Long Short-Term Memory

By,
Prof(Dr) Premanand Ghadekar

Outline

- Recurrent Neural Network
- Introduction to LSTM
- Why LSTM model…?
- Mathematical Model of LSTM
- Working of LSTM
- Applications of LSTM etc.

What is Natural Language Processing?

Natural Language Processing (NLP) is a subfield of Linguistics, Computer Science, and Artificial Intelligence concerned with the interaction between computers and human language, in particular how to program computers to process and analyze large amount of natural language data.

NLP using Machine Learning

Machine Learning is the application of Artificial Intelligence for making Computers learn from the data given to it and then make decisions on their own similar to humans.

NLP using Deep Learning

Deep Learning is also sometimes called Deep Structures Learning, is a class of machine learning algorithms.

Deep Learning uses a multi-layer approach to extract high-level features from the data that is provided to it.

Two issues of Standard RNNs

A gradient is a partial derivative. A gradient quantifies how much the output of a function varies when the inputs are changed slightly.

A function's slope is also known as its gradient.

The steeper the slope, the faster a model can learn, the higher the gradient.

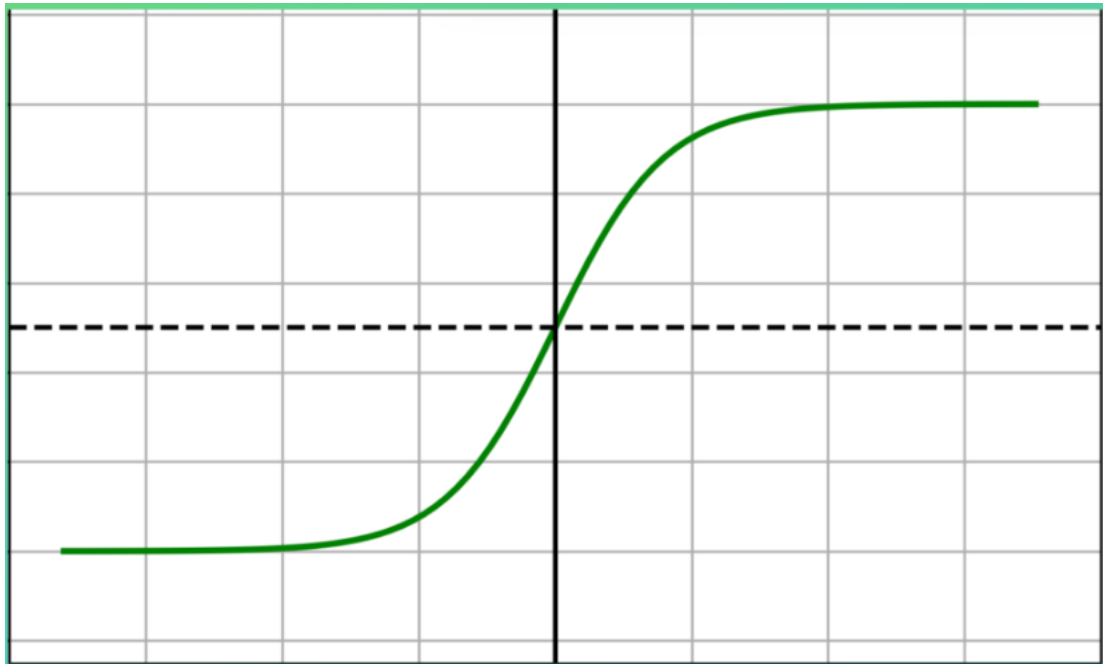
The model, on the other hand, will stop learning if the slope is zero.

A gradient is used to measure the change in all weights in relation to the change in error.

Two issues of Standard RNNs

Vanishing Gradients: Vanishing gradients occur when the gradient values are too small, causing the model to stop learning or take far too long. This was a big issue in the 1990s, and it was far more difficult to address than the exploding gradients. Fortunately, Sepp Hochreiter and Juergen Schmidhuber's LSTM concept solved the problem.

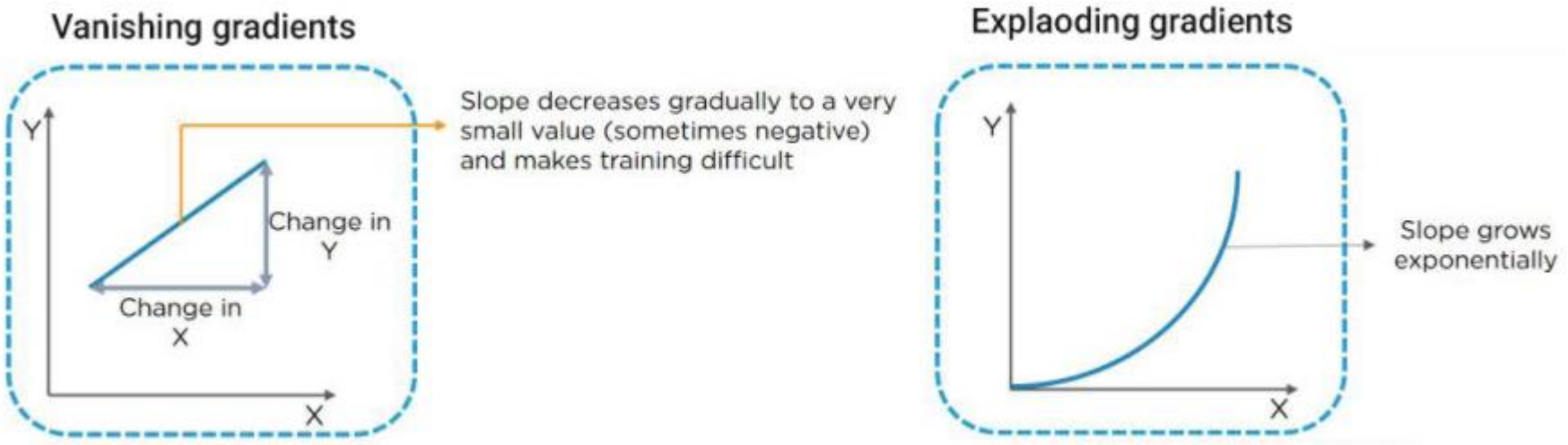
The vanishing gradient problem describes a situation encountered in the training of neural networks where the gradients used to update the weights shrink exponentially. As a consequence, the weights are not updated anymore, and learning stalls



Two issues of Standard RNNs

Exploding Gradients: Exploding gradients occur when the algorithm gives the weights an absurdly high priority for no apparent reason. Fortunately, truncating or squashing the gradients is a simple solution to this problem.

Exploding gradients can cause problems in the training of artificial neural networks. When there are exploding gradients, **an unstable network can result and the learning cannot be completed. The values of the weights can also become so large as to overflow and result in something called NaN values**



Why Vanishing Gradient Problem Occurs

- **Recurrent Neural Networks** uses **a hyperbolic tangent function**, what we call the tanh function.
- The range of this activation function lies between **[-1,1],** with its derivative ranging from [0,1].
- Now we know that **RNNs are a deep sequential neural network.** Hence, **due to its depth, the matrix multiplications continually increase in the network as the input sequence keeps on increasing.**
- Hence, while we use the **chain rule of differentiation during calculating backpropagation, the network keeps on multiplying the numbers with small numbers.**
- **When you keep on multiplying a number with negative values with itself? It becomes exponentially smaller, squeezing the final gradient to almost 0, hence weights are no more updated, and model training halts.**
- It leads to poor learning, which we say as **"cannot handle long term dependencies"** when we speak about RNNs.

Vanishing & Exploding Gradient Problem

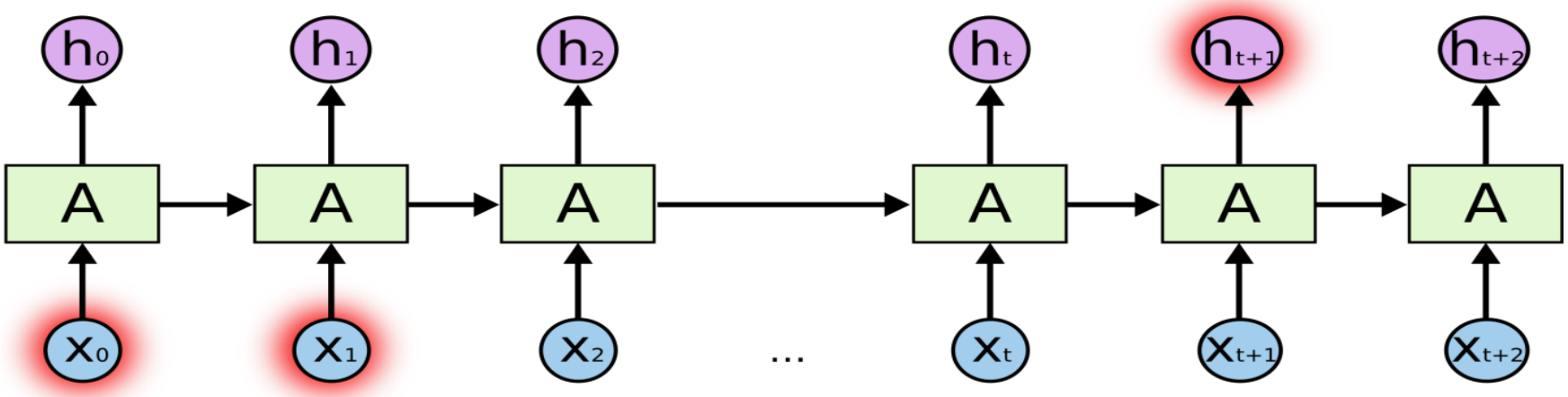
- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large).
- Makes it very difficult train deep networks, or simple recurrent networks over many time steps.

Why Exploding Gradient Problem Occurs

- Similar concept to the vanishing gradient problem, but just the opposite of the process.
- Suppose in this case our **gradient value is greater than 1 and multiplying a large number to itself makes it exponentially larger leading to the explosion of the gradient.**

Long Distance Dependencies

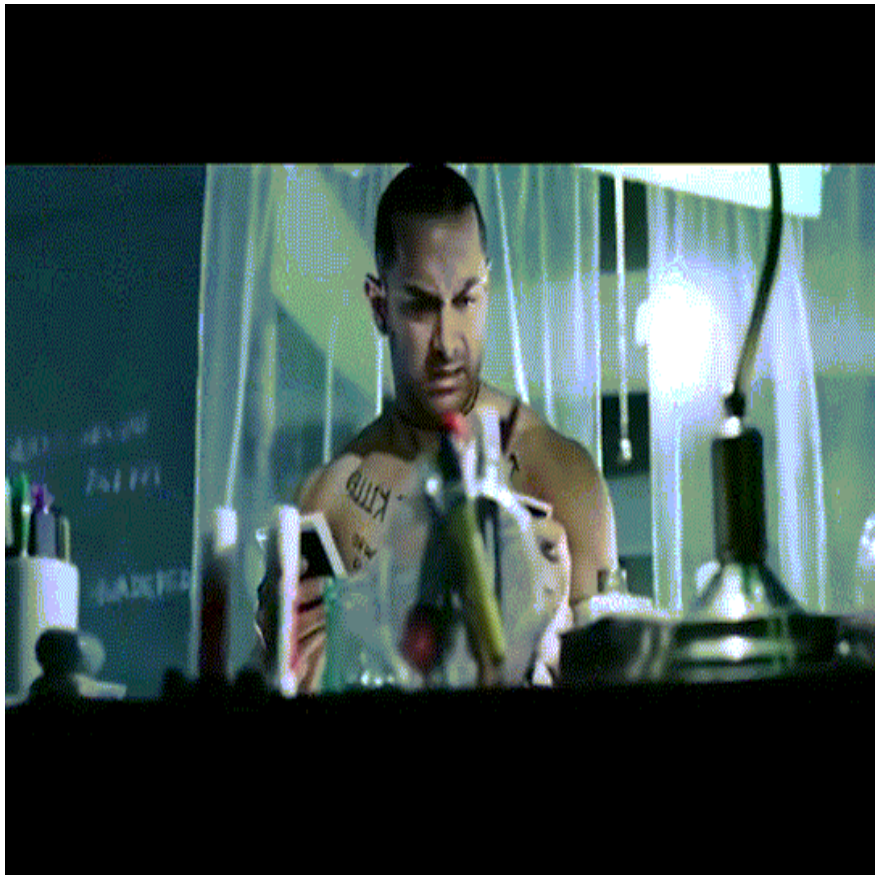
- It is very difficult to train RNNs to retain information over many time steps
- This make is very difficult to learn RNNs that handle long-distance dependencies, such as subject-verb agreement.

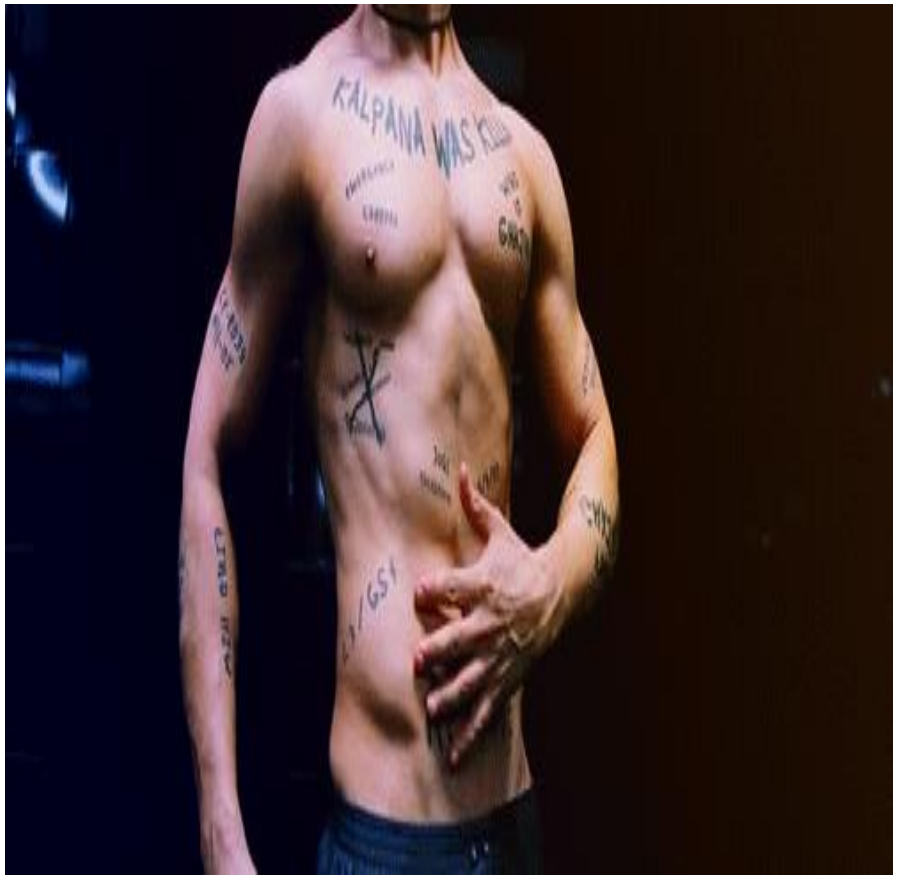


Vanishing Gradient Problem

Cannot handle Long Term Dependencies

Long Short-Term Memory





Can handle Long Term Dependencies

Long Term Dependency Issue in RNNs

Let us consider a sentence-

"I am an IT student and I love Machine _____."

- We know the blank has to be filled with 'Learning'. But had there been many terms after "I am an IT student" like, "I am an IT student pursuing MS from University of…… and I love Machine _____".

- This time, however, RNNS fails to work. Likely in this case we do not need unnecessary information like "pursuing MS from University of……".
- **What LSTMs do is, leverage their forget gate to eliminate the unnecessary information, which helps them handle long-term dependencies.**

Why LSTM when we have RNN?

- A sentence or phrase only holds meaning when every word in it is associated with its previous word and the next one.
- LSTM, short for Long Short-Term Memory, as opposed to RNN, extends it by creating both **short-term and long-term memory components** to efficiently study and learn sequential data.
- Hence, it's great for Machine Translation, Speech Recognition, time-series analysis, etc.

Long Short-Term Memory

Long Short-Term Memory is an artificial recurrent neural network architecture used in the field of Deep Learning.

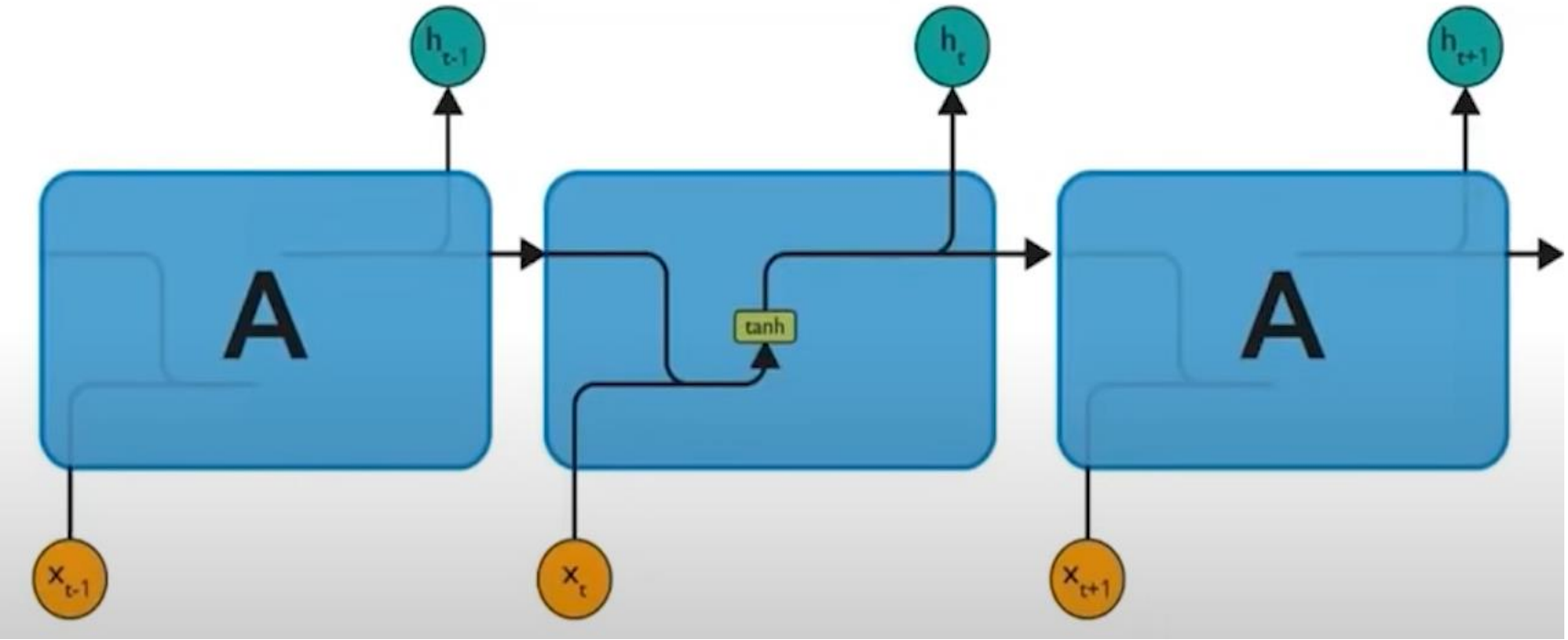
Unlike standard Feedforward neural Networks, LSTM has feedback connections.

It can **not only process Single data points, but also entire sequence of data.**

Long Short-Term Memory

Long Short-Term Memory Networks-Usually just called "LSTMs"-are a special kind of RNN.

They are capable of learning long-term dependencies.



The repeating module in a Standard RNN contains a single layer.

LSTM Specific Terms

1. Cell—Every unit of the LSTM network is known as a "cell". Each cell is composed of 3 inputs—

- x(t)—token at timestamp t
- h(t−1)—previous hidden state
- c(t-1)—previous cell state,
- h(t)—updated hidden state, used for predicting the output
- c(t)—current cell state

2. Gates

LSTM uses a special theory of controlling the memorizing process. Popularly referred to as gating mechanism in LSTM.

What the gates in LSTM do is, store the memory components in analog format, and make it a probabilistic score by doing point-wise multiplication using sigmoid activation function, which stores it in the range of 0–1.

Gates in LSTM regulate the flow of information in and out of the LSTM cells.

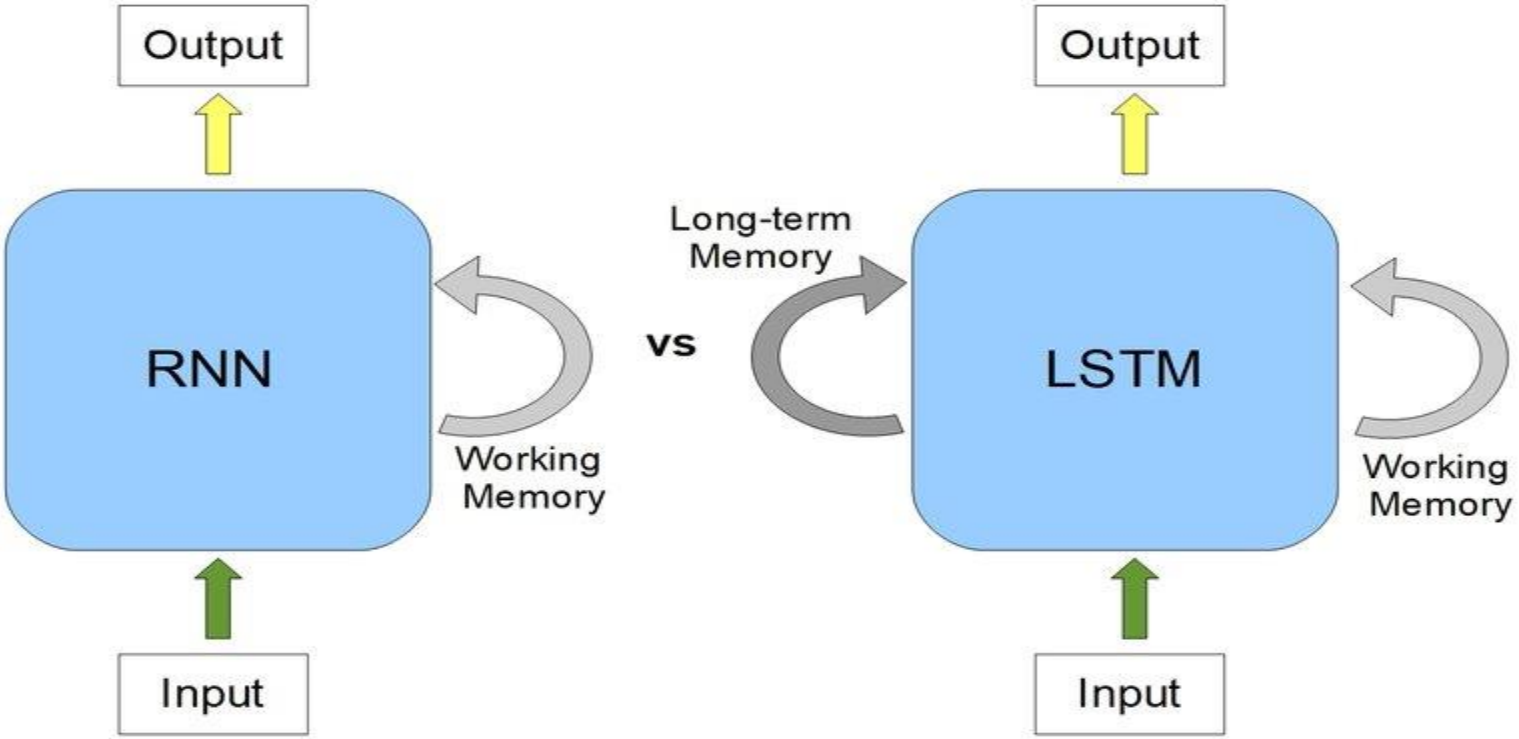
LSTM Specific Terms

Gates are of 3 types

1. **Input Gate—**This gate lets in optional information necessary from the current cell state. It decides which information is relevant for the current input and allows it in.
2. **Output Gate—**This gate updates and finalizes the next hidden state. Since the hidden state contains critical information about previous cell inputs, it decides for the last time which information it should carry for providing the output.
3. **Forget Gate—**Pretty smart in eliminating unnecessary information, the forget gate multiplies 0 to the tokens which are not important or relevant and lets it be forgotten forever.

Why does LSTM outperform RNN?

- **First, let's take a comparative look into an RNN and an LSTM**



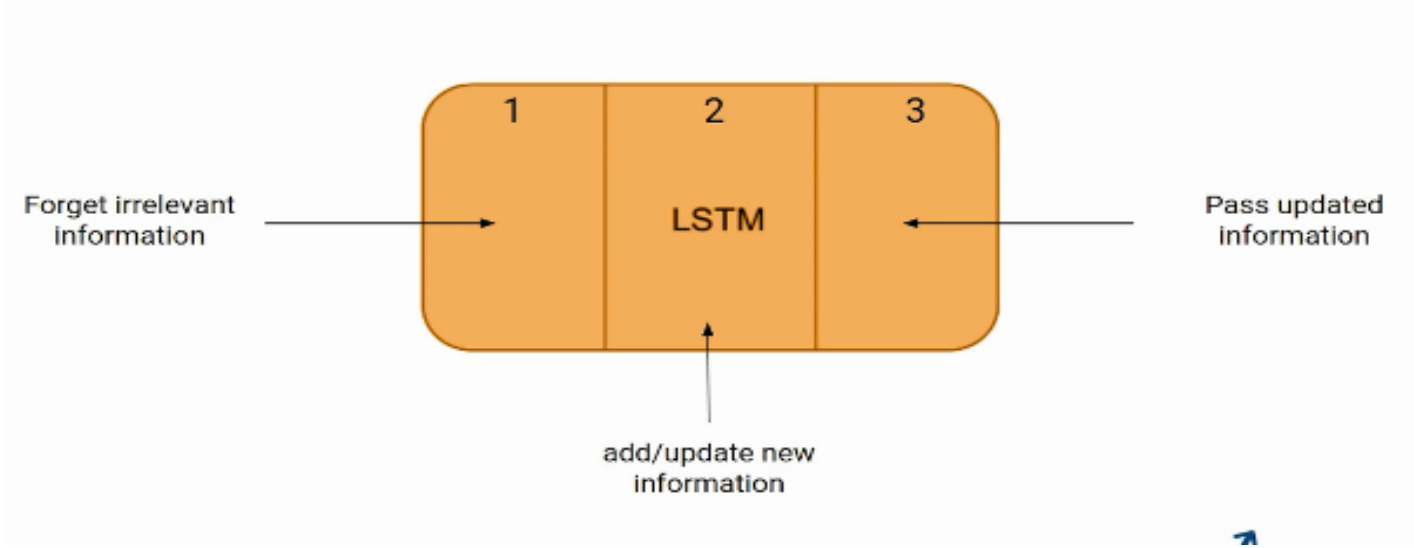
RNNs have quite massively proved their incredible performance in sequence learning.

But it has been remarkably noticed that RNNs are not sporty while handling long-term dependencies.

LSTM Architecture

LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network.

The LSTM network architecture consists of three parts, as shown in the image below,



https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

LSTM Architecture

These three parts of an LSTM unit are known as gates.

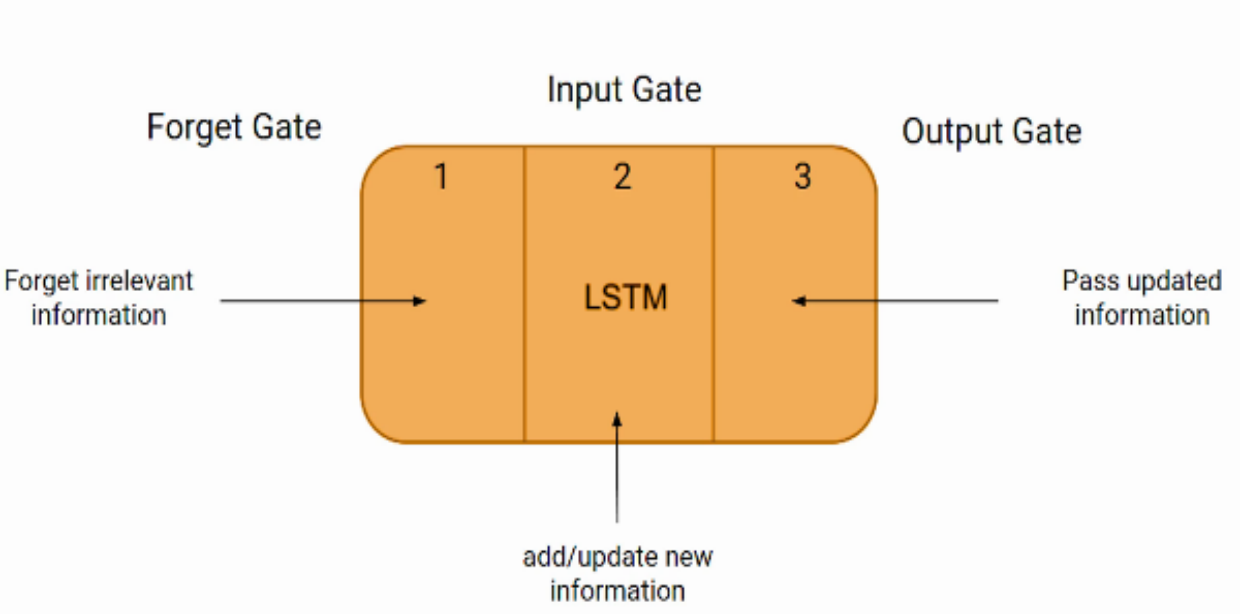
They control the flow of information in and out of the memory cell or LSTM cell.

The first gate is called **Forget gate,**

The second gate is known as **the Input gate.**

The last one is **the Output gate**.

An LSTM unit that consists of these three gates and a memory cell or LSTM cell can be considered as a layer of neurons in traditional feedforward neural network, with each neuron having a hidden layer and a current state.



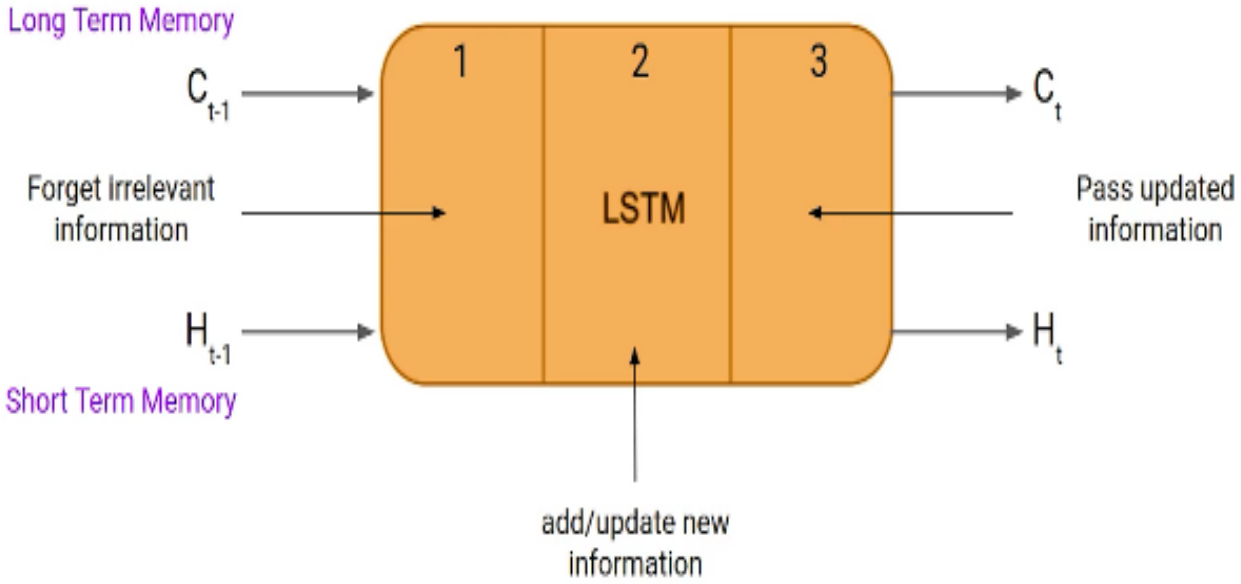
LSTM Architecture

Just like a simple RNN, **an LSTM also has a hidden state where H(t-1)** represents the hidden state of the previous timestamp.

Ht is the hidden state of the current timestamp.

In addition to that, **LSTM also has a cell state represented by C(t-1) and C(t) for the previous and current timestamps, respectively.**

Here the **hidden state is known as Short term memory**, and **the cell state is known as Long term memory.**



LSTM Architecture

It is interesting to note that the cell state carries the information along with all the timestamps.

LSTM

Bob is a nice person. Dan on the other hand is evil.

Let's take an example to understand how LSTM works. Here we have two sentences separated by a full stop.

The first sentence is **"Bob is a nice person**," and the second sentence is **"Dan, on the Other hand, is evil".** It is very clear, in the first sentence, we are talking about Bob, and as soon as we encounter the full stop(.), we started talking about Dan.

As we move from the first sentence to the second sentence, our network should realize that we are no more talking about Bob. Now our subject is Dan. Here, the Forget gate of the network allows it to forget about it.

Forget gate

In a cell of the LSTM neural network, the first step is to decide whether we should keep the information from the previous time step or forget it. Here is the equation for forget gate.

Forget Gate:

- $f_t = \sigma (x_t * U_f + H_{t-1} * W_f)$

Let's try to understand the equation, here

- Xt: input to the current timestamp.
- Uf: weight associated with the input
- Ht-1: The hidden state of the previous timestamp
- Wf: It is the weight matrix associated with the hidden state

Later, **a sigmoid function is applied to it.** That will make ft a number between 0 and 1. This ft is later multiplied with the cell state of the previous timestamp, as shown below.

$C_{t-1} * f_t = 0$ ...if $f_t = 0$ (forget everything)

$C_{t-1} * f_t = C_{t-1}$ ...if $f_t = 1$ (forget nothing)

Input Gate

Input Gate:

- $i_t = \sigma (x_t * U_i + H_{t-1} * W_i)$

Here,

•Xt: Input at the current timestamp t

•Ui: weight matrix of input

•Ht-1: A hidden state at the previous timestamp

•Wi: Weight matrix of input associated with hidden state

Again, we have applied the sigmoid function over it. As a result, the value of I at timestamp t will be between 0 and 1.

Let's take another example.

"Bob knows swimming. He told me over the phone that he had served the navy for four long years."

The input gate is used to quantify the importance of the new information carried by the input.

New Information

- $N_t = \tanh(x_t * U_c + H_{t-1} * W_c)$ (new information)

Now the new information that needed to be passed to the cell state is a function of a hidden state at the previous timestamp t-1 and input x at timestamp t.

The activation function here is tanh. **Due to the tanh function, the value of new information will be between -1 and 1.**

If the value of Nt is negative, the information is subtracted from the cell state, and if the value is positive, the information is added to the cell state at the current timestamp.

However, the Nt won't be added directly to the cell state. Here comes the updated equation:

$$C_t = f_t * C_{t-1} + i_t * N_t \text{ (updating cell state)}$$

Here, Ct-1 is the cell state at the current timestamp, and the others are the values we have calculated previously.

Output Gate

Here is the equation of the Output gate, which is pretty similar to the two previous gates.

Output Gate:

- $o_t = \sigma\,(x_t * U_o + H_{t-1} * W_o)$

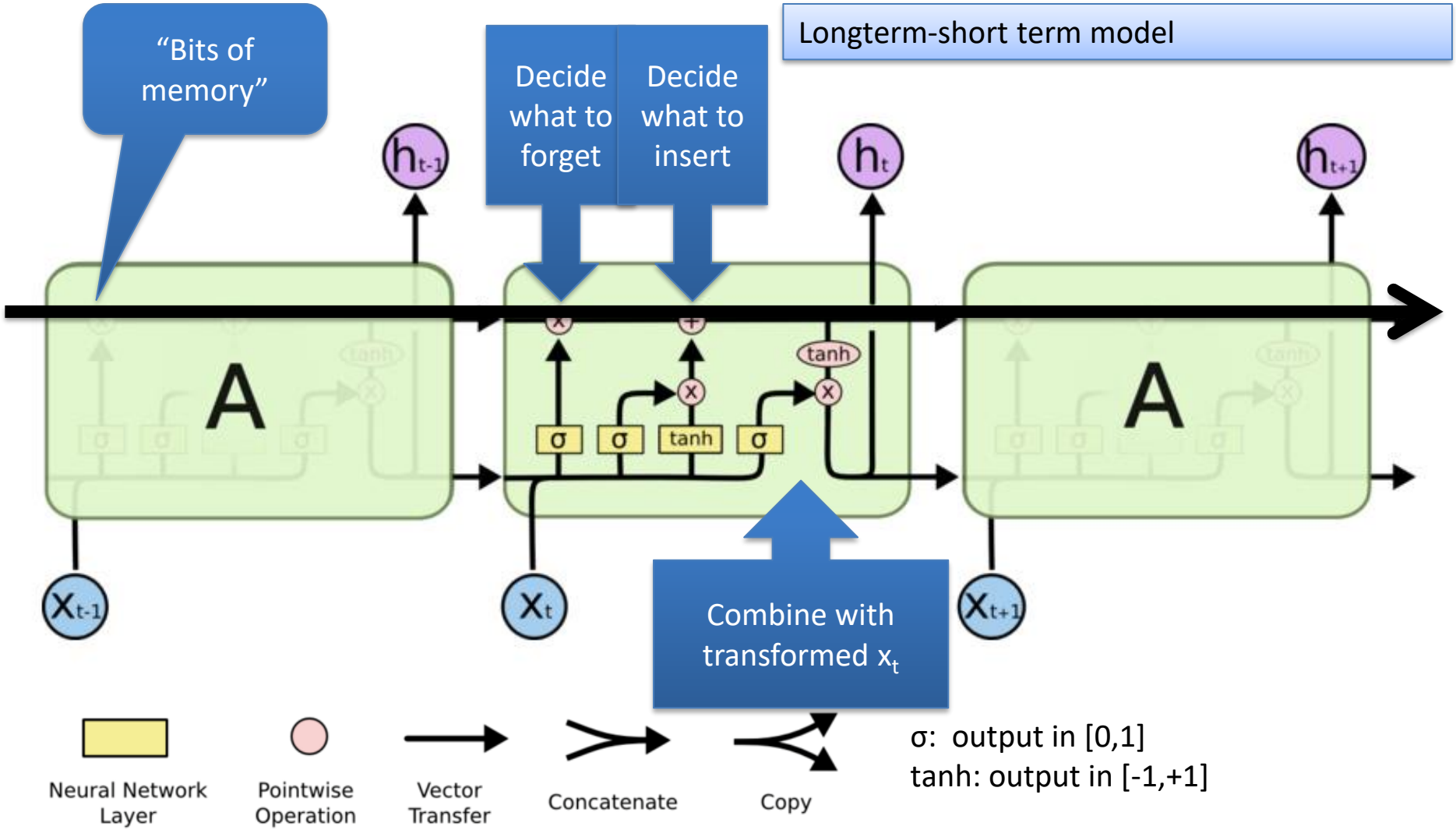
Its value will also lie between 0 and 1 because of this sigmoid function. Now to calculate the current hidden state, we will use Ot and tanh of the updated cell state. As shown below.

$$H_t = o_t * \tanh(C_t)$$

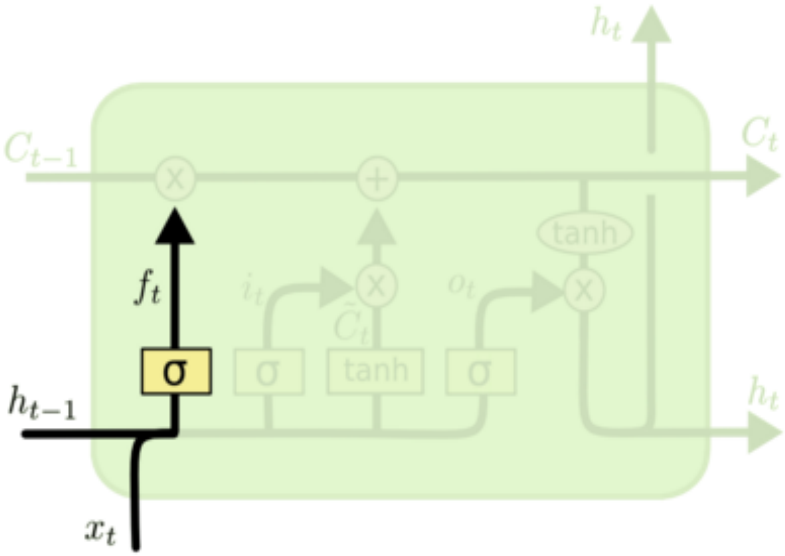
It turns out that the hidden state is a function of Long term memory (Ct) and the current output. If you need to take the output of the current timestamp, just apply the SoftMax activation on hidden state Ht.

$$\text{Output} = \text{Softmax}(H_t)$$

Architecture for an LSTM



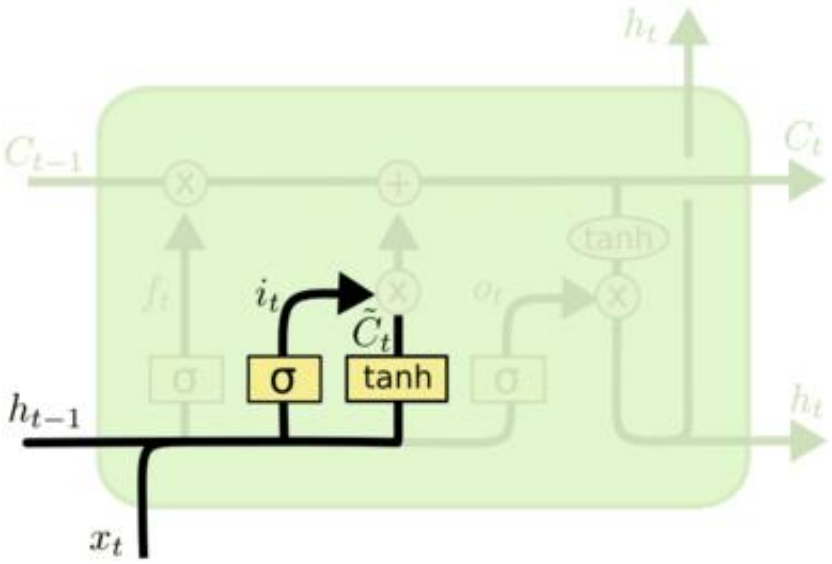
Walkthrough



What part of memory to "forget" – zero means forget this bit

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

Walkthrough



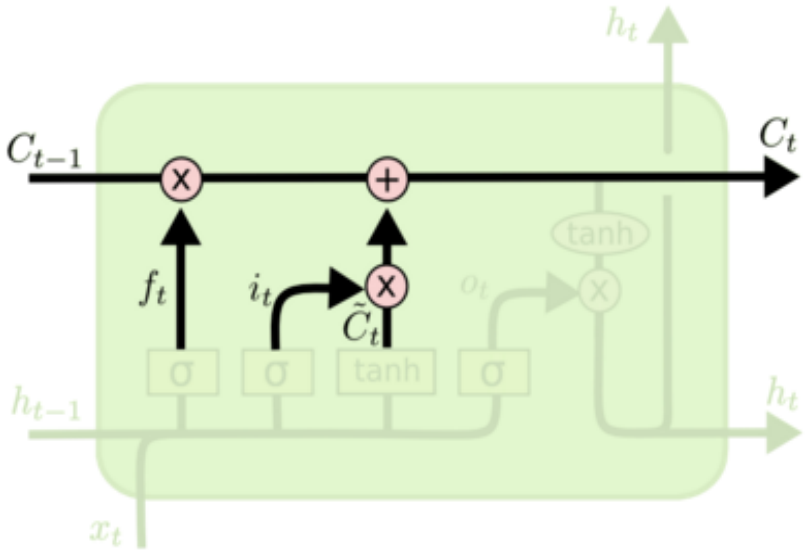
What bits to insert into the next states

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

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

What content to store into the next state

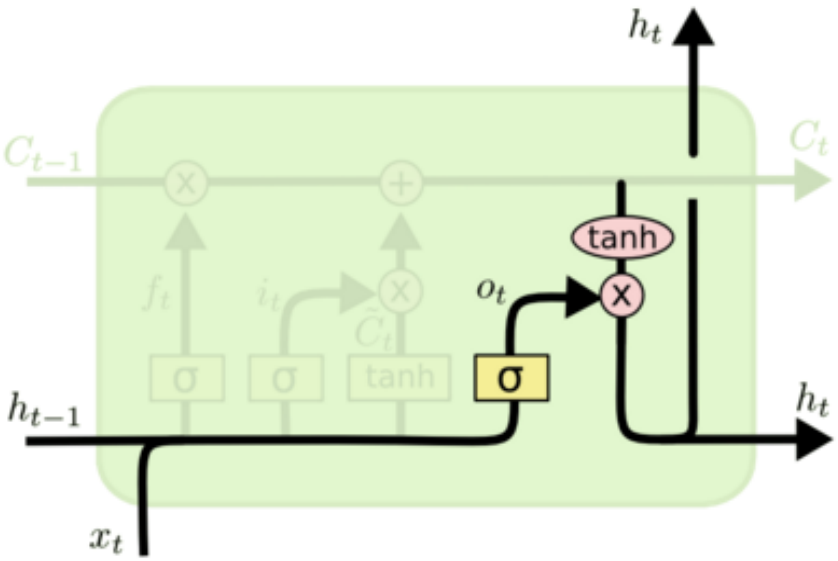
Walkthrough



Next memory cell content – mixture of not-forgotten part of previous cell and insertion

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

Walkthrough



What part of cell to output

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

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

tanh maps bits to [-1,+1] range

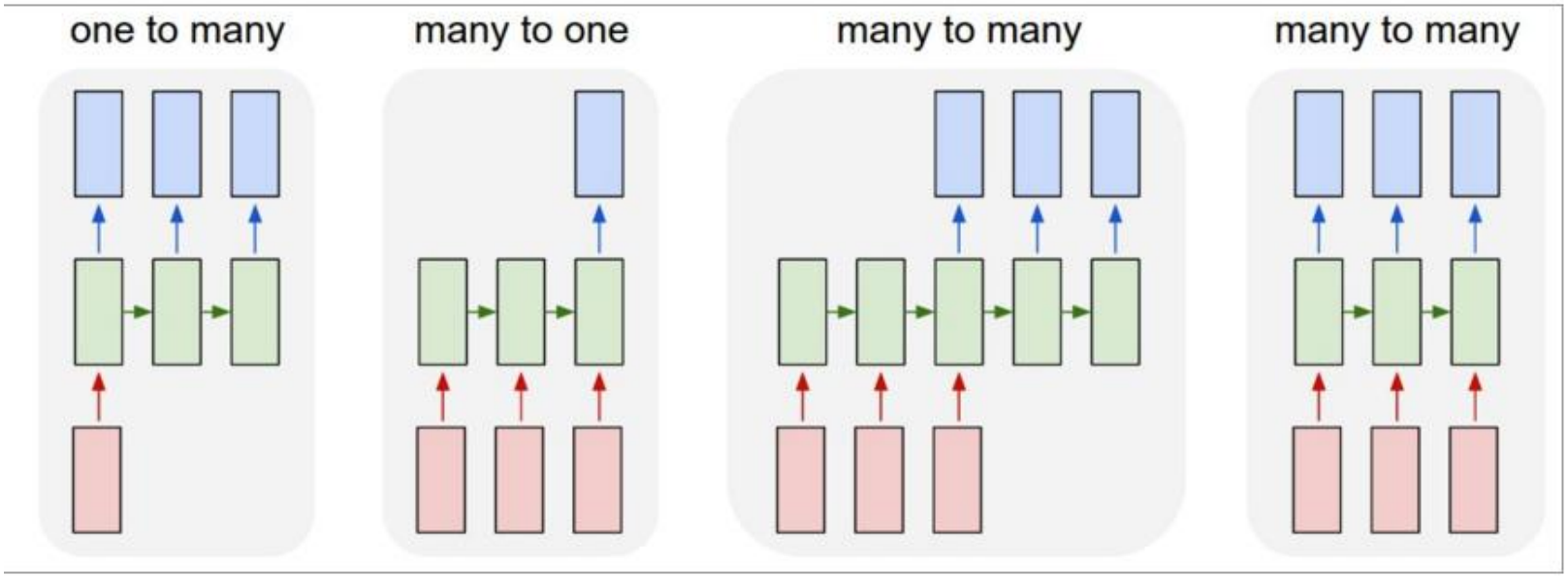
LSTMs can be used for other sequence tasks

image captioning

sequence classification

translation

named entity recognition

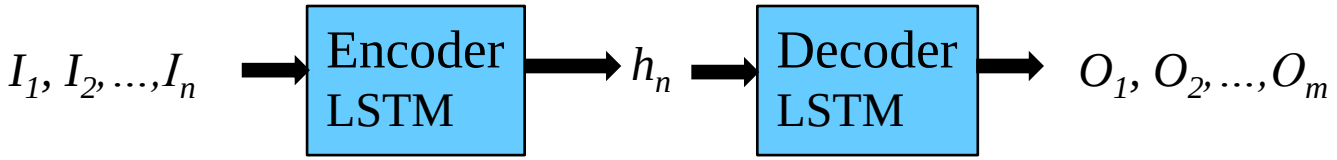


General Problems Solved with LSTMs

- Sequence labeling
  - Train with supervised output at each time step computed using a single or multilayer network that maps the hidden state (h_t) to an output vector (O_t).
- Language modeling
  - Train to predict next input ($O_t = I_{t+1}$)
- Sequence (e.g. text) classification
  - Train a single or multilayer network that maps the final hidden state (h_n) to an output vector (O).

Sequence to Sequence Transduction (Mapping)

- Encoder/Decoder framework maps one sequence to a "deep vector" en another LSTM maps this vector to an output sequence.



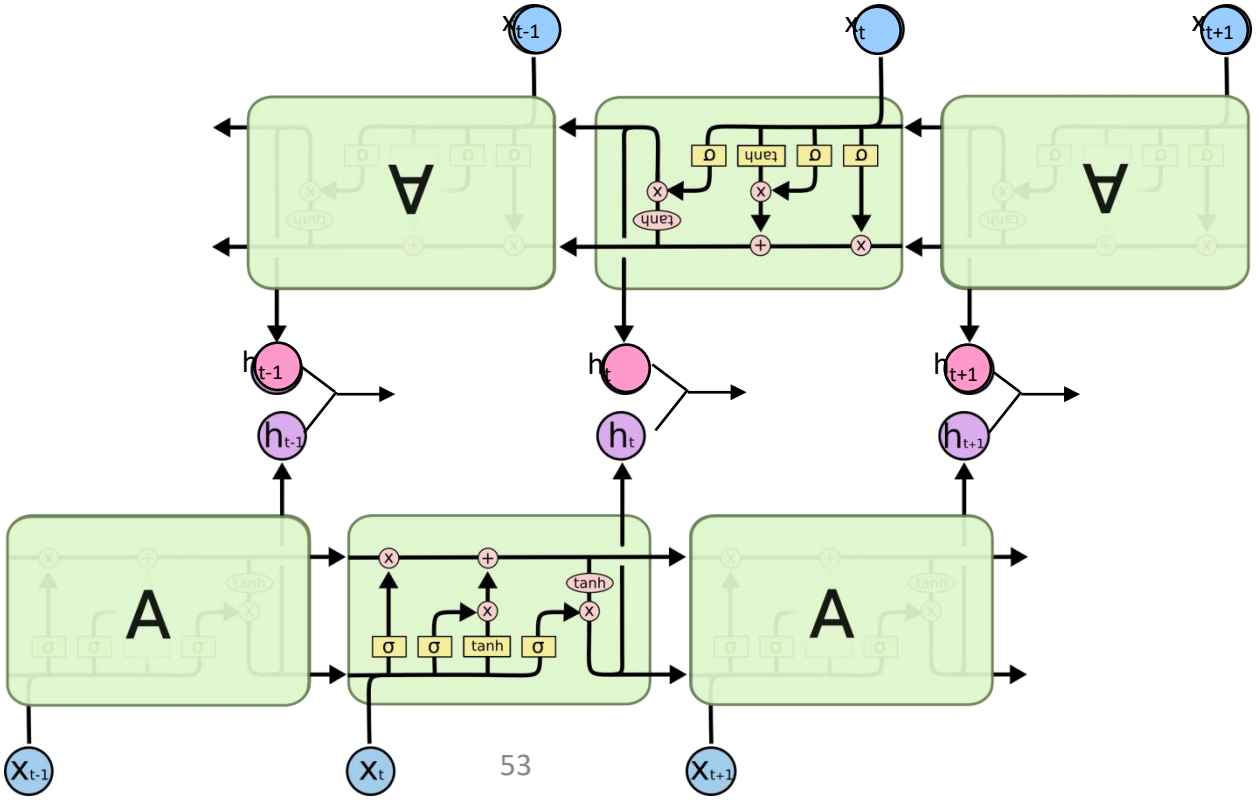
- Train model "end to end" on I/O pairs of sequences.

Successful Applications of LSTMs

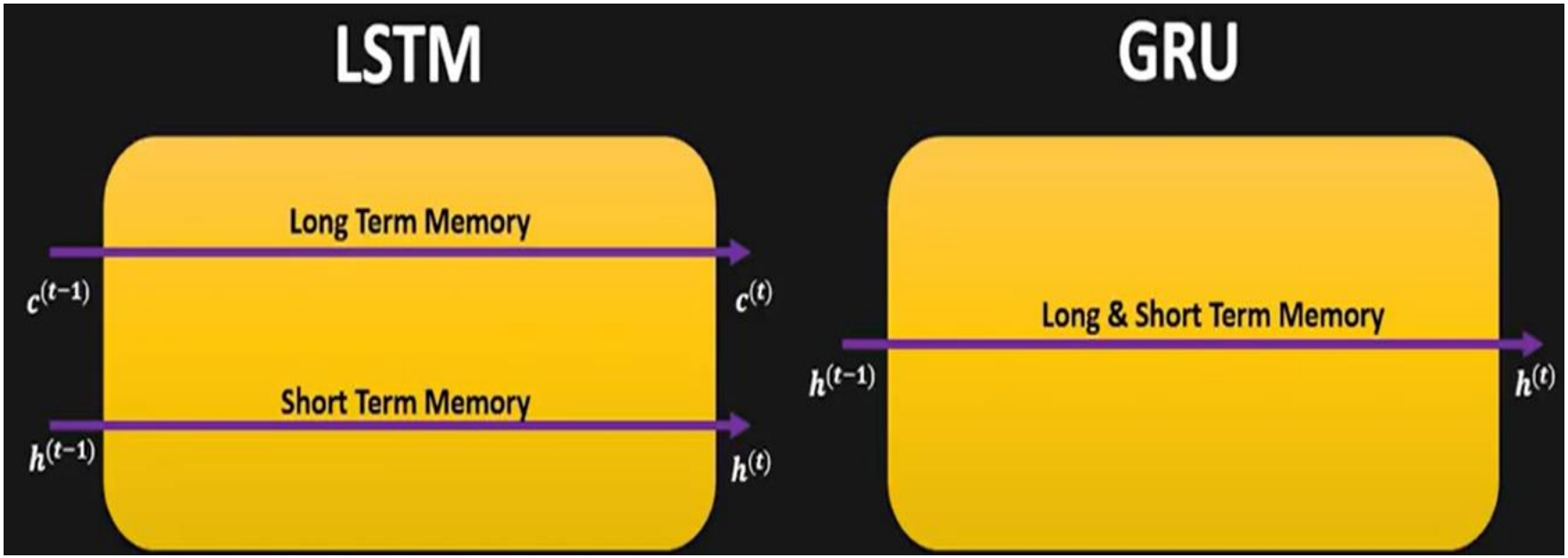
- Speech recognition: Language and acoustic modeling
- Sequence labeling
  - Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning: CNN output vector to sequence
- Sequence to Sequence
  - Machine Translation (Sustkever, Vinyals, & Le, 2014)
  - Video Captioning (input sequence of CNN frame outputs)

Bi-directional LSTM (Bi-LSTM)

- **A Bidirectional LSTM**, or biLSTM, is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction.
- The output layer can simultaneously get information from past (backward), and future (forward) states.



Gated Recurrent Unit (GRU)

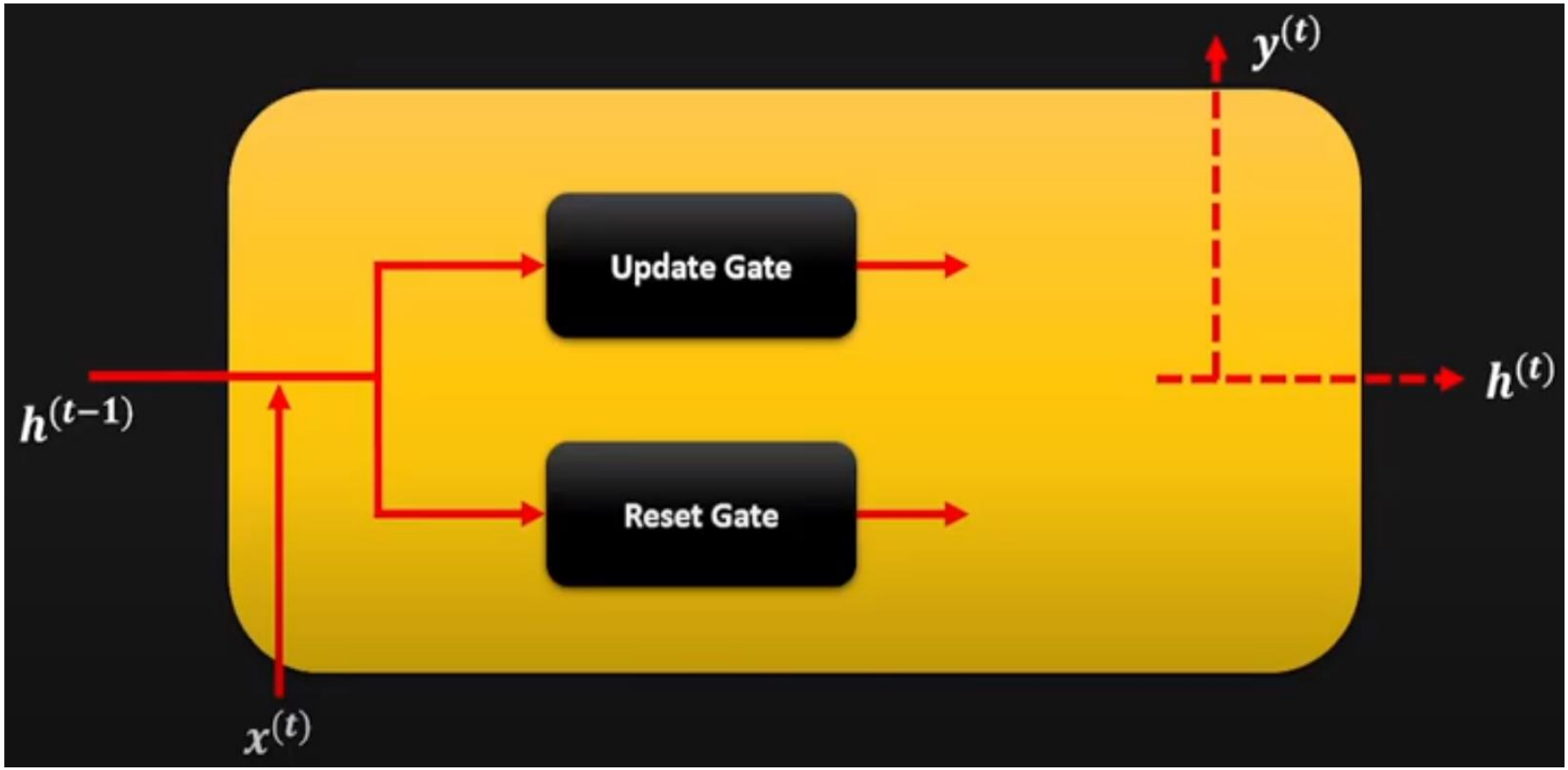


The Key Difference between GRU and LSTM is that GRU's bag has two gates that are reset and update while LSTM has three gates that are input, output, forget.

GRU is less complex than LSTM because it has less number of gates.

If the dataset is small, then GRU is preferred otherwise LSTM for the larger dataset.

Gated Recurrent Unit (GRU)



LSTM has three gates-Input Gate, Output Gate and Forgot Gate.

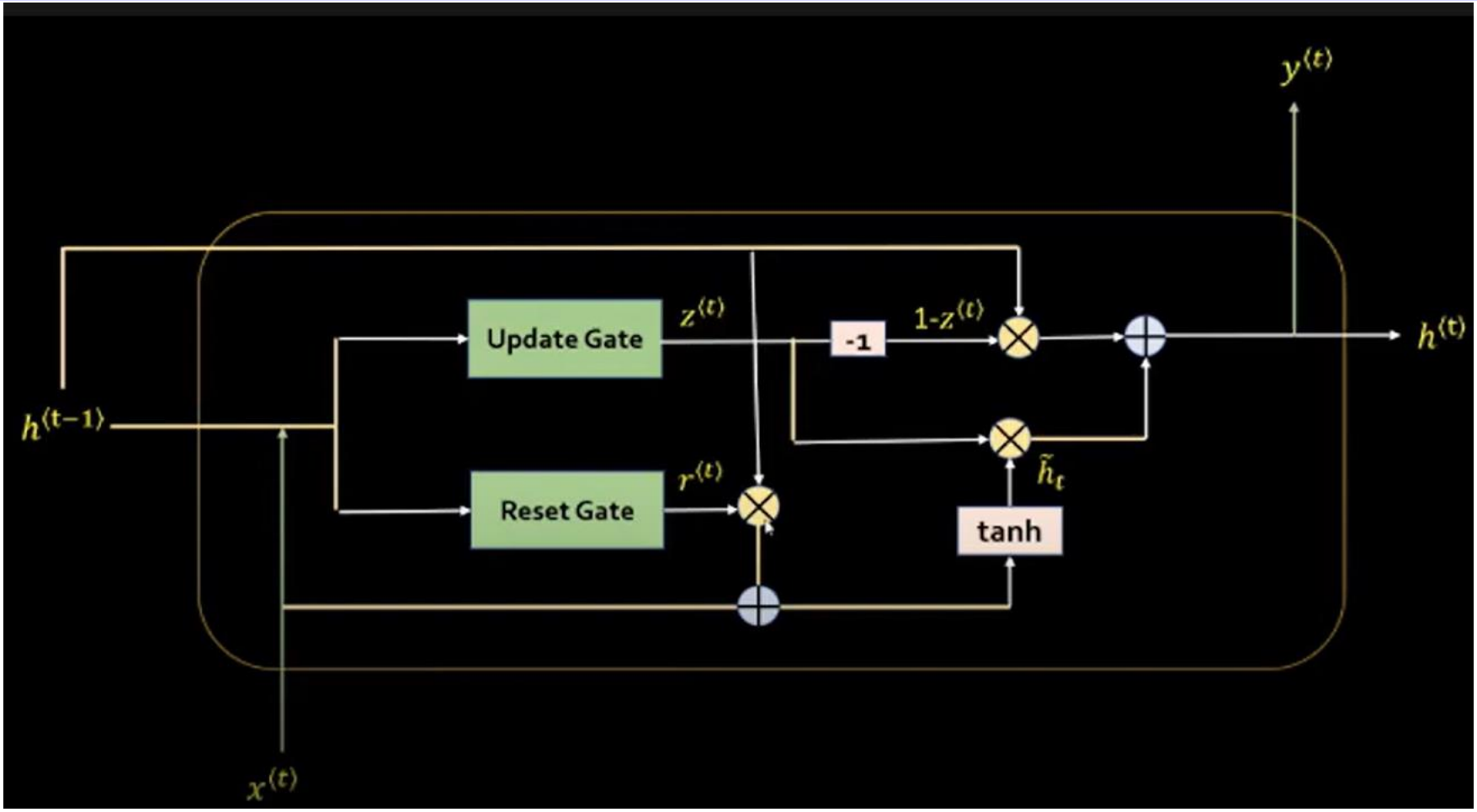
GRU has two Gates

Update Gate and Reset Gate.

Update Gate-How much of the Past memory to retain.

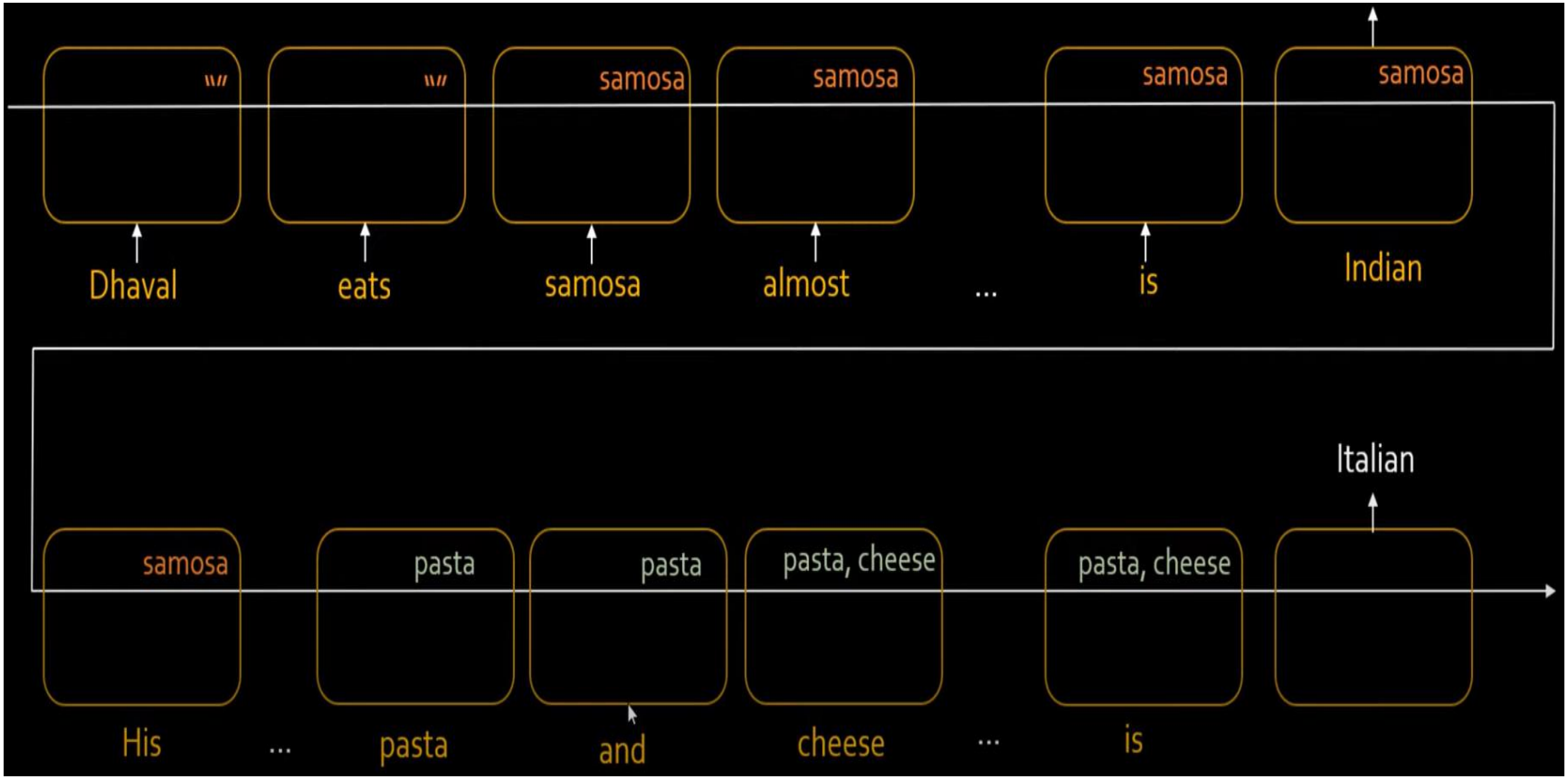
Reset Gate-How much of Past memory to forget

Gated Recurrent Unit (GRU)

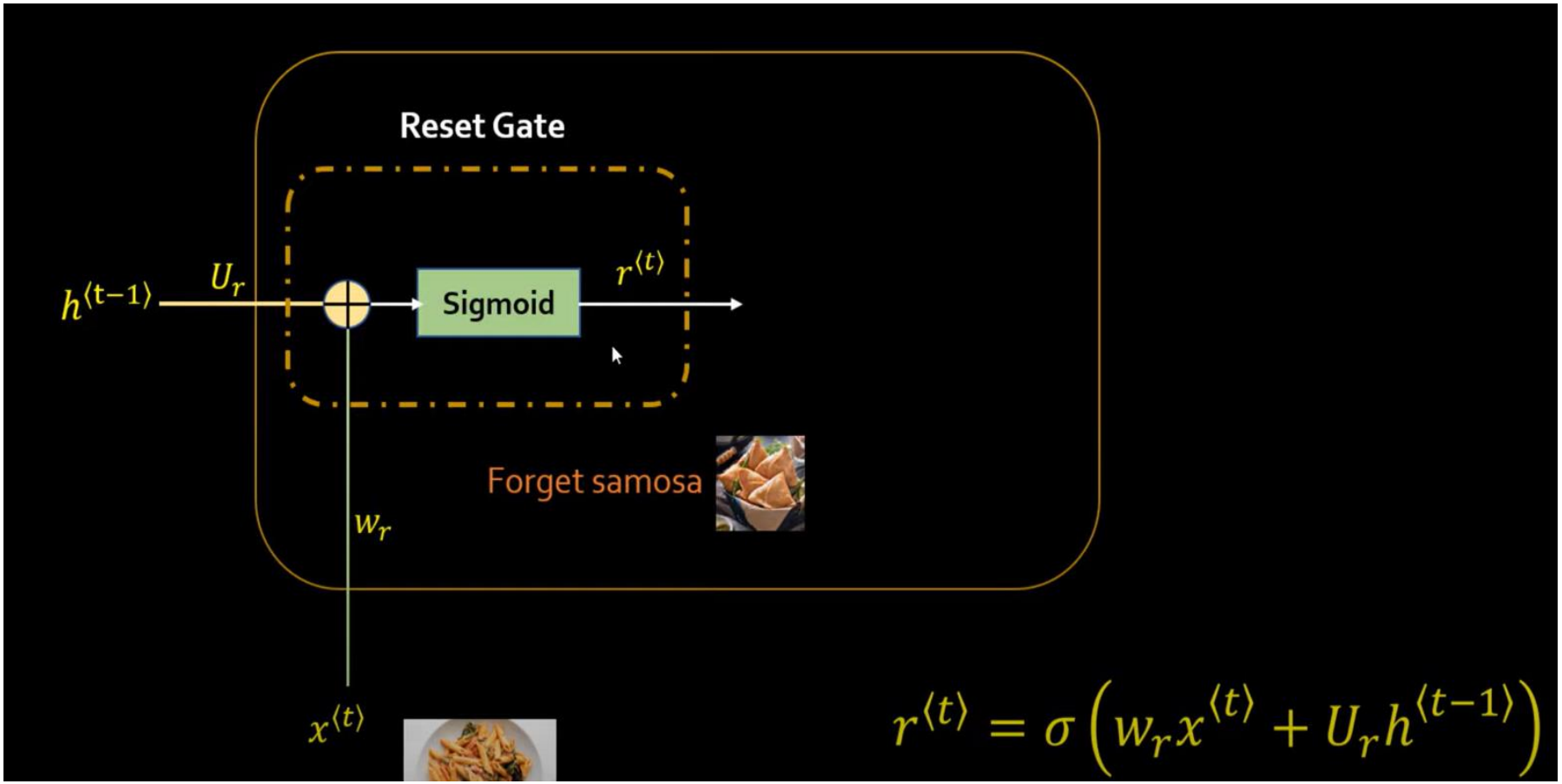


Gated Recurrent Unit (GRU)

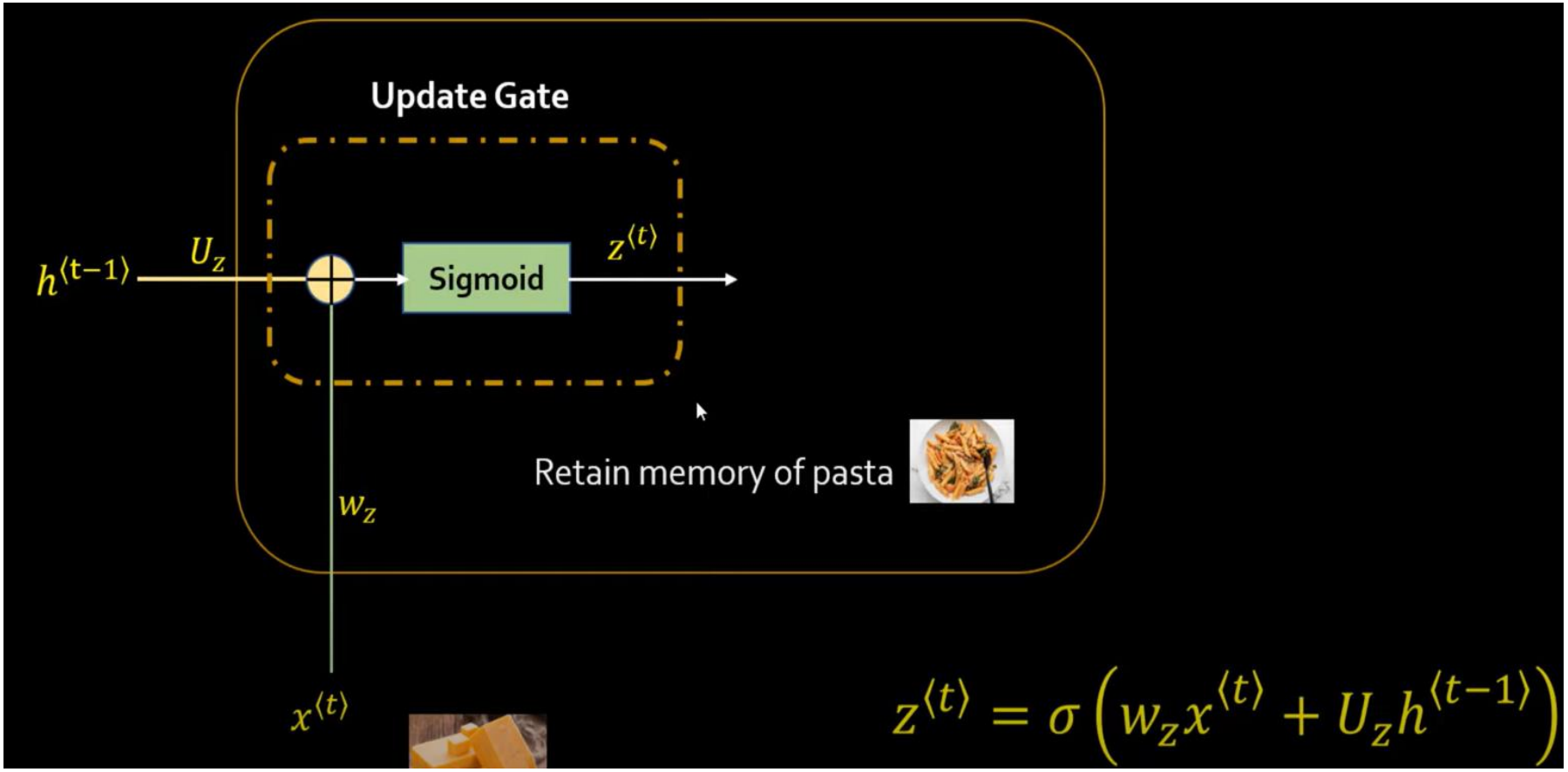
Dhaval eats **samosa** almost everyday; it shouldn't be hard to guess that his favorite cuisine is **Indian.** His Brother Bhavin however is a lover of **pasta and cheese** that means Bhavin's favorite cuisine is **Italian.**



Gated Recurrent Unit (GRU)



Gated Recurrent Unit (GRU)



GRU vs. LSTM

- GRU has significantly fewer parameters and trains faster.
- Experimental results comparing the two are still inconclusive, many problems they perform the same, but each has problems on which they work better.

LSTM	GRU
3 Gates: Input, output, forget	2 Gates: reset, update
More accurate on longer sequence, less efficient	More efficient computation wise. Getting more popular
Invented: 1995 - 1997	Invented: 2014

Conclusions

- By adding "gates" to an RNN, we can prevent the vanishing/exploding gradient problem.
- Trained LSTMs/GRUs can retain state information longer and handle long-distance dependencies.
- Recent impressive results on a range of challenging NLP problems.


ThankYou
Merci
Hvala
Kop
Salamat
Xie
Gra
kun
Tack
Dhanyavad
Khopjai
Kruthagnathalu
Arigatou
Dhonnobaad
Gomapsupnida
Terima
Danke
Enkosi
dank
Euxaristo
Kun
Shukriya
Gracias
Nandree
Blagodariya
Fyrir
Grazzi
raibh
krap
Dakujem
Waad
Daw
Dhanyavaadaalu
Dank
Takk
Gamsahapnida
Dekuju/Dekujeme
Ngiyabonga
Shokriya
Dziękuję
Shokrun
Cam
Dankie
Go
Grazie
Falemiderit
Dhanyavaad
Spaas
Mul
Ačі
or
al
Ederim
Hain
Dhan
Asante
daa
Diolch
Kiitos
Sheun
umesc
Shnorhakalutiun
Kasih
Mamnoon
Todah
Tesekkür