

## RNN Vs LSTM

Let's try to recognize images and we fitted this image. And neural network guesses that the image is most likely a dog with a small chance of being a wolf and an even smaller chance of being a goldfish.

But, what if this image is actually a wolf?

How would neural network know ?



Do let's say we are watching a Tv show about nature and the previous image before the wolf a bear and the previous one was a fox.



So, in this case, we want to use the information to hint to us that the last image is a wolf and not dog.

So what we do is analyze each image with the same copy of a neural network.

But we use the output of the neural network as a part of the input of the next one. And, that will actually improve our results.

**Mathematically, this is simple.**

**We just combine the vectors in a linear function, which will then be squished with an activation function which could be**

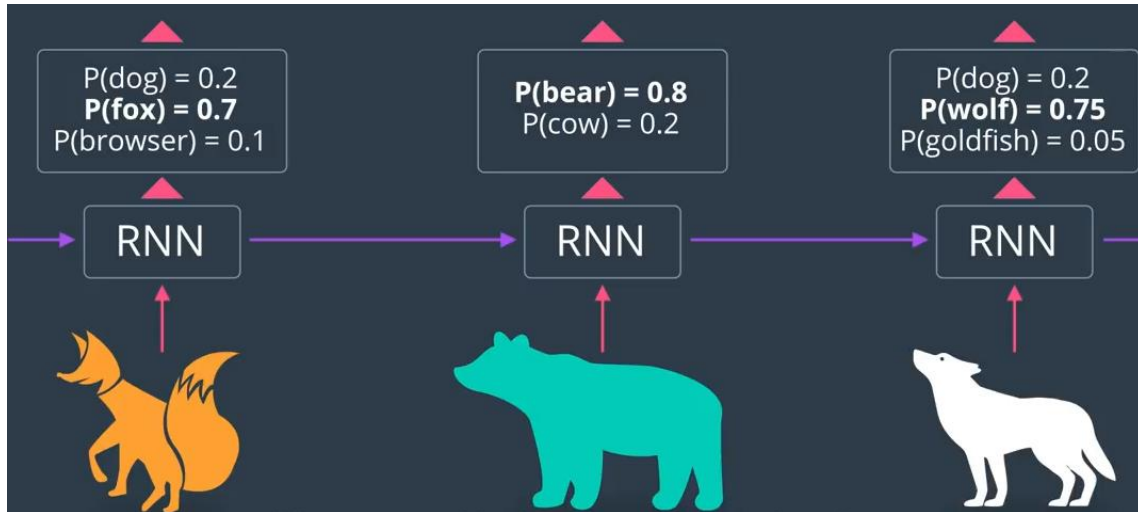
- 1- Sigmoid
- 2- Hyperbolic tan.

## LSTM

This way, we can use previous information and the final neural network will know that the show is about animals in the forest and actually use this information to correctly predicate the image is of a wolf and not a dog.

And, this is basically how recurrent neural network work.

However, this has some drawbacks.

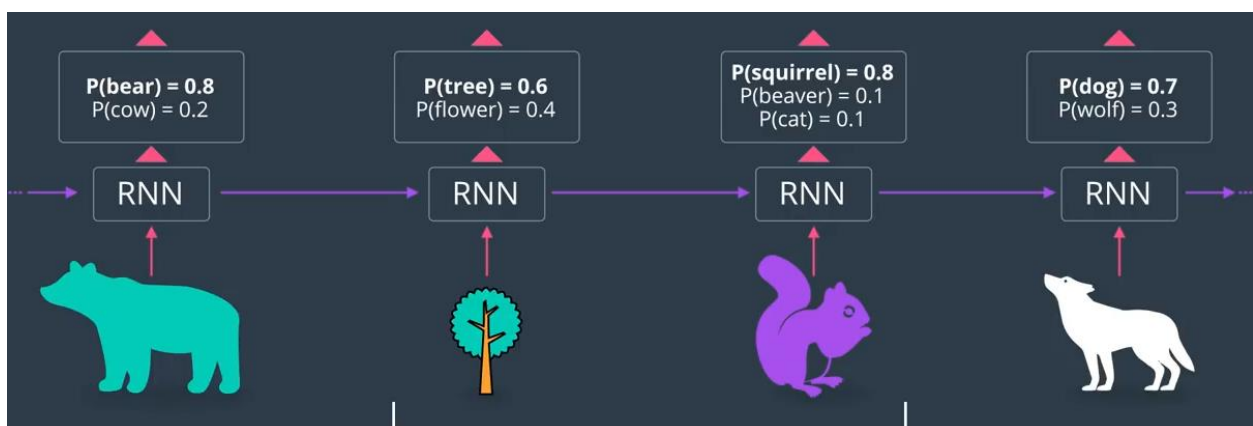


Let's say the bear appeared a while ago and the two recent images are the tree and a squirrel based on those two.

We don't really know if the new image is dog or a wolf. Since the trees and squirrels just are associated to domestic animals as they are with forest animals.

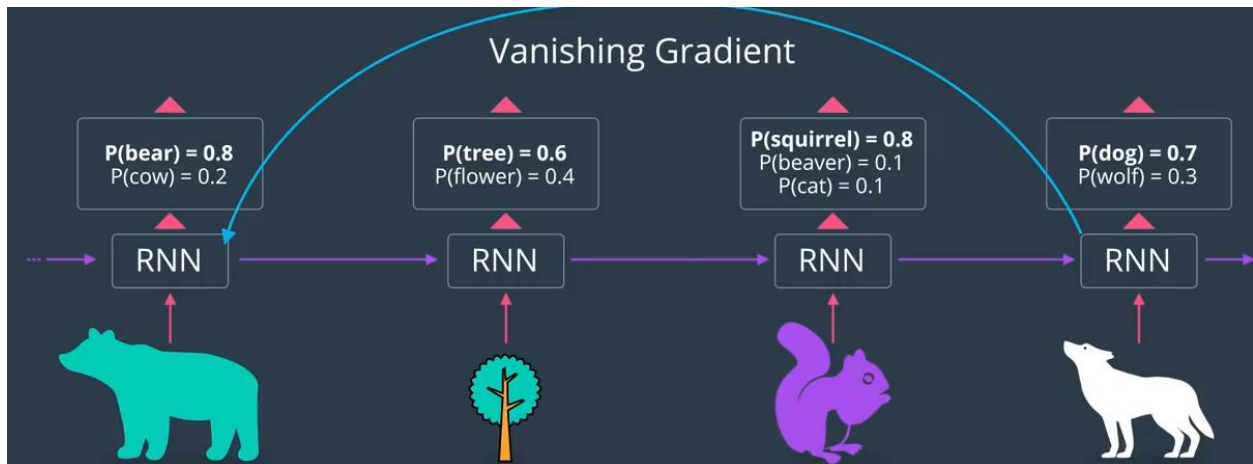
So, the information about being in the forest comes all the way back from the bear But, we have already experience information coming in gets repeatedly squished by

- 1- Sigmoid
- 2- Even worse that



Training the network using backpropagation all the way back, will lead to problems such as the

### 1- Vanishing gradient problem

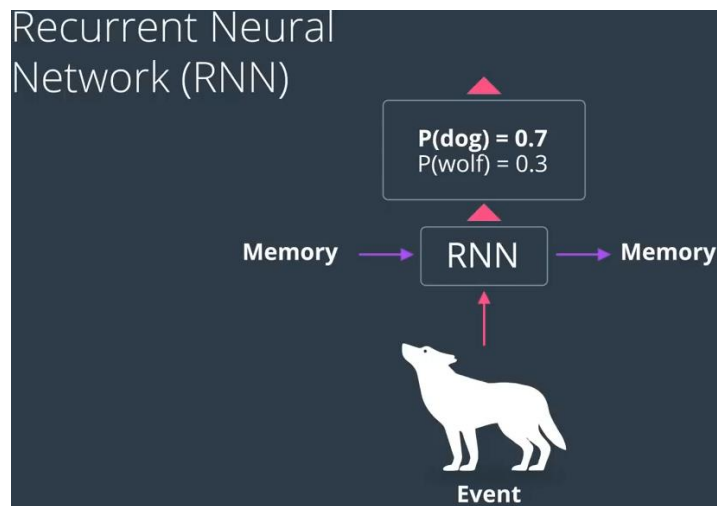


So, by this point pretty much all the bear information has been lost. That a problem with recurring networks. That the memory that is stored is normally short term memory.

RNNs, have a hard time storing long term memory and this is where LSTMs or long short term memory networks will come to the rescue.

So, small summary and RNN works as follows;

Memory comes in and merges with a **current event** and the output comes out as a prediction of what the input is.



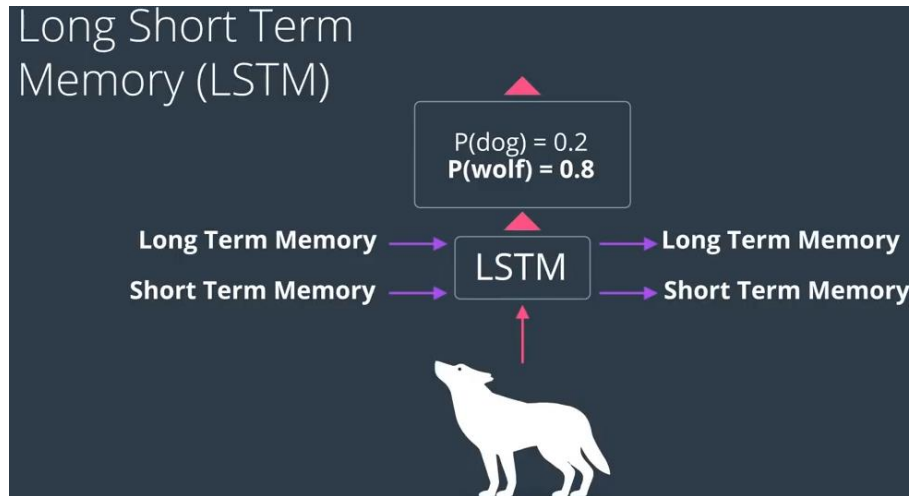
And similar way , an LSTM works as follows.

It keep track not just of memory but of long term memory which comes in and come out. And also , short term memory which also comes in and comes out. And every stage, the long- and short-term memory in the event get merged.

And form there, we get a new long term memory short term memory and prediction.

We protect all old information more.

If we deem it necessary, the network can remember things from long time ago.



### Basic architecture

#### Long Short-Term Memory (LSTM)

Let`s recap, and following the problem. Watching the tv show and we have al long term memory which is that the show is about nature and science a and lots of forest animal have appeared.

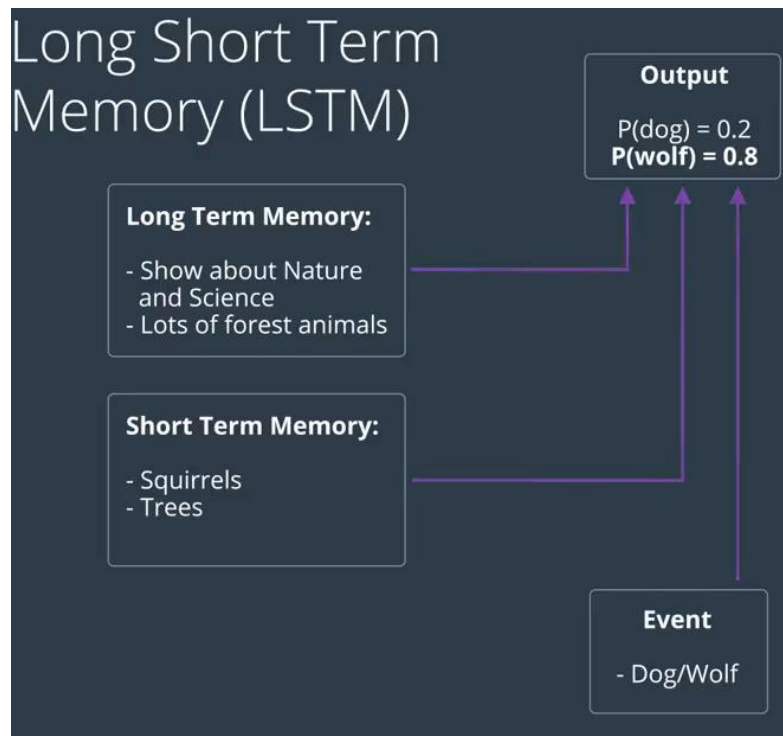
We also have a short-term memory which is what we have recently seen which is

- 1- Squirrels
- 2- Trees

And we have a **current event** which is what we just saw, the image of a dog which could also be a wolf.

And we want these three things to combine to form a prediction of what our image is.

In this case, the long term memory which says that the show is about forest animals will give us a hint that the picture is of a **wolf** and not a **dog**.



We also want the three pieces of information

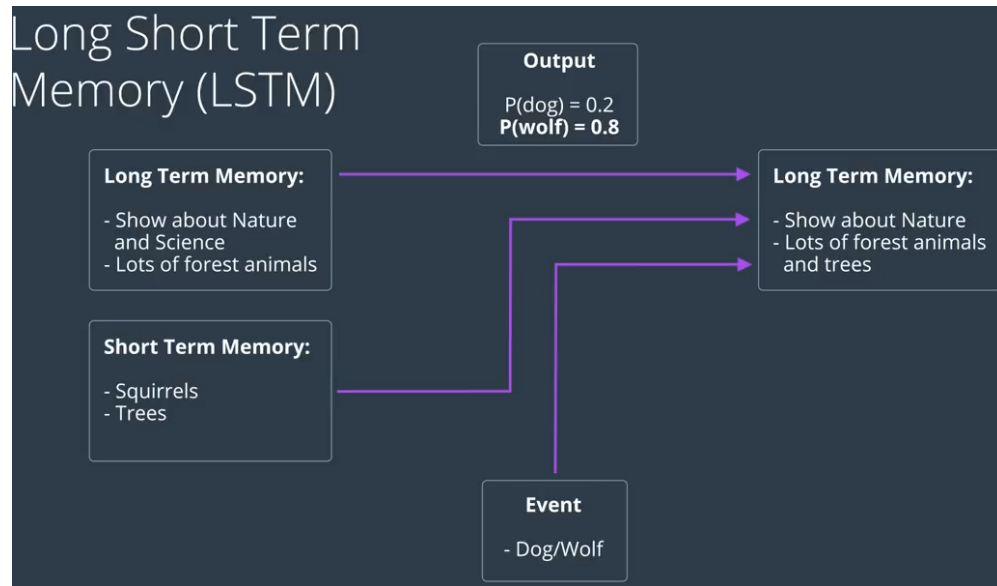
- 1- long term memory
- 2- Short term memory
- 3- Event

to help us update the long-term memory.

So, let's say we keep the fact that the show is about nature

- 1- we forget that it's about science.
- 2- And we also remember that the show is about forest animal and trees
- 3- Since we recently saw a tree.

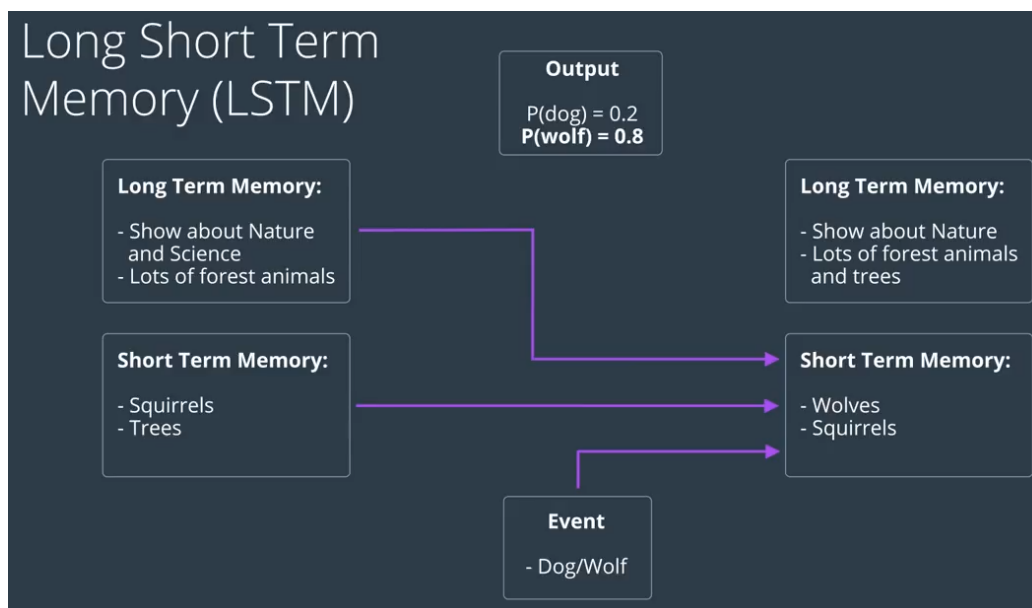
So, we add a bit and remove a bit to the long-term memory.



And Finally we also want to use these pieces of information to help us update the short term memory.

So let's say in our short term memory you want to

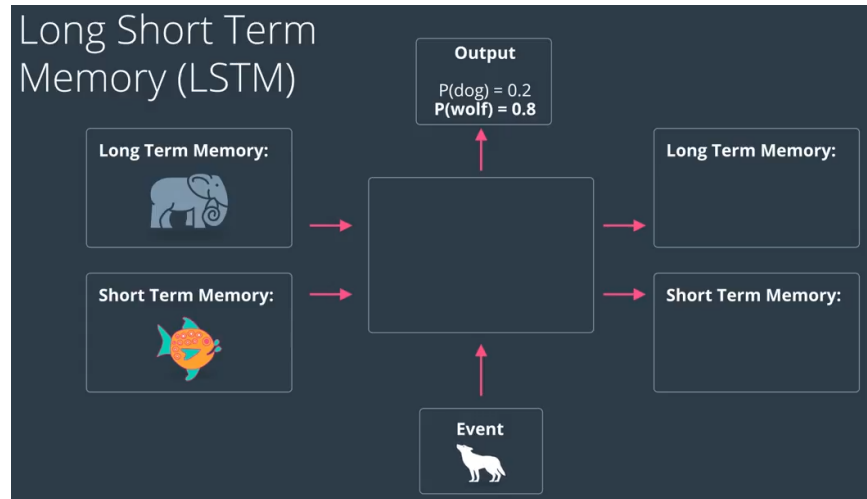
- 1- **forget** the show has trees
- 2- And remember that it has wolves since the trees happened a few images ago and we just saw a wolf.



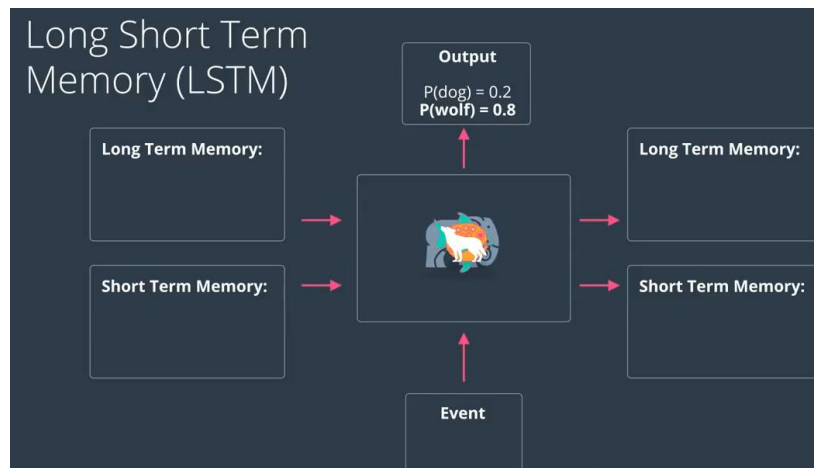
## LSTM

So, basically, we have an architecture like this and we use even more animals to represent our stages of memory.

- 1- The long-term memory is represented by an elephant since have long term memory.
- 2- The short term memory is represented by a forgetful fish
- 3- And the event will still represented by a wolf we just saw.



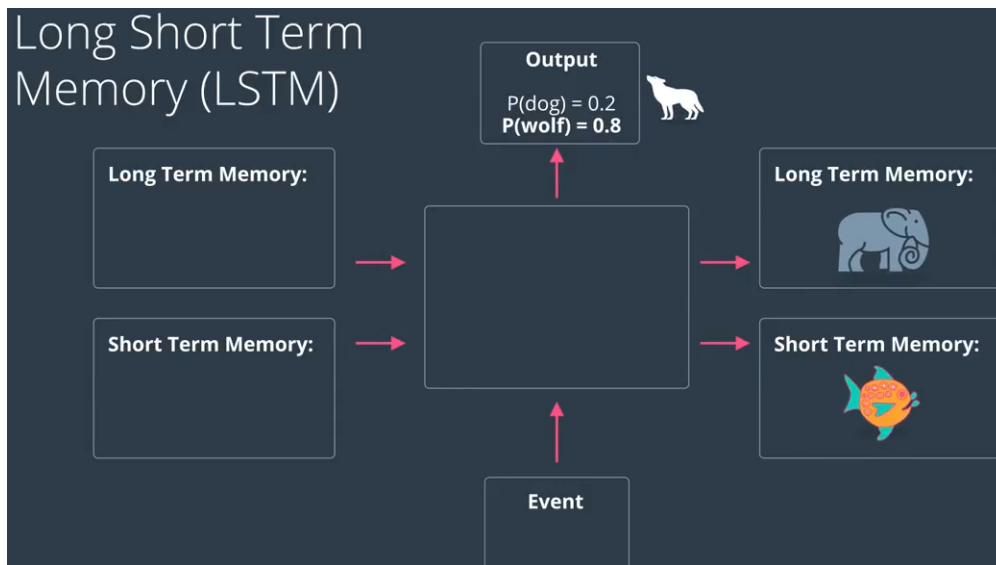
So LSTM works as follows three pieces of information go inside the node and then some math happens.



And then the new pieces of information get updated and come out.

- 1- There is a long-term memory
- 2- A short-term memory
- 3- And predication of the event.

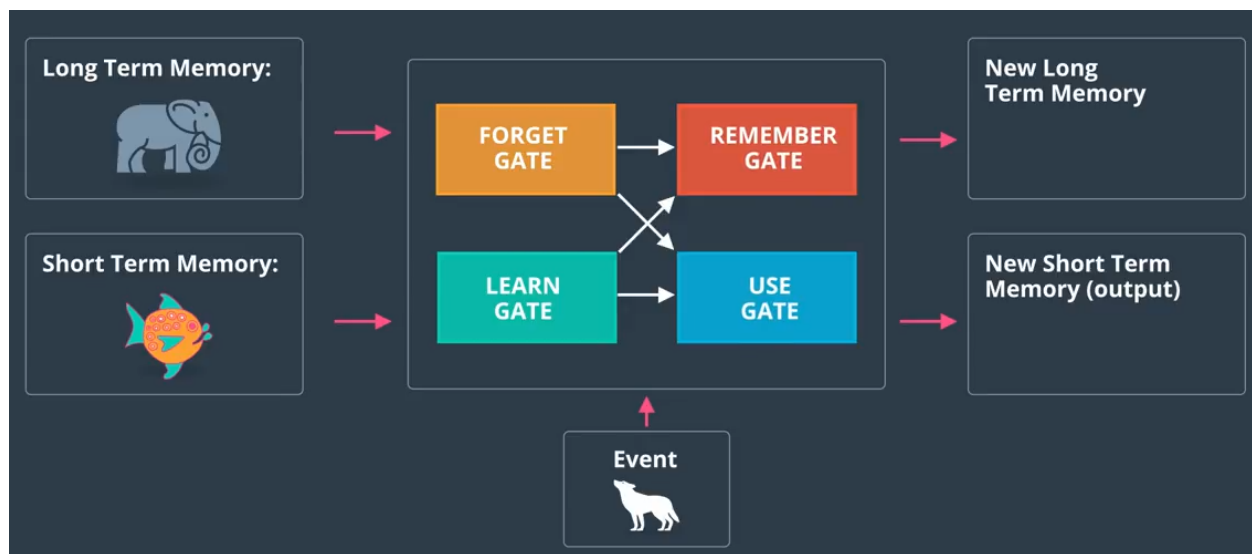
## LSTM



More specifically the architecture of LSTM contains a few gates.

- 1- It contains a forget gate
- 2- A learn gate
- 3- Remember gate
- 4- Use gate

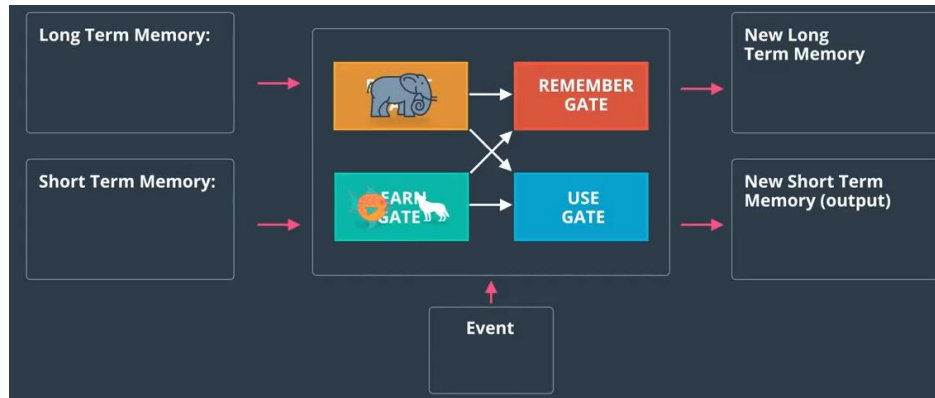
And here's basically how they work.





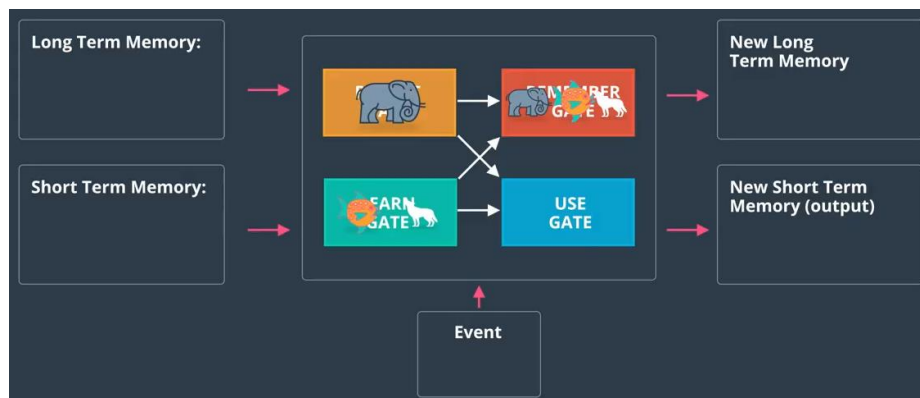
## LSTM

- 1- So, the long-term memory goes to the forget gate where it forgets everything it doesn't consider useful.
- 2- The short memory and the event are joined together in the **learn gate** containing the information that we have recently learned and it removes any necessary information.

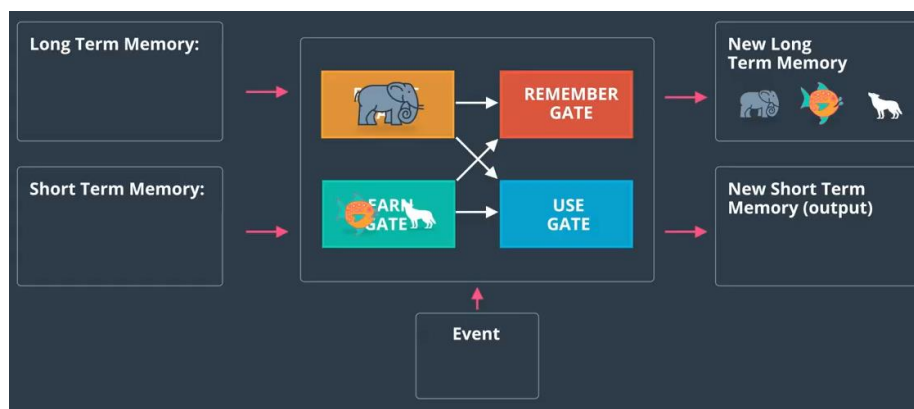


Now the long-term memory that we haven't forgotten yet plus the new information that we have learned get joined together in the remember gate.

This gate puts these two together and since it's called remember gate,



what it does is it outputs an updated long term memory. So this is what we will remember for the future.

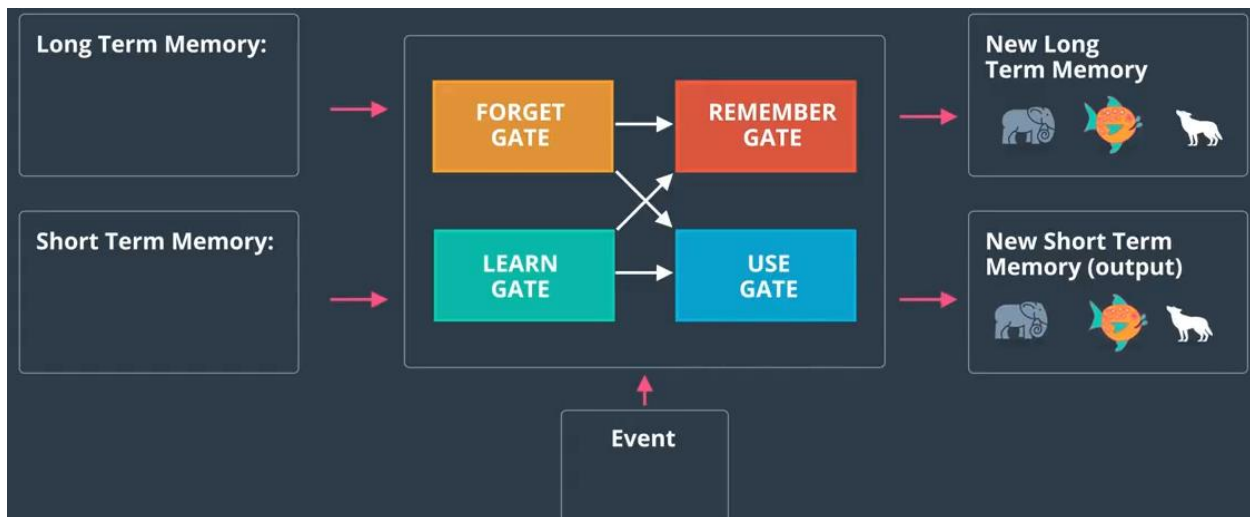


## LSTM

And finally, the **use gate** is the one that **decides what information we use from** what we previously know plus **what we just learned to make a prediction** so it also takes those **inputs** the long-term memory, and the new information joins them and decides what to output.

The output becomes both

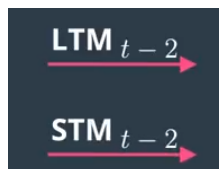
- 1- The prediction
- 2- The new short-term memory



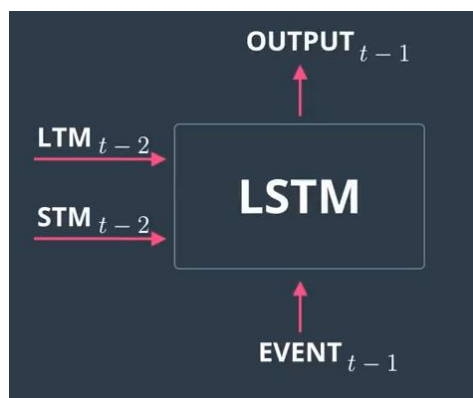
And so the big unfolded picture that we have is as follows :

- 1- We have the long term memory
- 2- The short-term memory

coming in which we call LTM and STM

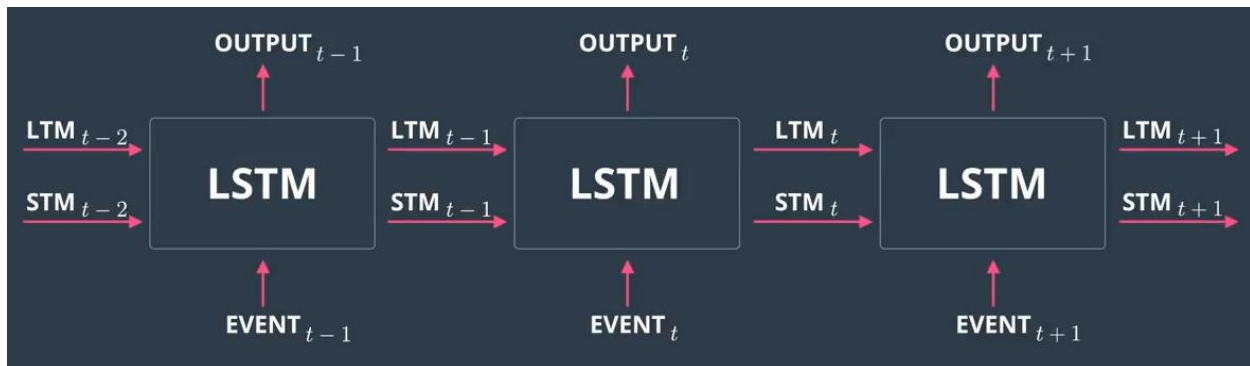


And then an event and the output are coming in and out of the LSTM.



## LSTM

And then this passes to the next node, and so on and so forth.

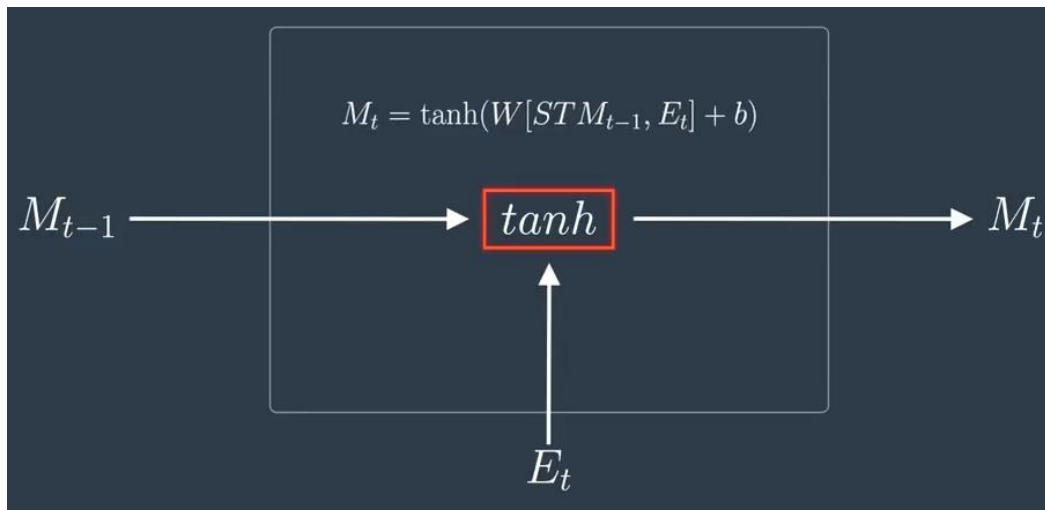


So, in general at time  $t$  we label everything with and underscore  $t$  as

- 1- We can see information passes from time  $t-1$  to time  $t$

So let's study the architecture an LSTM quickly recall the architecture of an RNN.

Basically what we do is we take our event  $E_t$  and our memory  $M_{t-1}$  coming from the previous in time, and we apply a simple tanh or sigmoid activation function to obtain the output and then your memory  $M_t$ .



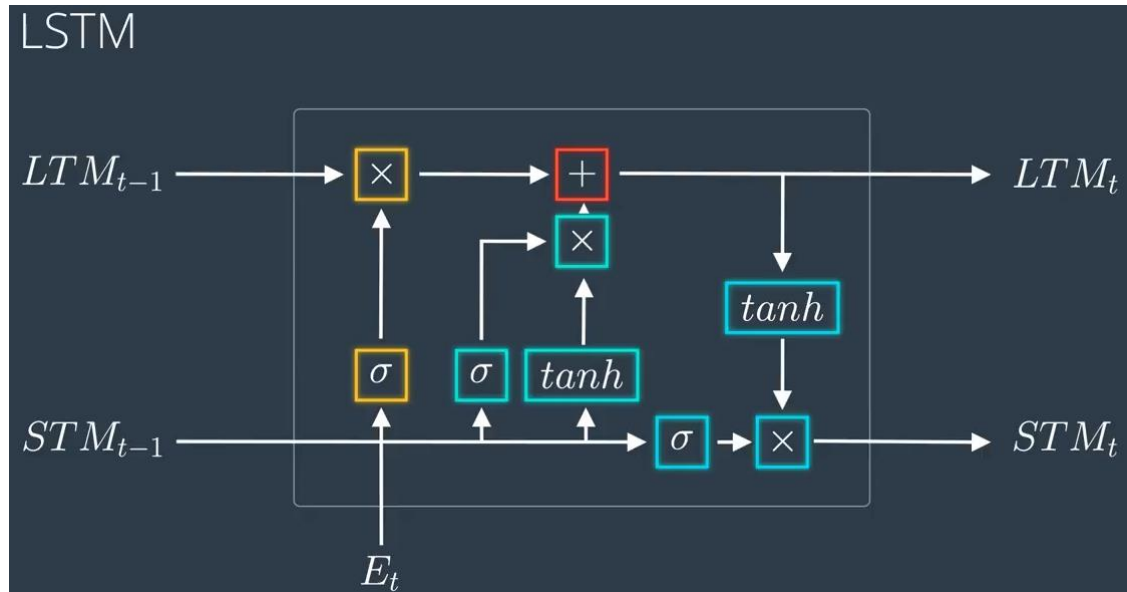
So to more specific, we join these two **vectors** and **multiply them** by a matrix  $W$  and add a bias  $b$  and then squish this with the tanh function, and that gives us the output  $M_t$ .

The output is prediction and also the memory that we carry to the next node.

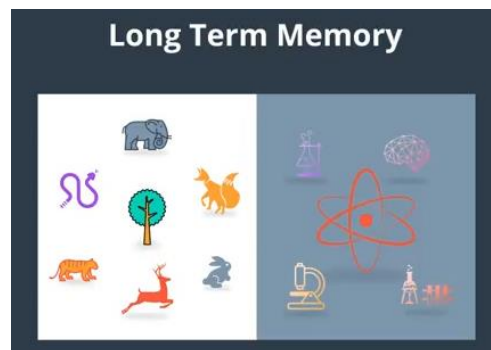
## LSTM

The LSTM architecture is very similar, except with a lot **more nodes inside** and with **two inputs** and **outputs** since it keeps track of **the long- and short-term memories**. And as I said, the short-term memory is, again, the output or prediction.

**Don't get scared. These are actually not as complicated as they look.**



So, let's keep start with base case we have a long term memory which is at the show we are watching it's about nature and science.



We also have a short-term memory which is what we have recently seen, a squirrel and a tree.



And finally, we have our current event which is a picture which is a picture we just saw that looks like a dog but it could also be a wolf.

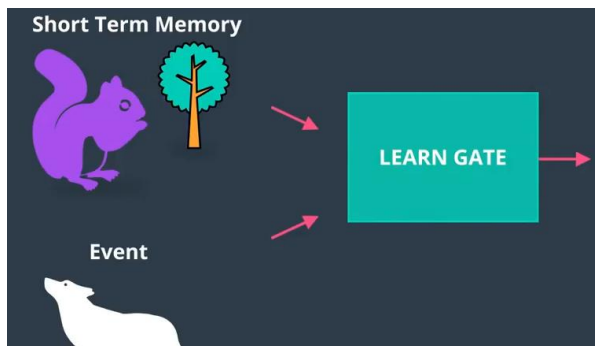
Learn Gate



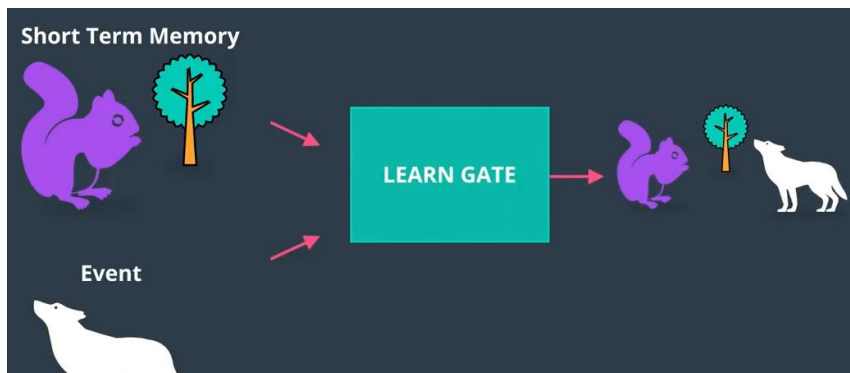
What the learn gate does is the following. It takes

- 1- Short-term memory
- 2- The event

And it joins it. Actually, it does a bit more

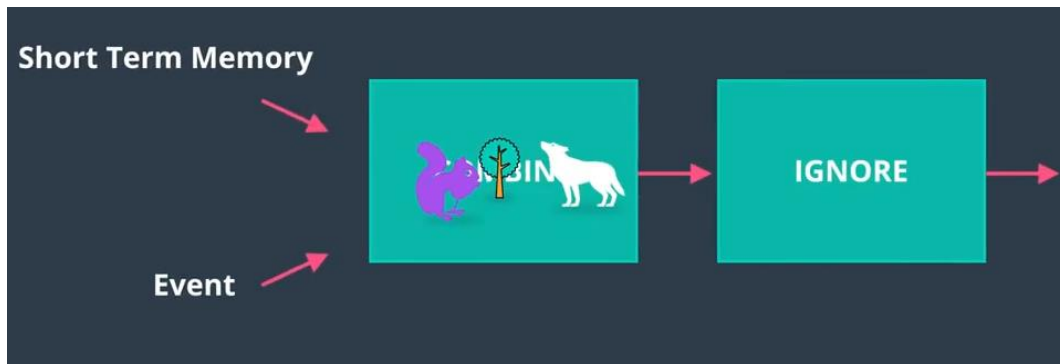


It takes a short-term memory and the event and it joins it. Actually, it does a bit more

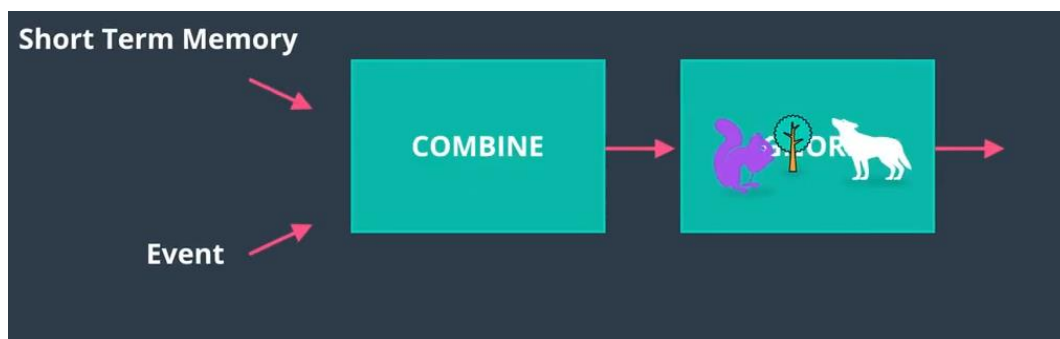


## LSTM

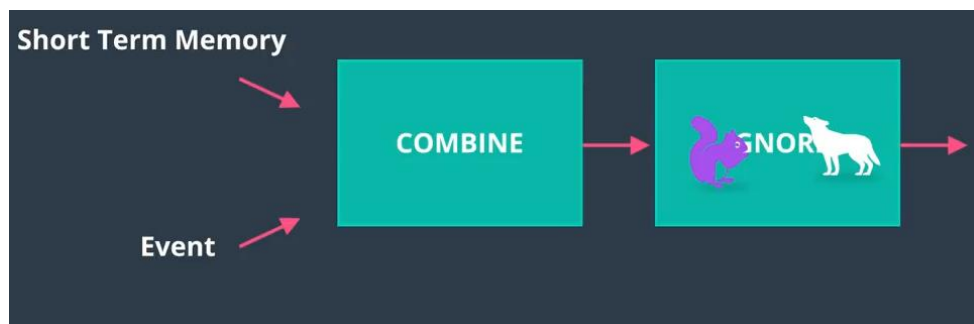
It takes the short-term memory and the event and it combines



Then and then it ignores a bit of it keeping the important part of it.



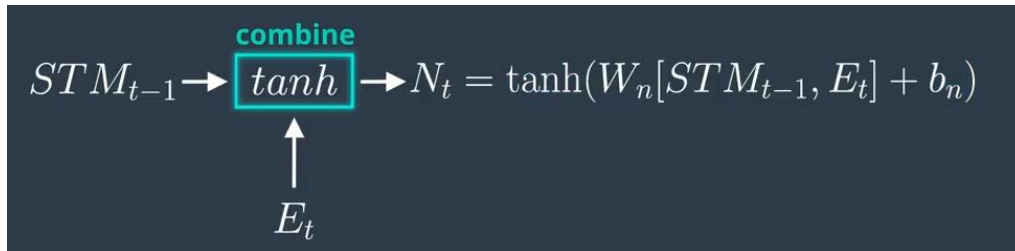
So here it forgets the fact that there's a tree and it remembers how we recently saw a squirrel and a dog/wolf.



And how does this work mathematically?



Well, it works like this. We have the short term memory  $STM_{t-1}$  minus one and the event  $E_t$  and it combines them by putting them through **a linear function** which consists of **joining the vectors multiplying by a matrix** adding a bias and finally squishing the result with a tanh activation function.

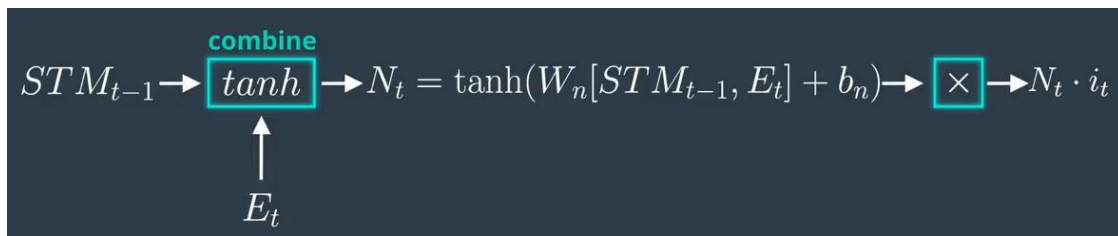


Then the new information  $N_t$  has this form over here.

**Now, how do we ignore part of it?**

Well, by multiplying by an ignore factor, I-T.

The **ignore factor**, I-T is actually a vector but its multiples element wise.



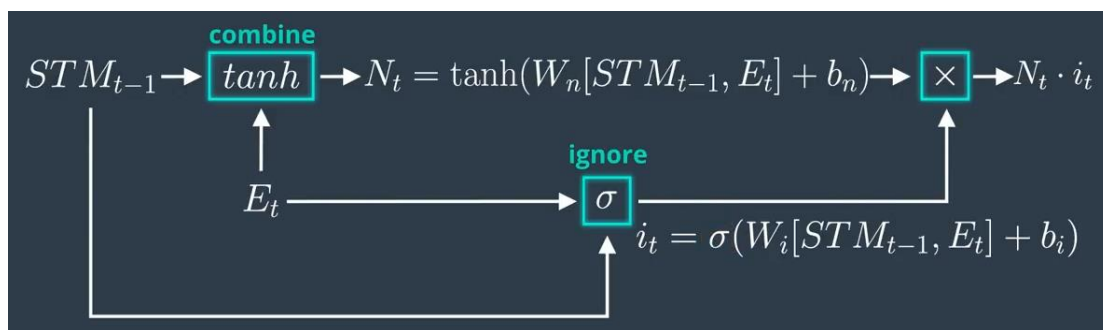
And how we do calculate I-T?

Well, we use our previous information of the short term memory and the event.

So, again we create a small neural network whose inputs

- 1- The short-term memory
- 2- The event

Will pass them through a small linear function with a new matrix and a new bias and squish them with the sigmoid function to keep it between zero and one.



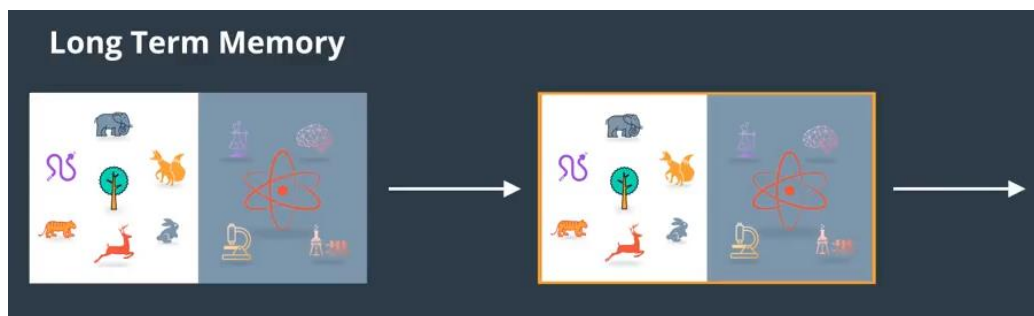
**So that's it. That's how the learn gate works.**

## Forget gate

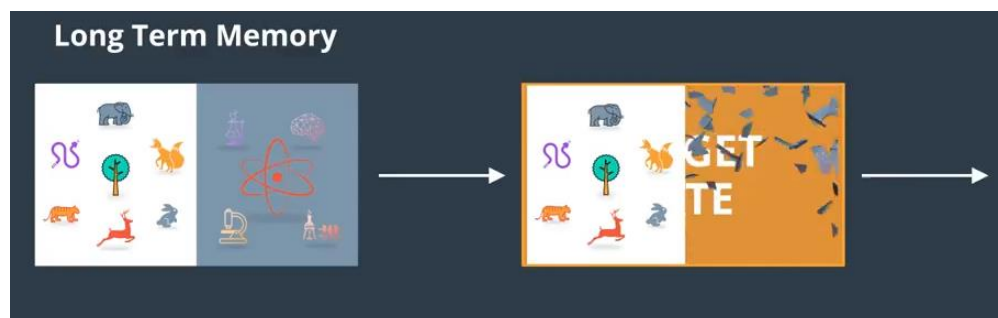
## Forget Gate



This one works as follows: it takes a long term memory and it decides what a parts to keep and to forget.



In this case, the show is about nature and science and the forget gate decides to forget that the show is about science and keep the fact that it's about nature.





## LSTM

### Long Term Memory



How does the forget gate work mathematically?

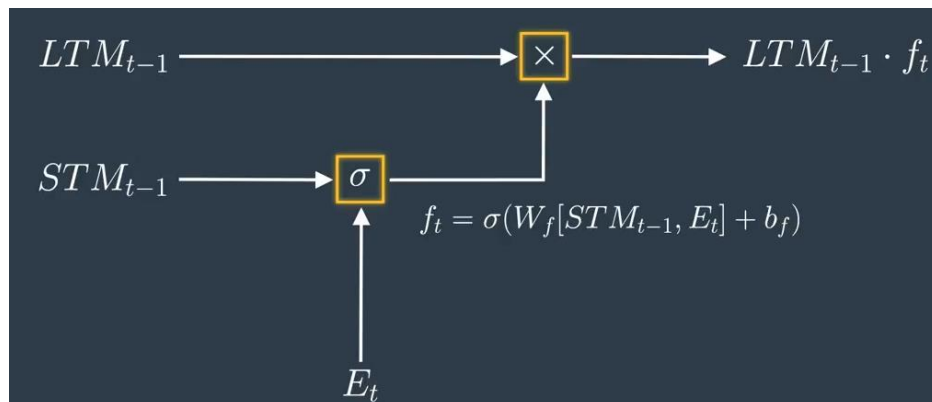
Very simple . The long-term memory LTM from T minus 1 comes in, and it gets multiplied by a forget factor  $f_t$ . And how does the forget factor  $f_t$  get calculated ?

### Forget Gate

$$LTM_{t-1} \longrightarrow \boxed{\times} \longrightarrow LTM_{t-1} \cdot f_t$$

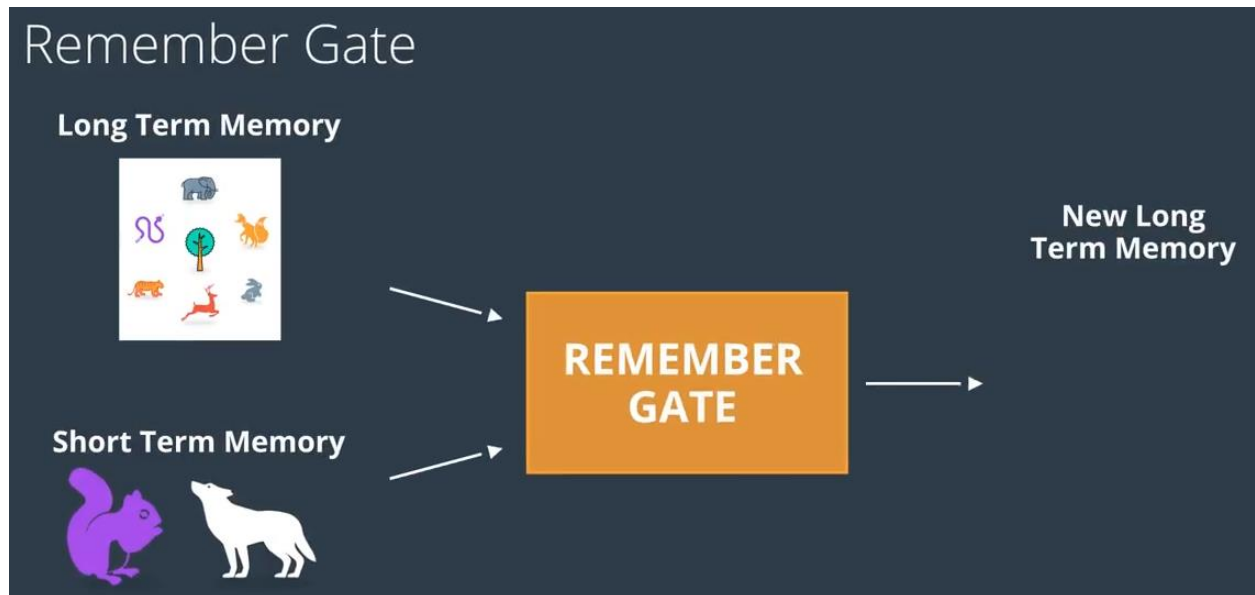
Well simple, we will use

A **short-term memory STM** and **the event information** to calculate  $f_t$ . So, just as before, we run a small one-layer neural network with a linear function combined the sigmoid function to calculate this forget factor and that's how the forget gate works.



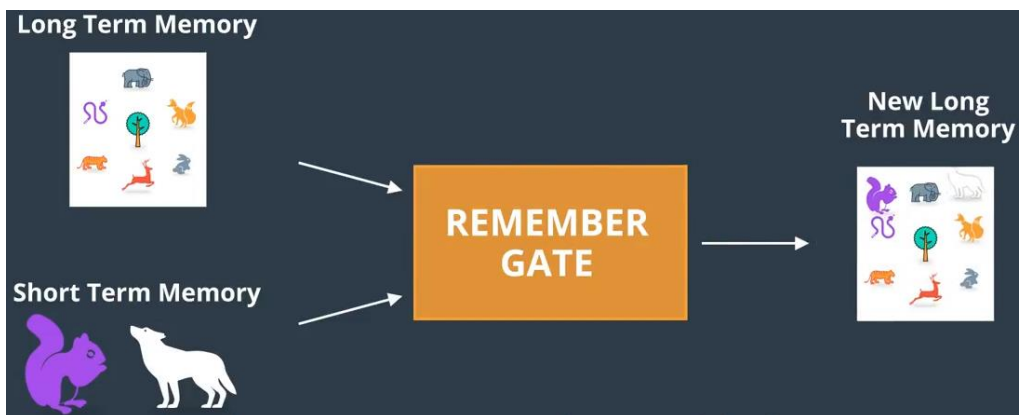
**Remember gate**

## Remember Gate



Learn our they remember gate it is simplest!

It takes a long-term memory coming out the of the forget Gate and the short memory coming out of the learn Gate and simply combines them together.

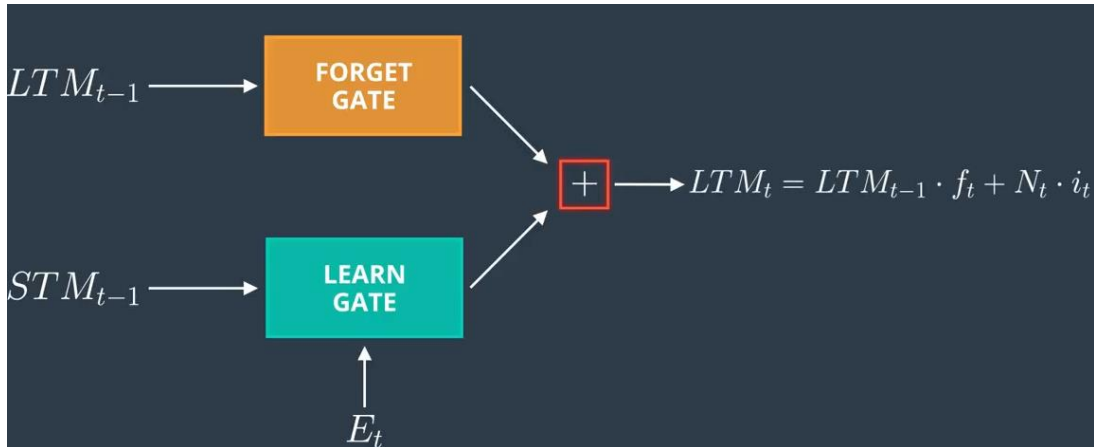


And how does this work mathematically? Again, very simple.

We just take the outputs coming from

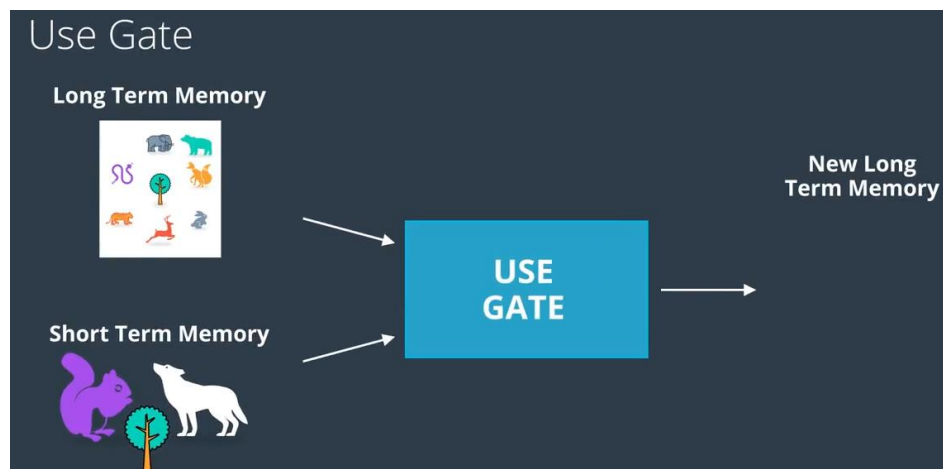
- 1- the forget Gate
- 2- from the learn gate

And we just add them. That's it, that's all we do.



Use Gate

And finally, we come to use gate or output gate



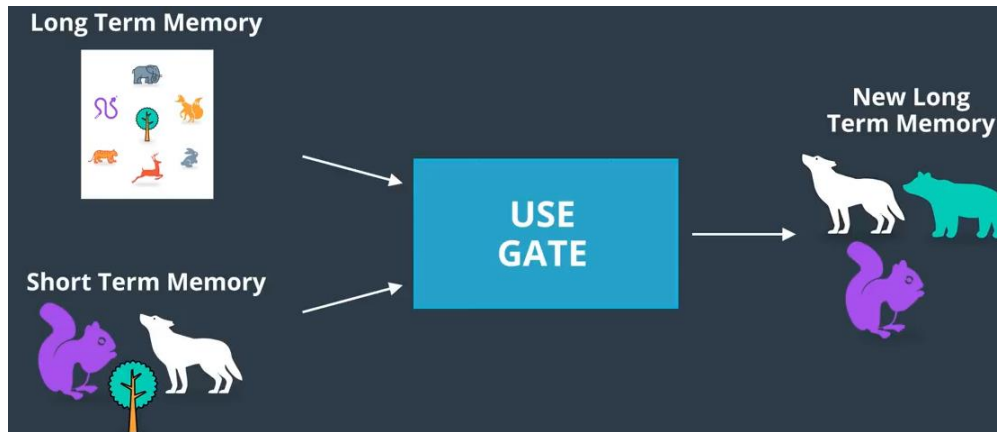
This is the one that uses the long-term memory that just came out of the forget gate and the short memory that just came out of the learned gate, to come up with a new short-term memory and an output. These are the same thing.

In this case, we will take what's useful from the long-term memory

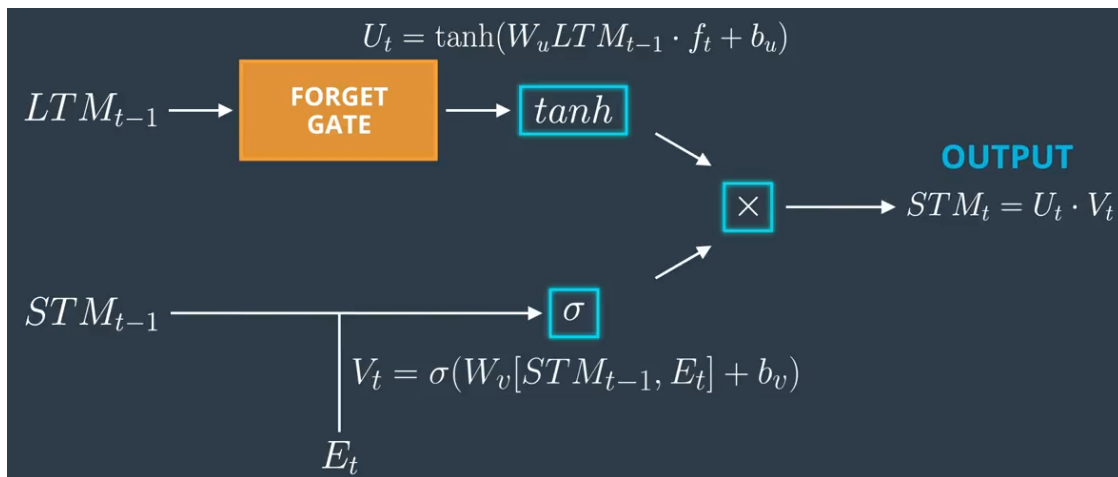
- 1- which is this bear over here
- 2- And what's useful from the short-term memory which is these dark wolf
- 3- And the squirrel and that's what's going to be our new short-term memory.

## LSTM

So our output basically says, your image is most likely a wolf but also carry some of the other animals that I have seen recently.



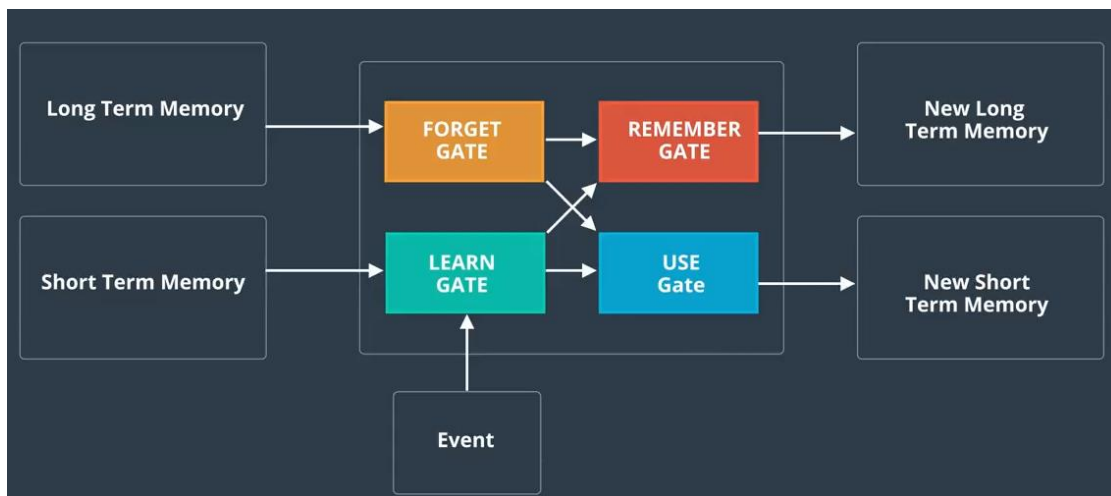
**And mathematically what this does is the following:** it applies a small neural network on the output of the forget gate using the tanh activation function and it applies to another small neural network on the short-term memory and the events using the sigmoid activation function. And the final step, it multiplies these two in order to get the new output.



**The output also worth of the new short-term memory. And that's how they use gate works.**

Here is the Architecture for an LSTM with the four gates.

- 1- There is the **Forget Gate**, which take the long-term memory and forget part of it
- 2- The **Learn Gate** puts the short-term memory together with the event as the information we have recently learned.
- 3- The **Remember Gate** joins the long-term memory that we haven't yet forgotten plus the new information we have learned in order
  - a. to update our long-term memory
  - b. And output it
- 4- And finally, the **Use Gate** also takes the information we just learned together with long-term memory we haven't yet forgotten, and it uses it to
  - a. make a prediction
  - b. And update the short-term memory.



So, this is how it looks all put together. It's not so complicated after all, isn't it?

Now you may be thinking, wait a minute, this looks too arbitrary

Why use tanh sometimes and sigmoid other times?

Why multiply sometimes and add other times, and other times apply a more complicated linear function?

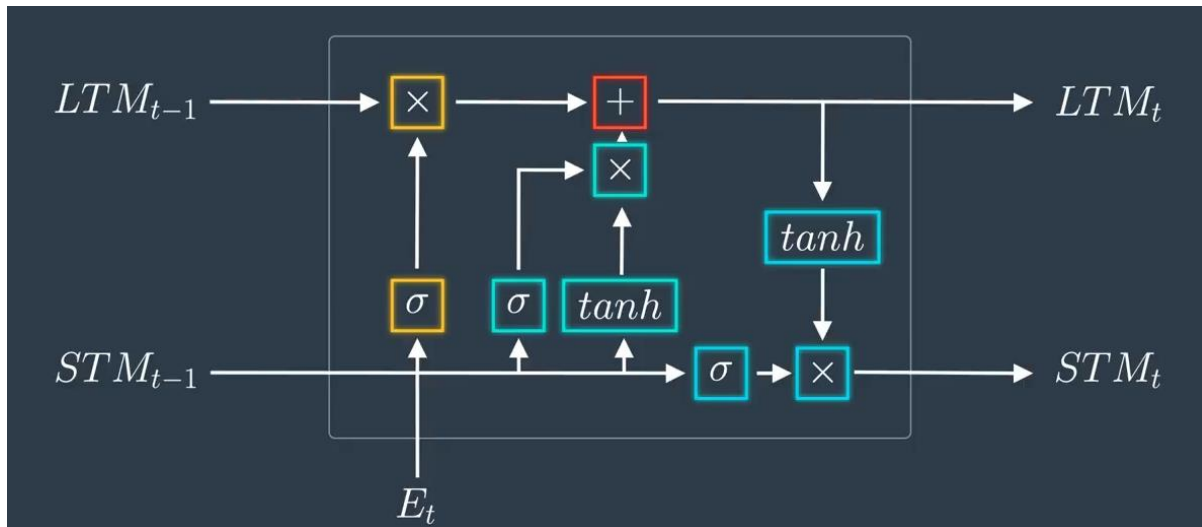
You can probably think of different architectures that make more sense or that are simpler and you are absolutely right

This is an arbitrary construction. And as many things in machine learning the reason it is like is because it works. And in the following section we will see some other architectures which can be simpler or more complex and that also do the job.

But you are welcome to look for others and experiment.

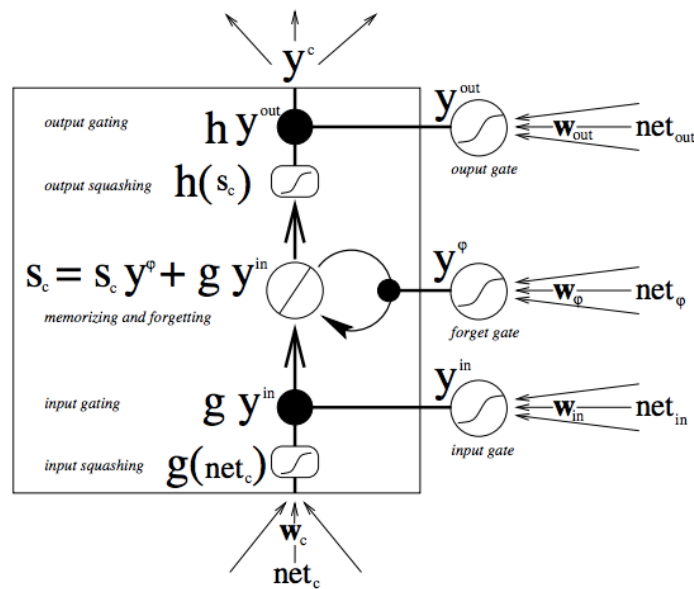
This is area very much under development so if you come up with a different architecture and it works, that is wonderful.

## LSTM



If you would like to deepen your knowledge even more, go over the following [tutorial](#). Focus on the overview titled: **Long Short-Term Memory Units (LSTMs)**.

If you are feeling confident enough, skip the overview and jump right into our next question:



The LSTM cell- taken from the Deep Learning tutorial

## LSTM

he illustration above is of a LSTM cell. What would be the values of the three gates in situations where the cell retains information for a long period, without accepting a new input or producing an output?



The inputs gate: close to 0; the forget gate: close to 1; the output gate: close to 0



The inputs gate: close to 1; the forget gate: close to 1; the output gate: close to 0



The inputs gate: close to 0; the forget gate: close to 0; the output gate: close to 0



The inputs gate: close to 0; the forget gate: close to 1; the output gate: close to 1

**Sol : The inputs gate: close to 0; the forget gate: close to 1; the output gate: close to 0**

## Character-wise RNN

You implement a character-wise RNN.

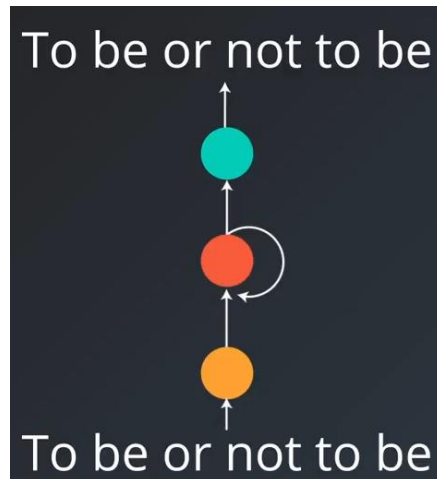
The network will learn about **some text one character** at a time, and then generate new text one character at a time.

Let's say we want to generate a new **Shakespeare** play.

As an example, "To be or not to be."

1- We would pass the sequence into our RNN one character at a time.

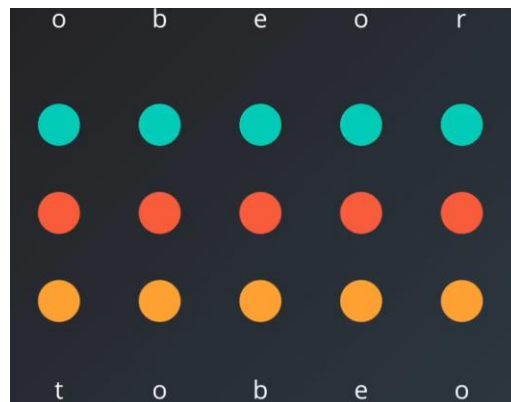
Once trained, the network will generate **new text** by predicting the **next character** based on the characters it's already seen. So then, to train this network, we want it to predict the next character in the input sequence.



In this way, the network will **learn** to produce a sequence of characters that look like the original text.

Let's consider what the architecture of this network will look like.

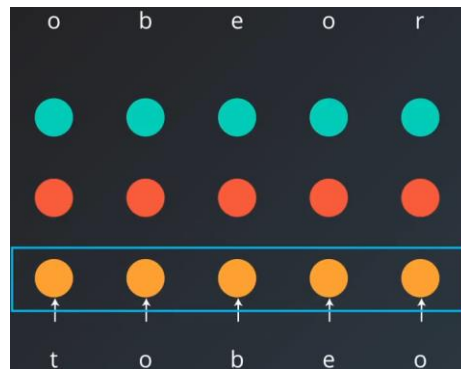
1- First, let's unroll the RNN, so we can see how this all works as a sequence.



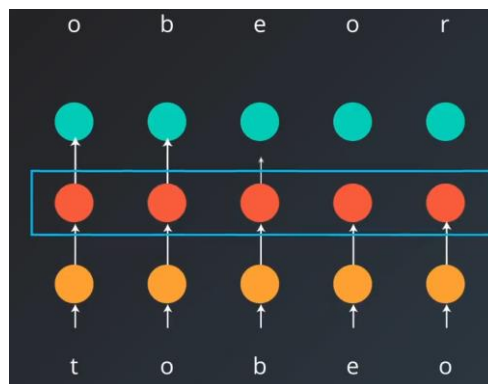


## LSTM

Here, we have our input layer where we will pass a charcutier as one-hot encoded vectors.



These vectors go to the hidden layer.

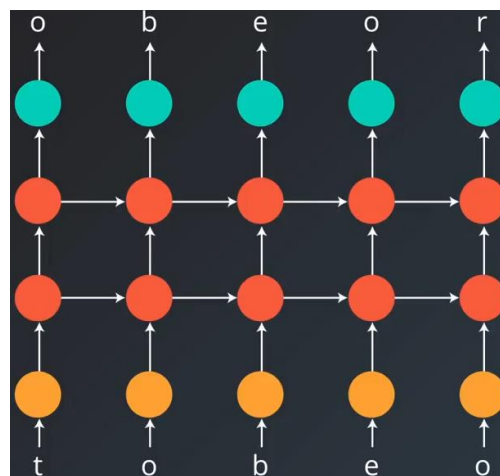


The hidden layer is built the LSTM cells where

- 1- The hidden state
- 2- And cell state

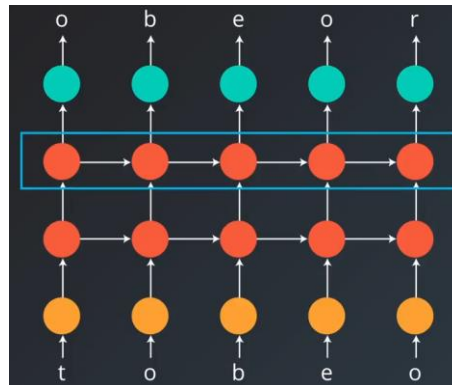
Pass from one cell to the next in the sequence.

in practice, we will actually use multiple layers of LSTM cells. You just stack them up like this.

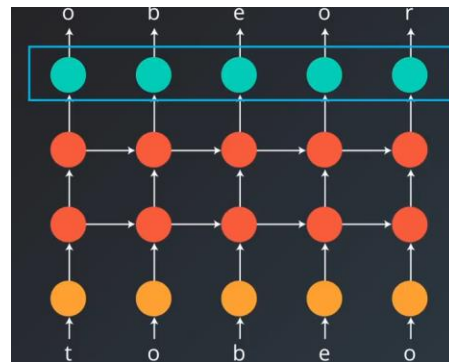


## LSTM

The output of these cells go to the output layer.



The output layer is used to predict the next character.



We want the probabilities for each character the same way you did image classification with covenant.

That means that we want a SoftMax activation on the output.

Our target here will be the input sequence but shifted over one, so that, **each character** is predicting the next character in the sequence.

Again, we will use cross entropy loss for training with gradient descent.

When is network trained up, we can pass in one character and get out probability distribution for the likely next character.

Then we can sample from that distribution to get the next character.

Then we can take that character, pass it in , and get another one .

We keep doing this and eventually we will build up some completely new text.

We will be training this network on the text from Anna Karenina. It is public domain

## LSTM



## Sequence batching

Difficult part works for me is getting batch

It's more of a programming challenge than anything deep learning specific.

So here, I'm going to walk you through how batching works for RNNs.

With RNNs, we are training on sequence of data like text, stock values , audio

By taking a sequence and splitting it into multiple shorter sequences, we can take a advantage of matrix operations to make training more efficient.

In fact, the RNN is training on multiple sequences in parallel

Let's look at a simple example.

```
[ 1 2 3 4 5 6 7 8 9 10 11 12 ]
```

A sequence of numbers from one to 12.

We can pass these into an RNN as one sequence.

But what's better, we could split it in half and pass in two sequences.

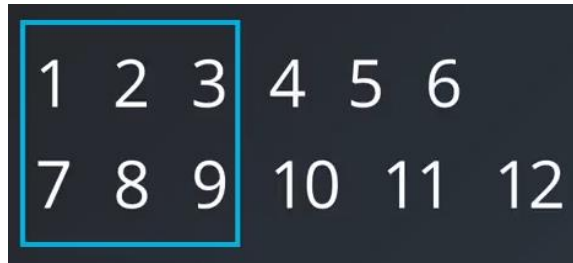
```
[ 1 2 3 4 5 6 ]  
[ 7 8 9 10 11 12 ]
```

The batch size corresponds to the number of sequences we are using. So here we would say the batch size is two.

```
1 → [ 1 2 3 4 5 6 ]  
2 → [ 7 8 9 10 11 12 ]
```

Along the batch size, we also choose the length of the sequences we feed to the network.

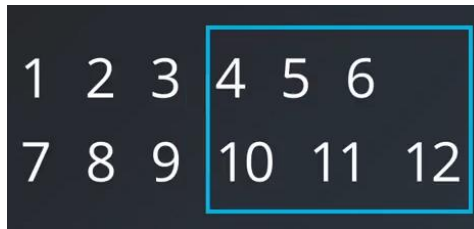
For example, let's consider using a sequence length of three.



1	2	3	4	5	6
7	8	9	10	11	12

Then the first batch of data we pass into the network are the first three values in each mini-sequence.

The next batch contains the next three values, and so on until we run out of data.



1	2	3	4	5	6
7	8	9	10	11	12

We can retain the hidden state from one batch and use it at the start of the next batch.

This way the sequence information is transferred across batches for each mini-sequences.

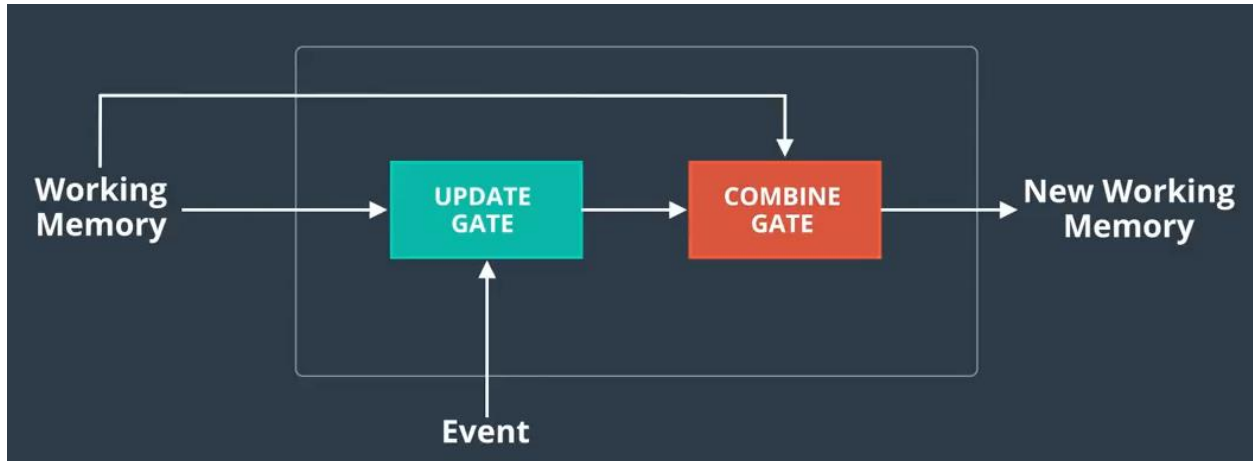
Next up, you will see how to actually build a recurrent network.

### Gated Recurrent Unit (GRU)

It is similar architecture also worked well but there are many variations to LSTMs and we encourage you to study them further.

Here's a simple architecture which also works well.

It is called the gated recurring unit or GRU for short.



It combines the forget and the learn gate into an update gate and then runs this through combine gate.

It only returns one **working memory** instead of a pair of **long- a short term memories**, but actually seems to work in practice very well too. I won't go much into details, but in the instructor comments I will recommend some very good reference to learn more about gated recurrent units.

Here's another observation.

Let's remember the forget gate.

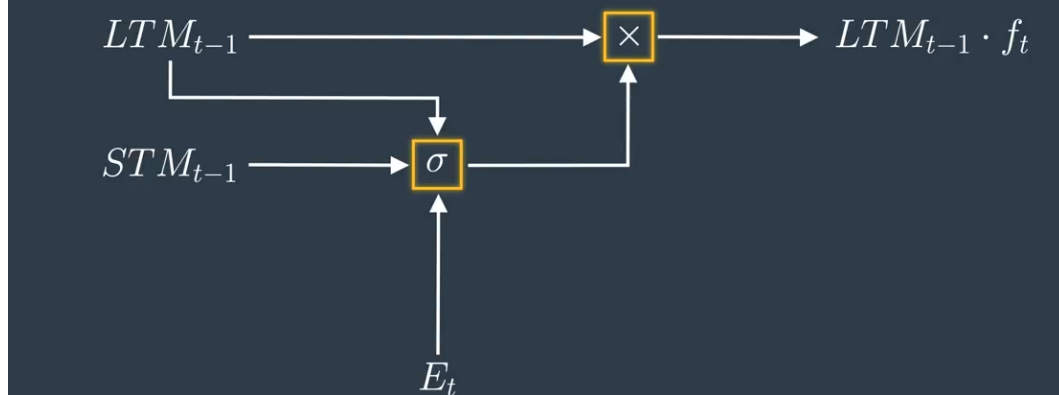
The forget factor  $f_t$  was calculating using as input a combination of the short-term memory and the event.

But what about the long-term memory? It seems like we left it away from the decision.

Why does a long-term memory not have a say into which things **get remembered** or not? Well let's fix that.

Let's also connect the long-term memory into the neural network that calculates the forget factor.

## Peephole Connections



Mathematically, this just means the input matrix is larger since we are also concatenating it with the long-term memory matrix.

$$f_t = \sigma(W_f[LTM_{t-1}, STM_{t-1}, E_t] + b_f)$$

This is called a **peephole connection** since now the long-term memory has more access into the decision made inside the LSTM.

We can do this for every one of the forget-type nodes, and this is what we get an LSTM with peephole connections.

## LSTM with Peephole Connections

