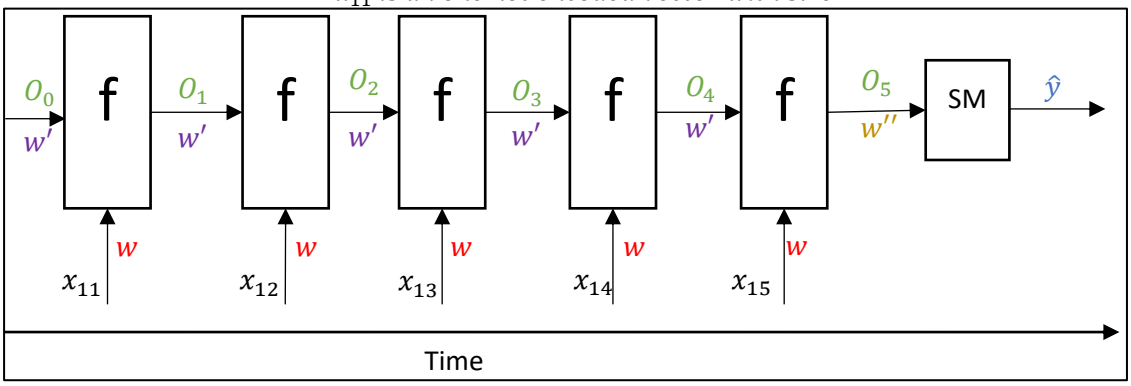
- **Recurrent (Repeating) Neural Networks**
  - Recurrent Neural Networks (RNN) are sequence models that are more advanced alternative to traditional Neural Networks. Right from Speech Recognition , Natural Language Processing , Music Generation, Machine Translation, image captioning RNNs play a transformative role in handling sequential datasets. E.g., time series data, sentences
  - Let's start with an example. The task is to classify sequence of words into positive or negative (1,0)

$$x_1 \rightarrow [x_{11}, x_{12}, x_{13}, x_{14}, x_{15}] : y_1$$

$$x_{11} \text{ is an one hot encoded vector with size } \mathbb{R}^D$$

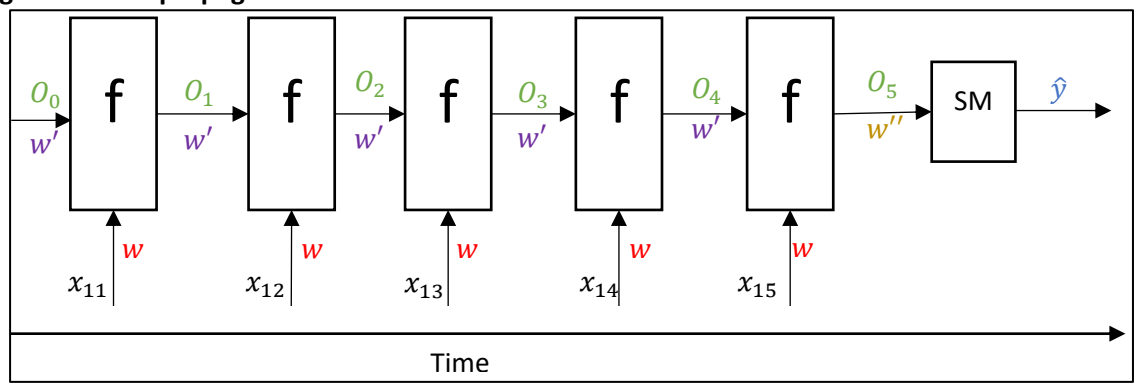


$$O_1 = f(w * x_{11} + w' * O_0)$$

  - Because of the repetitive structure it is called as recurrent neural networks
  - For each sentence we have 3 sets of weights i.e., w (used with the input values), w' (used with the prebious output values i.e., o) and w'' used in the SoftMax layer
  - Let's take an approach of how RNN trains on these sentences:
    - 1st sent: My name is Ram.
    - 2nd sent: My city is Delhi.
    - Vocab: [my, name, is, ram, city, Delhi]
    - vectorizer:

      Sentence1: [[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]]
      Sentence2: [[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1]]
    - c. After forward propagation of one datapoint we backpropagate and send in the next datapoint
      - time step 1 (word 'my'): input0 -> [1, 0, 0, 0, 0, 0]
      - weight input -> w
      - output0 -> [0, 0, 0, 0, 0, 0]
      - weight output -> w'
      - output1 = f (w*input0 + w' * output0)
      - y1 = SoftMax(w''*output1)
      - time step 2(word 'name'): input1 -> [0, 1, 0, 0, 0, 0]
      - weight input -> w
      - output1 -> output1 from previous time step
      - weight output -> w'
      - output2 = f (w*input1 + w' * output1)
      - y2 = SoftMax(w''*output2)
    - and repeats till it reaches word 'Ram'.
    - Now it backpropagates through time and updates the weights. After this sentence 2 - > 'My city is Delhi' undergoes the same process. This continues till we reach an optimal accuracy (or any other metric such as recall, precision. etc. that we choose)
  - We follow the below procedure-
    1. Initialize weight matrix
    2. Forward propagation to compute predictions
    3. Compute the loss
    4. Backpropagation to compute gradients
    5. Update weights based on grad
    6. Repeat steps 2–5
  - Note: The above layers are not multiple layers they are actually one layers over time

- **Training RNNs: Backpropagation over time**



  - The forward propagation happens in following way

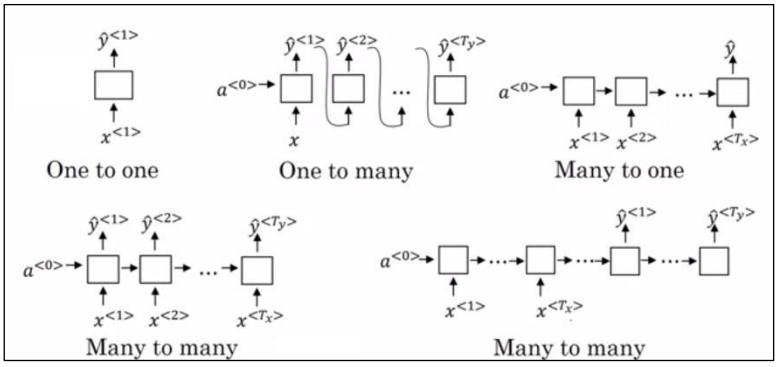
$$o_1 = f(w * x_{i1} + w' * O_0)$$
$$o_2 = f(w * x_{i2} + w' * O_1)$$
$$\vdots$$
$$y_i' = g(w'' * o_{10})$$

  - In backpropagation, we update w, w', w'' such that loss is minimized

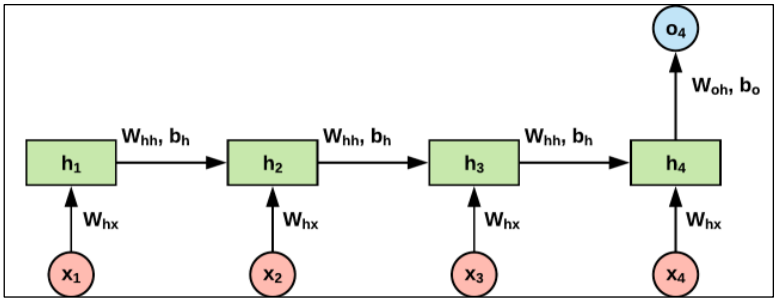
$$\frac{\partial L}{\partial w''} = \frac{\partial L}{\partial y_i'} * \frac{\partial y_i'}{\partial w''}$$
$$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial y_i'} * \frac{\partial y_i'}{\partial O_{10}} * \frac{\partial O_{10}}{\partial w'}$$
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y_i'} * \frac{\partial y_i'}{\partial O_{10}} * \frac{\partial O_{10}}{\partial w}$$

  - Due to these many multiplication computations when we reach first word, we run into a vanishing gradient problem

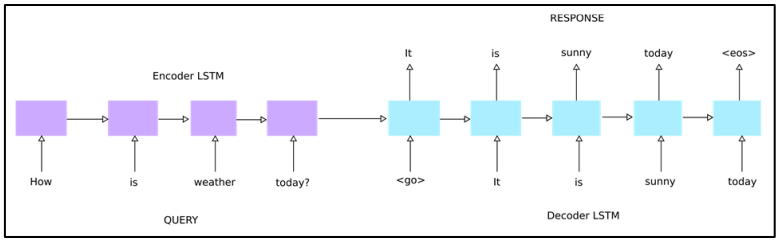
- **Types of RNNs.**



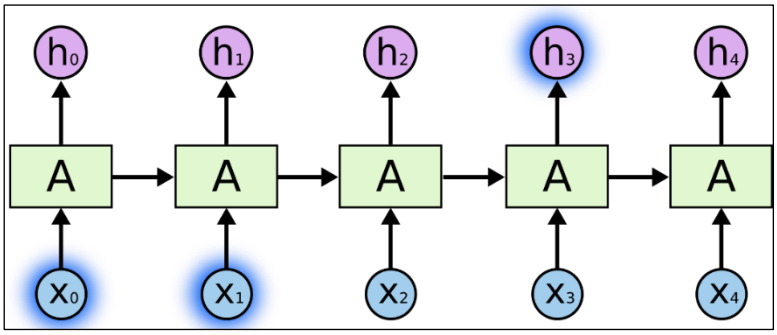
  - Many to one: many inputs will be provided but output will be only one. E.g., sentiment analysis, review rating prediction



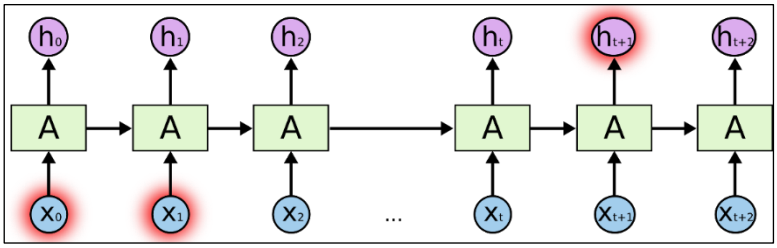
  - One to many: input will be only one but outputs will be multiple e.g., Image caption prediction
  - Many to many
    - Input is sequence & so is the output where input & output will be of same size**.** E.g., part of speech detection
    - Input is sequence & so is the output where input & output will be of different size E.g., machine translation



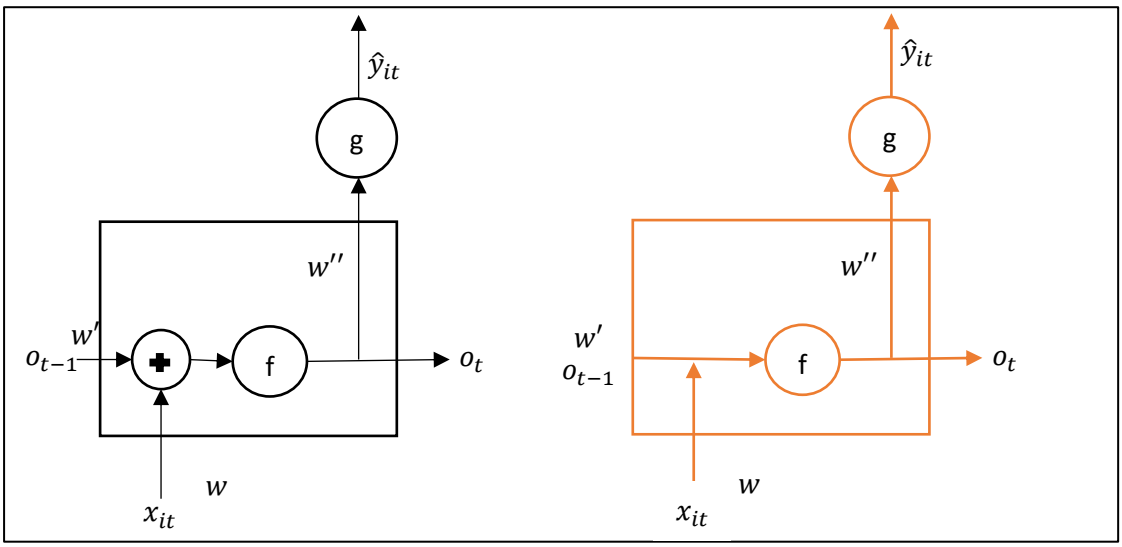
- **Need for LSTM/GRU**
  - Sometimes, we only need to look at recent information to perform the present task. For example, if we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



  - But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France… I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
  - Unfortunately, as that gap grows, RNNs unable to learn to connect the information.



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



  - In RNN,

$$O_t = f(w * x_{it} + w' * o_{t-1})$$

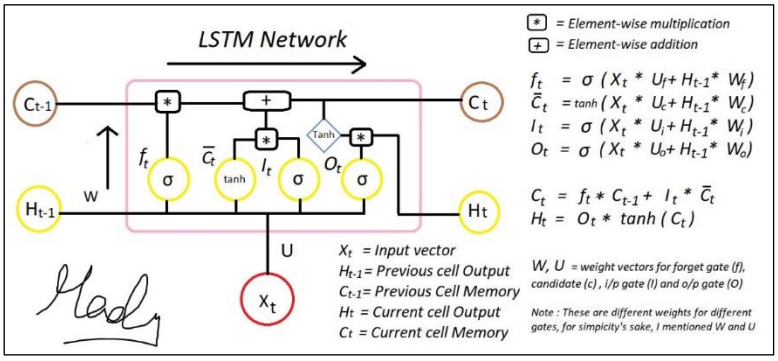
$$\hat{y}_{it} = g(w'' * o_t)$$

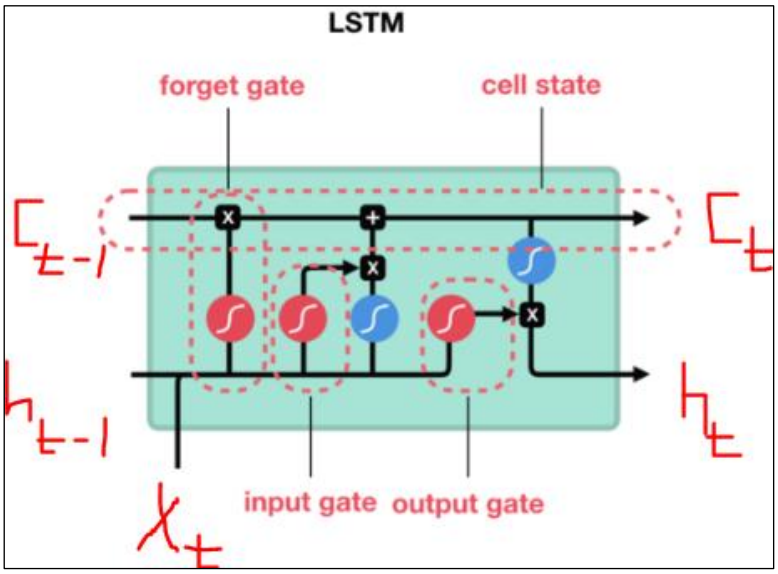
  - In LSTM, instead of plus we perform merge/concatenation operation to form new vector/ matrix.

$$O_t = f([w', w], [x_{it}, o_{t-1}])$$

$$\hat{y}_{it} = g(w'' * o_t)$$

- Have a look at this: https://imgur.com/a/Ho2EkBm . The internal structure of LSTM can be broken down into 3 different gates, primarily the input, output and the forget gate.



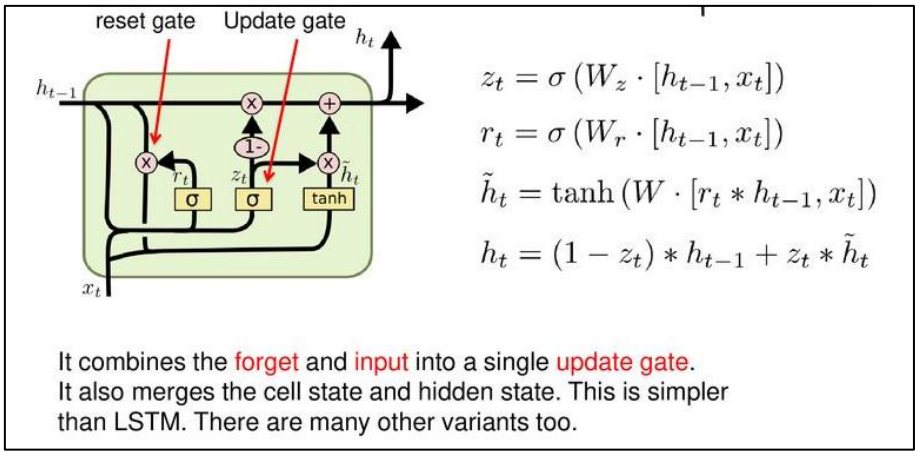


1. **Forget Gate (weights corresponding to W_f):** First, we have the forget gate (W_f). This gate decides what information should be thrown away or kept. Information from the previous hidden state$\big(h(t-1)\big)$ and information from the current input $\big(x(t)\big)$ is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.
2. **Input Gate (weights corresponding to W_i):** To update the cell state, we have the input gate. First, we pass the previous hidden state$(h(t-1))$ and current input $x(t)$ into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state$(h(t-1))$ and current input$(x(t))$ into the $tanh$ function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.
3. **Cell State:** Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state(c(t)).
4. **Output Gate:** Last we have the output gate. The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The

output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

- To review, the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.
- This is a good video explanation of internals of LSTM: https://www.youtube.com/watch?v=8HyCNIVRbSU .

- **GRU (Gated Recurrent Unit)**



- Introduced in 2014 which is inspired from LSTM which is faster to train & as powerful as LSTM