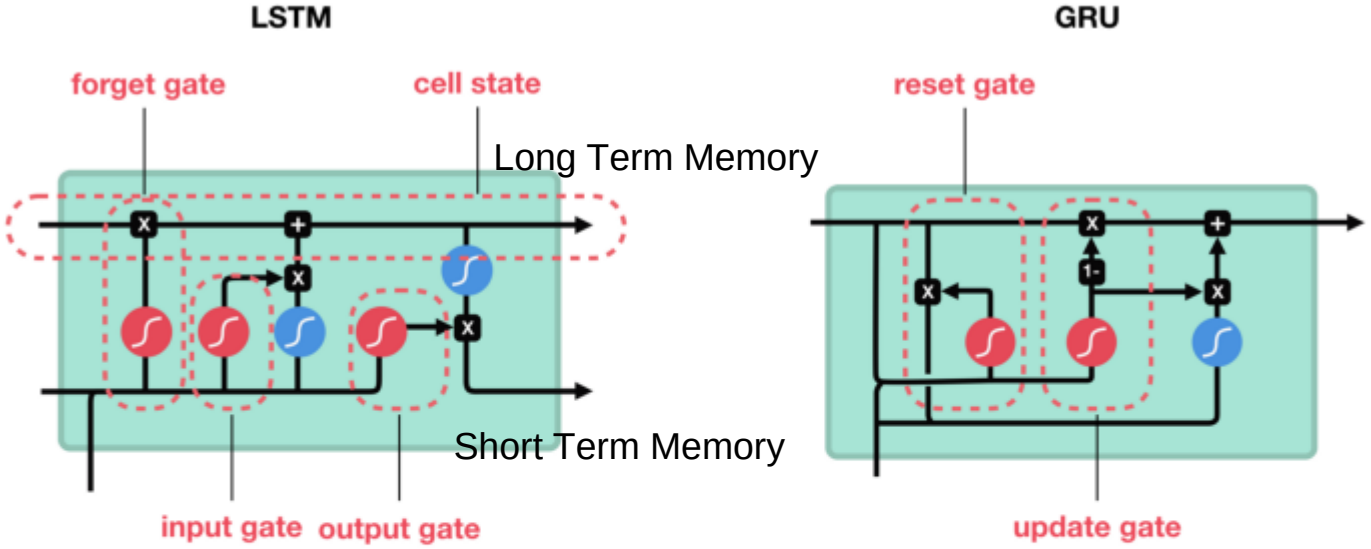
Hidden State : h_{t_1}

- A typical RNN cell

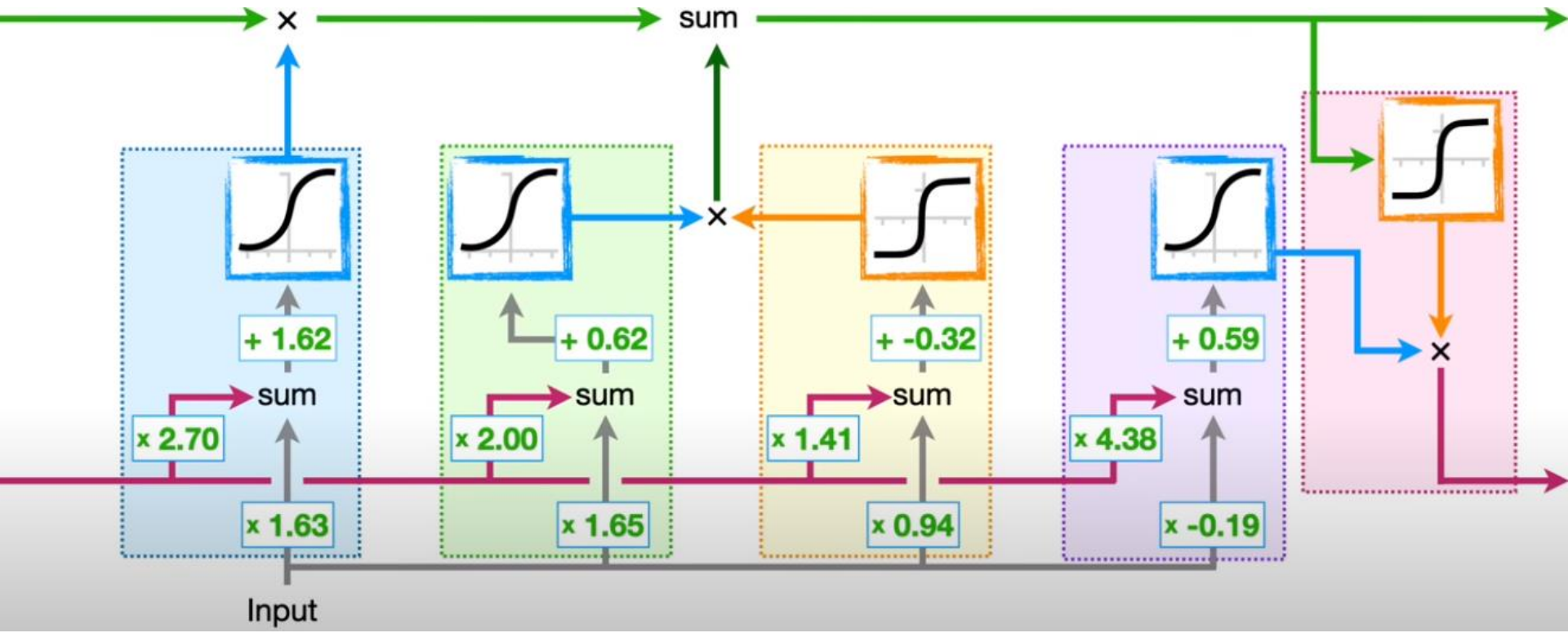


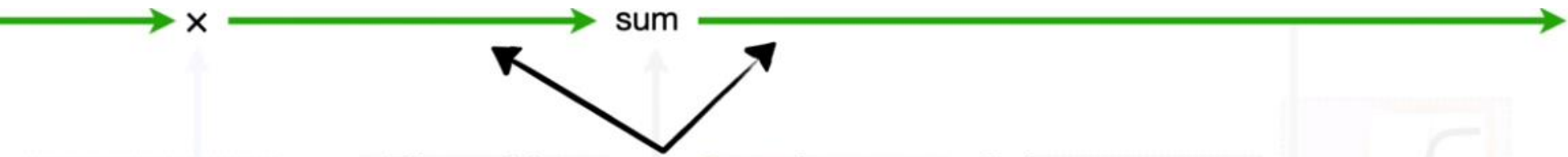
The Hidden State may Vanish Or Explode

An LSTM Cell


LSTM
forget gate
cell state
Long Term Memory
Short Term Memory
input gate
output gate
GRU
reset gate
update gate
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

Long Short Term Memory Cell (LSTM) Unit

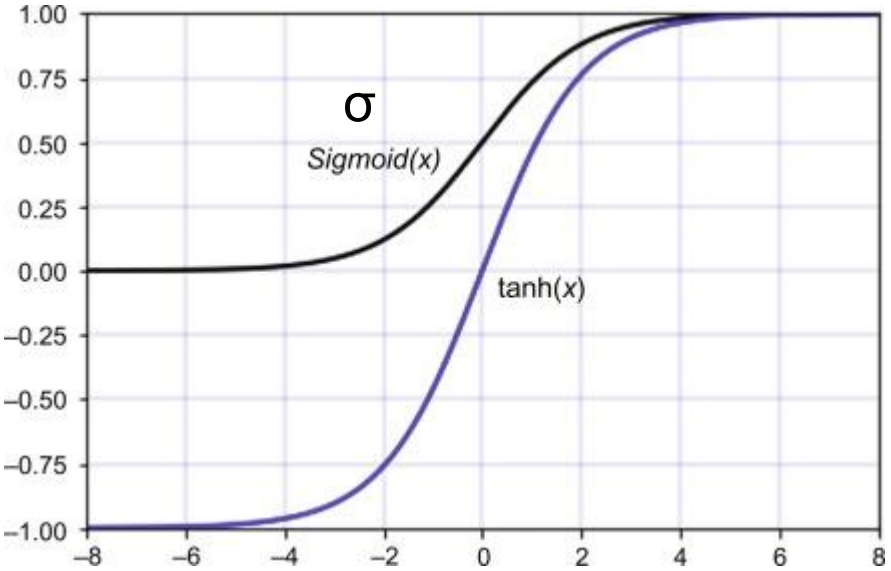




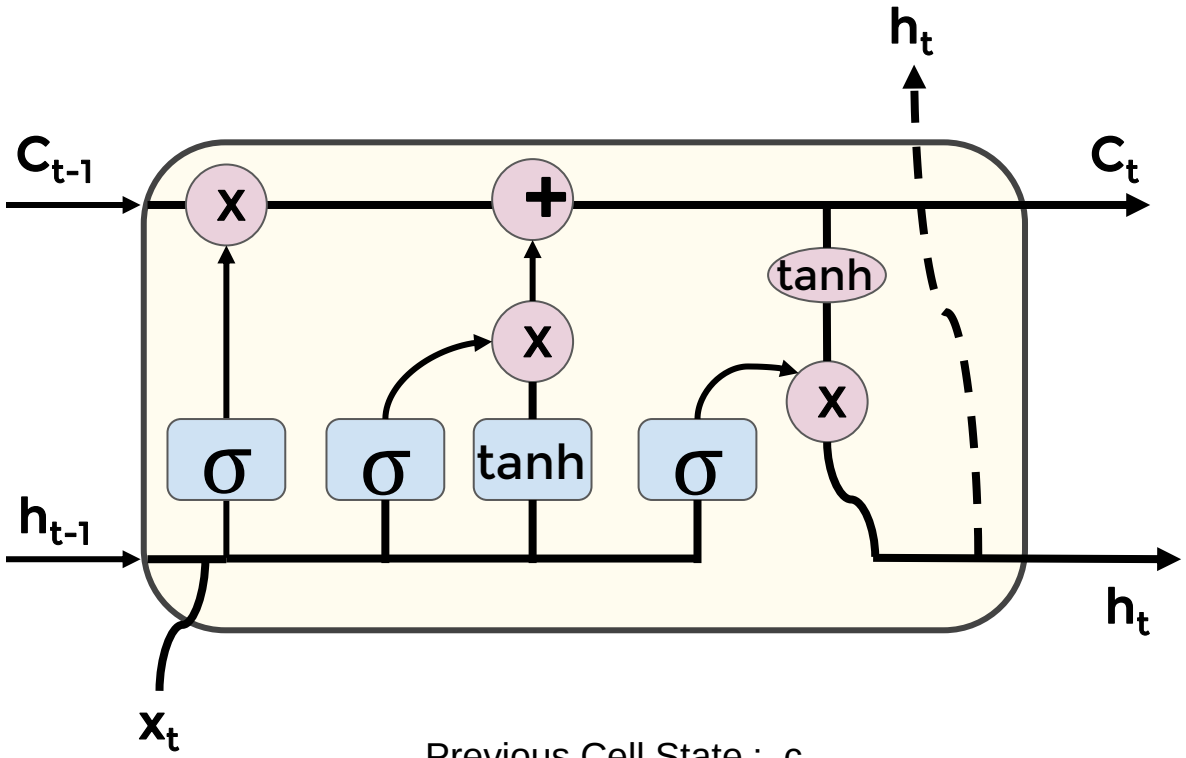
First, this **green line** that runs all the way across the top of the unit is called the **Cell State** and represents the **Long-Term Memory**.

Now, this **pink line**, called the **Hidden State**, represents the **Short-Term Memories**.

Sigmoid vs Tanh



An LSTM Cell

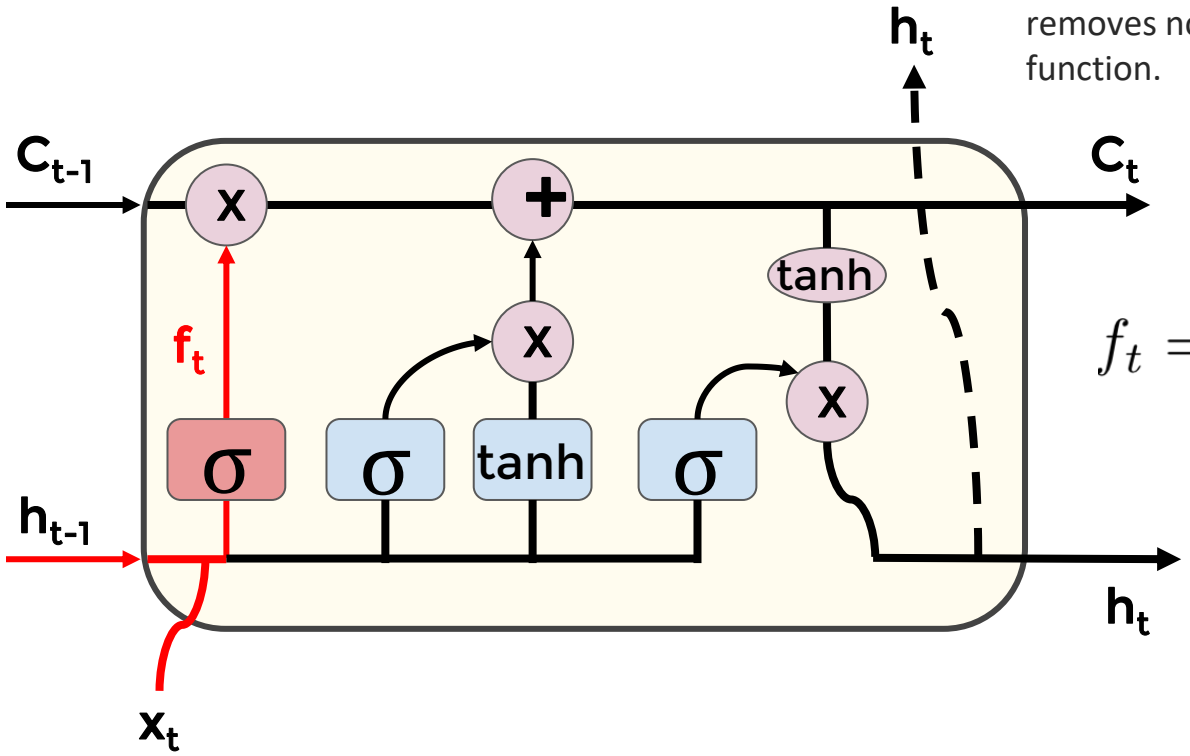


Previous Cell State : c_{t_1}

Here we can see the entire LSTM cell.

Let's go through the process!

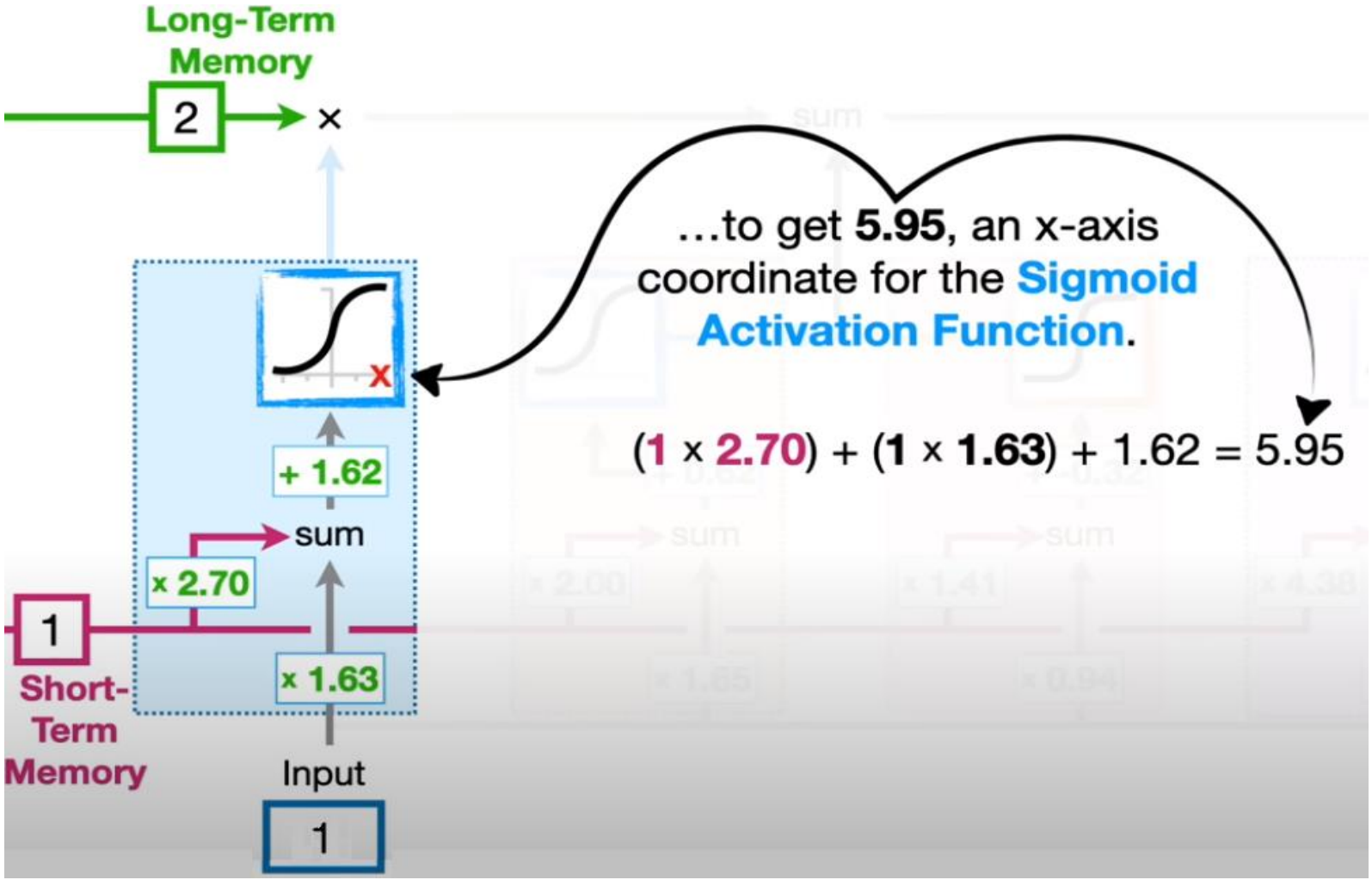
An LSTM Cell



- First, the previous hidden state and the current input get concatenated. We'll call it *combine*.
- *Combine* get's fed into the forget layer. This layer removes non-relevant data through the sigmoid function.

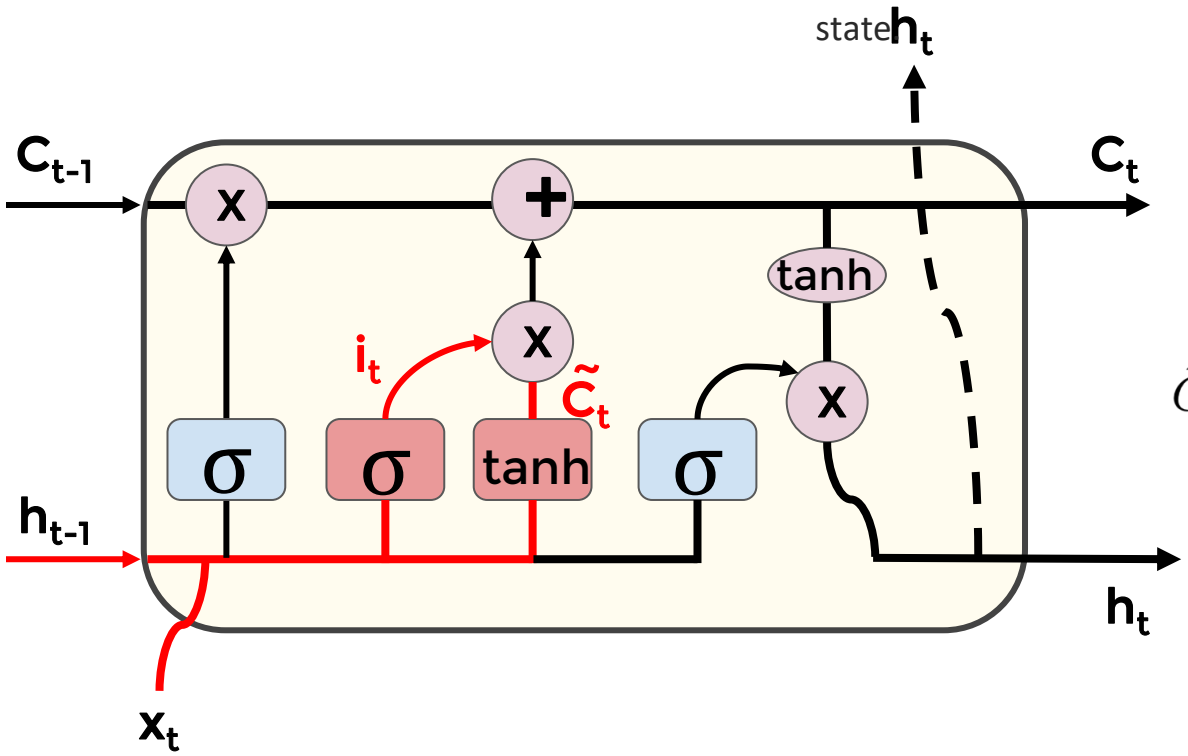
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Learn parameters Wf and bf


Long-Term Memory
2
×
x
+ 1.62
sum
× 2.70
1
Short-Term Memory
× 1.63
Input
1
…to get 5.95, an x-axis coordinate for the Sigmoid Activation Function.
(1 × 2.70) + (1 × 1.63) + 1.62 = 5.95

An LSTM Cell

- A candidate layer is created using *combine*. The candidate holds possible values to add to the cell state.
- *Combine* also get's fed into the input layer. This layer decides what data from the candidate should be added to the new cell state

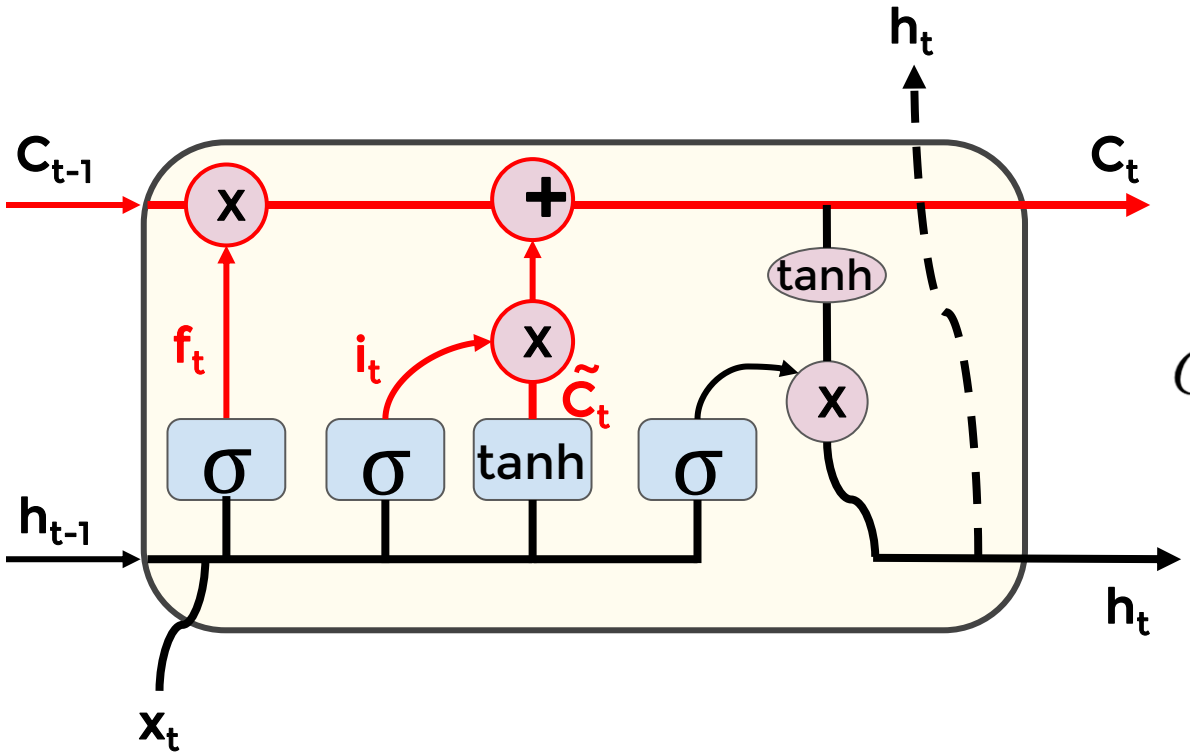


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

An LSTM Cell

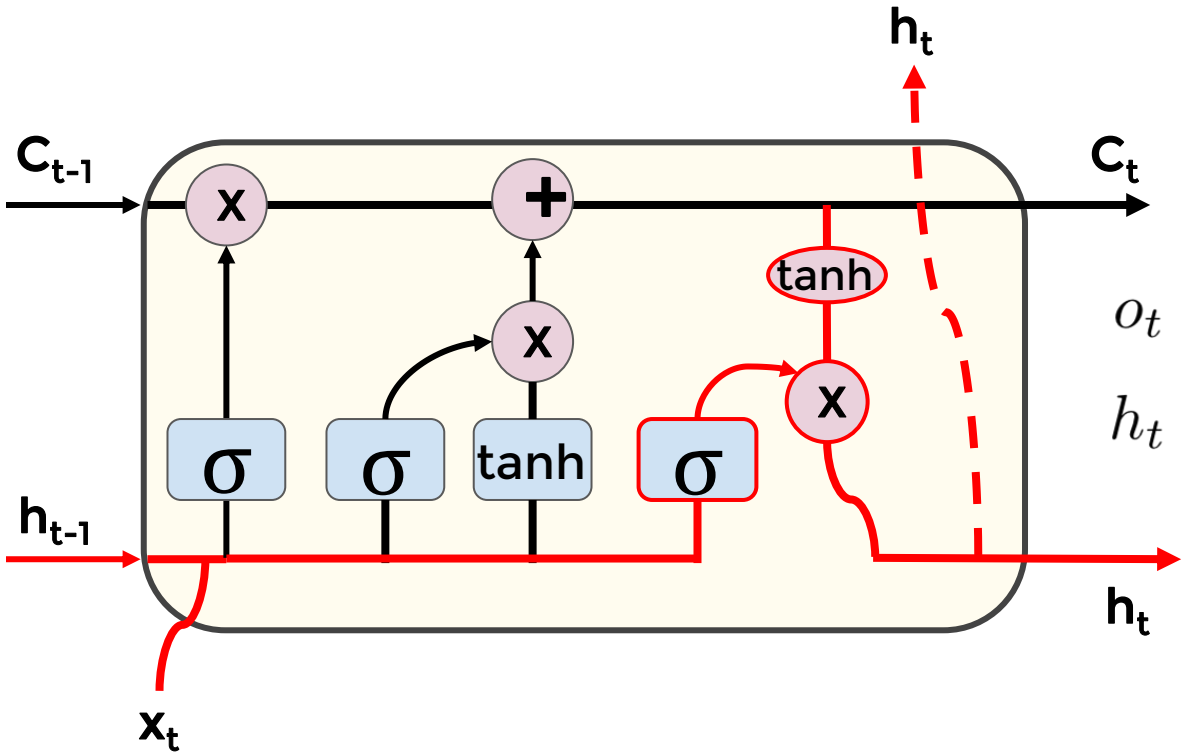
After computing the forget layer, candidate layer, and the input layer, the cell state is calculated using those vectors and the previous cell state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

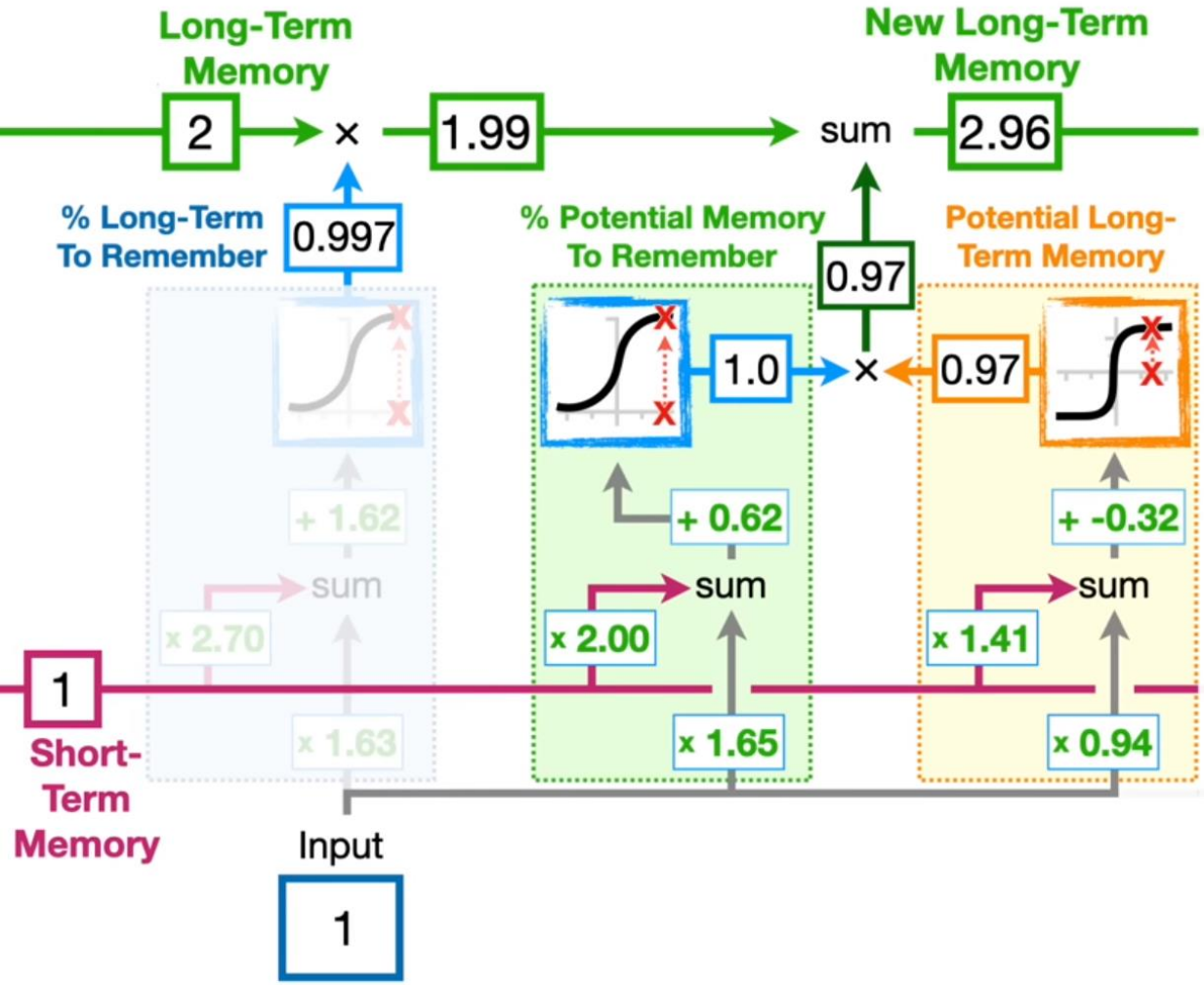
An LSTM Cell

The output is then computed.
Pointwise multiplying the output and the new cell state gives us the new hidden state.



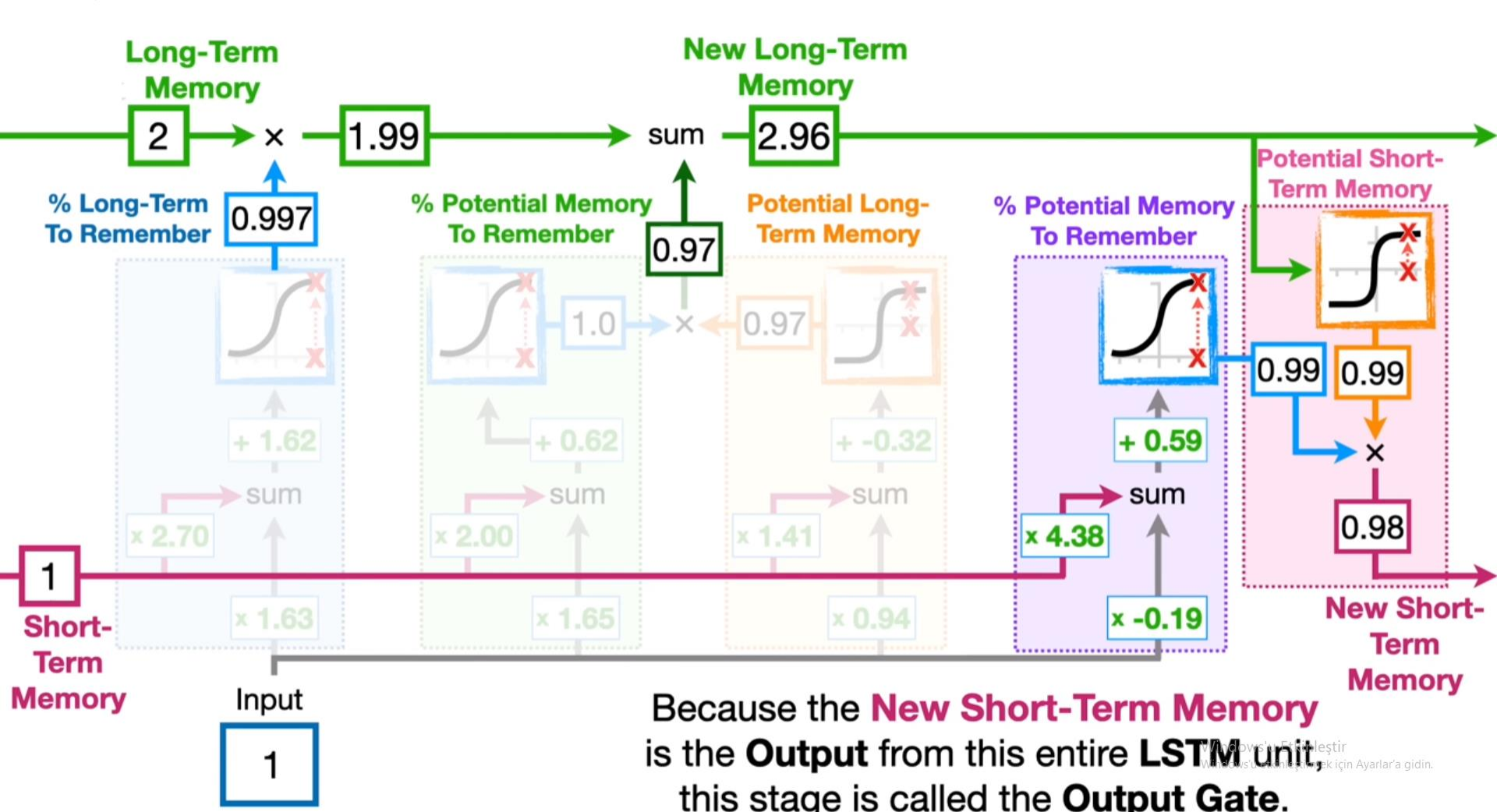
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



Even though this part of the **Long Short-Term Memory** unit determines how we should update the **Long-Term Memory**…

…it is usually called the **Input Gate**.


Long-Term Memory
2
×
1.99
sum
New Long-Term Memory
2.96
% Long-Term To Remember
0.997
% Potential Memory To Remember
0.97
Potential Long-Term Memory
% Potential Memory To Remember
Potential Short-Term Memory
1.0
×
0.97
+ 1.62
sum
x 2.70
x 1.63
+ 0.62
sum
x 2.00
x 1.65
+ -0.32
sum
x 1.41
x 0.94
+ 0.59
sum
x 4.38
x -0.19
0.99
0.99
×
0.98
1
Short-Term Memory
New Short-Term Memory
Input
1
Because the New Short-Term Memory is the Output from this entire LSTM unit, this stage is called the Output Gate.

Initial Long Term Memory

Initial Short Term Memory

0 0 1 Day 1

0.5 0.3 0.5 Day 2

0.9 0.6 0.25 Day 3

1.5 0.9 1 Day 4

2.4 1.0

And that means that the **Output** from the **LSTM** correctly predicts **Company B**'s value for **Day 5**.

An LSTM Cell Illustration

- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

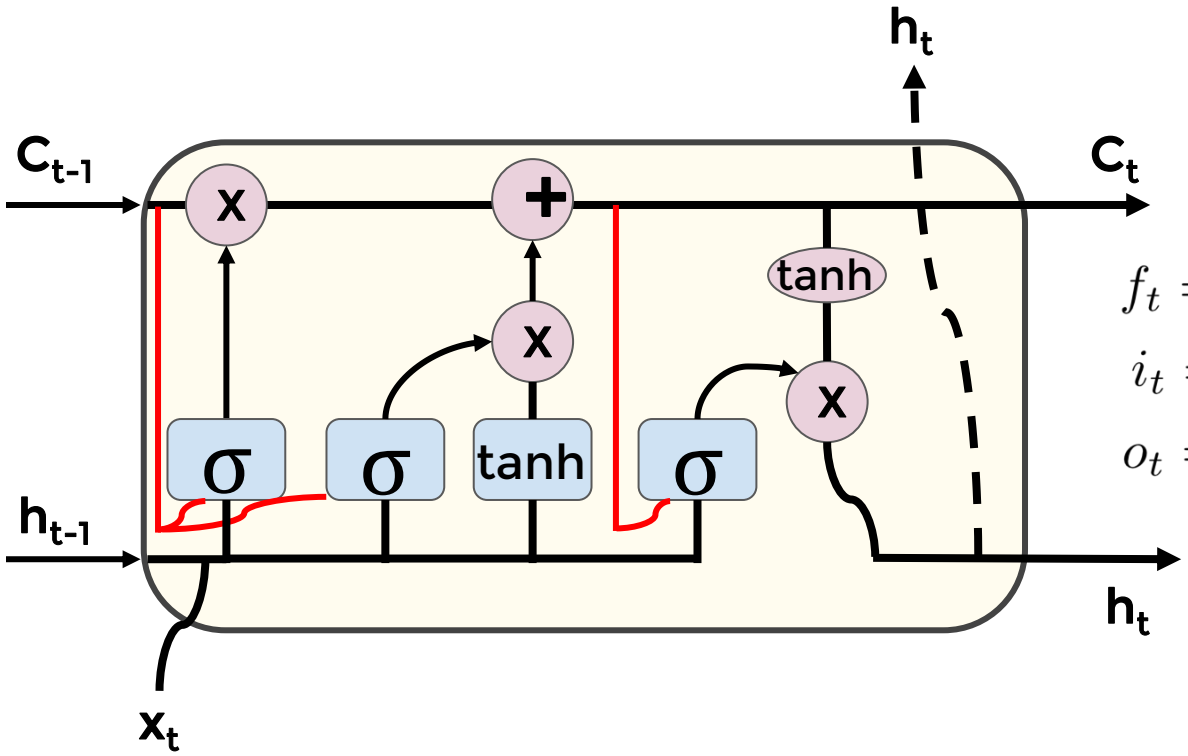
```
def LSTMCELL(prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)
    Ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(Ct)
    return ht, Ct



ct = [0, 0, 0]
ht = [0, 0, 0]

for input in inputs:
    ct, ht = LSTMCELL(ct, ht, input)
```

A Modified LSTM Cell Variant



Added Cell State to forget, input and output layers

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

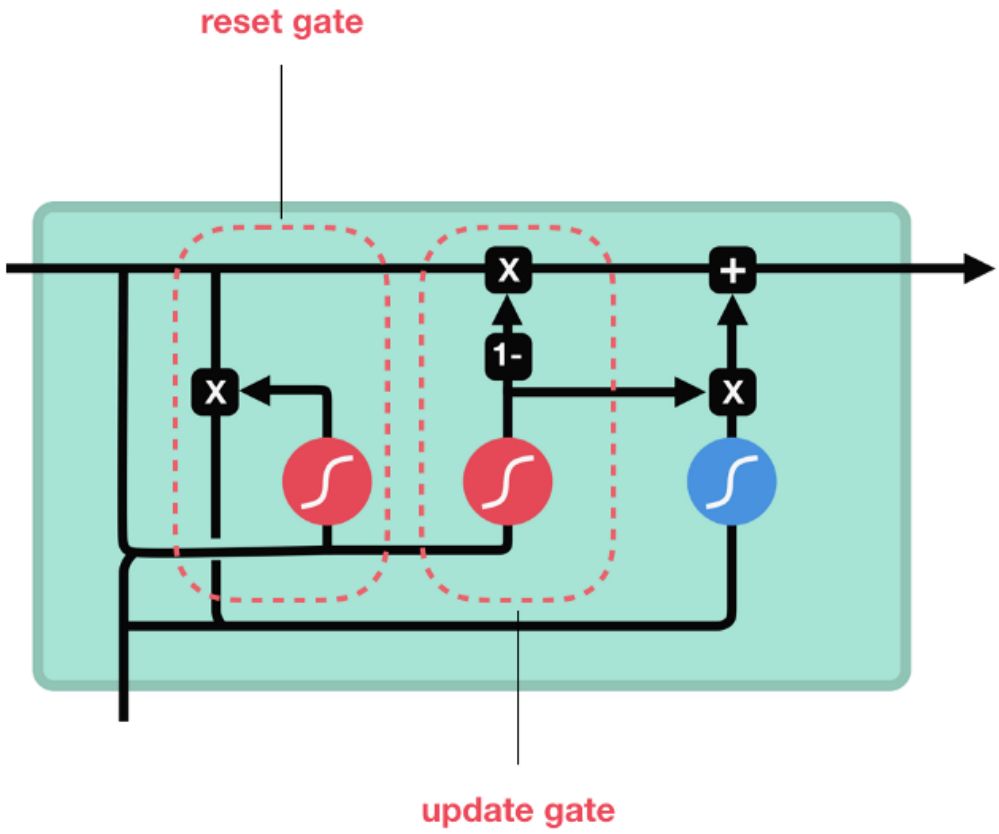
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$

$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Gated Recurrent Unit (GRU)

- The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM.
- GRU’s got rid of the cell state and used the hidden state to transfer information.
- It also only has two gates, a reset gate and update gate.

GRU



GRU Gates

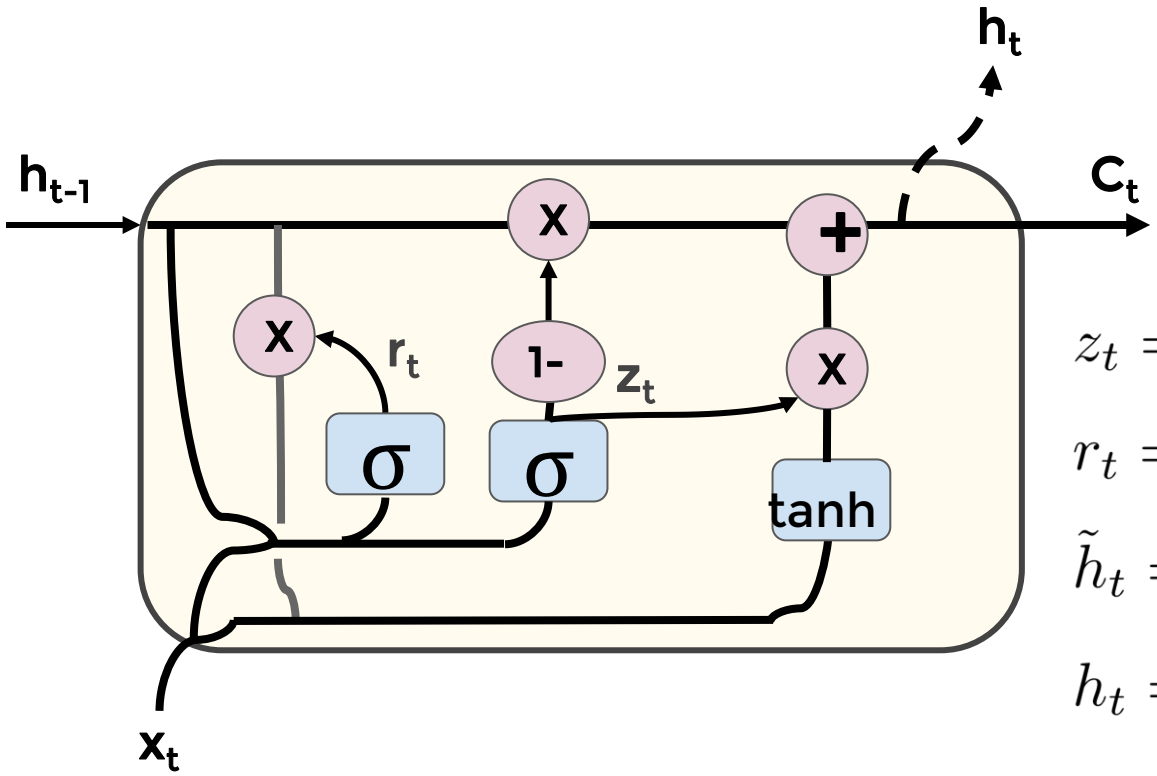
Update Gate

- The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.

Reset Gate

- The reset gate is another gate is used to decide how much past information to forget.

Gated Recurrent Unit (GRU)



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

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

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

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

GRU vs LSTM

- GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's.

- There isn't a clear winner which one is better. Researchers and engineers usually try both to determine which one works better for their use case.

Deep Learning

- Fortunately Keras has a really nice API that makes LSTM and RNN easy to work with.

- Coming up next, we'll learn how to format data for RNNs and then how to use LSTM for text generation.

Text Generation With Python and Keras

Part Two

Natural Language Processing Bootcamp

- Create the LSTM Based Model
- Split the Data into Features and Labels
  - X Features (First n words of Sequence)
  - y Label (Next Word after the sequence)
- Fit the Model

Text Generation With Python and Keras

Part Three

Natural Language Processing Bootcamp

- Generate New Text Based off a Seed