Recurrent Neural Network-RNN

Renu Khandelwal
Oct 26, 2018 · 8 min read ★

If you are interested to know how does Google voice search or Amazon's Alexa or Apple's Siri works. You need to have a basic understanding of Neural networks and CNN's then read on to know the technology.

For a basic understanding of neural networks

For a basic understanding of CNN

we will start with

- What is missing with Artificial Neural Networks and Convolution Neural Network that Recurrent Neural Network helps solve.
- Where can we use RNN?
- What is RNN and how it works?
- Challenges with vanilla RNN like vanishing and exploding gradients
- How LSTM(Long Short Term Memory)and GRU (Gated recurrent unit) solves these challenges

Fasten your seat belts and get ready for an exciting journey on RNN

Let's say we are writing a message "Let's meet for___" and we need to predict what would be the next word. The next word could be lunch, or dinner or breakfast or coffee. It is easier for us to make inferences based on the context. Let’s say if we knew that we were meeting in the afternoon and that information persisted in our memory then we can easily make prediction that we are possibly meeting for lunch.

When we need to work on sequential data that needs to be persisted over several time steps then we use Recurrent Neural network(RNN)

Traditional neural network and CNN’s need a fixed input vector, apply activation function on fixed set of layers to produce a fix sized output.

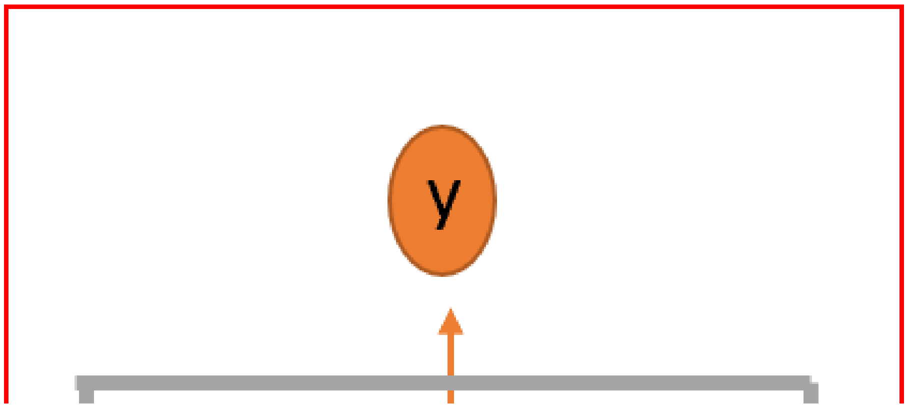
For example we take an input image of 128 by 128 sized vector to predict images of dogs or cats or cars. we cannot take a variable sized image to make the prediction

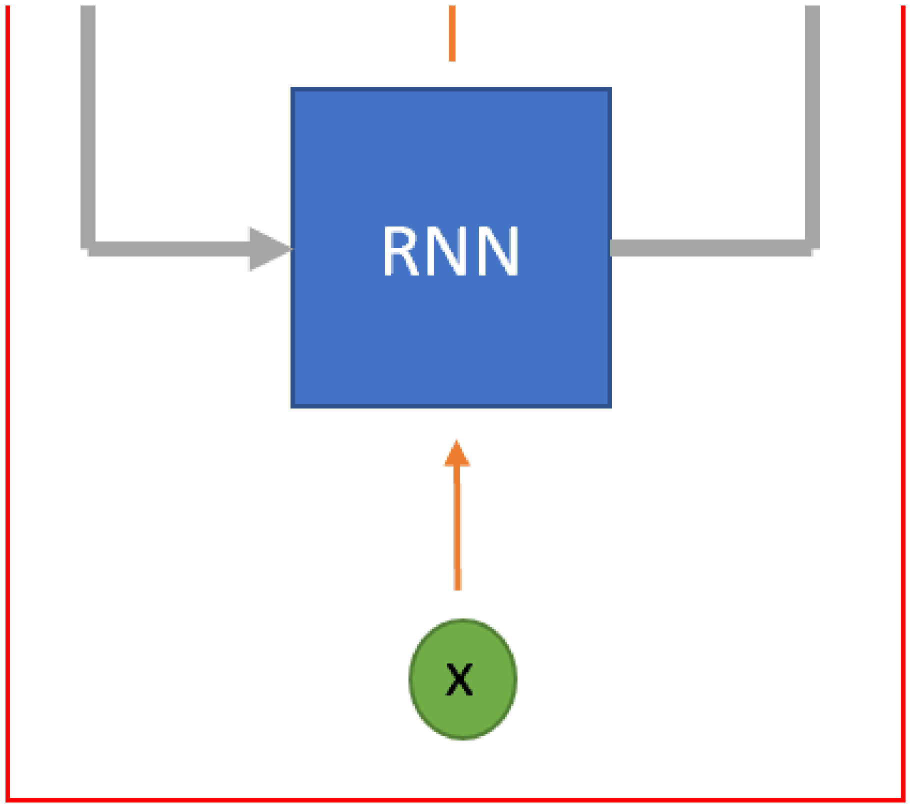
Now what if we need to operate over sequential data that is dependent on previous input state like our message or when sequential data can be in input or output or both, and that is exactly where we use RNNs.

In RNN, we share the weights and feed the output back into the inputs recursively.This recurrent formulation helps process sequential data.

RNN’s make use of sequential data to make inferences like who is talking, what is being spoken and what might be the next word etc.

RNN’s are neural networks with loops to persist information. RNN are called a **recurrent as they perform the same task for every element in the sequence and output elements are dependent on previous elements or states**. This is is how RNN’s persist information to use context to draw inferences.





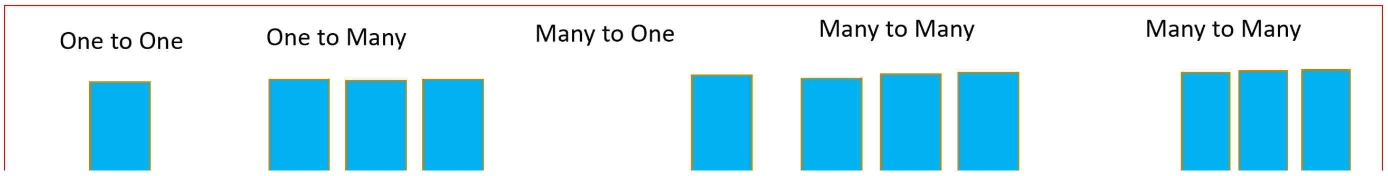
RNN are neural networks with loops to persist information

Where are RNN's used?

As stated earlier RNN can have one or more input and one or more output i.e. variable input and variable output.

RNN's are used for

- Image classification
- Image captioning
- Machine Translation
- Video classification
- Sentiment Analysis



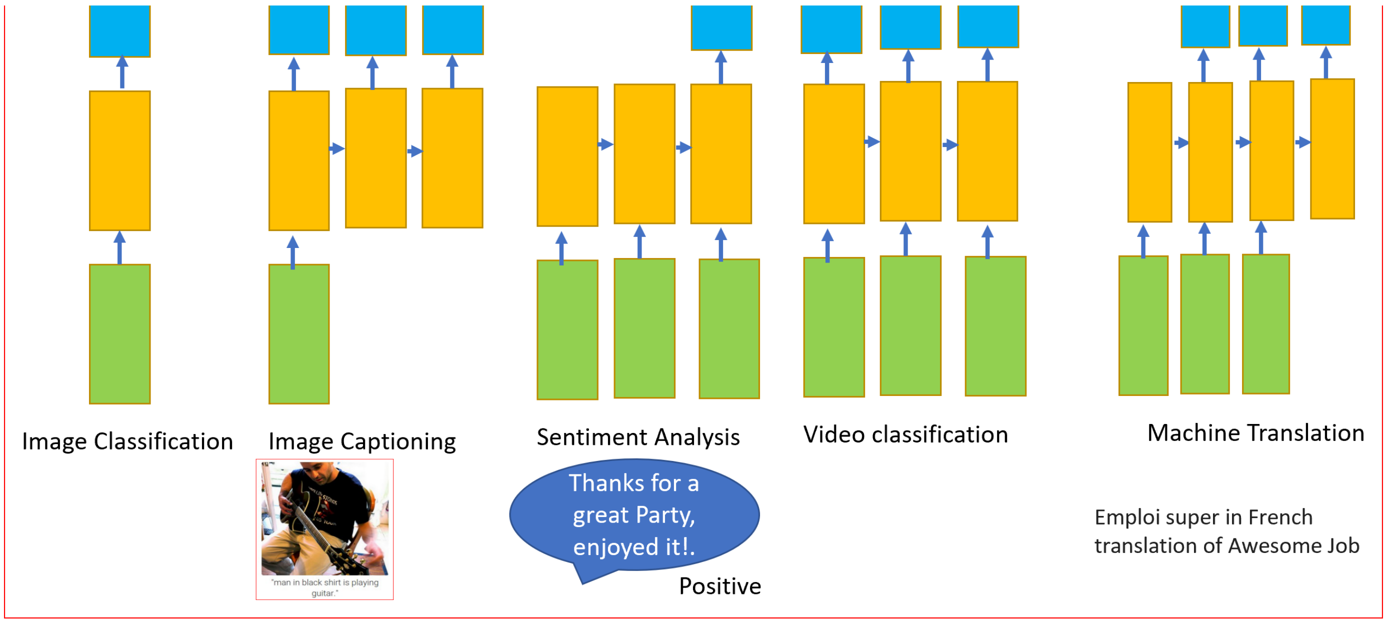


Image Classification
Image Captioning
"man in black shirt is playing guitar."
Sentiment Analysis
Thanks for a great Party, enjoyed it!.
Positive
Video classification
Machine Translation
Emploi super in French translation of Awesome Job

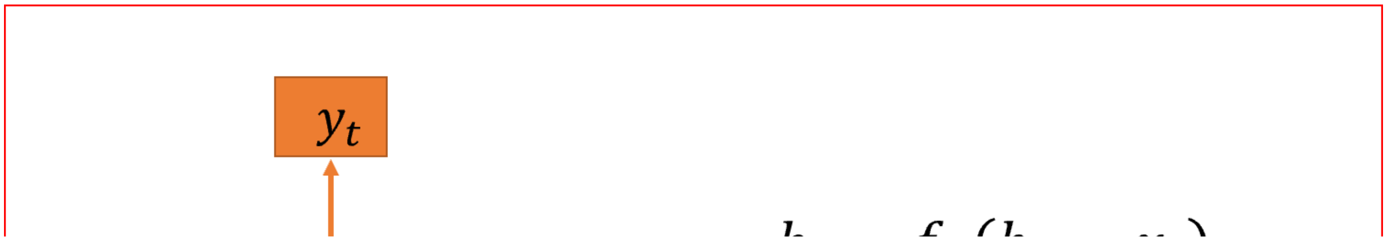
Source:http://karpathy.github.io/

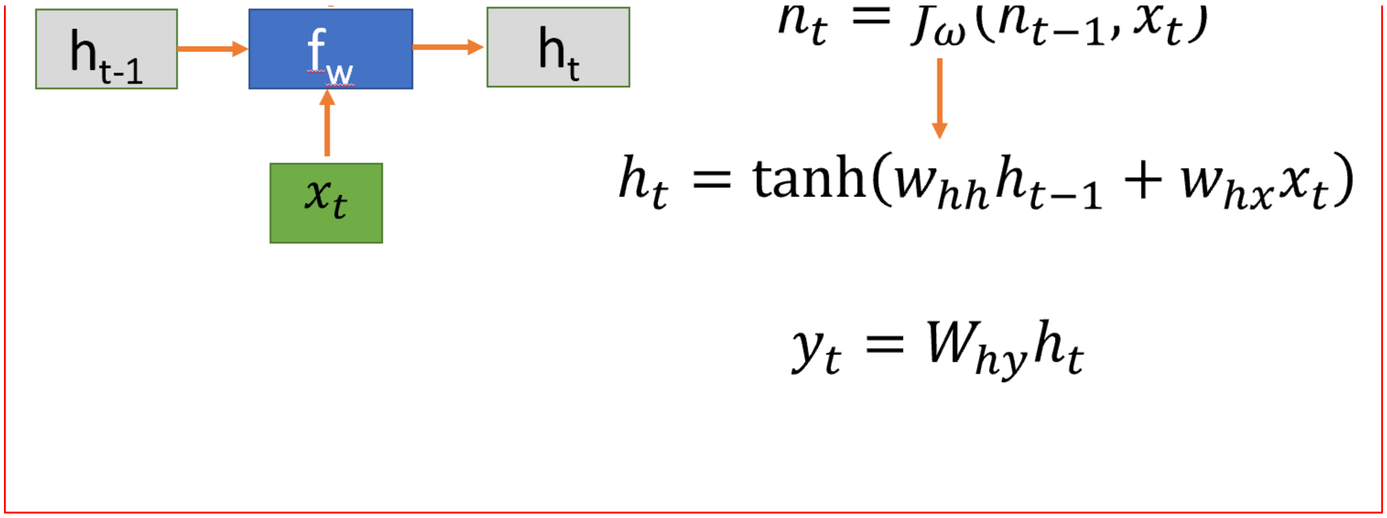
How does RNN work?

Slowly unravelling RNN, explaining the notations first.

- h is hidden state
- x is input
- y is output
- W is the Weights
- t is time step

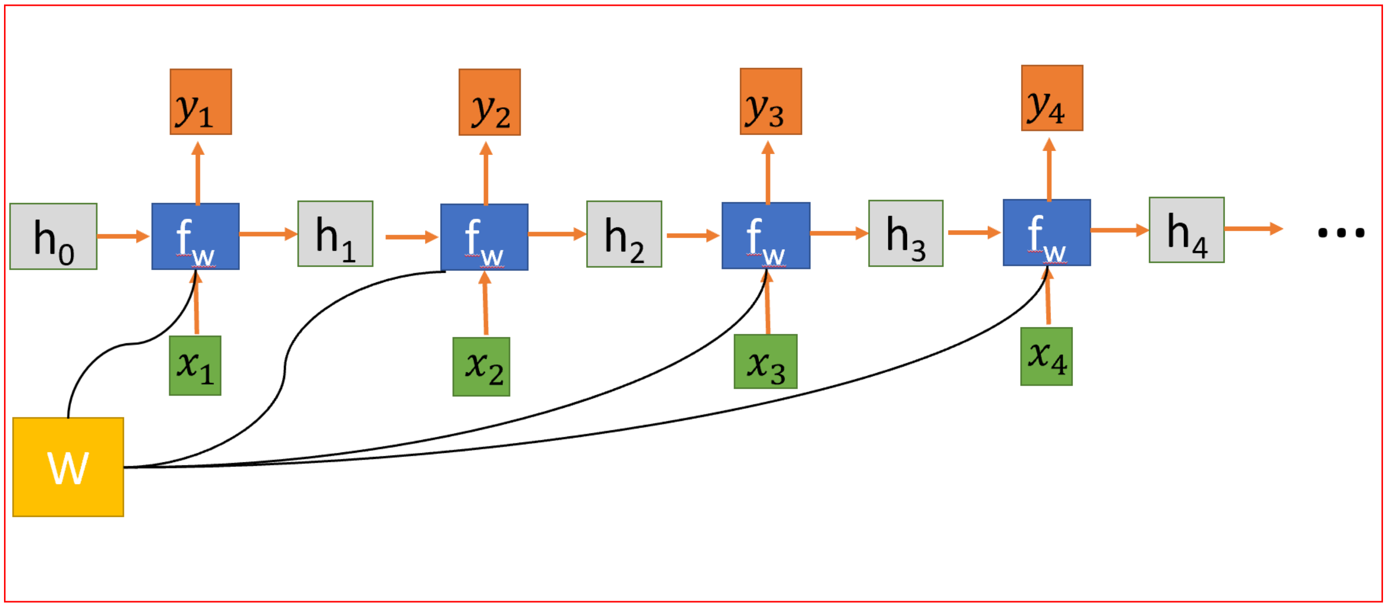
As we are processing sequential data, RNN takes an **input x at a time step t.** RNN takes the **hidden state value at time step t-1** to calculate the **hidden state h at time step t** and **applying a tanh activation function**. we use tanh or ReLU for non linearity in the **output y at time time t.**




h_{t-1}
f_w
h_t
x_t
$h_t = f_\omega(h_{t-1}, x_t)$
$h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t)$
$y_t = W_{hy}h_t$

Vanilla RNN at time step t with equations for hidden state and output

Unrolling RNN into a 4 layer neural network where weight matrix W is shared at every step.

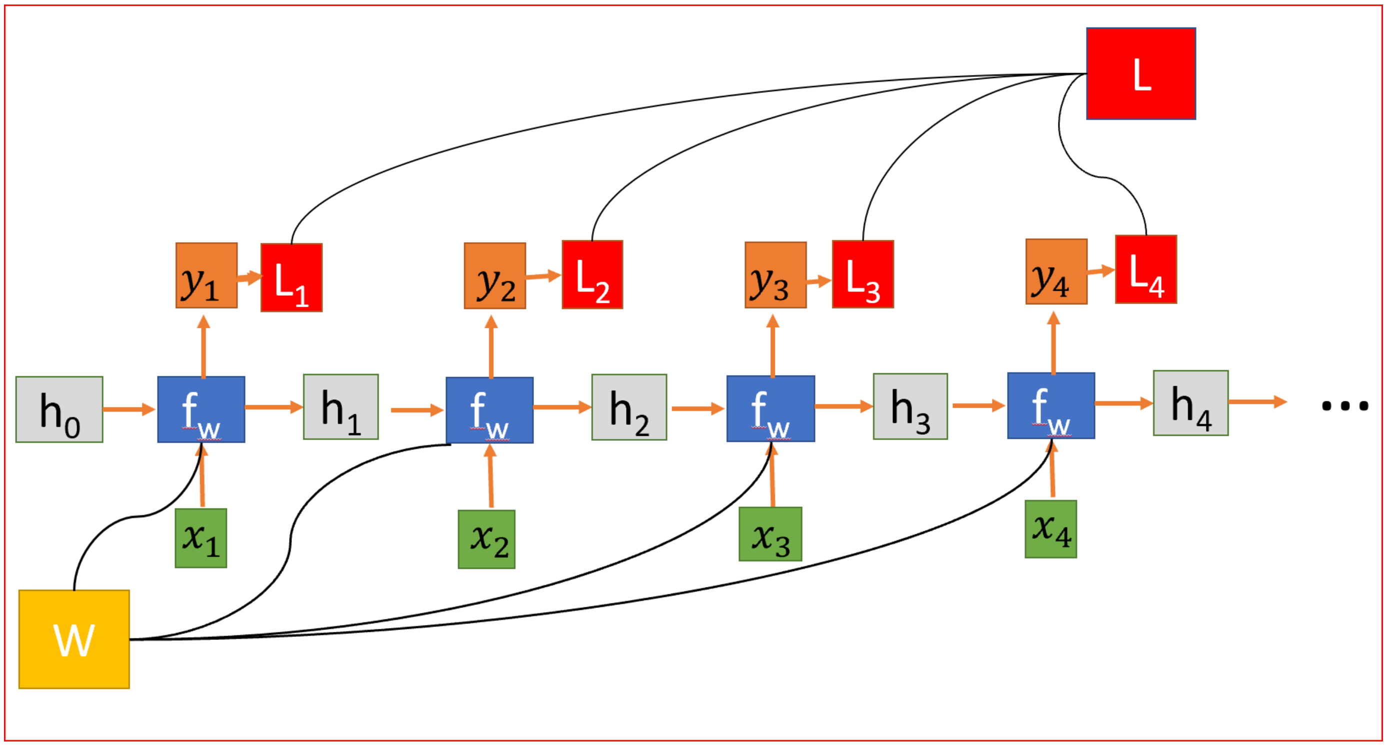


Unrolled RNN

Hidden state connects the information from the previous state and thus acts like memory for the RNN. The output at any time step depends on the the current input as well as the previous states.

Unlike other deep neural networks which uses a different parameter for each hidden layer, RNN shares the same weight parameter at each step.

we randomly initialize the weight matrices, and during the training we need to find the values of the matrices that give us desirable behaviour, so we calculate the **loss function L.** Loss function L is calculated by measuring difference between actual output and the predicted output.. To calculate L, we use Cross Entropy function.



RNN where loss function L is sum of all the loss across layers

To reduce the loss, we use back propagation but unlike traditional neural nets, RNN's share weights across multiple layers or in other words it shares weight across all the time steps. This way the gradient of error at each step is also dependent on the loss at previous steps.

In the above example to calculate the gradient at step 4, we need to sum up the losses of the previous 3 steps along with the loss at the fourth time step. This is called **Backpropagation through Time -BPPT.**

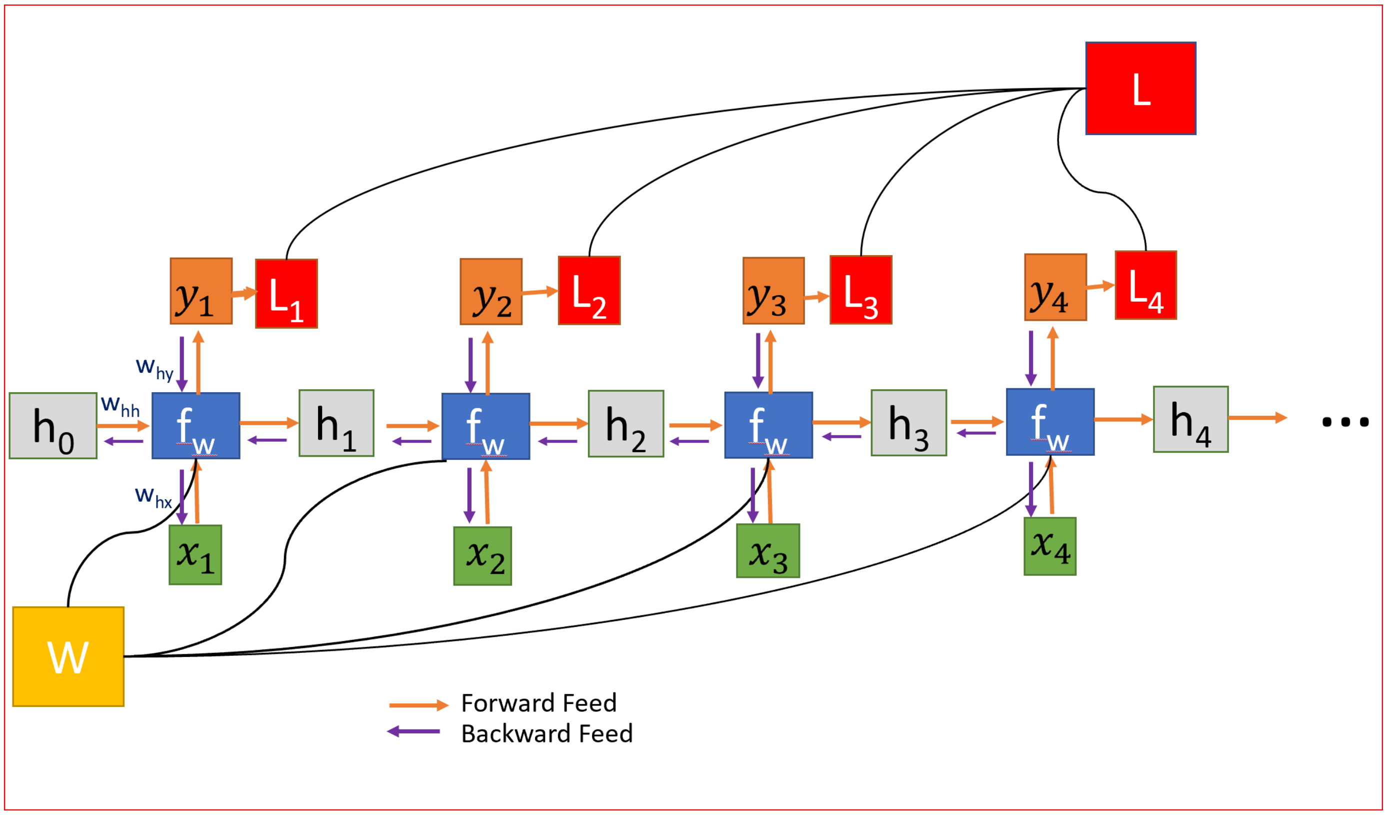
We calculate gradient of the error with respect to weights for us to learn the right weights for us get a desirable output.

As W is used in every step till the output we care about, we back propagate from t=4 to t=0. In traditional neural net we do not share weight so we do not need to sum the gradients however in RNN we share weights and we need to sum up the gradients for W at each step in time.

Computing gradient of h at time step t =0 involves many factors of W as we need to back propagate through each of the RNN cells. Even if we forget the weight matrix and multiply the same scalar value again and again for let's say 100 time steps and this will can be a challenge.

If the largest singular value is greater than 1 then the gradient would explode, called as **Exploding gradient**.

If the largest singular value is less than 1 then the gradient would vanish, called as **Vanishing gradient.**

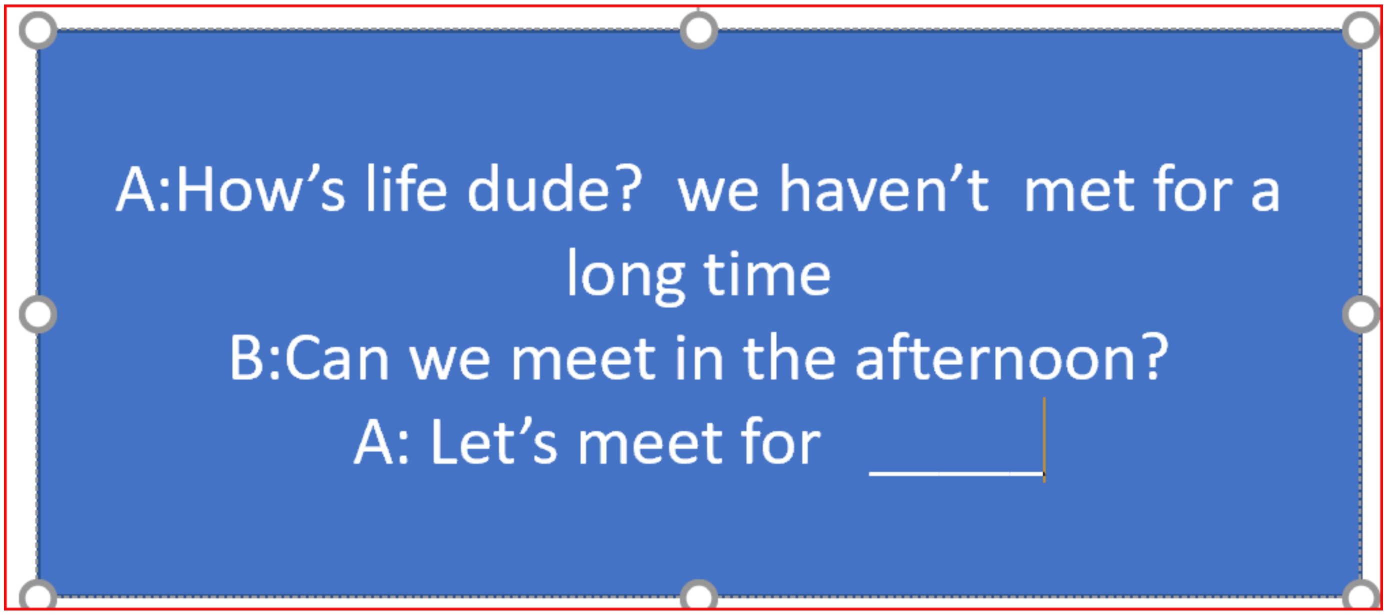


Forward and Backward feed and weights are shared across all layers causing either Exploding or Vanishing gradient

we can use **Gradient Clipping for exploding gradient** problem where we can pre set a threshold value and if the gradient value is greater than the threshold value we can clip it.

For solving vanishing gradient popular way it to use LSTM(Long Short Term Memory) or Gated Recurrent Unit(GRU).

In our message example, to predict the next word, we need to go back a few time steps to know the previous discussion. It is possible that we may have sufficient gap between two relevant information. As the gap grows it becomes difficult for RNN to learn and connect the information. This is when LSTM comes to our rescue.



A:How's life dude? we haven't met for a long time
B:Can we meet in the afternoon?
A: Let's meet for ______

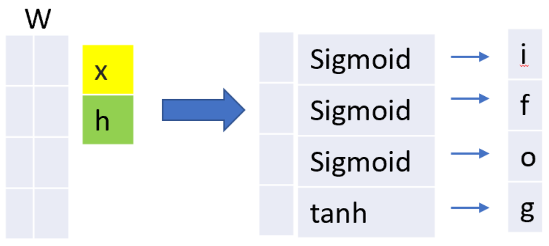
we need to go a few steps back to predict the next word. LSTMs can go back 1000's of time step

Long Short Term Memory(LSTM)

LSTMs are capable of learning long term dependencies much faster. LSTMs can learn to bridge time intervals in excess of 1000 steps. This is achieved by an efficient, gradient based algorithm that uses constant error flow through the internal states

To predict next word in the message, we can store the context to the very start of the message so that we have the correct context. This is exactly how our memories work.

Let's dive in into the LSTM architecture to understand how it works



i= input gate – whether to write to cell
f= forget gate – whether to erase the cell
o=output gate – how much to reveal cell
g=gate gate – how much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tan h \end{pmatrix} w \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

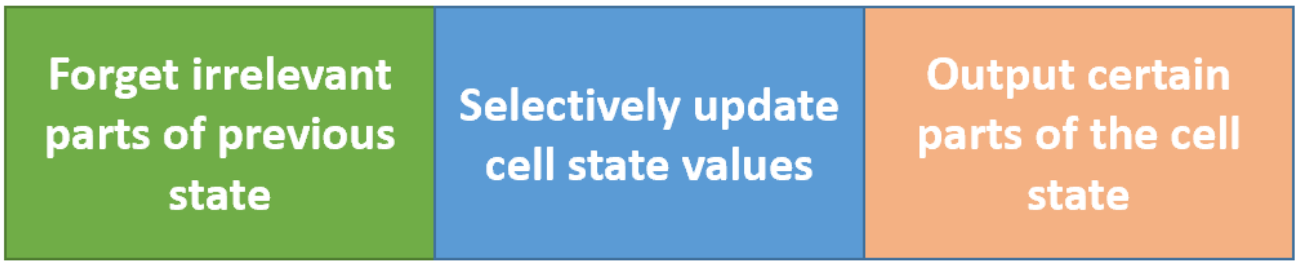
$$C_t = f \Theta C_{t-1} + \mathrm{i} \Theta g$$
$$h_t = 0 \Theta \tanh(C_t)$$

Long Short Term Memory-LSTM

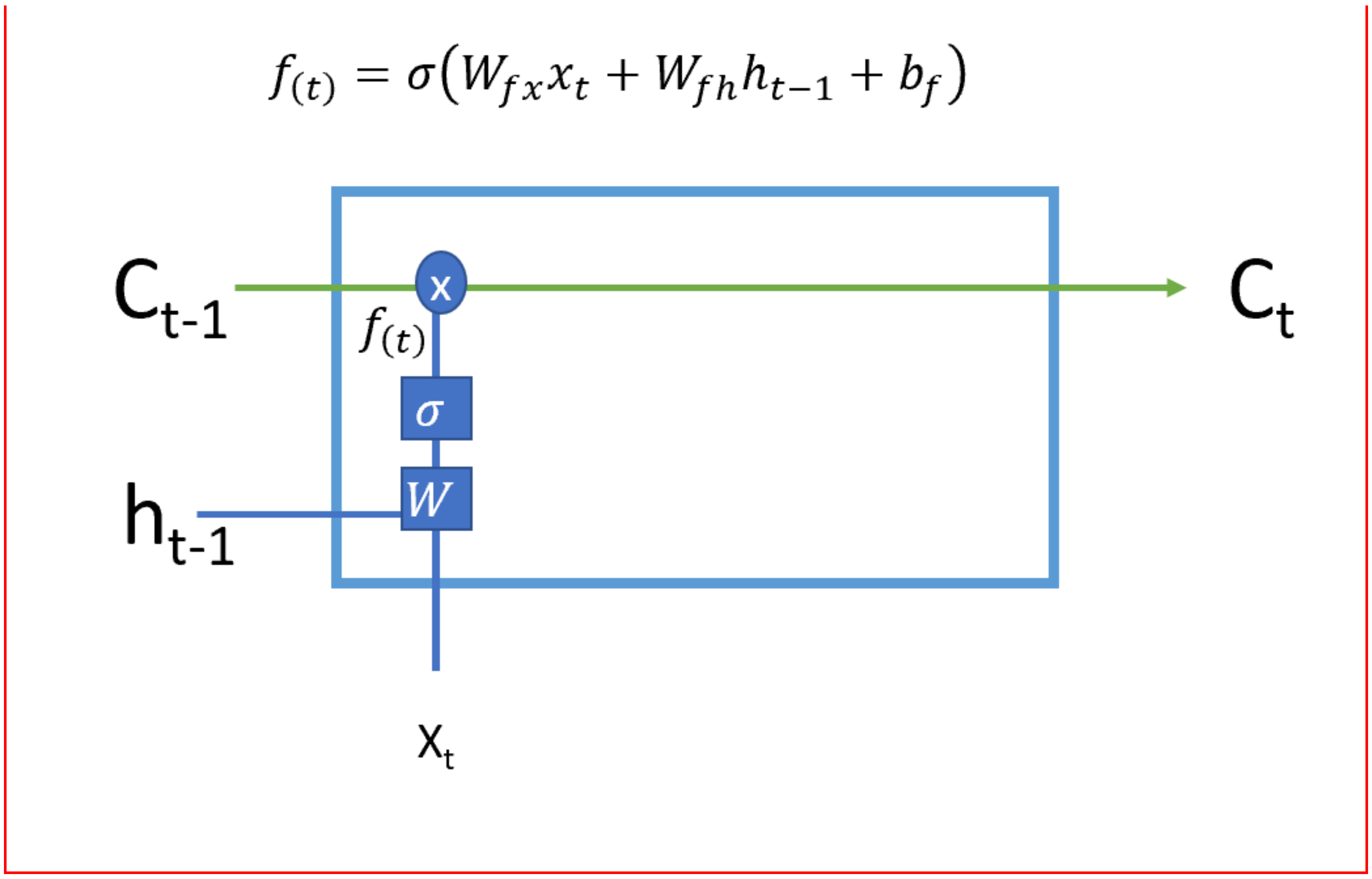
LSTMs behaviour is to remember information over a long time step, so it needs to know what to remember and what to forget.

LSTM uses 4 gates, you can remember them as ifog, to know if we need to remember the previous state. Cell states play a key role in LSTMs. LSTM can decide if they want to add or remove information from a cell state using the 4 regulated gates.

These gate acts like faucets that determine how much information should flow through.

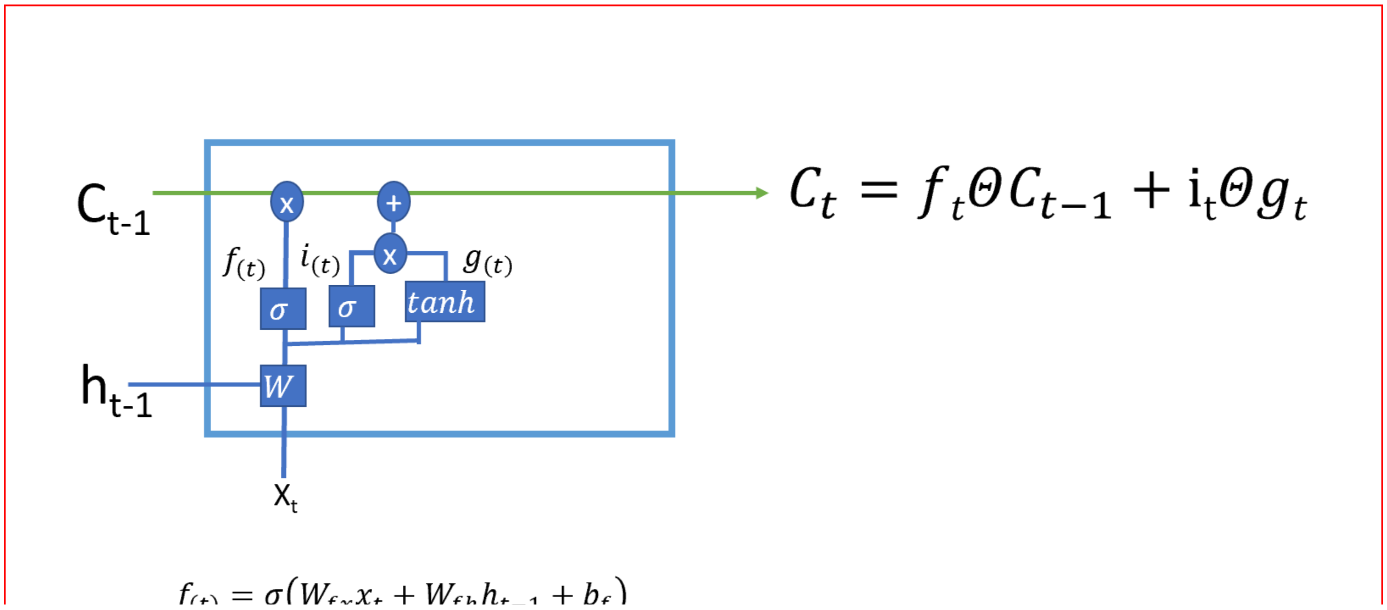

Forget irrelevant parts of previous state
Selectively update cell state values
Output certain parts of the cell state

Three steps in LSTM



Forget gate uses sigmoid activation to decide whether to forget to remember the value of previous cell state

1. First step in LSTM is to decide if we need to **remember or forget** the cell state. **Forget gate** uses sigmoid activation function which gives an output value of 0 or 1. An output of 1 from forget gate tells us that we want to keep the value and a value of 0 tells us that we want to forget the value.


C_{t-1}
x
+
x
$f_{(t)}$
$i_{(t)}$
$g_{(t)}$
σ
σ
$tanh$
W
h_{t-1}
X_t
$C_t = f_t \Theta C_{t-1} + \mathrm{i}_t \Theta g_t$
$f_{(t)} = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$

$$i_{(t)} = \sigma(W_{ix}x_t + W_{ix}h_{t-1} + b_i)$$

$$g_{(t)} = tanh\left(W_{gx}x_t + W_{gh}h_{t-1} + b_g\right)$$

2. Second step to decide what new information will we store in the cell state. This has two parts one is the input gate, which decide whether to write to the cell state using the sigmoid function and the gated gate, which decides how much to write to the cell state using tanh activation function. Gated gate creates a vector or new candidate values that can be added to the cell state

$$h_t = 0_t \Theta \tanh(C_t)$$

hidden state output

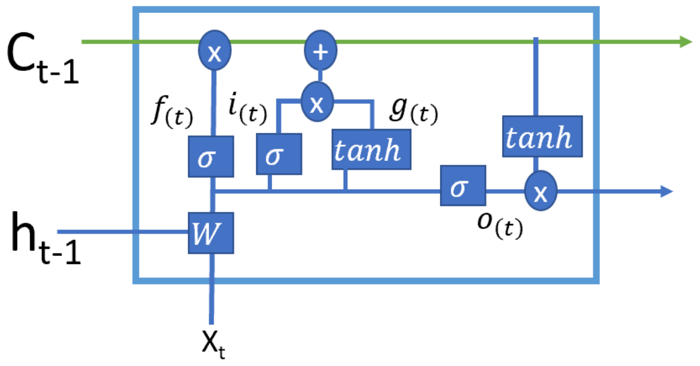
3. In the last step we create the cell state by combining the outputs from step 1 and step2 which is multiplying the cell state after applying a tanh activation function for the current time step by the output gate's output. Tanh activation function gives and output range between -1 and +1

4. Cell state which is the internal memory of the unit combines the previous cell state multiplied by the forget gate and then new computed hidden state in g, multiplied by the input gate i's output.

$$C_t = f_t \Theta C_{t-1} + \mathrm{i}_t \Theta g_t$$

Cell state for current time step

Finally the output will be based on the cell satte but will be a filtered version.



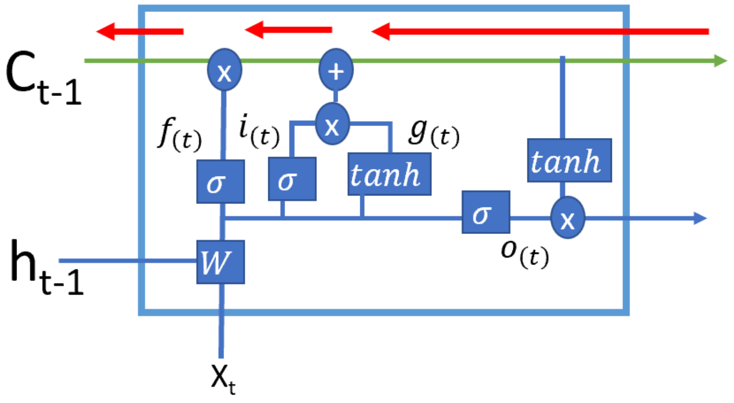
$$C_t = f_t \Theta C_{t-1} + \mathrm{i}_t \Theta g_t$$

$$h_t = 0_t \Theta \tanh(C_t)$$

$$f_{(t)} = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$
$$i_{(t)} = \sigma(W_{ix}x_t + W_{ix}h_{t-1} + b_i)$$
$$g_{(t)} = tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g)$$
$$o_{(t)} = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$

LSTM

Backpropagation from current cell state to previous cell sate only has element wise multiplication by forget gate, there is no matrix multiply by W. This creates a back propagation super highway using the cell states eliminating the vanishing and exploding gradient issue.



$$C_t = f_t \Theta C_{t-1} + \mathrm{i}_t \Theta g_t$$

$$h_t = 0_t \Theta \tanh(C_t)$$

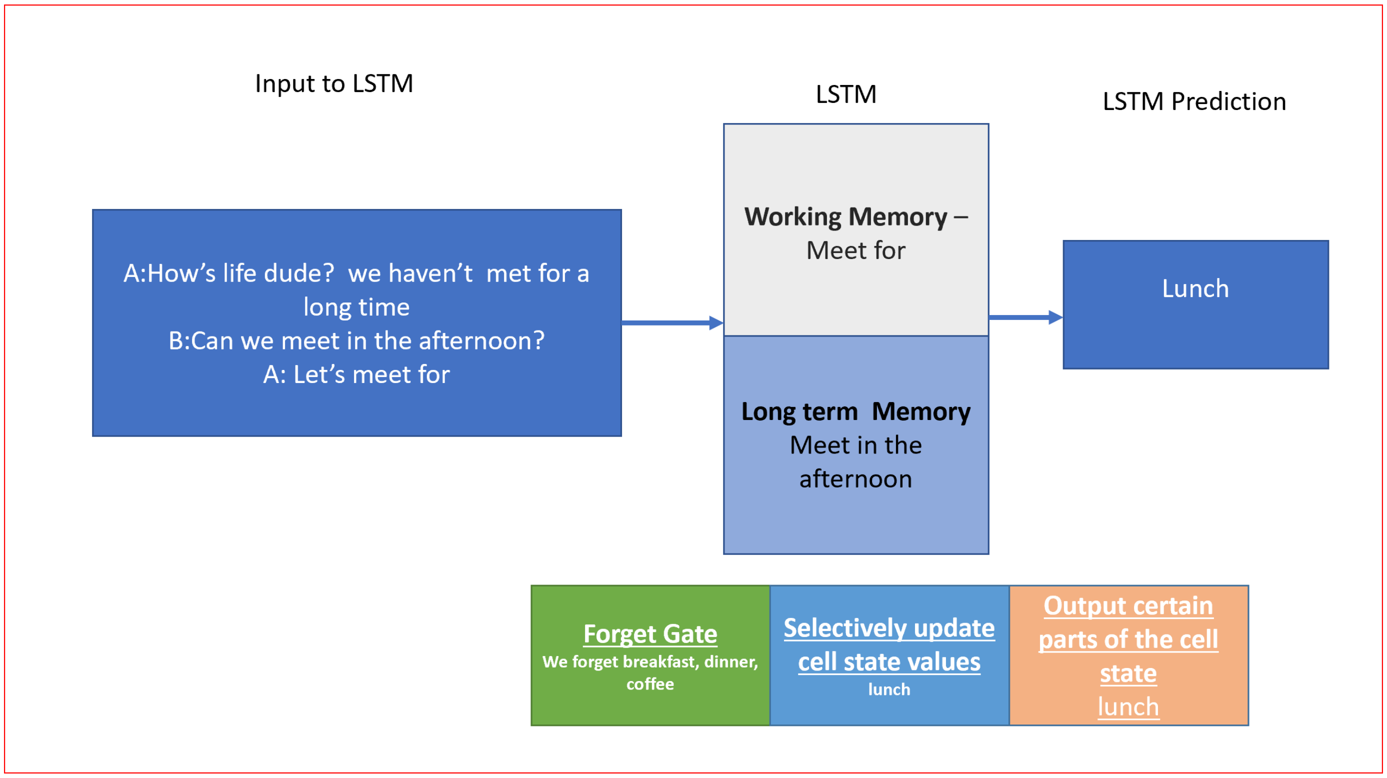
$$f_{(t)} = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$
$$i_{(t)} = \sigma(W_{ix}x_t + W_{ix}h_{t-1} + b_i)$$
$$g_{(t)} = tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g)$$

$$o_{(t)} = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$

Back propagation super highway using cell states

LSTM decides when and how to transform the memory at each time step by deciding what to forget and what to remember, which information to update. This is how LSTMs helps with storing long term memory.

A sample of how LSTM would make prediction for our message


Input to LSTM
LSTM
LSTM Prediction
A:How's life dude? we haven't met for a long time
B:Can we meet in the afternoon?
A: Let's meet for
Working Memory – Meet for
Long term Memory Meet in the afternoon
Lunch
Forget Gate
We forget breakfast, dinner, coffee
Selectively update cell state values
lunch
Output certain parts of the cell state
lunch

LSTM prediction

GRU — a variant of LSTM

GRU uses two gates **reset gate and an update gate** unlike the three steps in LSTM. GRU does not have an internal memory

Reset gate decides how to combine the new input with the previous time steps's memory.

Update gate decides how much of the previous memory should be kept. Update gate is a combination of input and forget gate that we understood in LSTM.

GRU is simpler variant of LSTM to solve vanishing gradient problem

Inspiration and Reference sources

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

http://cs231n.stanford.edu/

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Read it, share it and give claps if it helped you gain a better understanding.

Machine Learning Deep Learning Recurrent Neural Network Lstm