LSTM

Long Short Term Memory

RNN: Recurrent Neural Network

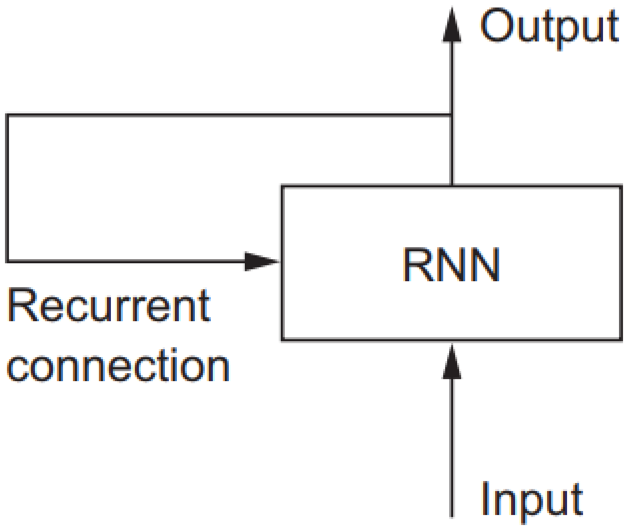


Figure 6.9 A recurrent network: a network with a loop

- Output at each time step (t)
  - Becomes input of next time step (t+1)

- Model Input @ time (t)
  - Integration of two parts
    - Real input
    - Recurrent input = output of previous time (recurrent) (t-1)

https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470

RNN: Recurrent Neural Network

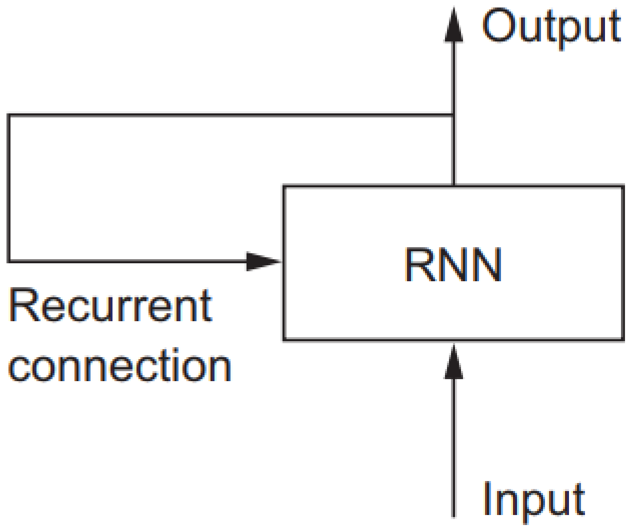


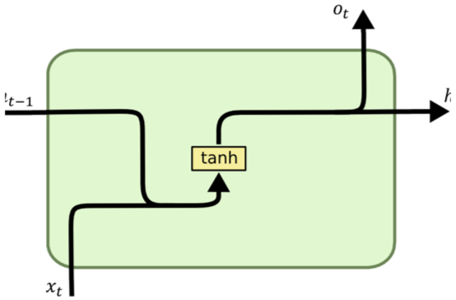
Figure 6.9 A recurrent network: a network with a loop

- Recurrent connection
  - Allow to pass past information to current process
    - As model memory
      - Help to capture long term dependency in a sequence of data

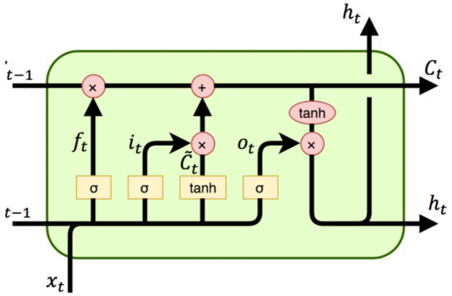
https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470

RNN: Recurrent Neural Network

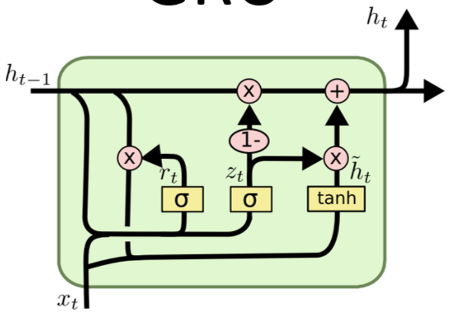
RNN



LSTM



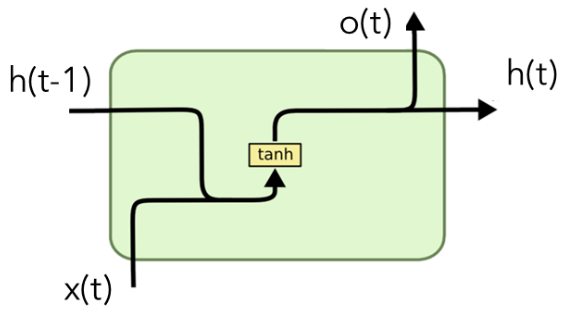
GRU



- Variation in RNN Architecture
  - Standard RNN
  - LSTM
    - Long Short Term Memory
  - GRU
    - Gated Recurrent Unit

https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470

RNN: Recurrent Neural Network



$h_\theta(\,K(x_t, h_{t-1})\,)$

$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$$

$h_\theta(\,K(h_t)\,)$

- Standard RNN cell
  - Consists of 2 Neurons
    - With recurrent connection
    - 1st Neuron
      - Process input (x(t)) and past hidden state (h(t-1))
        - With tanh activation

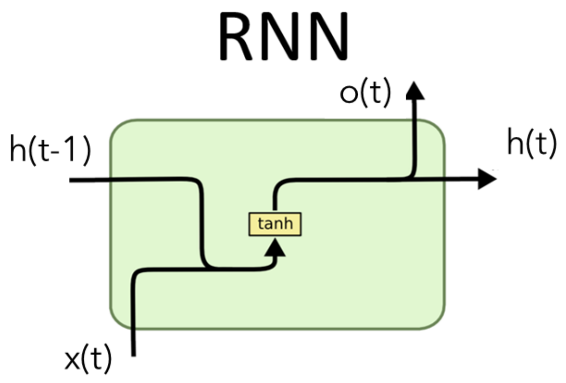
$h_\theta(\,K(x_t, h_{t-1})\,)$

    - 2nd Neuron
      - Process current hidden state to form output (o(t))

$h_\theta(\,K(h_t)\,)$

https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470

RNN: Recurrent Neural Network



$$h_\theta(\, K(x_t, h_{t-1})\,)$$

$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

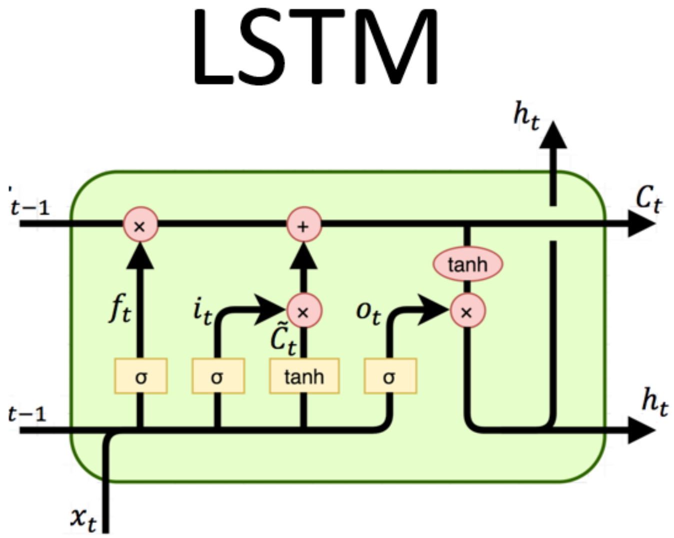
$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h)$$

$$h_\theta(\, K(h_t)\,)$$

- Standard RNN cell
  - Limitation
    - Vanishing Gradient
      - Like most neural networks, recurrent nets are old. By the early 1990s,
      - the vanishing gradient problem emerged as a major obstacle to recurrent net performance.
    - the gradient expresses the change in all weights
      - with regard to the change in error.
    - If we can't know the gradient,
      - we can't adjust the weights in a direction that will decrease error, and
      - network ceases to learn.

https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470

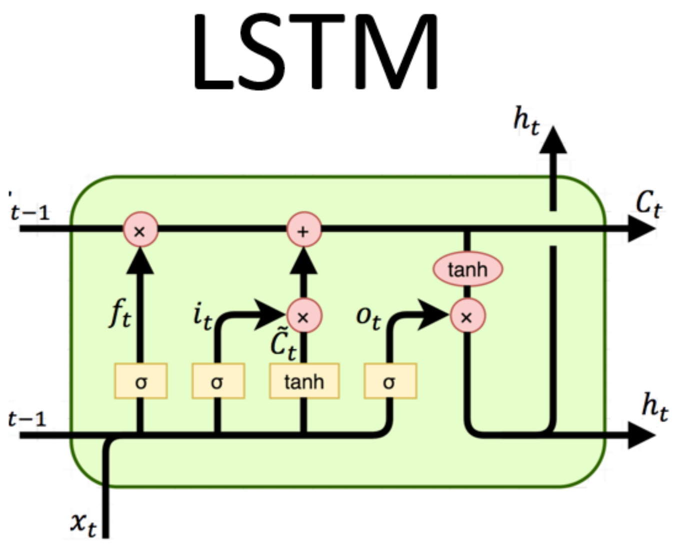
RNN: Recurrent Neural Network



- LSTM (mid-90s)
  - a variation of recurrent net
    - proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber
    - as a solution to the vanishing gradient problem.
      - allow recurrent nets to continue to learn over many time steps (over 1000)
      - The central plus sign is essentially the secret of LSTMs.
      - Simple, this basic change helps to preserve a constant error when it must be backpropagated at depth.

https://pathmind.com/wiki/lstm

RNN: Recurrent Neural Network



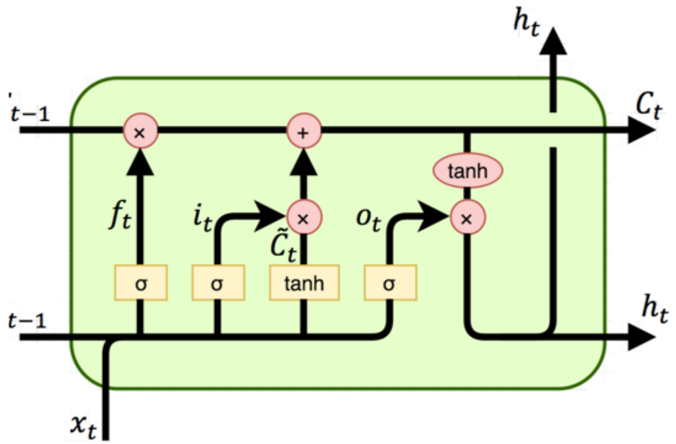
- LSTM (mid-90s)
  - a variation of recurrent net
    - proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber
    - as a solution to the vanishing gradient problem.
      - allow recurrent nets to continue to learn over many time steps (over 1000)
      - The central plus sign is essentially the secret of LSTMs.
      - Simple, this basic change helps to preserve a constant error when it must be backpropagated at depth.

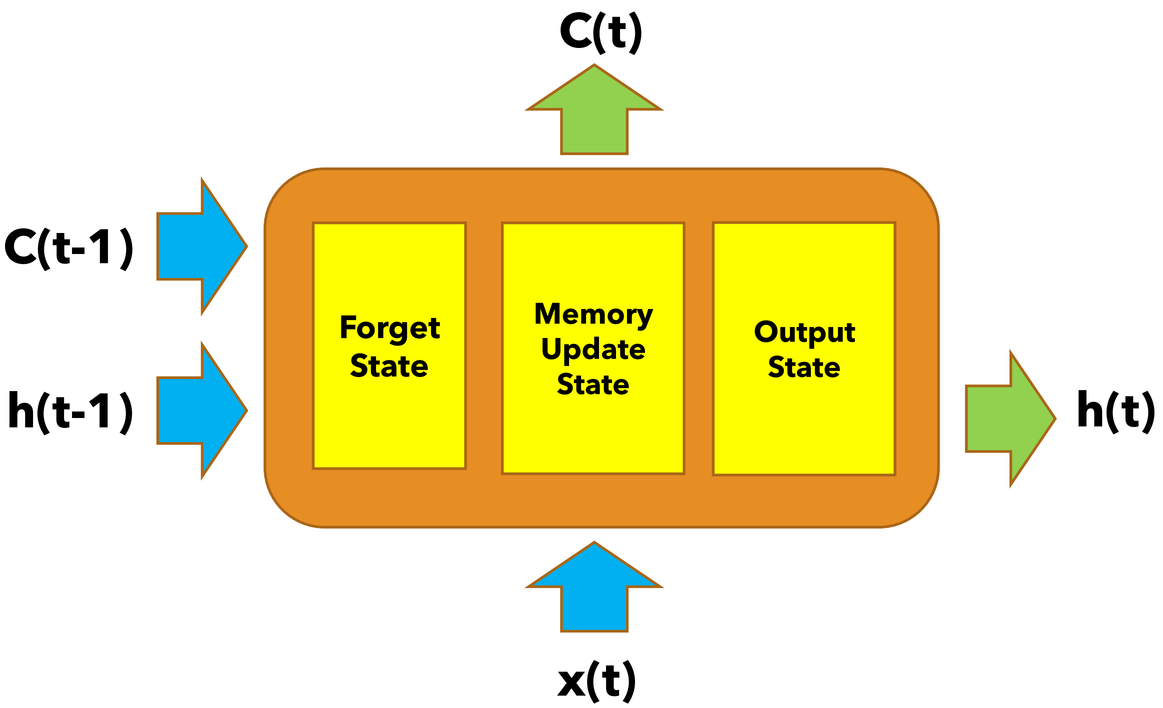
https://pathmind.com/wiki/lstm

RNN: Recurrent Neural Network

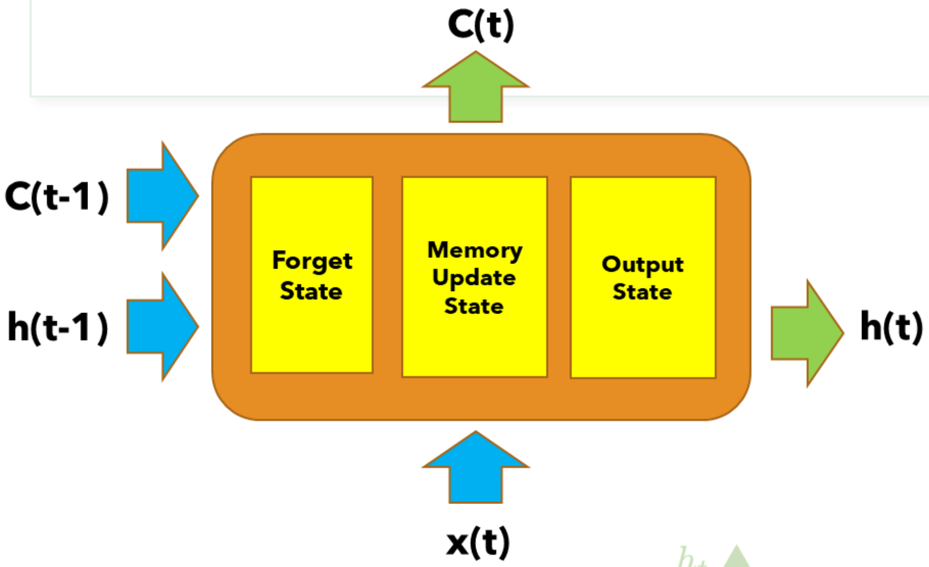
- Consists of 3 states

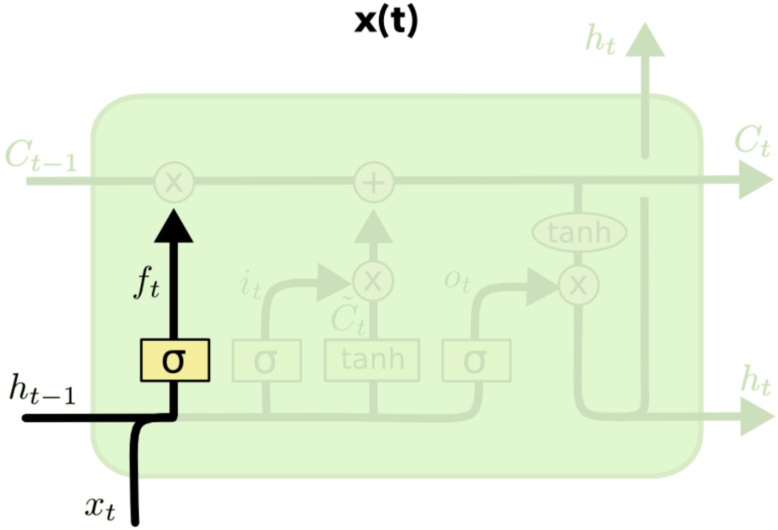
LSTM





RNN: Recurrent Neural Network





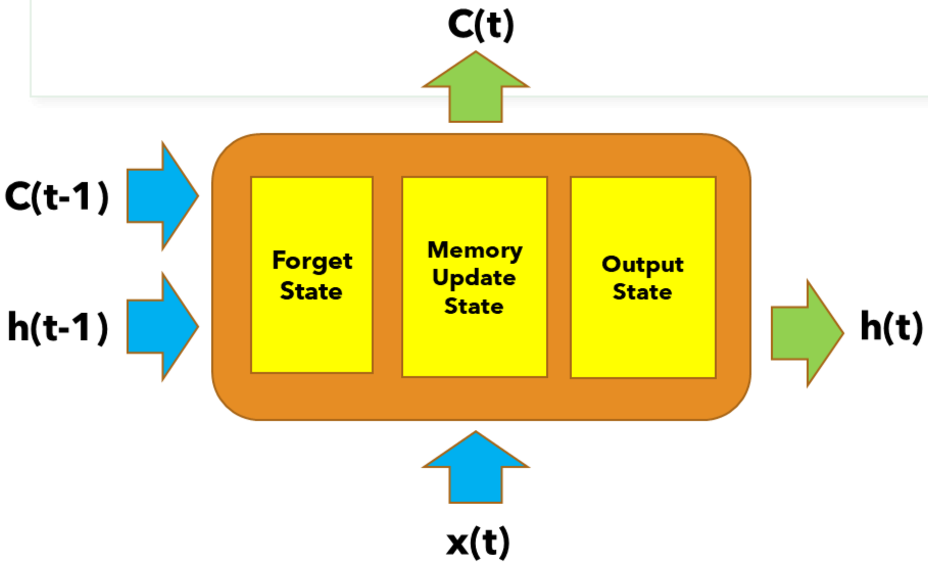
- Forget State (with Forget Gate)

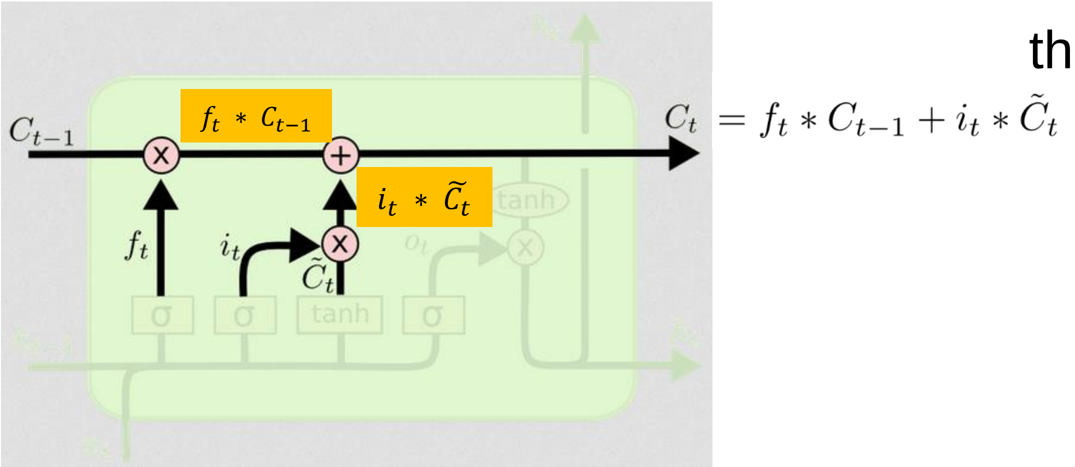
$$h_\theta(\, K(x_t, h_{t-1})\,)$$

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

- Identify how much past memory (C(t-1))
  - should be forget and passed to cell
  - can be viewed as
    - Filtering weights of past memory

RNN: Recurrent Neural Network





- Memory Update State $\boxed{C_t = f_t * C_{t-1} + i_t * \tilde{C}_t}$
  - Filtered past memory using forget weights

$$f_t \, * \, C_{t-1}$$

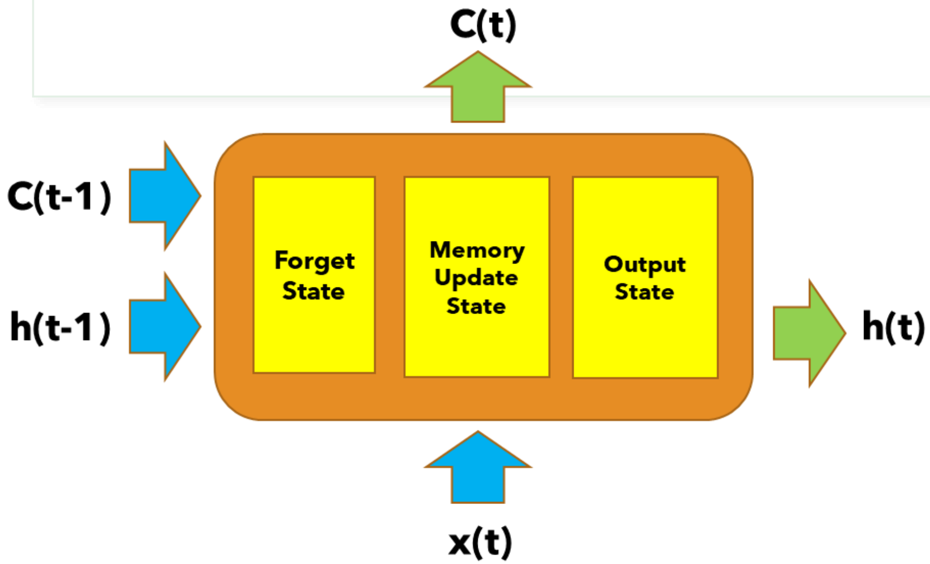
  - Integrate the new significant input information to the memory

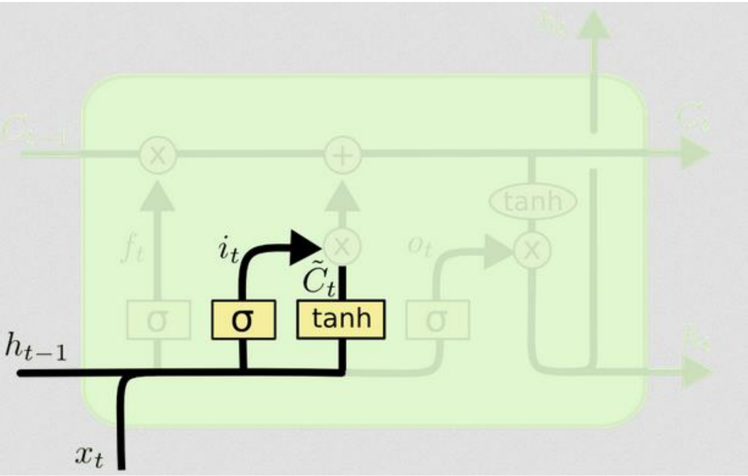
$$i_t \, * \, \tilde{C}_t$$

i_t : Input Gate Weight

$\tilde{C}_t$: significant input information

RNN: Recurrent Neural Network





- Estimate significant input information though
  - Input Gate

$$i_t \; x \; \tilde{C}_t$$

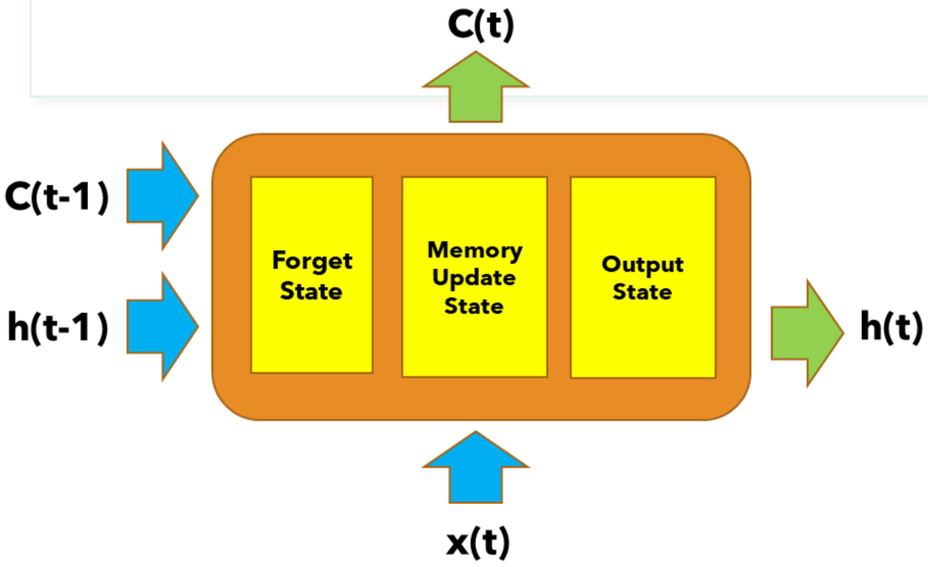
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$

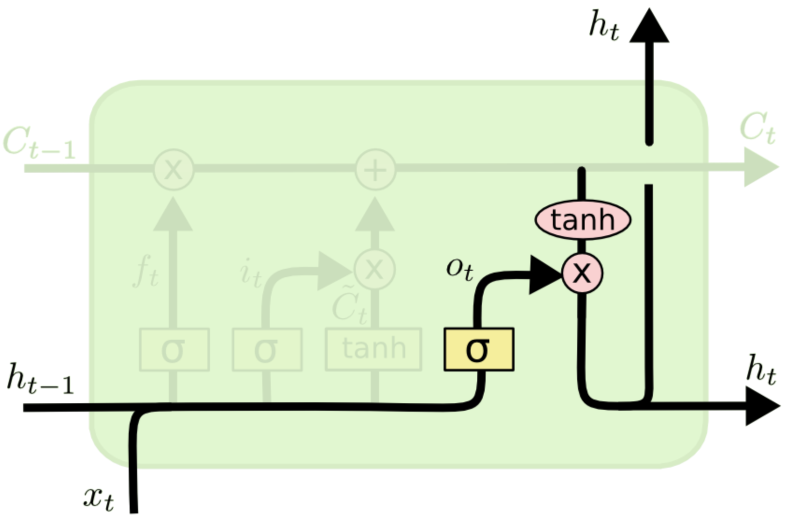
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

i_t : Input Gate Weight

$\tilde{C}_t$: significant input information

RNN: Recurrent Neural Network





- Output State
  - Filtering weights from past state and current input

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

  - Integrate the new cell state output estimated from current updated memory

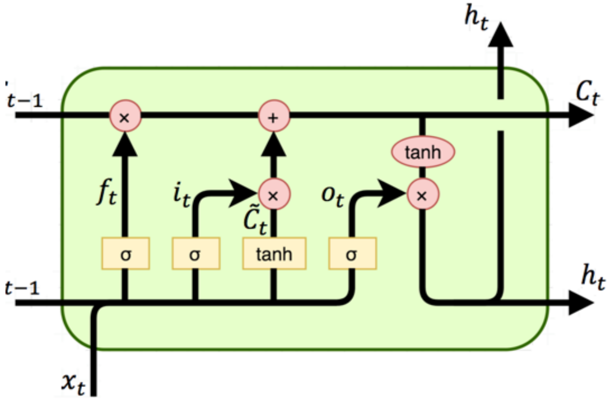
$$\tilde{h}_t = \tanh(\, C_t \,)$$

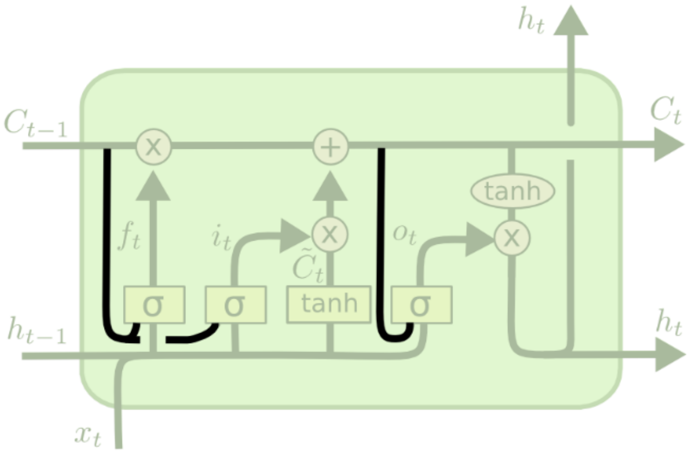
  - Update Output cell state (filtered estimated cell state)

$$h_t = O_t * \tilde{h}_t = O_t * \tanh(\, C_t \,)$$

LSTM variation

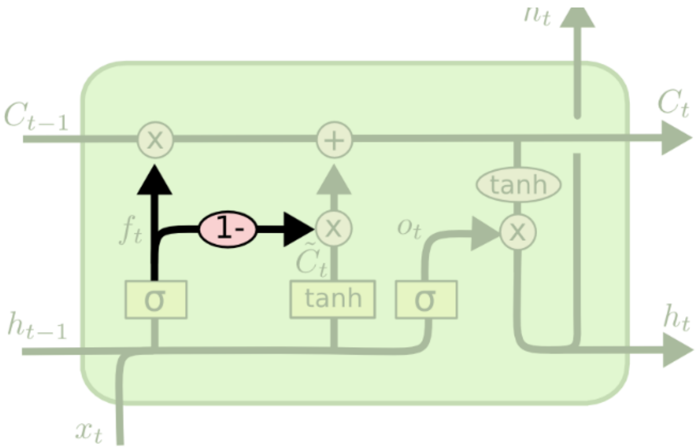
LSTM





introduced by Gers & Schmidhuber (2000)

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

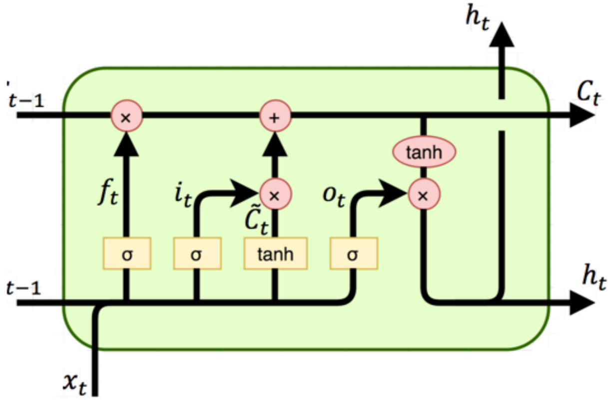


Cho, et al. (2014)

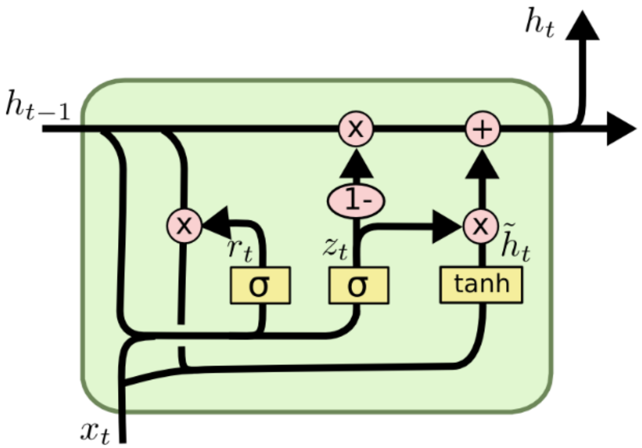
$$C_t = f_t * C_{t-1} + (\boldsymbol{1 - f_t}) * \tilde{C}_t$$

LSTM variation

LSTM



Yao, et al. (2015)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

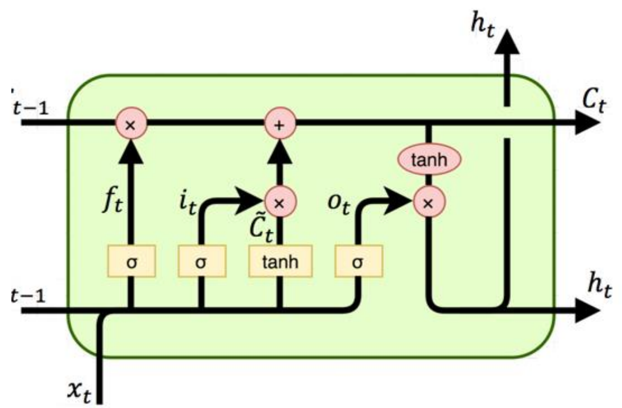
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

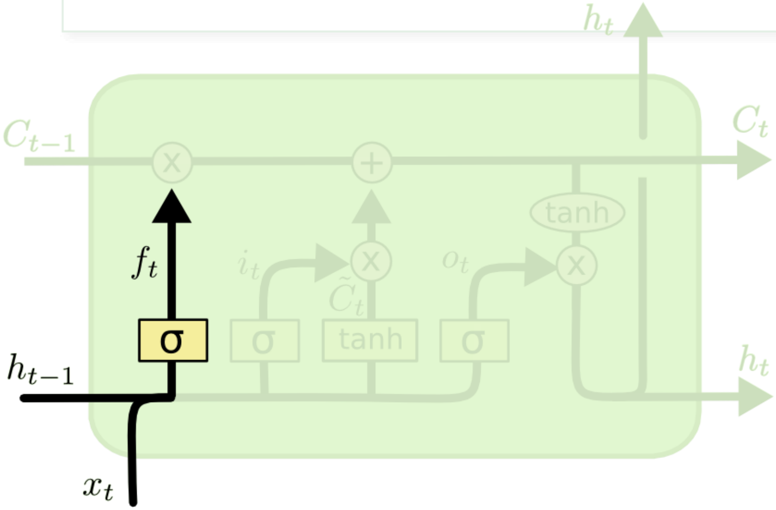
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM Numerical Example

- Forget State (with Forget Gate)





$$x_t = [1, 2, 3], h_{t-1} = [4, 5, 6],$$

concatenate $[x_t, h_{t-1}]$

$$[x_t, h_{t-1}] = [1, 2, 3, 4, 5, 6])$$

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

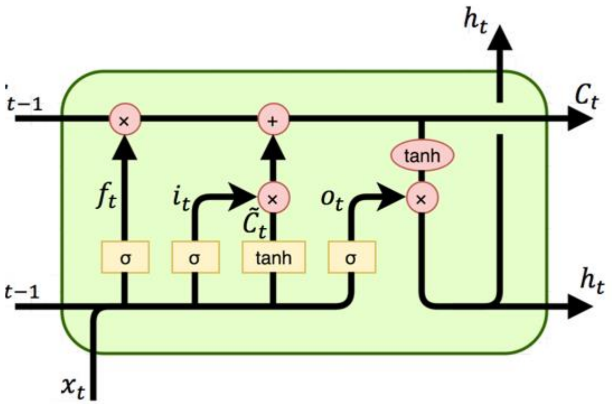
$$W_f . [h_{t-1}, x_t] = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 91 & 175 & 133 \end{bmatrix}$$

$$z = K(x_t, h_{t-1}) = W_f . [h_{t-1}, x_t] + b_f = \begin{bmatrix} 91 & 175 & 133 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 92 & 177 & 136 \end{bmatrix}$$

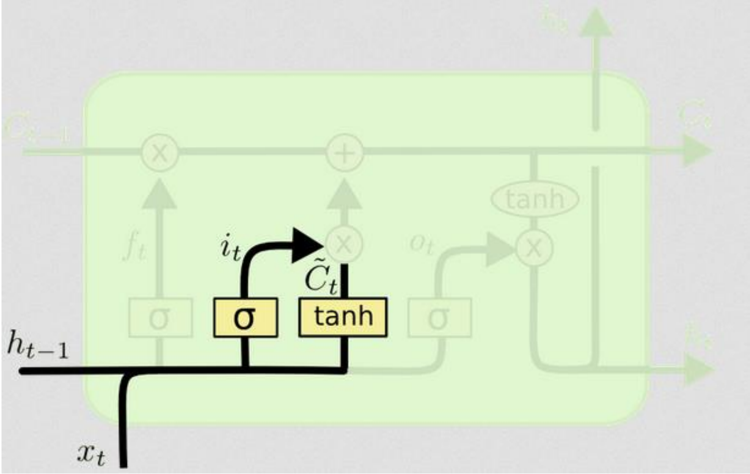
$$f_t = \sigma(z) = \frac{1}{1+\exp(-z)} = [\frac{1}{1+\exp(-92)}, \frac{1}{1+\exp(-177)}, \frac{1}{1+\exp(-136)}] = [1, 1, 1]$$

https://datascience.stackexchange.com/questions/19196/forget-layer-in-a-recurrent-neural-network-rnn/19269#19269

RNN: Recurrent Neural Network



$i_t \; x \; \tilde{C}_t$



- Estimate significant input information though
  - Input Gate

concatenate $[x_t, h_{t-1}]$

$$[x_t, h_{t-1}] = [1, 2, 3, 4, 5, 6])$$

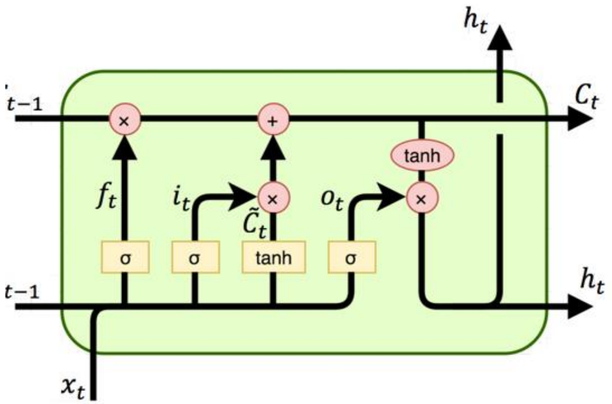
$$W_i = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix} \qquad b_i = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

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

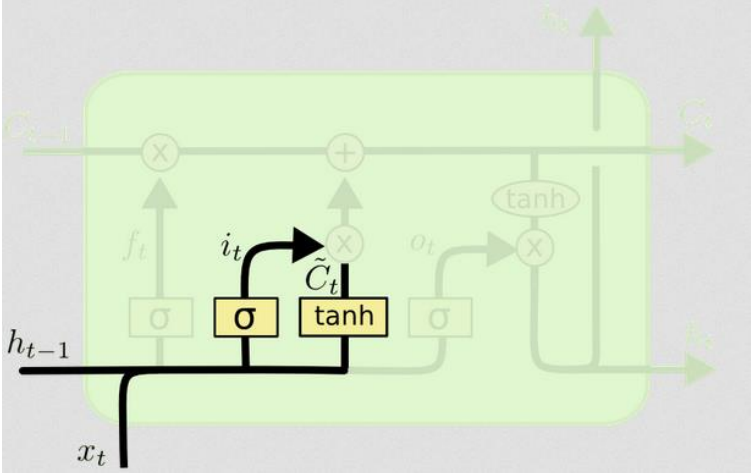
$$z = K(x_t, h_{t-1}) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 42 \\ 62 \end{bmatrix}$$

$$i_t = \sigma(z) = \frac{1}{1+\exp(-z)} = [\frac{1}{1+\exp(-22)}, \frac{1}{1+\exp(-42)}, \frac{1}{1+\exp(-62)}]$$
$$= [1, 1, 1]$$

RNN: Recurrent Neural Network



$i_t \; x \; \tilde{C}_t$



$$W_c = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 \end{bmatrix} \quad b_c = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- Estimate significant input information though
  - Input Gate

concatenate $[x_t, h_{t-1}]$

$$[x_t, h_{t-1}] = [1, 2, 3, 4, 5, 6])$$

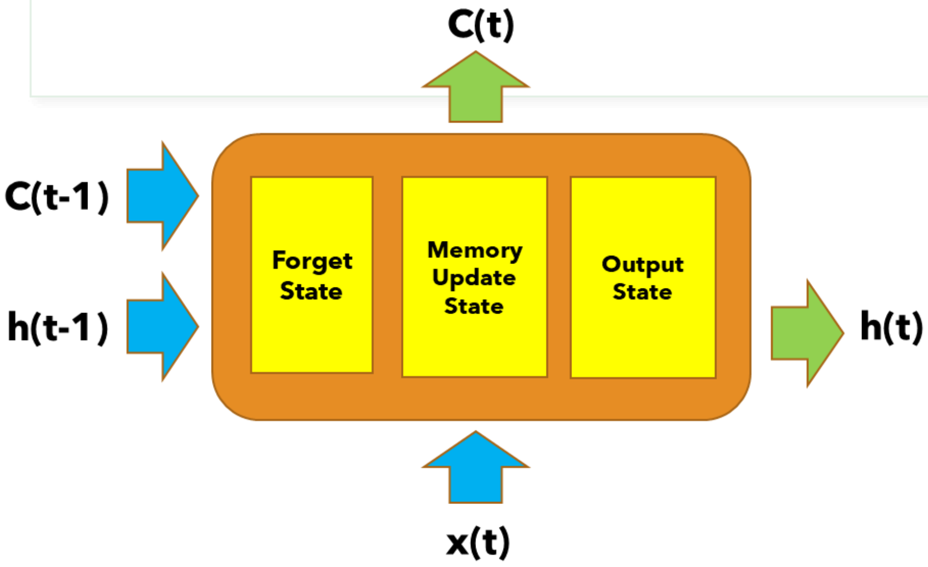
$$\tilde{C}_t = \tanh(W_C \cdot [x_t, h_{t-1}] \; + \; b_C)$$

$$z = K(x_t, h_{t-1}) = W_c[x_t, h_{t-1}] + b_c$$

$$= \begin{bmatrix} 4 & 4 & 4 & -4 & -4 & -4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & -6 & -6 & -6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -35 \\ 107 \\ -51 \end{bmatrix}$$

$$\tilde{C}_t = tanh(z) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

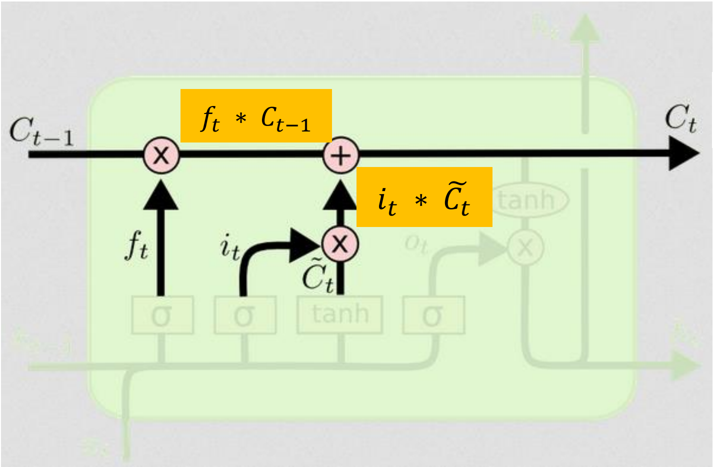
RNN: Recurrent Neural Network



- Memory Update State

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

$$f_t = \begin{bmatrix}1\\1\\1\end{bmatrix} \quad C_{t-1} = \begin{bmatrix}3\\-2\\1\end{bmatrix} \quad i_t = \begin{bmatrix}1\\1\\1\end{bmatrix} \quad \tilde{C}_t = \begin{bmatrix}-1\\1\\-1\end{bmatrix}$$

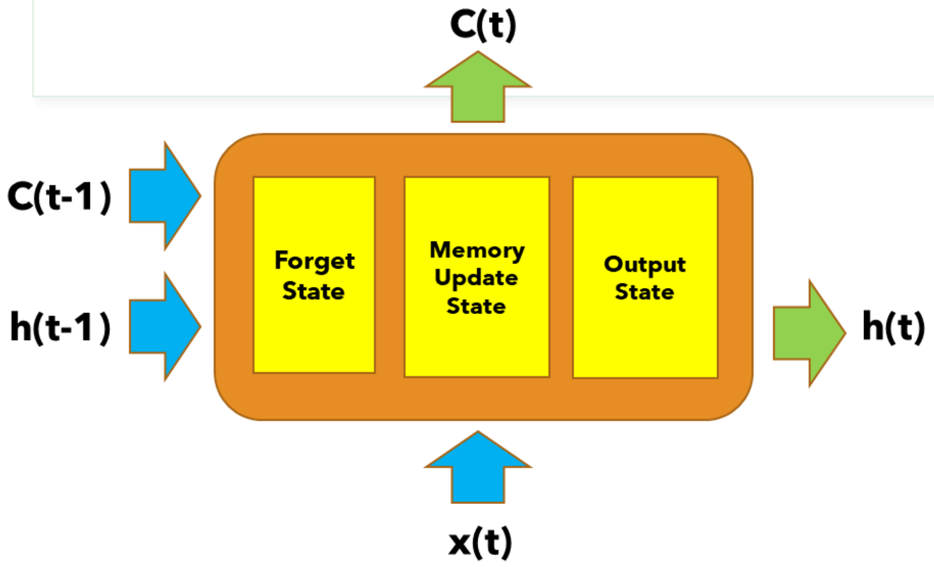


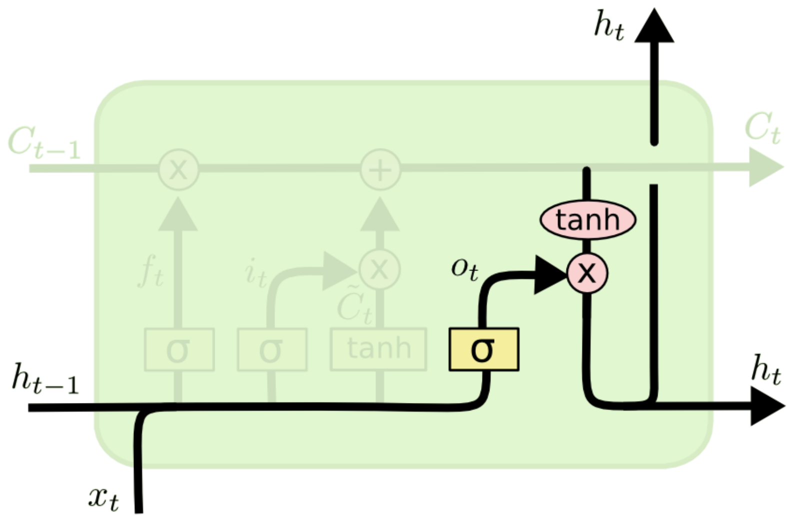
$$= f_t * C_{t-1} + i_t * \tilde{C}_t$$

f(t)	C(t-1)	i(t)	C~(t)	C(t)
1	3	1	-1	2
1	-2	1	1	-1
1	1	1	-1	0

$$C_t = \begin{bmatrix}2\\-1\\-0\end{bmatrix}$$

RNN: Recurrent Neural Network





- Output State

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

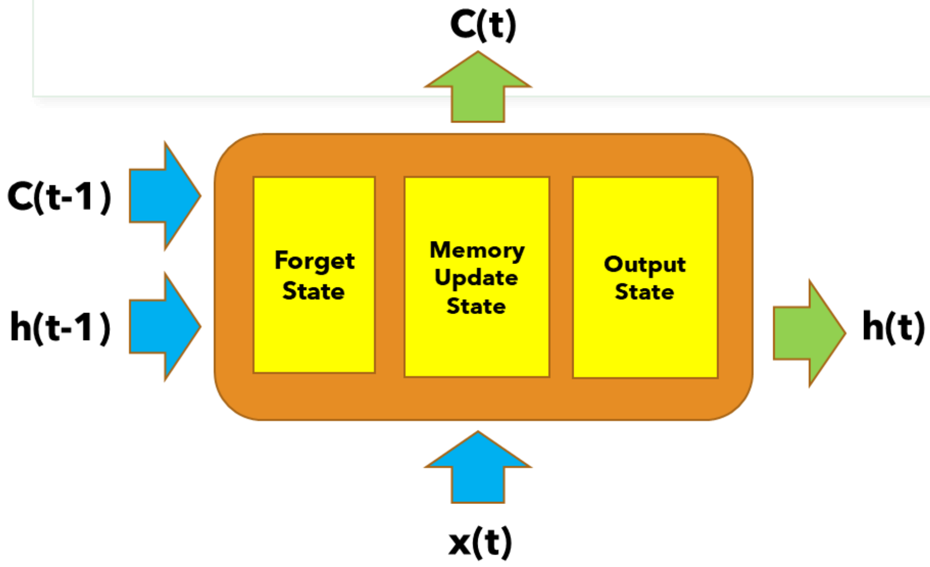
$$z = K(x_t, h_{t-1}) = W_o[x_t, h_{t-1}] + b_o$$

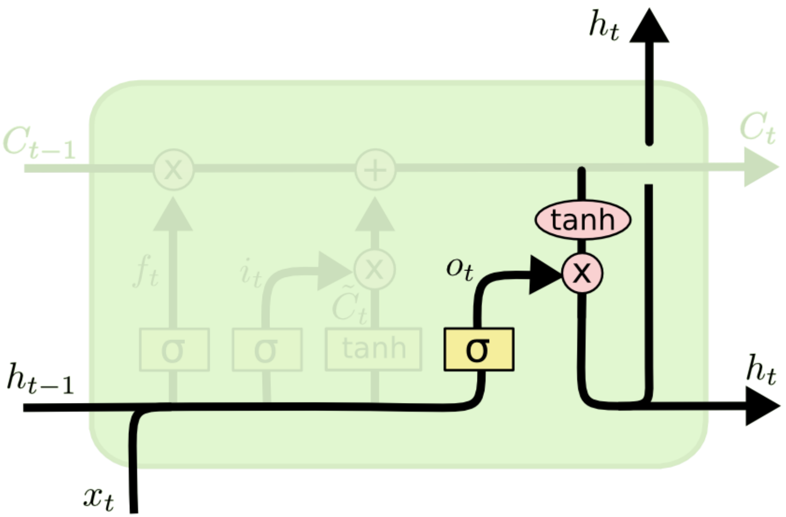
$$= \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} -20 \\ -40 \\ -100 \end{bmatrix} = \begin{bmatrix} 22 \\ 23 \\ -16 \end{bmatrix}$$

$$o_t = \sigma(z) = \frac{1}{1+\exp(-z)} = \left[\frac{1}{1+\exp(-22)}, \frac{1}{1+\exp(-23)}, \frac{1}{1+\exp(16)}\right]$$

$$= \begin{bmatrix} 1 \\ 1 \\ 1.12535\text{E-07} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

RNN: Recurrent Neural Network





- Output State

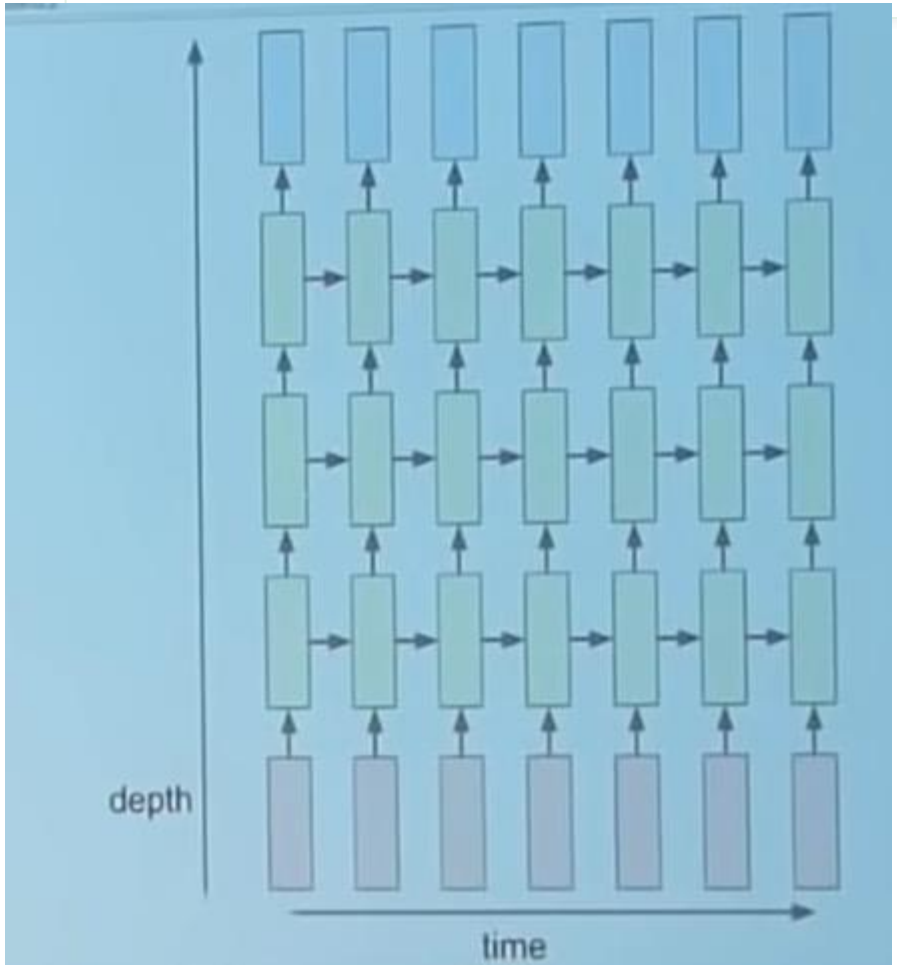
$$\tilde{h}_t = \tanh(\ C_t\ )$$

$$= \begin{bmatrix} 0.96402758 \\ \text{-}0.76159416 \\ 0 \end{bmatrix} \qquad C(t) = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$$

$$h_t = O_t * \tilde{h}_t = O_t * \tanh(\ C_t\ )$$

$$= \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} * \begin{bmatrix} 0.96402758 \\ \text{-}0.76159416 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.96402758 \\ -0.76159416 \\ 0 \end{bmatrix}$$

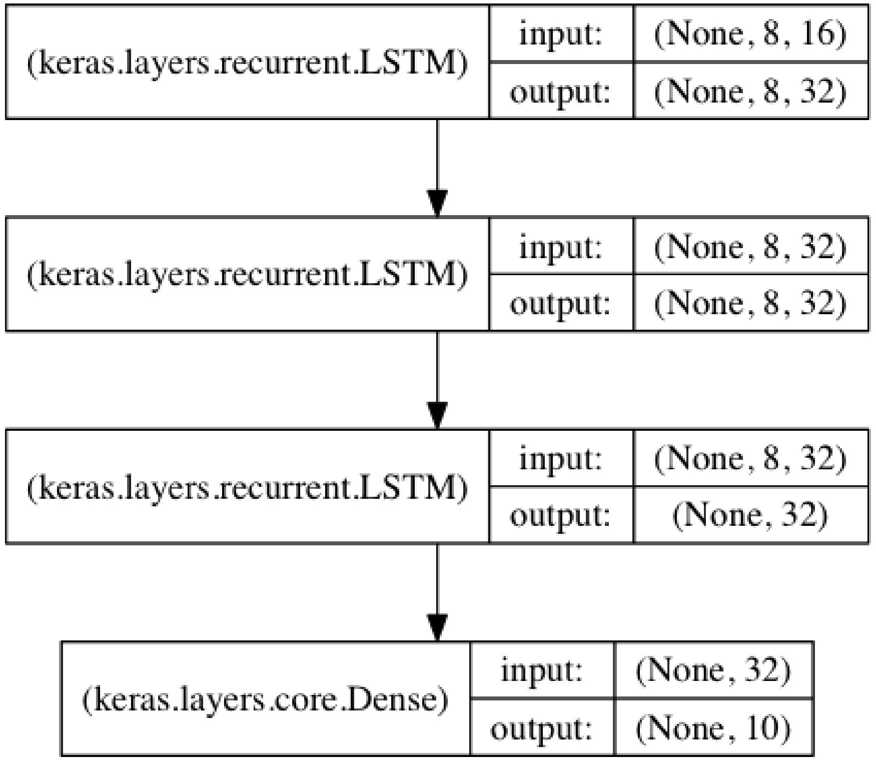
Stacked LSTM



- Each LSTM Layer
  - #LSTM node = timesteps = 8
  - Input dimension = vector_size or n_features = 16
  - Output dimension = กำหนดในโครงสร้าง
    - Ex. LSTM(50, activation='tanh', input_shape=(8,16))
      - Output dimension = (8, 50) -> 8 nodes / dim = 50/node

LSTM: text classification example

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

data_dim = 16
timesteps = 8
num_classes = 10

# expected input data shape: (batch_size, timesteps, data_dim)
model = Sequential()
model.add(LSTM(32, return_sequences=True,
               input_shape=(timesteps, data_dim)))  # returns a sequence of vectors of dimension 32
model.add(LSTM(32, return_sequences=True))  # returns a sequence of vectors of dimension 32
model.add(LSTM(32))  # return a single vector of dimension 32
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Generate dummy training data
x_train = np.random.random((1000, timesteps, data_dim))
y_train = np.random.random((1000, num_classes))

# Generate dummy validation data
x_val = np.random.random((100, timesteps, data_dim))
y_val = np.random.random((100, num_classes))

model.fit(x_train, y_train,
          batch_size=64, epochs=5,
          validation_data=(x_val, y_val))
```



https://towardsdatascience.com/multi-class-text-classification-with-lstm-using-tensorflow-2-0-d88627c10a35