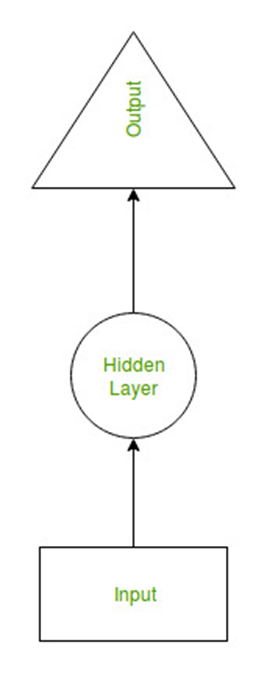
**\*\*Question 1\*\*.What are deep learning modelling techniques to handle textual data?**

**Ans:** Deep learning modelling techniques to handle textual data are:

1. RNN
2. LSTM
3. Bidirectional LSTM
4. Transfer Learning Models.
5. Attention Based Models

**Question 2. What is RNN? Why we use them?**

**Ans:** RNN are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer.



**Uses of RNN**

An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

**\*\*Question 3\*\* State down few problems in RNN?**

Disadvantages of Recurrent Neural Network:

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

**\*\*Question 4\*\* What is vanishing gradient and gradient explosion?**

**Ans: Vanishing gradient :** Gradient descent algorithm finds the global minimum of the cost function that is going to be an optimal setup for the network. Information travels through the neural network from input neurons to the output neurons, while the error is calculated and propagated back through the network to update the weights.

Similarly for RNNs, but here we’ve got a little bit more going on.

Firstly, information travels through time in RNNs, which means that information from previous time points is used as input for the next time points.

Secondly, you can calculate the cost function, or your error, at each time point.

Essentially, every single neuron that participated in the calculation of the output, associated with this cost function, should have its weight updated in order to minimize that error. And the thing with RNNs is that it’s not just the neurons directly below this output layer that contributed but all of the neurons far back in time. So, you have to propagate all the way back through time to these neurons.

The problem relates to updating wrec (weight recurring) – the weight that is used to connect the hidden layers to themselves in the unrolled temporal loop.

For instance, to get from xt-3 to xt-2 we multiply xt-3 by wrec. Then, to get from xt-2 to xt-1 we again multiply xt-2 by wrec. So, we multiply with the same exact weight multiple times, and this is where the problem arises: when you multiply something by a small number, your value decreases very quickly.

As we know, weights are assigned at the start of the neural network with the random values, which are close to zero, and from there the network trains them up. But, when you start with wrec close to zero and multiply xt, xt-1, xt-2, xt-3, … by this value, your gradient becomes less and less with each multiplication.

**Exploding gradients** are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. This has the effect of your model being unstable and unable to learn from your training data.

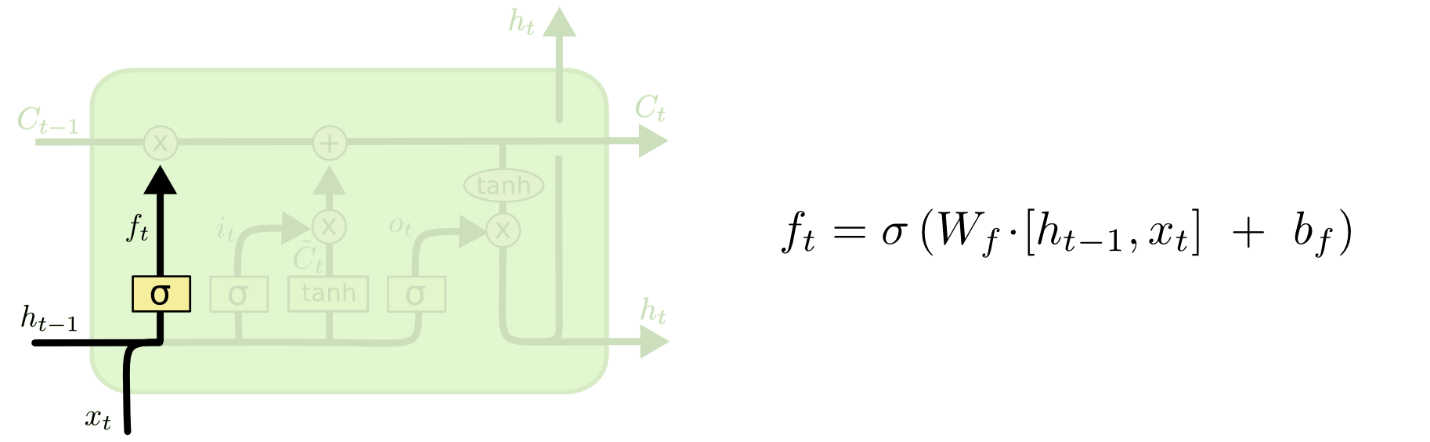
**\*\*Question 5\*\*.What is LSTM?**

**Ans:** Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

Steps followed in LSTM :

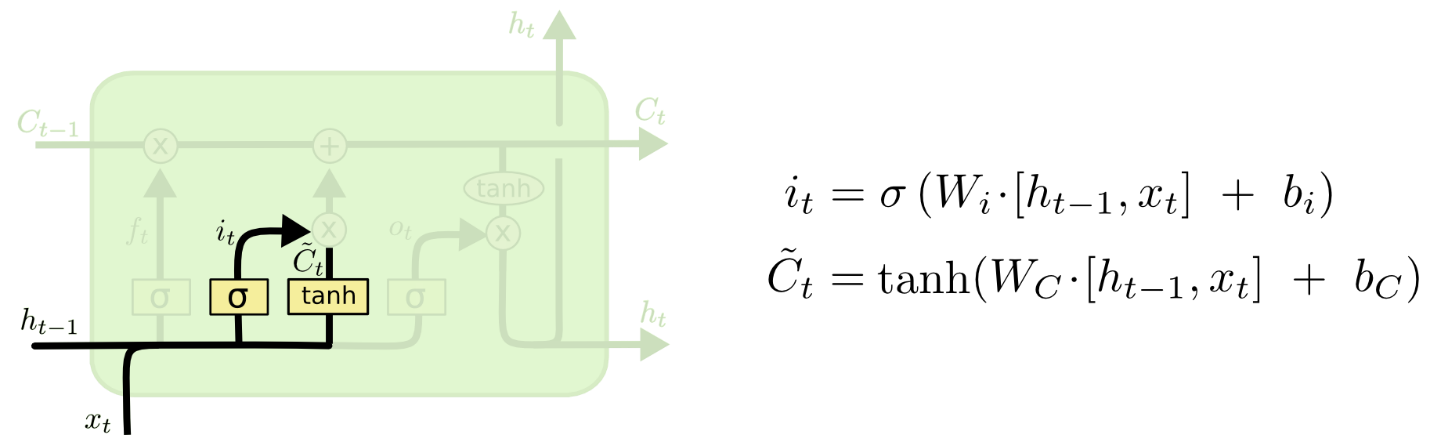
The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at ht−1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1. A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

For example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

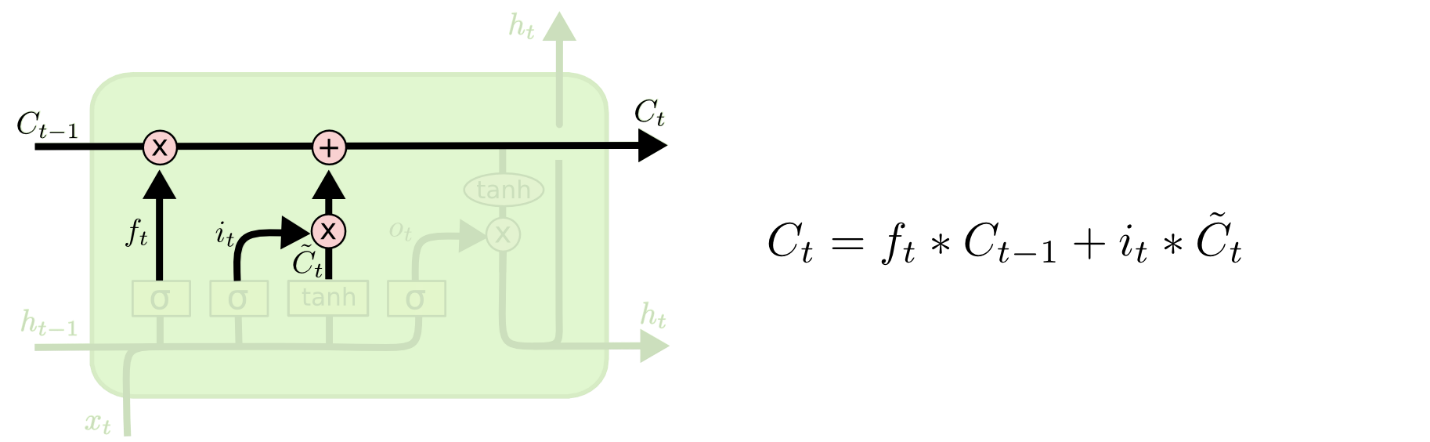
In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.



It’s now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.

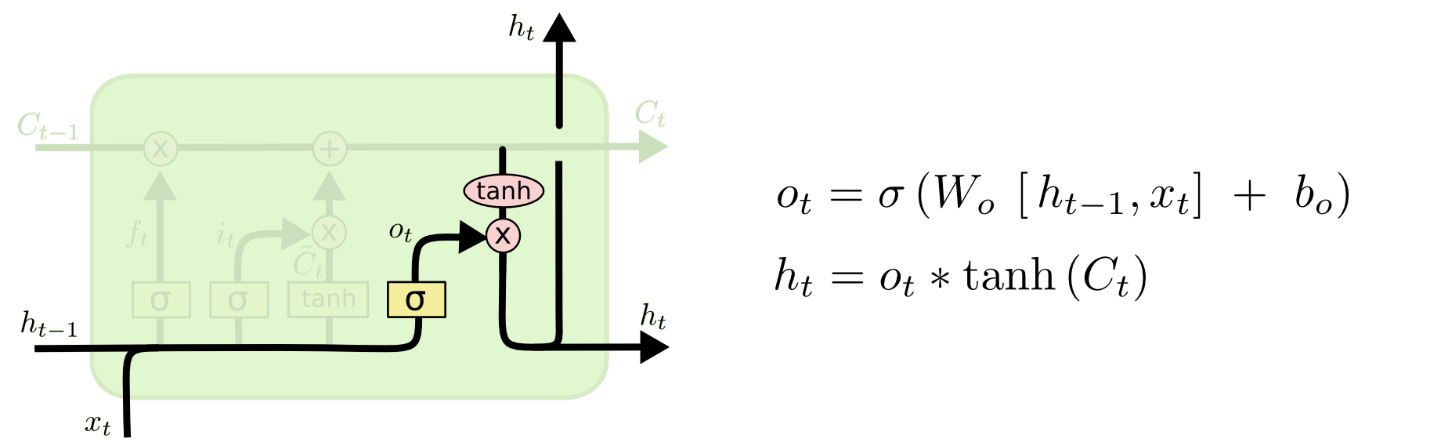
We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.



Finally, we need to decide what we’re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that’s what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that’s what follows next.



**\*\*Question 6\*\* Benifits of LSTM over RNN**

All RNNs have feedback loops in the recurrent layer. This lets them maintain information in 'memory' over time. But, it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time (called the vanishing gradient problem). LSTM networks are a type of RNN that uses special units in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time. A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten. This architecture lets them learn longer-term dependencies.