

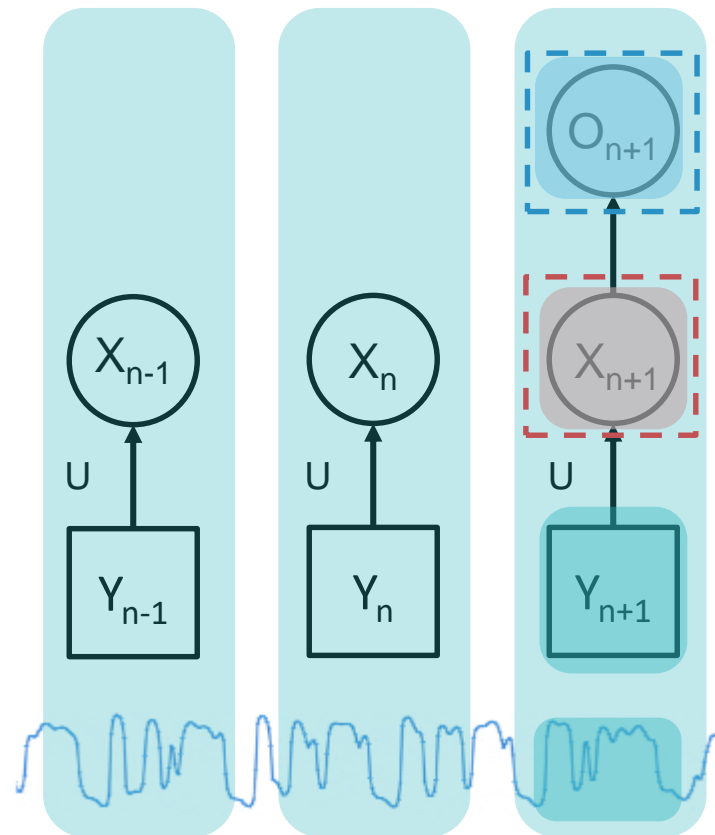


# Neural Network Architectures for Time-Series: Recurrent Neural Network (RNN)

**Industrial AI Lab.**  
**Prof. Seungchul Lee**

# Recurrent NN (RNN)

- Hidden state extraction and transformation

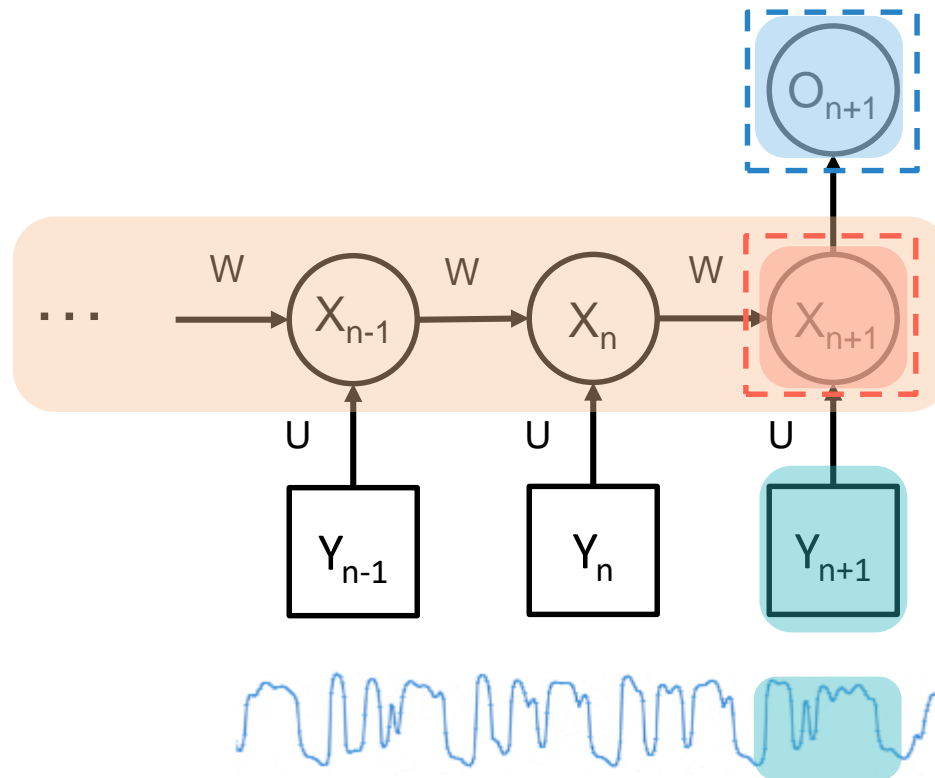


Classification **based on states**

**Learned latent state**

# Recurrent NN (RNN)

- Hidden state extraction and transformation
- Good for sequential data (dynamic behavior)



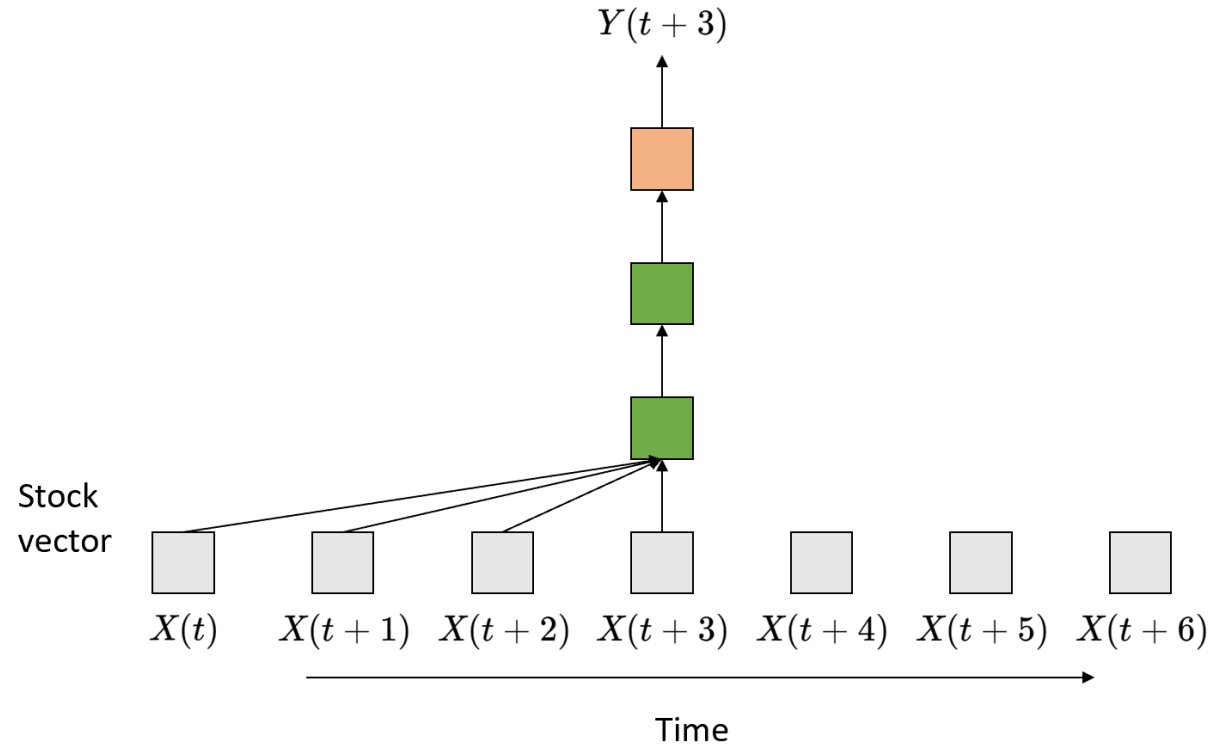
Classification **based on states**

**Learned latent state and its dynamics**

# Recurrent NN

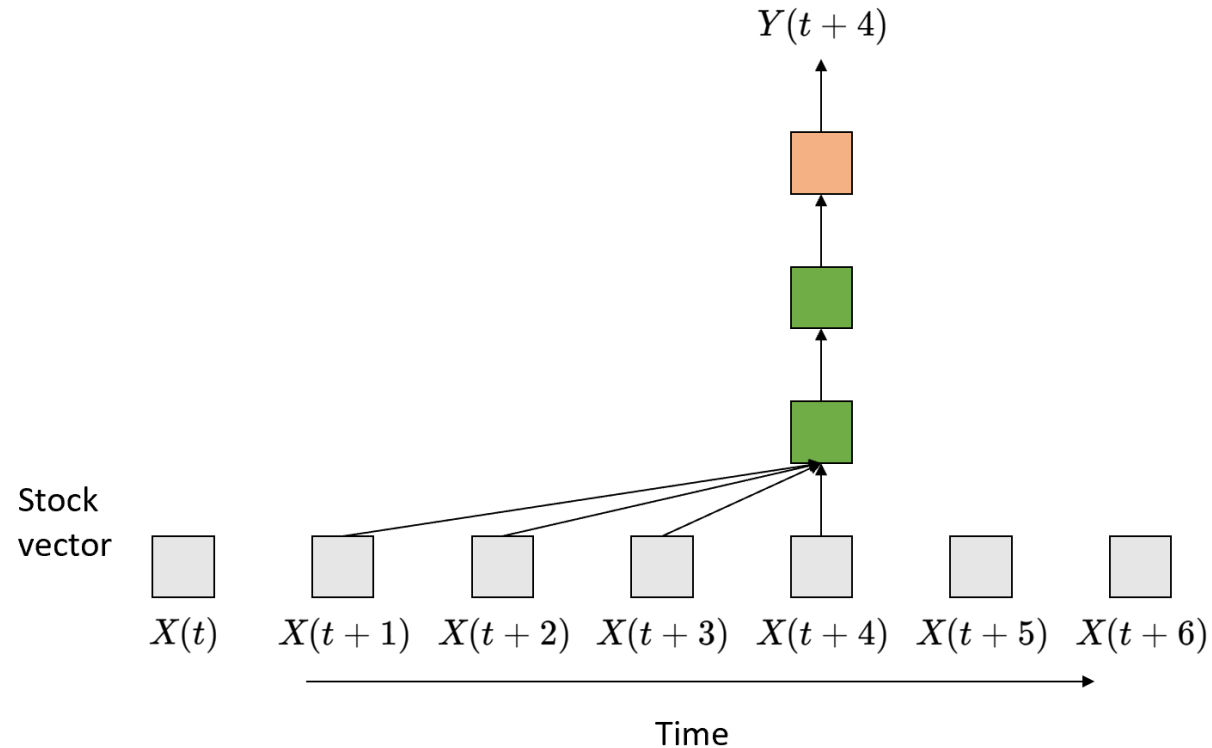
- Hidden state representation
- Learn sequential process

# Sliding Predictor



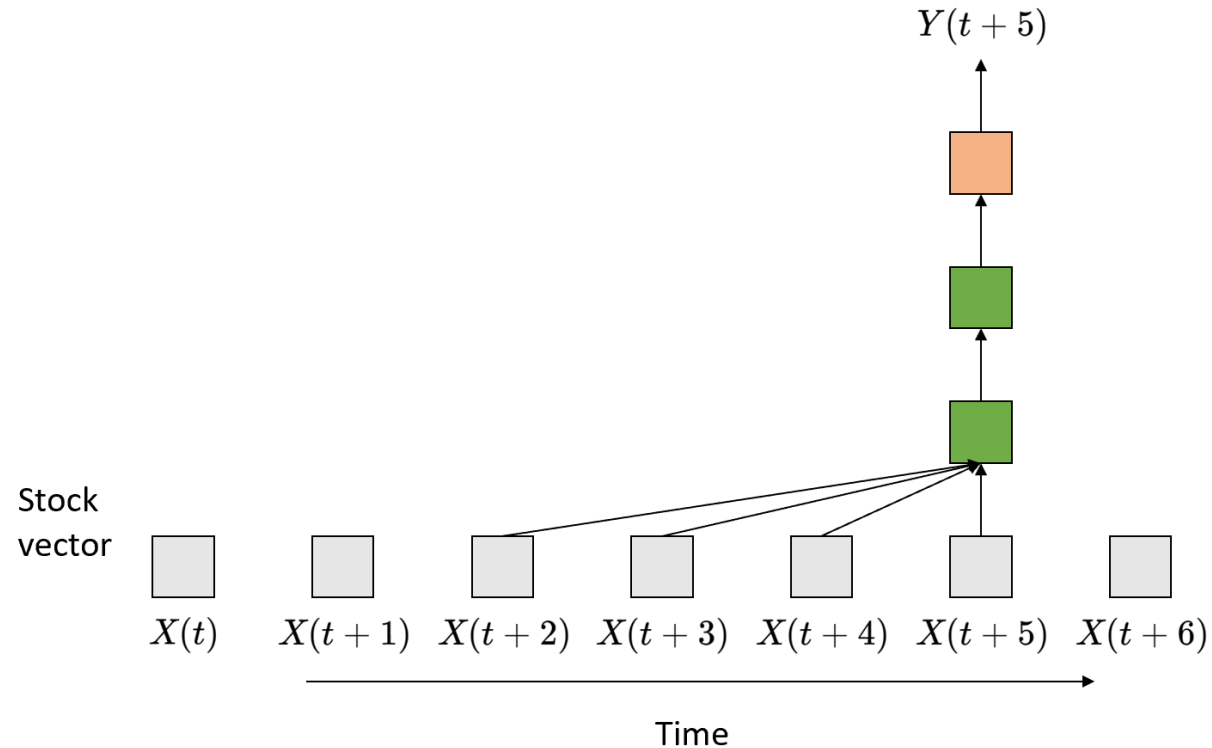
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

# Sliding Predictor



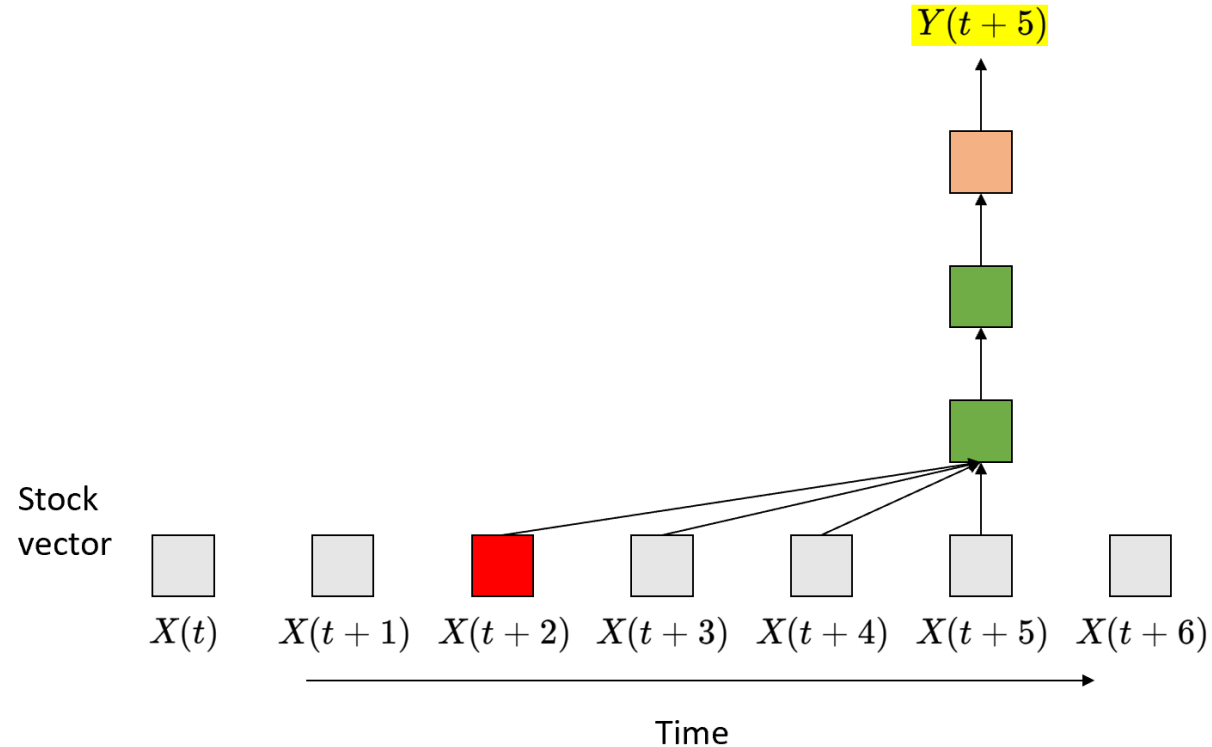
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

# Sliding Predictor



- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

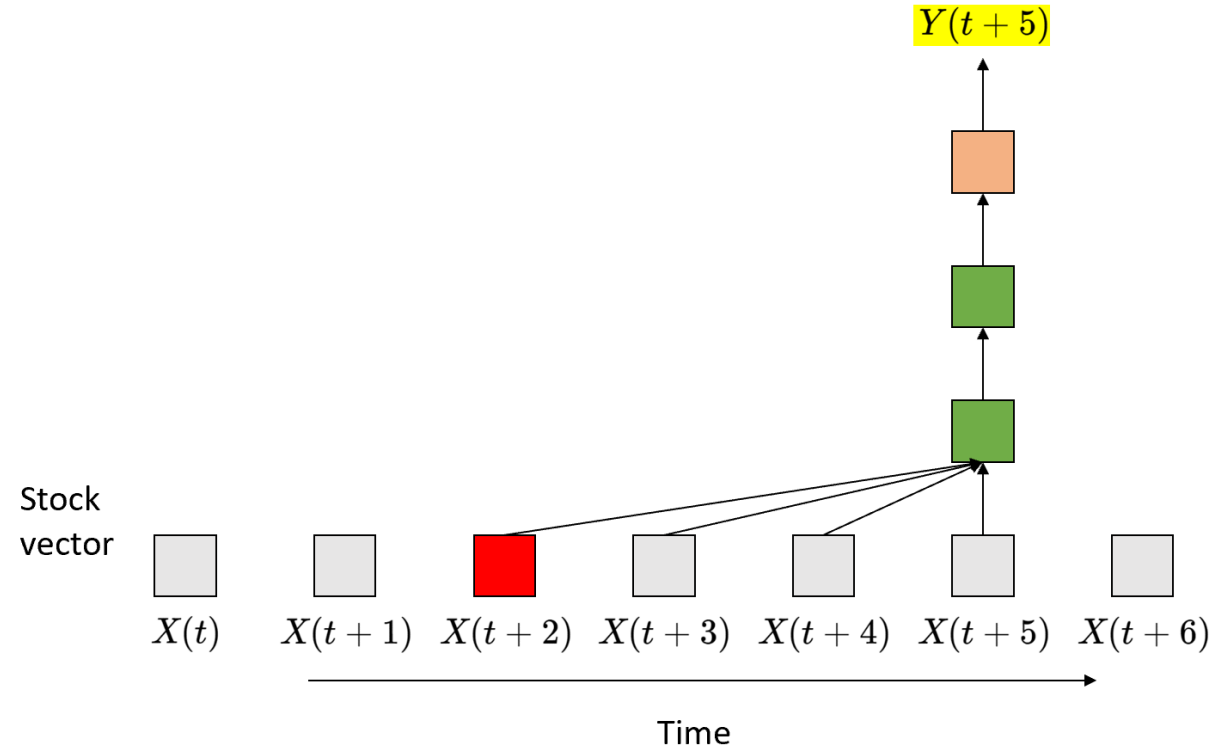
# Sliding Predictor



- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

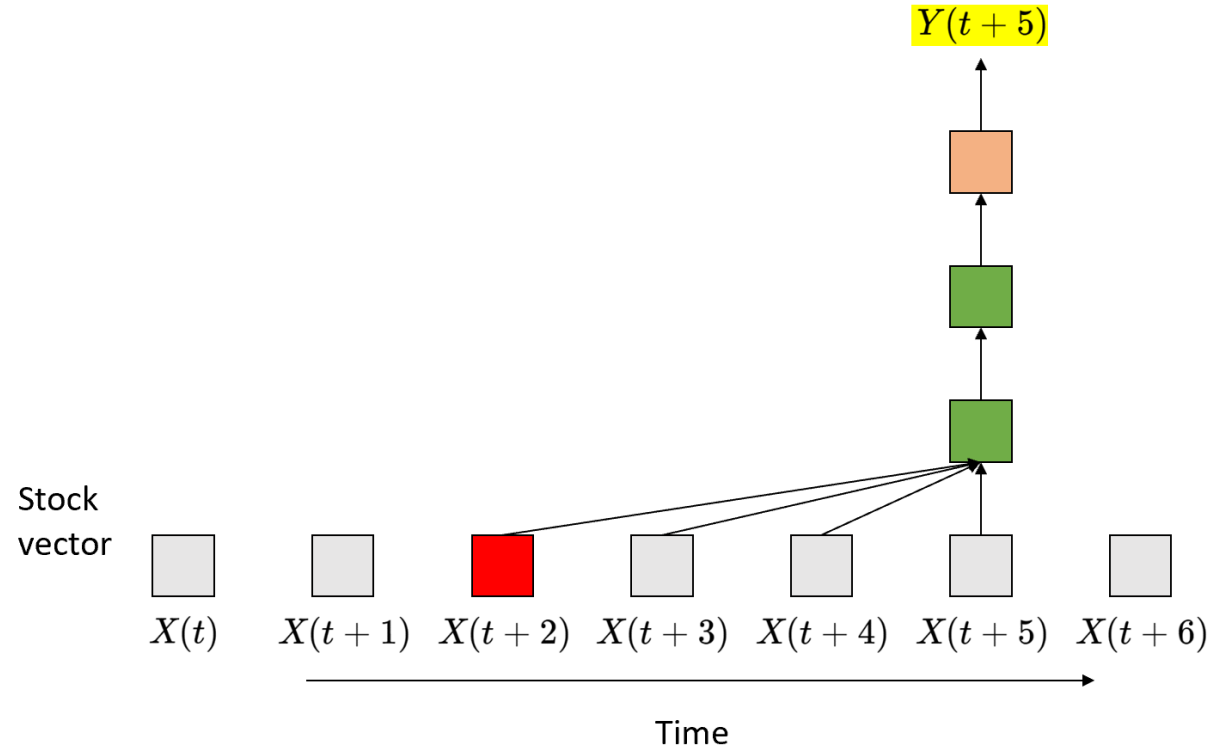


# Finite-Response Model



- This is a finite response system
  - Something that happens today only affects the output of the system for  $N$  days into the future
  - $N$  is the width of the system

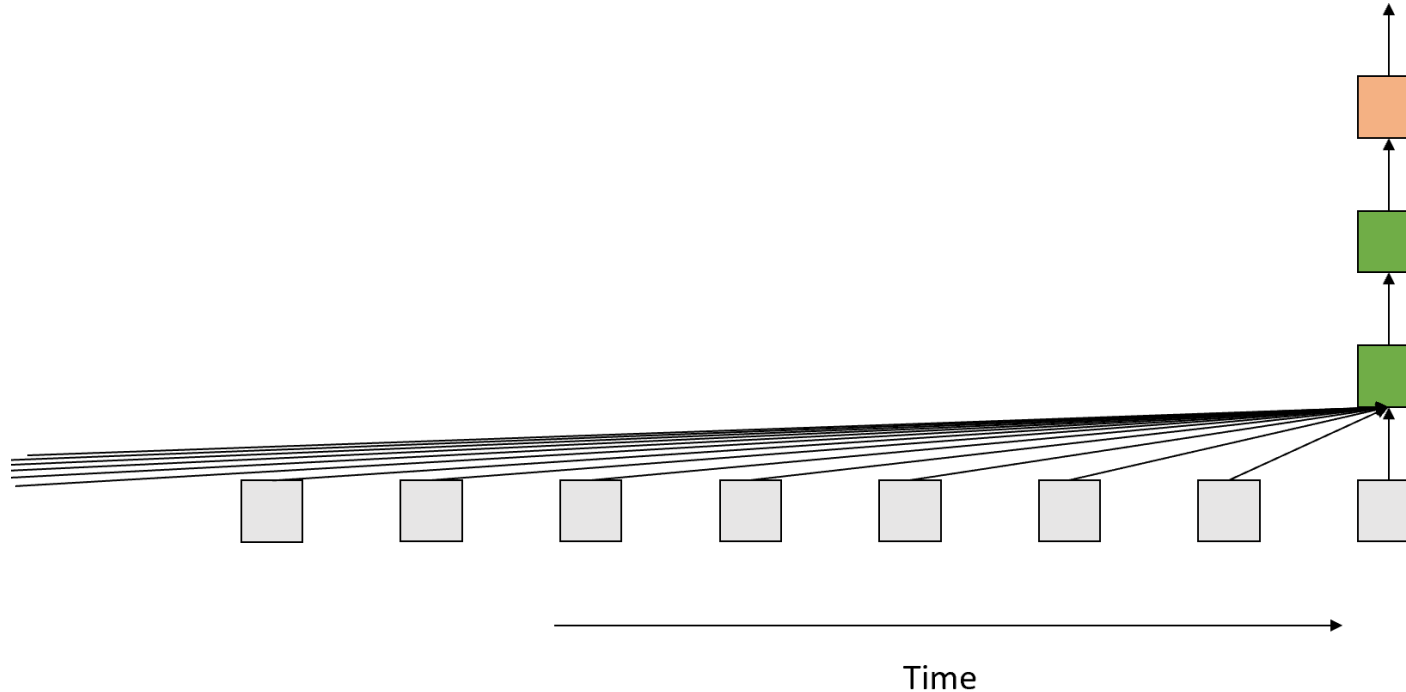
# Finite-Response Model



- Problem: Increasing the “history” makes the network more complex

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

# In Theory, We Want Infinite Memory



- Required: Infinite response systems
  - What happens today can continue to affect the output forever
  - Possibly with weaker and weaker influence

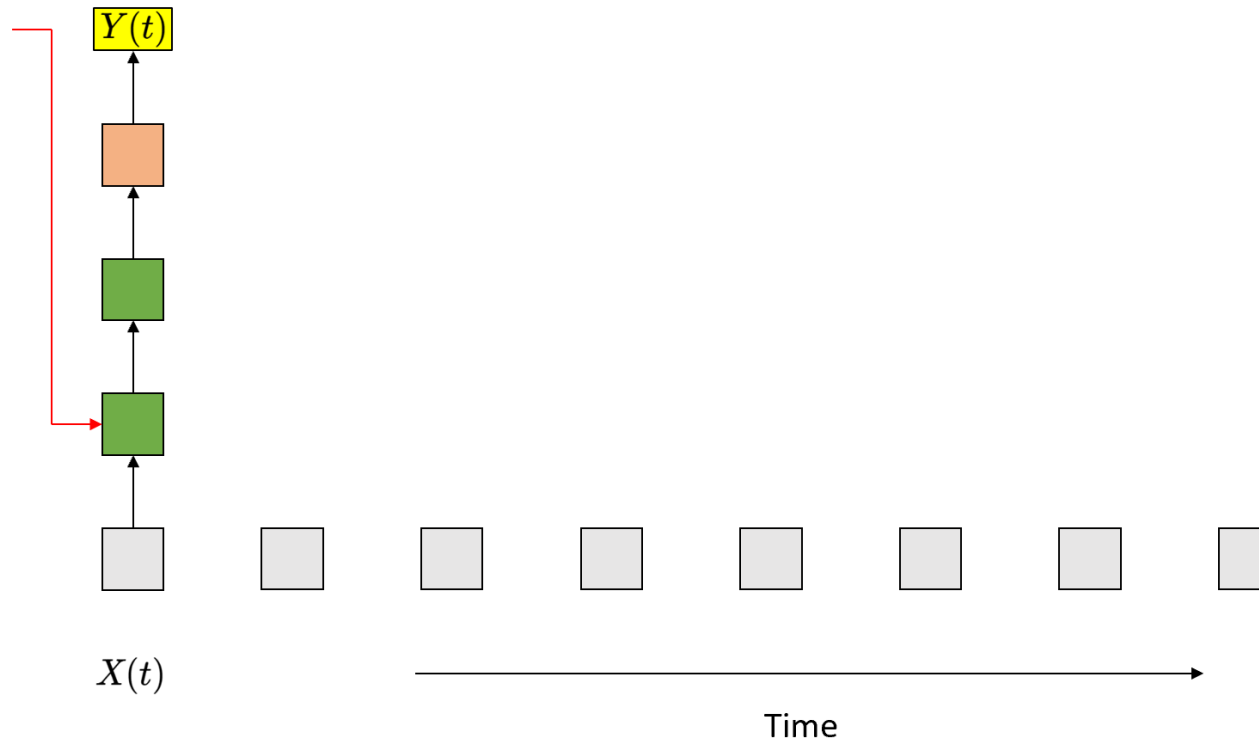
$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-\infty})$$

# Infinite Response Systems

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-\infty}) \implies Y_t = f(X_t, Y_{t-1})$$

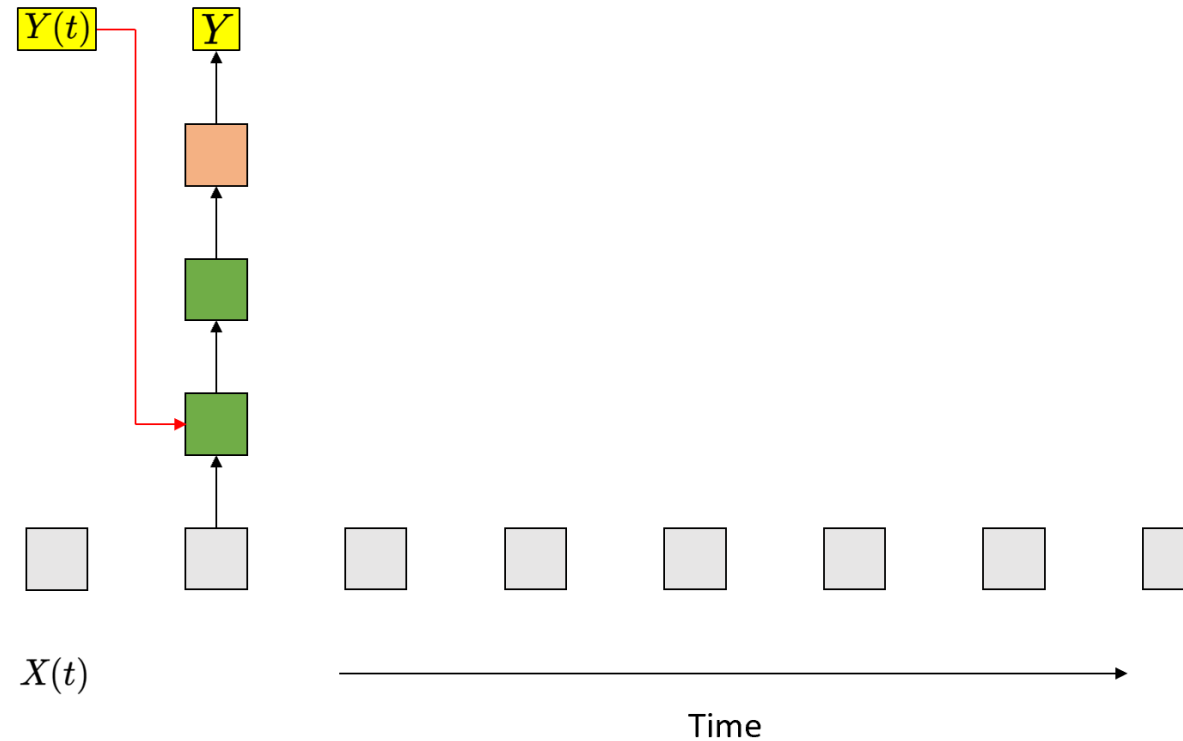
- Recursive
  - Required: Define initial output:  $Y_{t-1}$  for  $t = 0$
  - An input at  $X_0$  at  $t = 0$  produces  $Y_0$
  - $Y_0$  produces  $Y_1$  which produces  $Y_2$  and so on until  $Y_\infty$  even if  $X_1, \dots, X_\infty$  are 0
- Output contains information about the entire past

# Autoregression



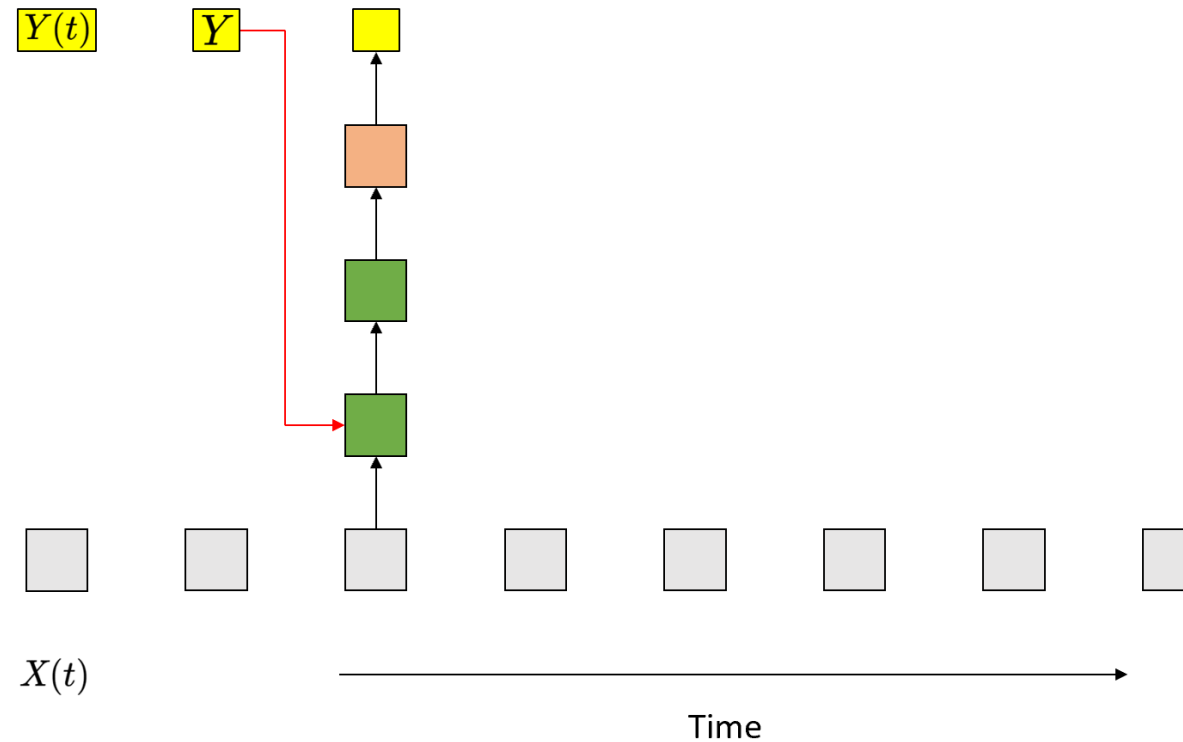
- An autoregressive net with recursion from the output

# Autoregression



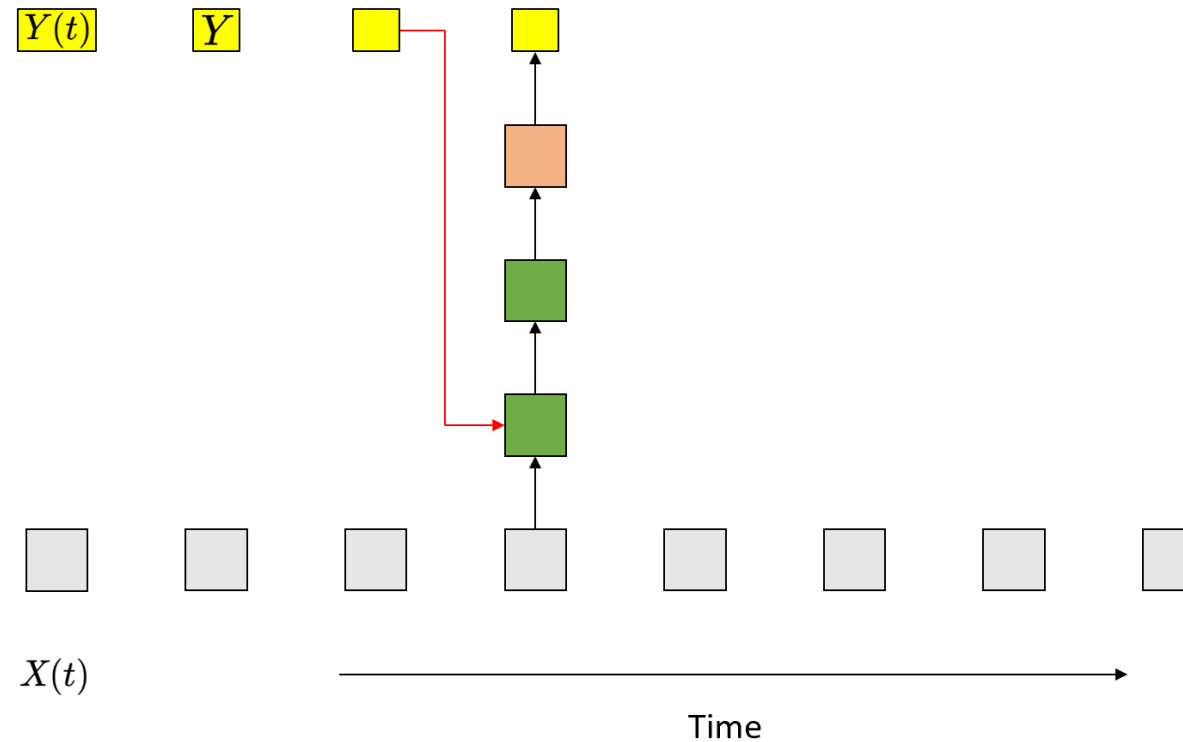
- An autoregressive net with recursion from the output

# Autoregression



- An autoregressive net with recursion from the output

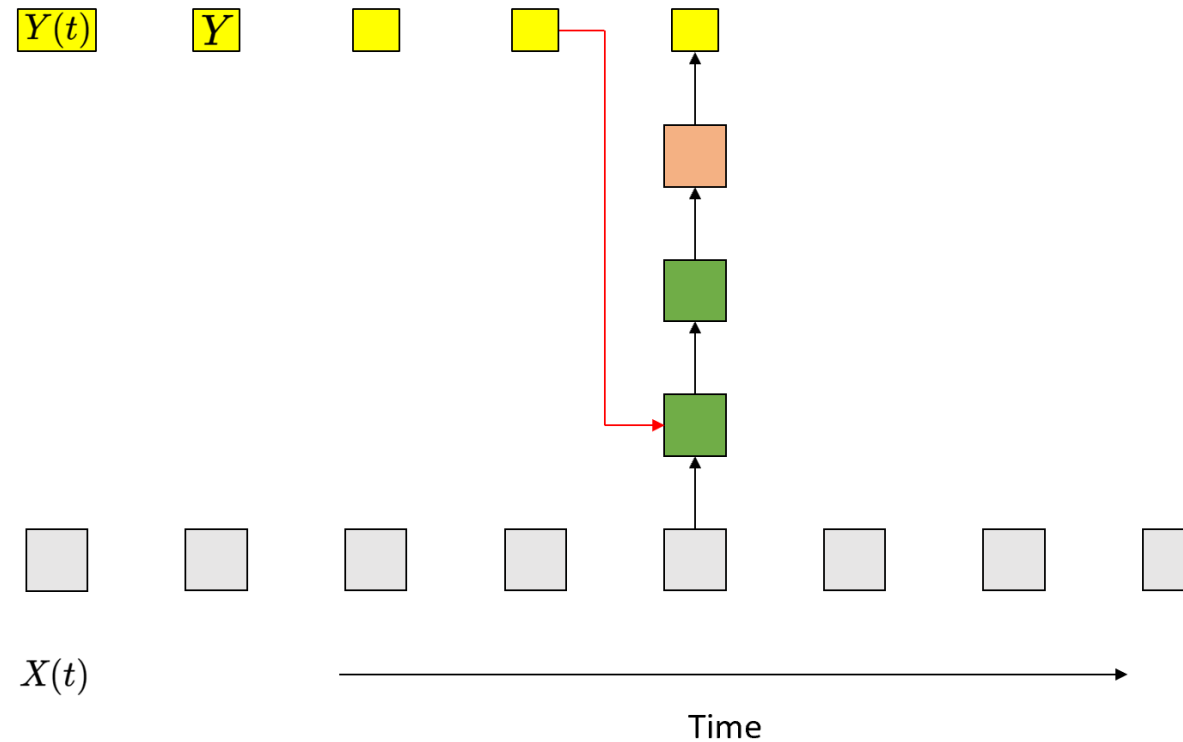
# Autoregression



- An autoregressive net with recursion from the output

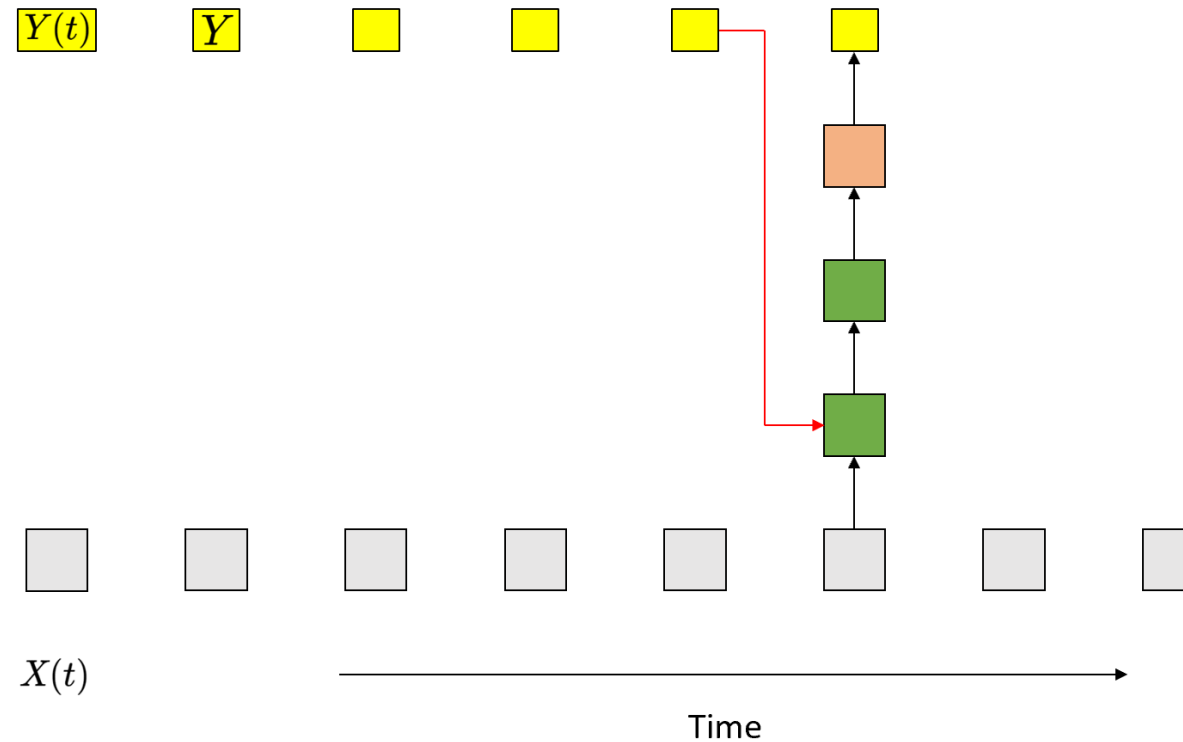


# Autoregression



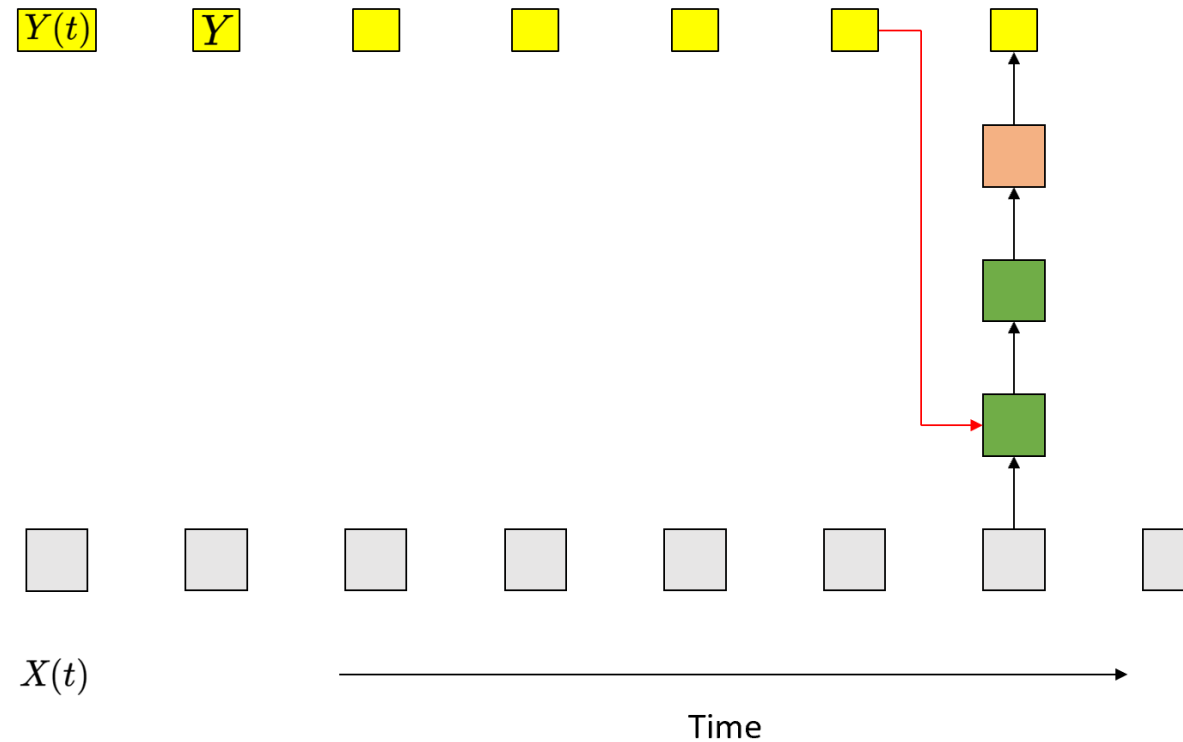
- An autoregressive net with recursion from the output

# Autoregression



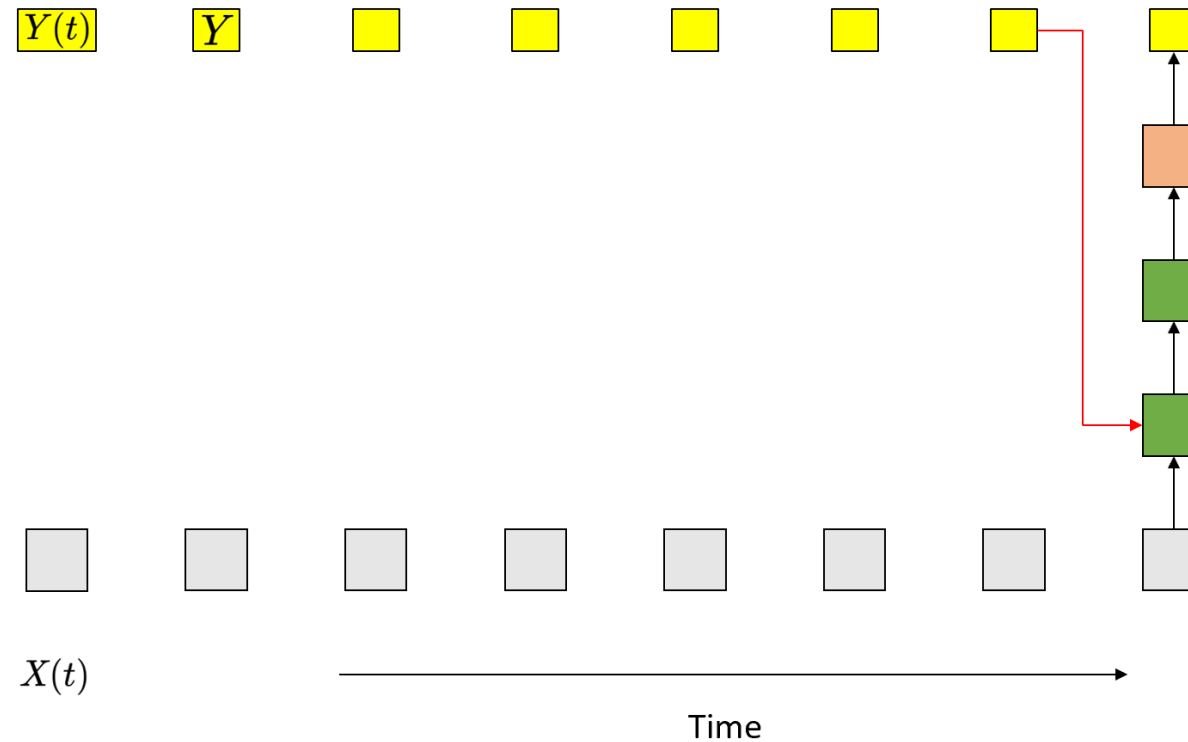
- An autoregressive net with recursion from the output

# Autoregression



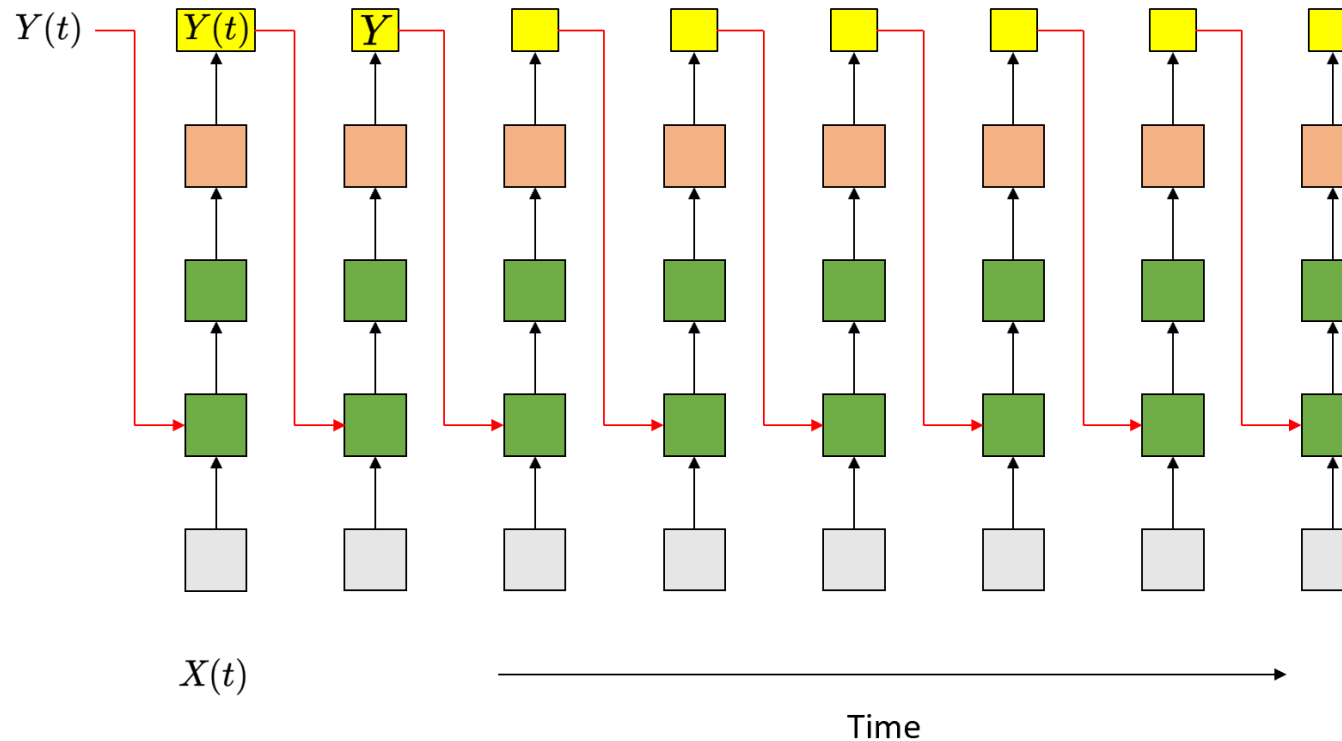
- An autoregressive net with recursion from the output

# Autoregression



- An autoregressive net with recursion from the output

# More Complete Representation



- An autoregressive net with recursion from the output
- Showing all computations
- All columns are identical
- An input at  $t = 0$  affects outputs forever

# An Alternate Model for Infinite Response Systems

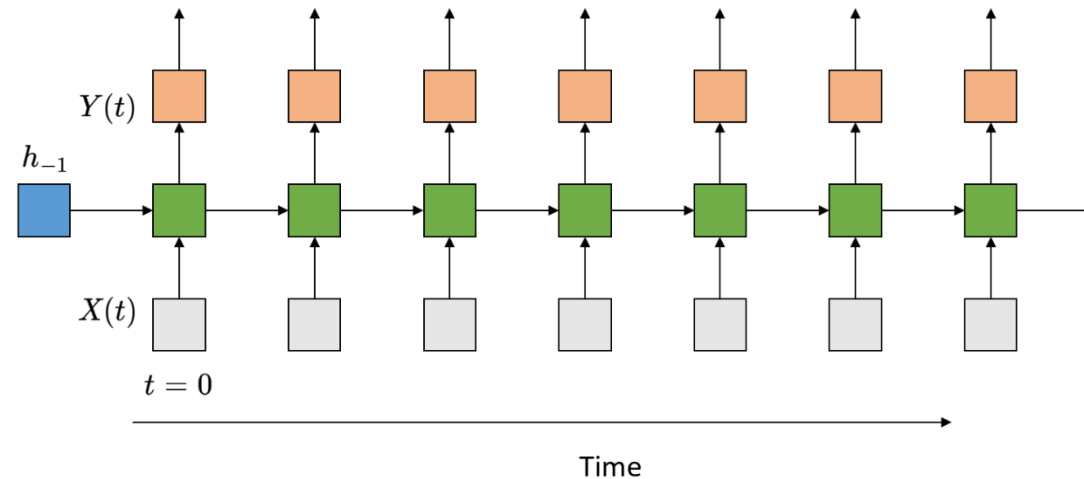
- the state-space model

$$h_t = f(x_t, h_{t-1})$$

$$y_t = g(h_t)$$

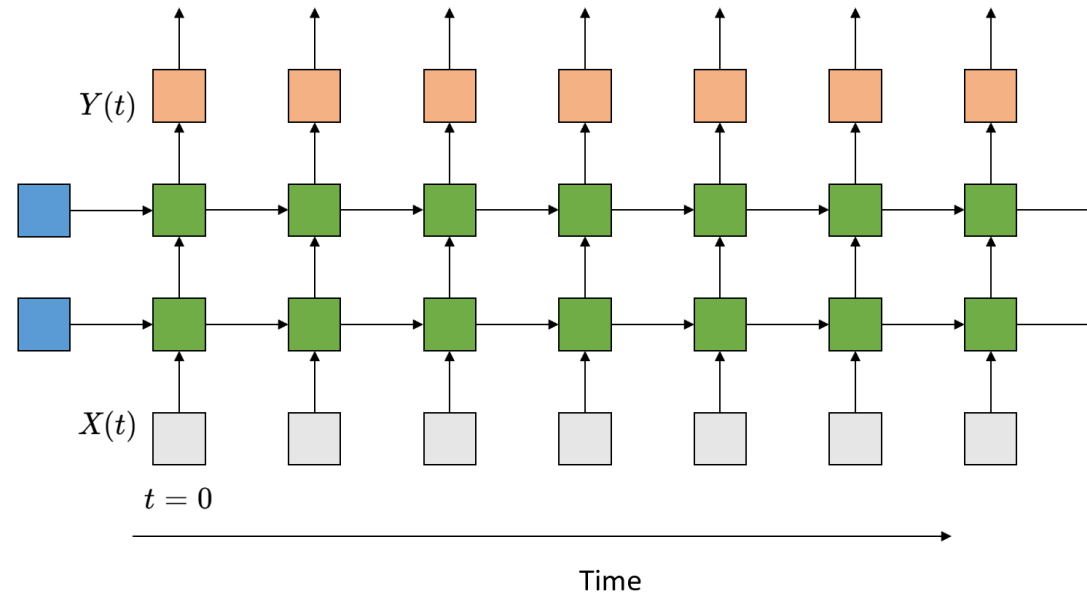
- $h_t$  is the state of the network
- Need to define initial state  $h_{-1}$
- This is a recurrent neural network
- State summarizes information about the entire past

# Single Hidden Layer RNN (Simplest State-Space Model)



- The state (green) at any time is determined by the input at that time, and the state at the previous time
- All columns are identical
- An input at  $t = 0$  affects outputs forever
- Also known as a recurrent neural net

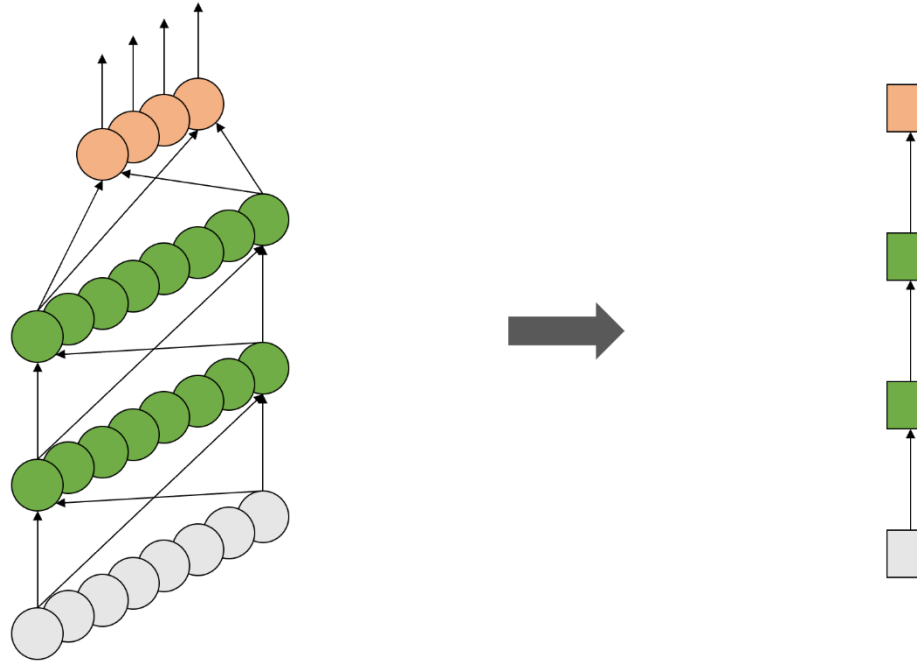
# Multiple Recurrent Layer RNN



- The state (green) at any time is determined by the input at that time, and the state at the previous time
- All columns are identical
- An input at  $t = 0$  affects outputs forever
- Also known as a recurrent neural net

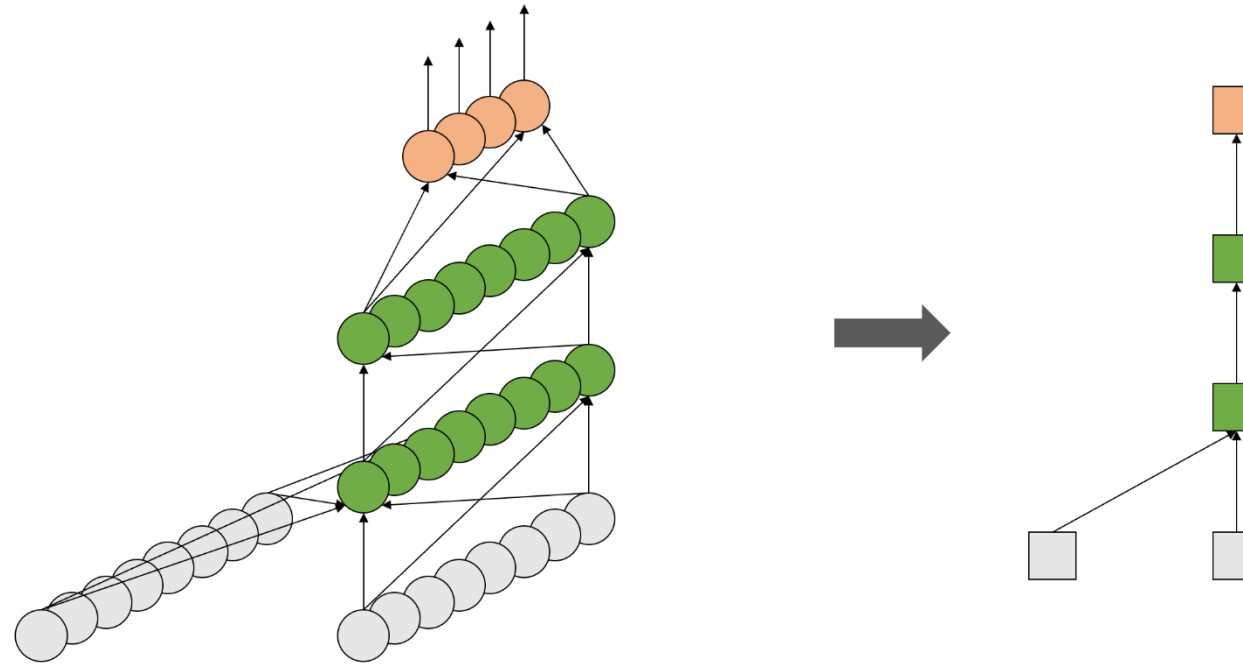


# Representation Shortcut



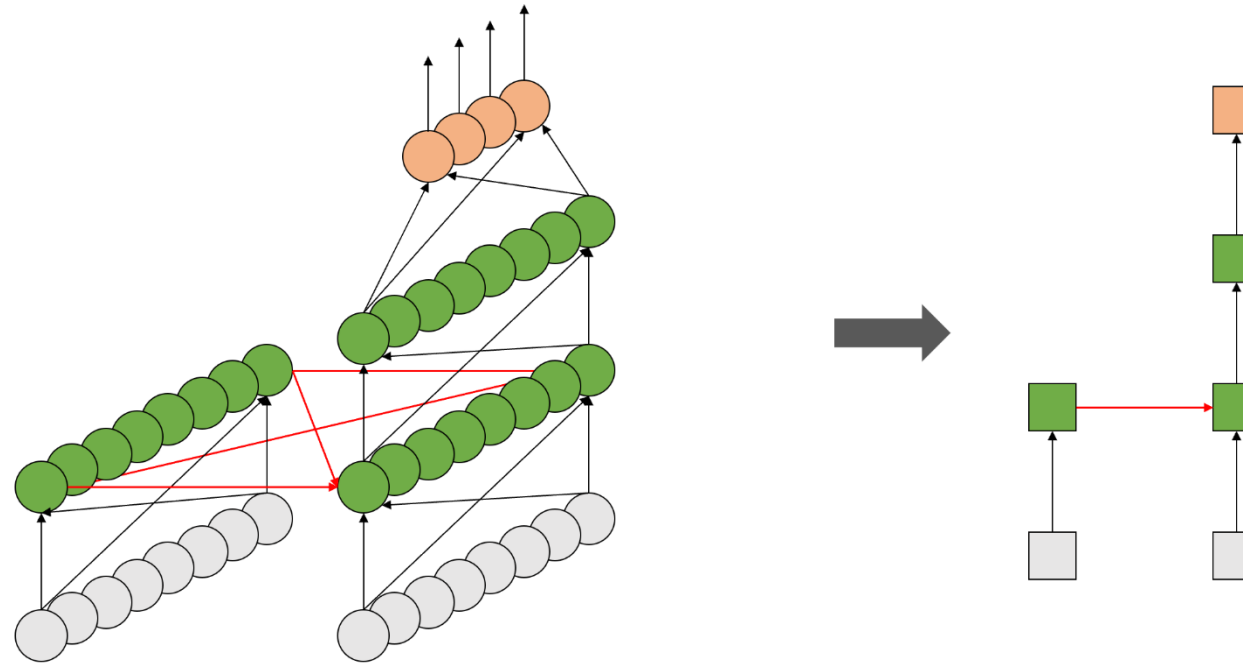
- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

# Representation Shortcut



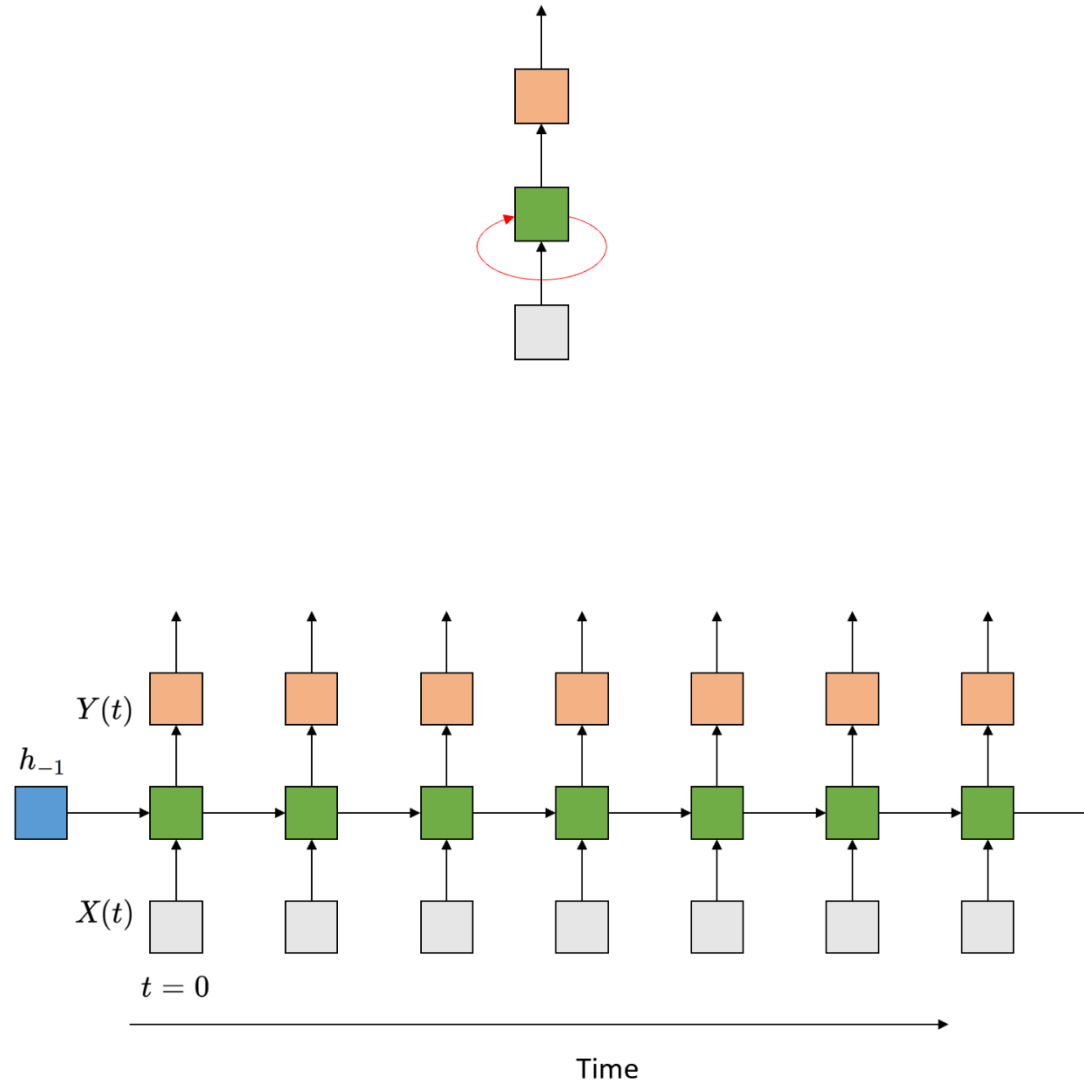
- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

# Representation Shortcut

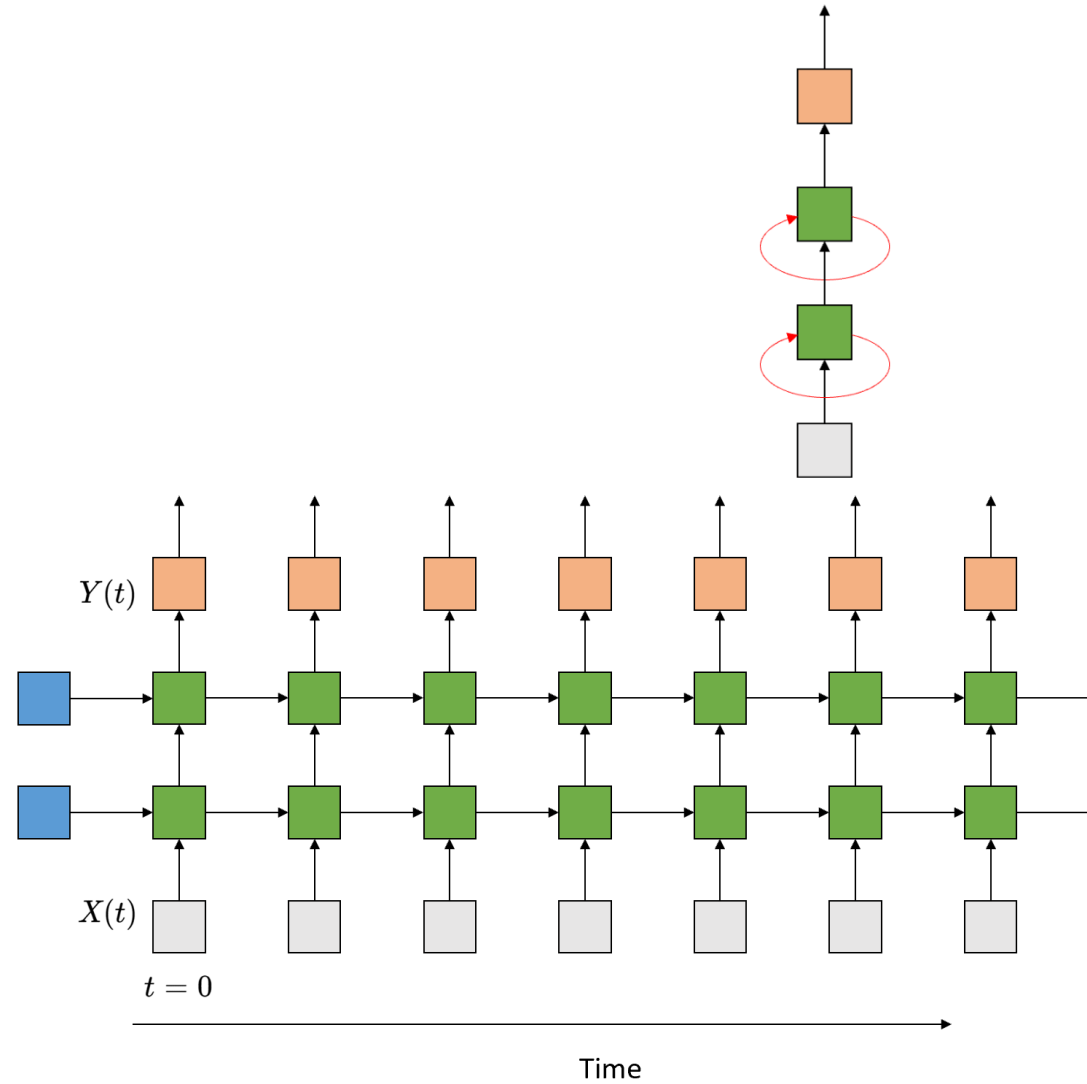


- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

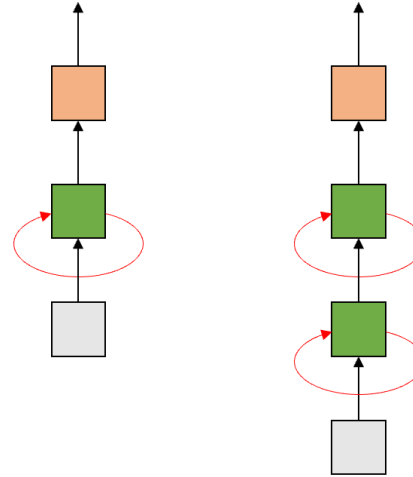
# The Folded Version of RNN



# The Folded Version of RNN



# Recurrent Neural Network



- Simplified models often drawn
- The loops imply recurrence

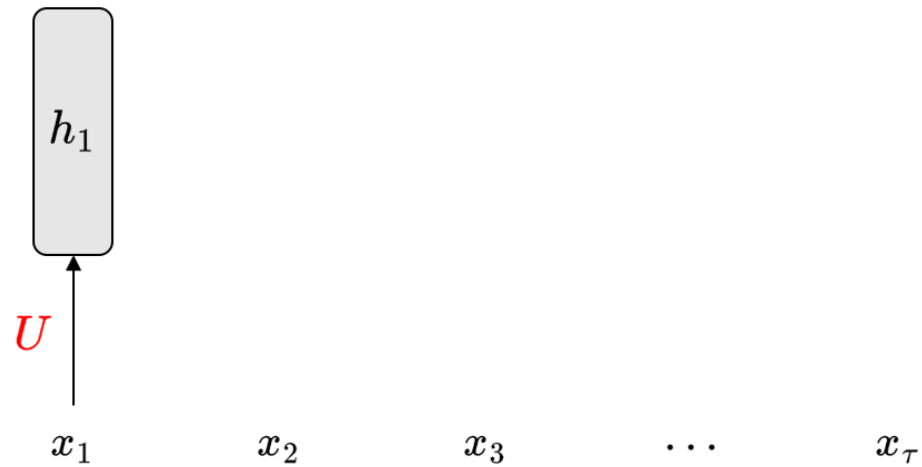
# Recurrent Neural Networks

# Unrolling the Recurrence

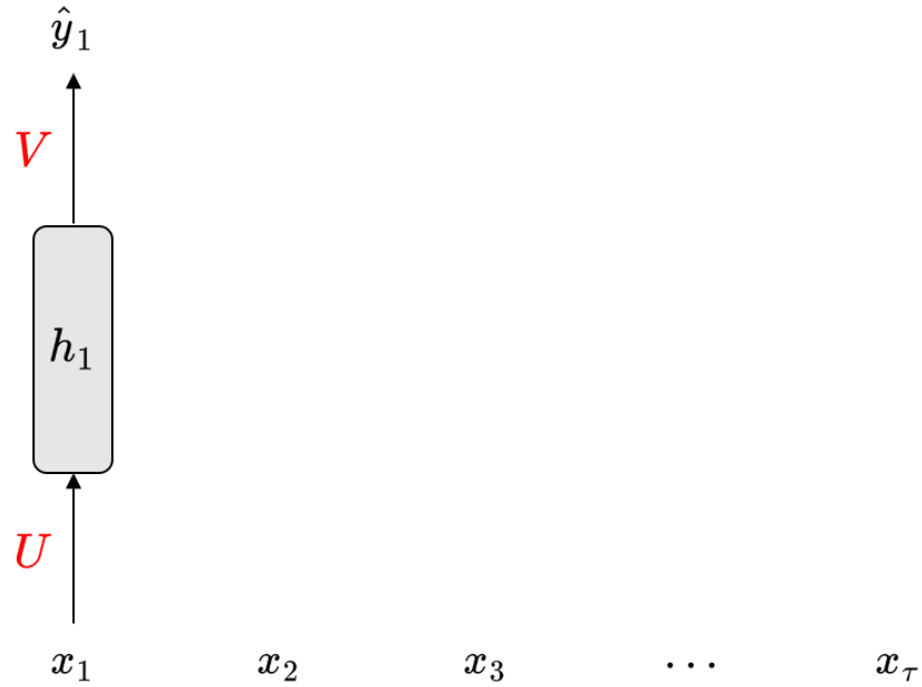
$x_1$        $x_2$        $x_3$        $\dots$        $x_\tau$



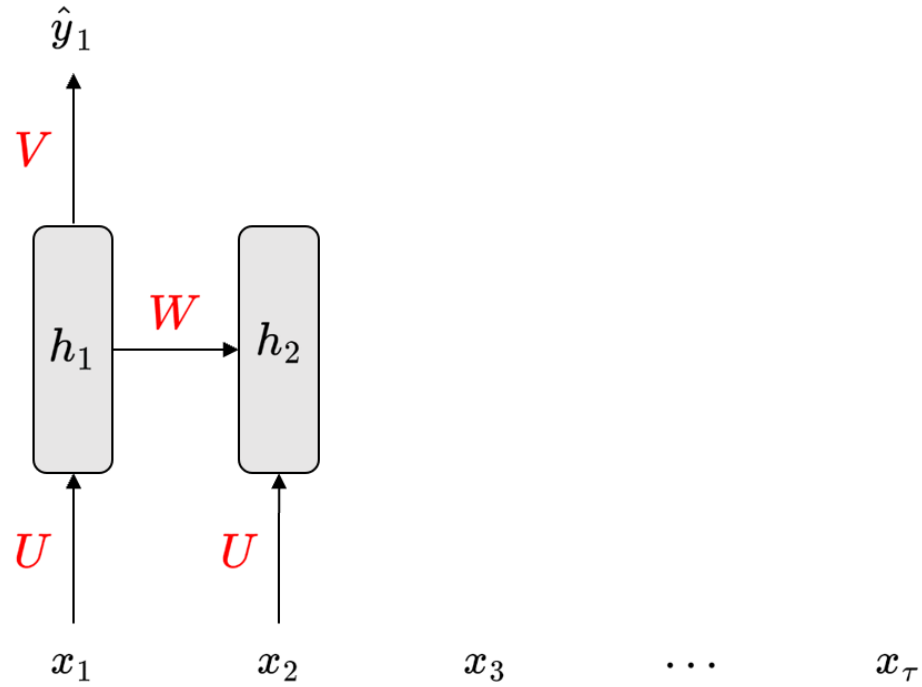
# Unrolling the Recurrence



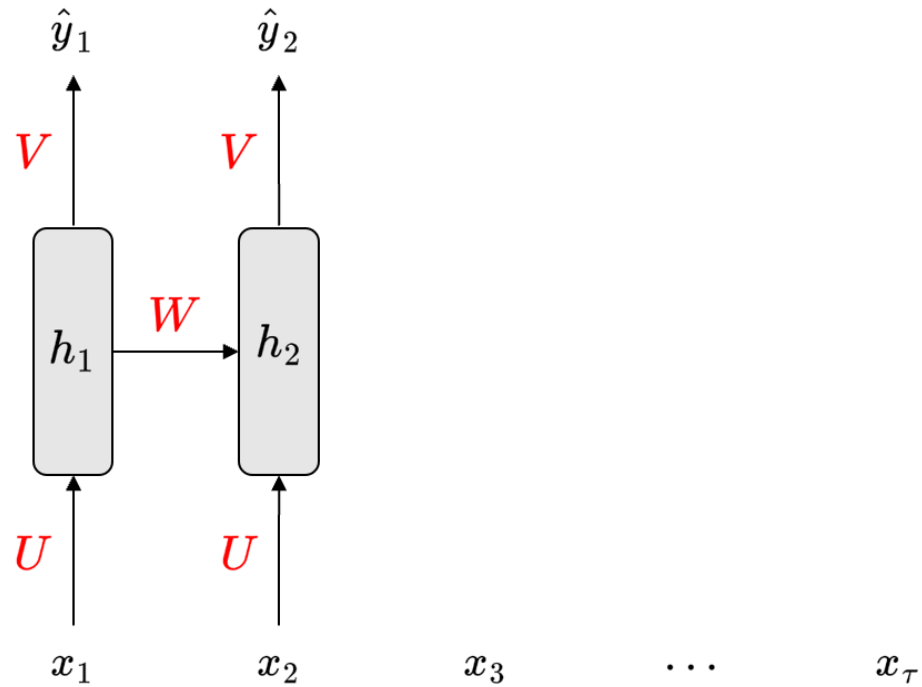
# Unrolling the Recurrence



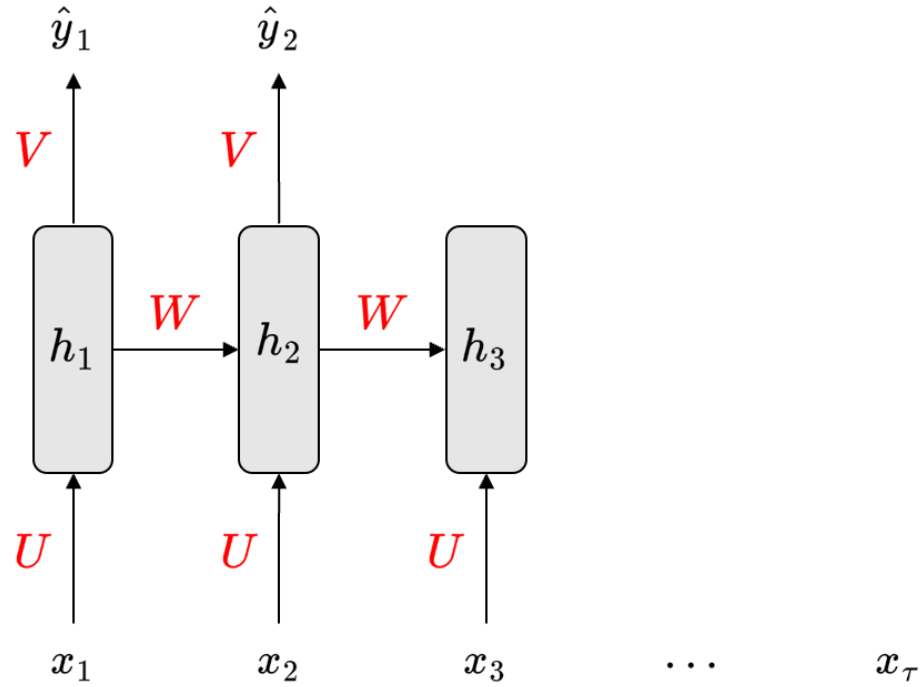
# Unrolling the Recurrence



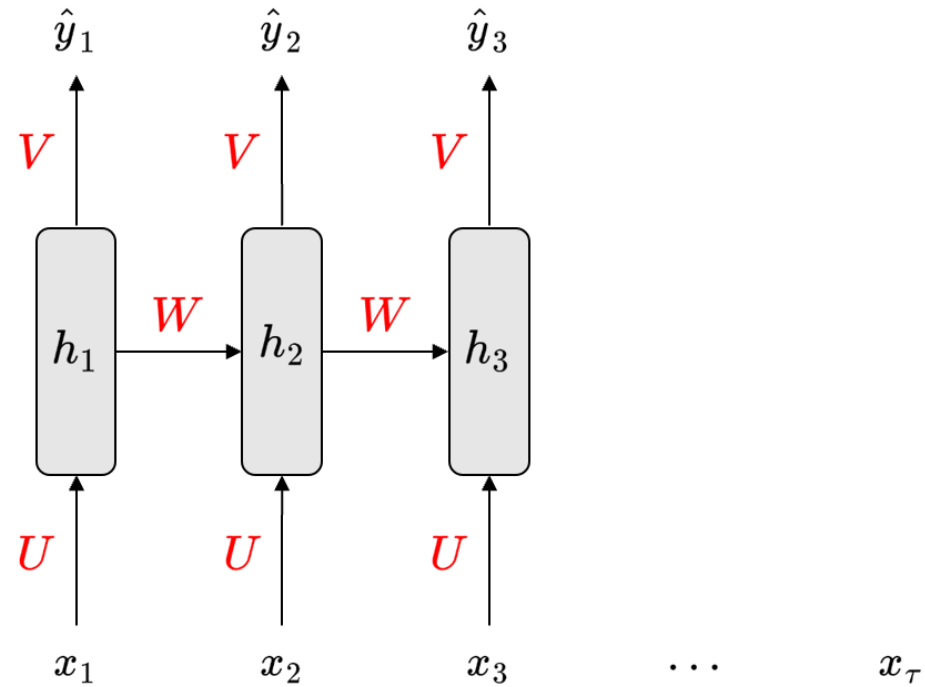
# Unrolling the Recurrence



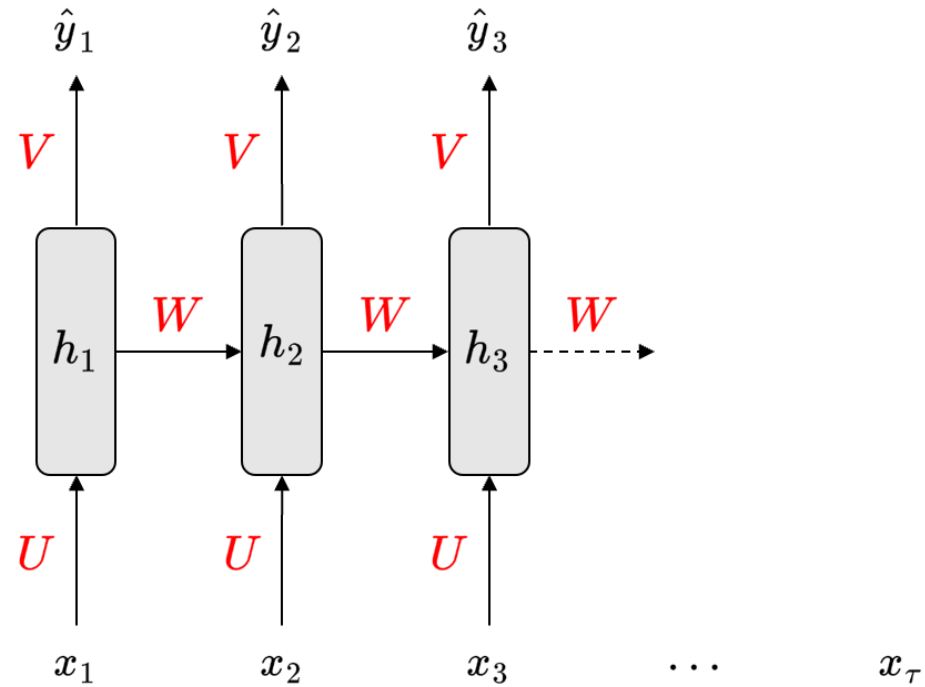
# Unrolling the Recurrence



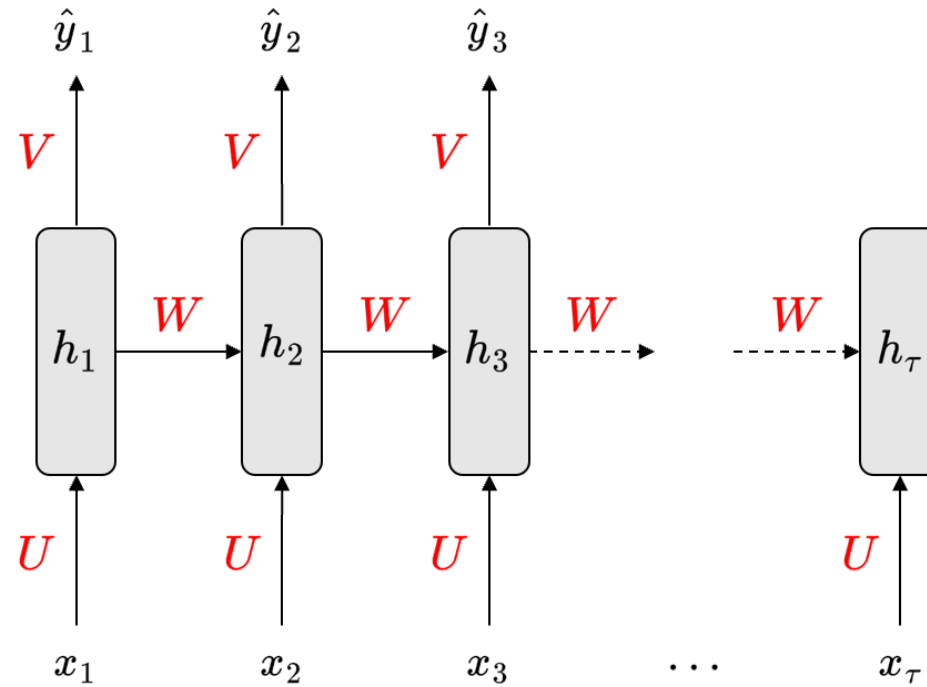
# Unrolling the Recurrence



# Unrolling the Recurrence

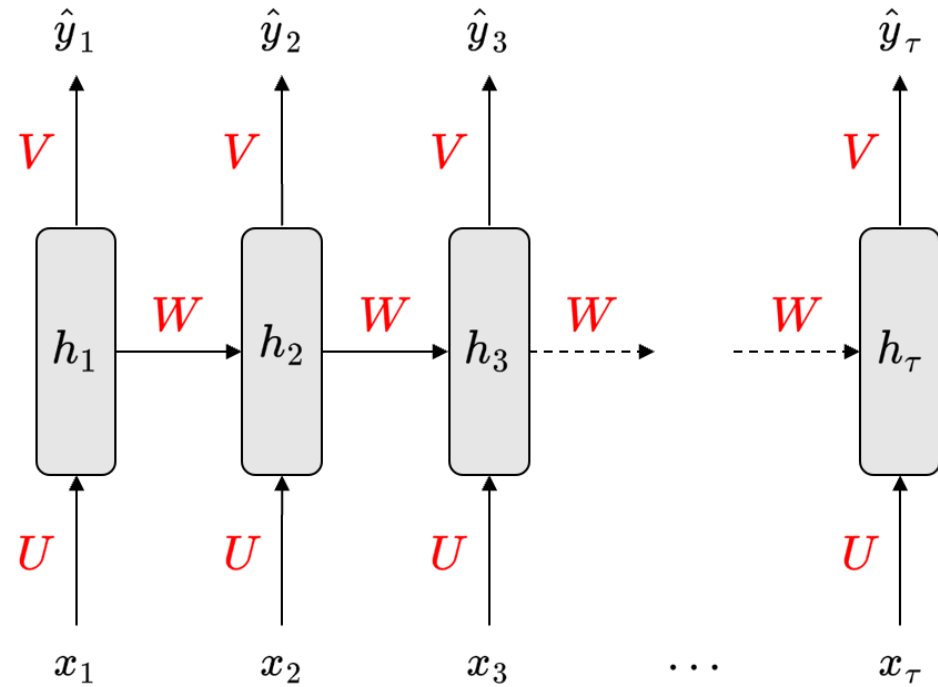


# Unrolling the Recurrence

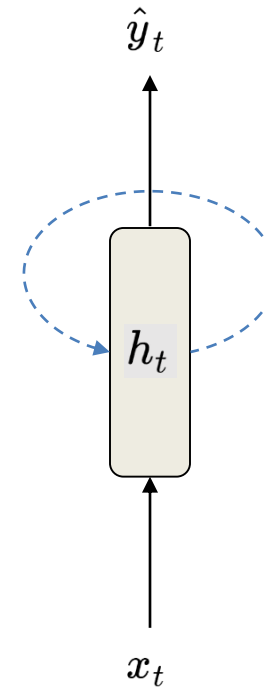




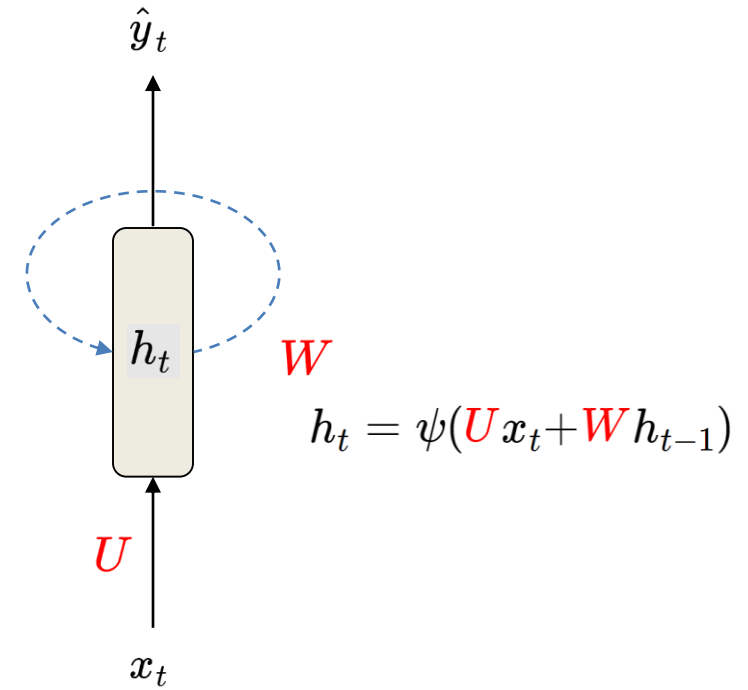
# Unrolling the Recurrence



# Recurrent Connections



# Recurrent Connections



# Feedforward Propagation

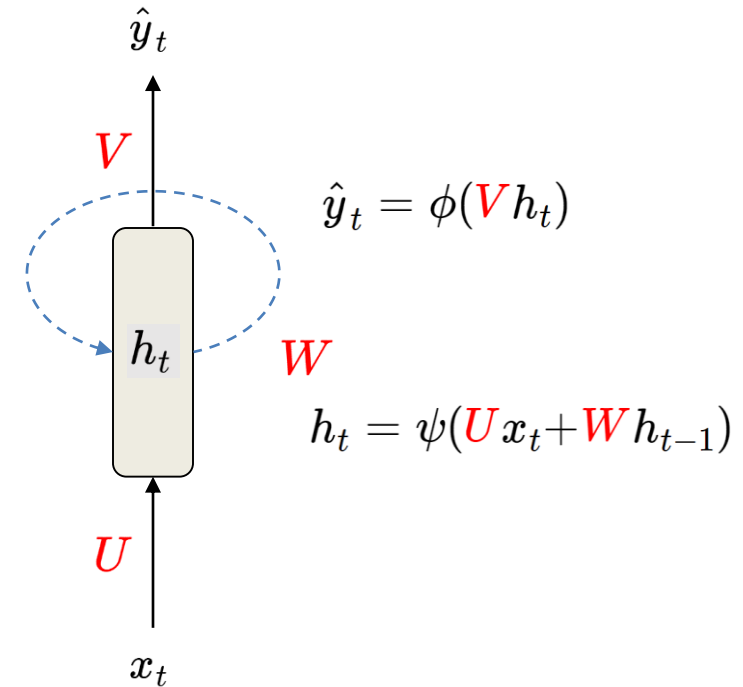
- This is a RNN where the input and output sequences are of the same length
- Feedforward operation proceeds from left to right
- Update Equations:

$$\mathbf{a}_t = b + W\mathbf{h}_{t-1} + U\mathbf{x}_t$$

$$\mathbf{h}_t = \tanh \mathbf{a}_t$$

$$\mathbf{o}_t = c + V\mathbf{h}_t$$

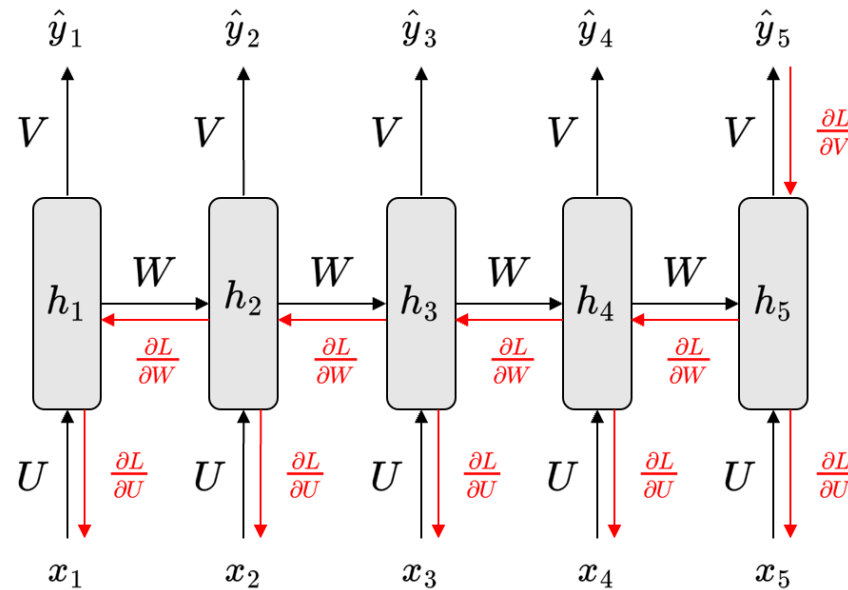
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{o}_t)$$



# How to Train RNN

# Backward Propagation

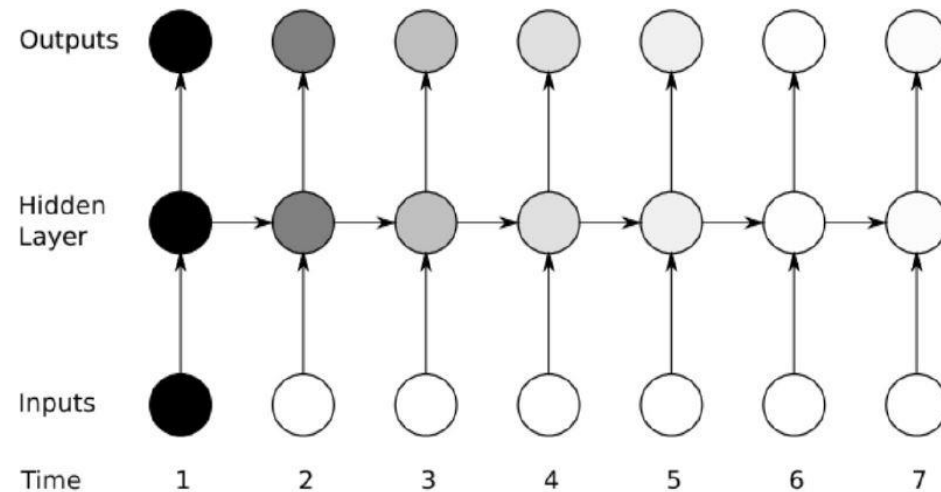
- Loss would just be the sum of losses over time steps
- Treat the recurrent network as a usual multilayer network and apply backpropagation on the unrolled network
- This is called **Backpropagation through time (BPTT)**



# Long Short-Term Memory (LSTM)

# Long Short-Term Memory (LSTM)

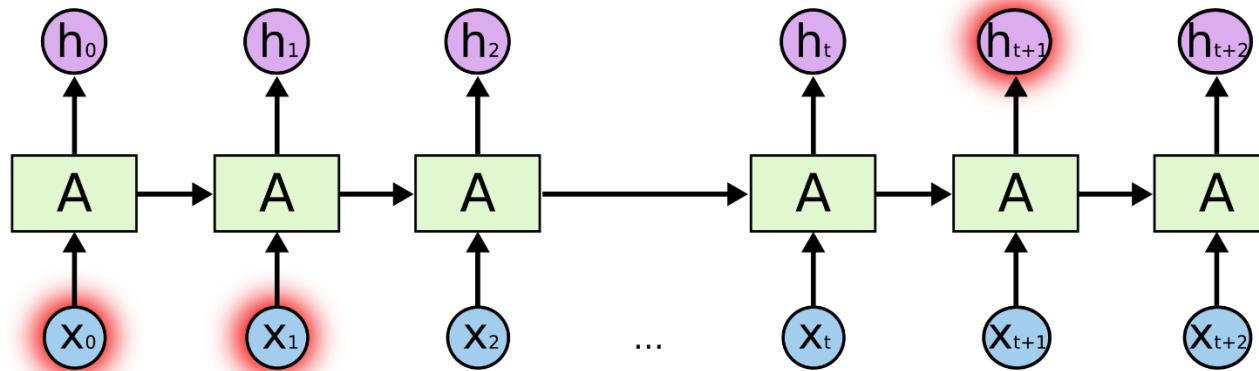
- Long-Term Dependencies
  - Gradients propagated over many stages tend to either **vanish** or **explode**
  - Difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions
  - Introduce a memory state that runs through only linear operators
  - Use gating units to control the updates of the state





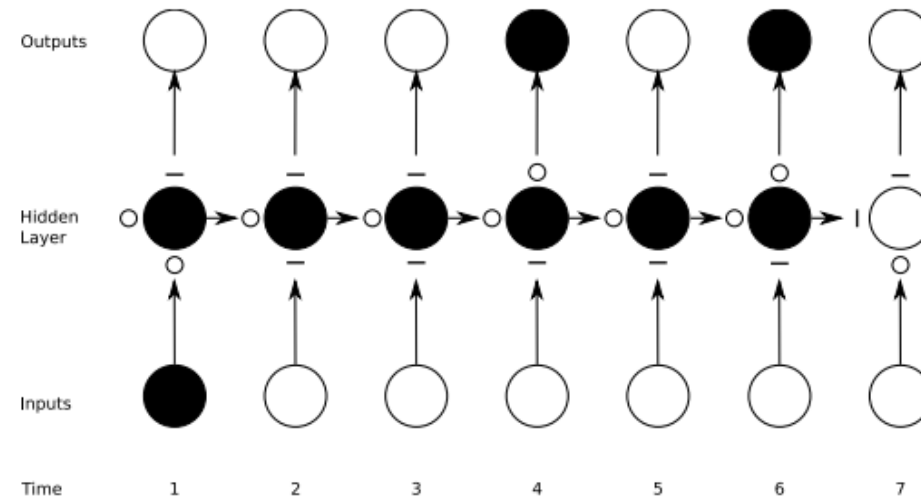
# Example

- “I grew up in France... I speak fluent *French*.”



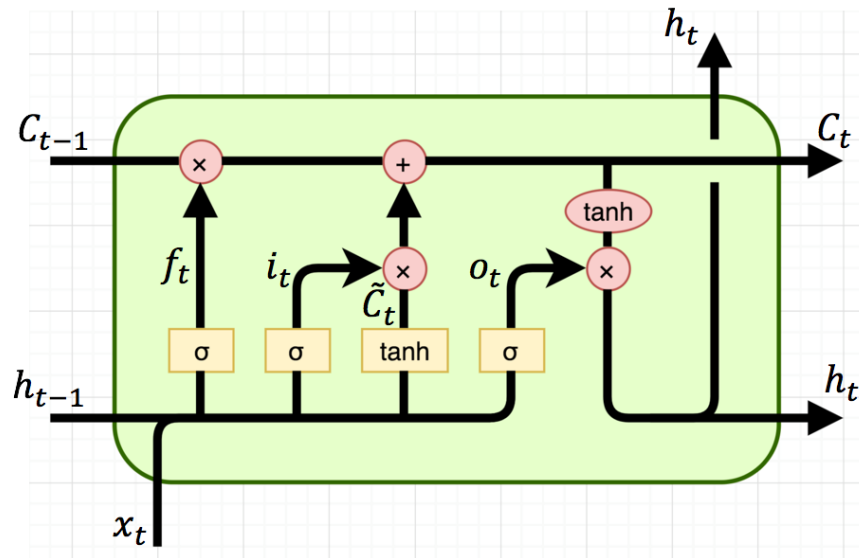
# Long Short-Term Memory (LSTM)

- Consists of a memory cell and a set of gating units
  - Memory cell is the context that carries over
  - Forget gate controls erase operation
  - Input gate controls write operation
  - Output gate controls the read operation



# Long Short-Term Memory (LSTM)

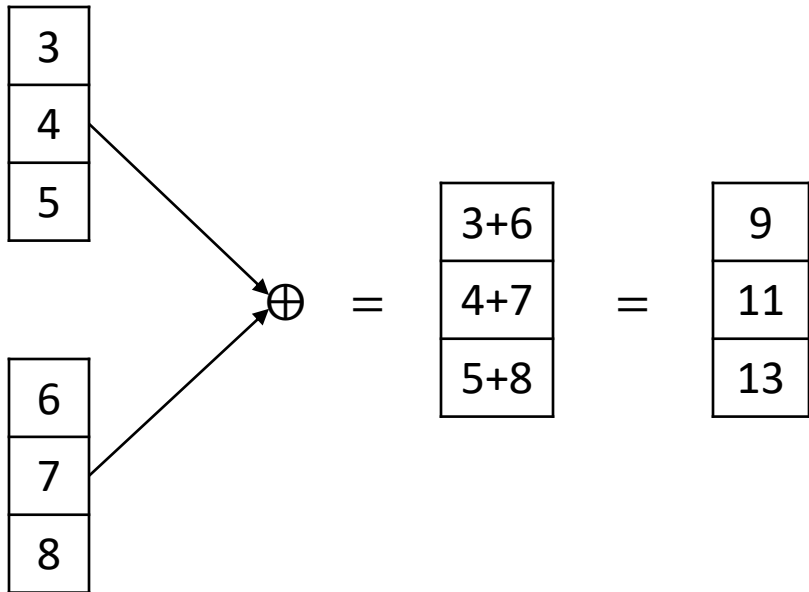
- Consists of a memory cell and a set of gating units
  - Memory cell is the context that carries over
  - Forget gate controls erase operation
  - Input gate controls write operation
  - Output gate controls the read operation



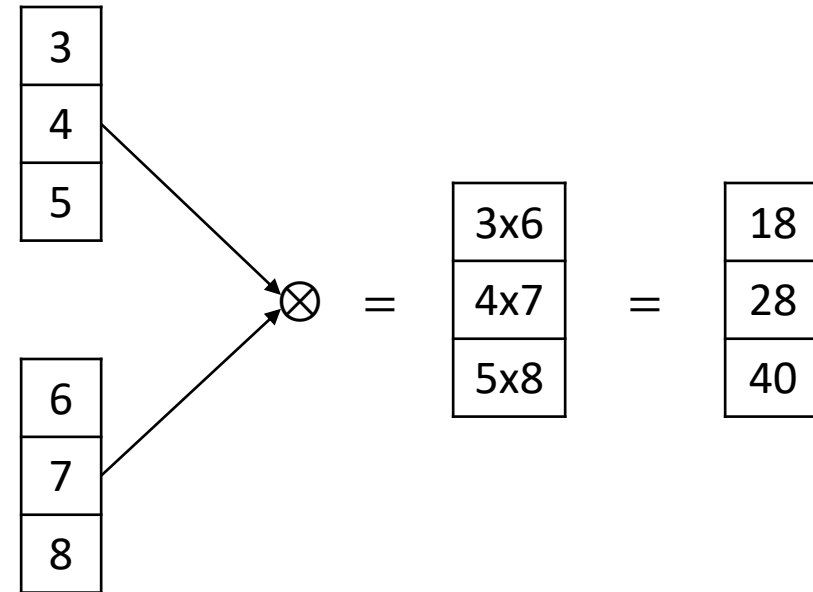
$$\begin{aligned}i_t &= \sigma(x_t U^i + h_{t-1} W^i) \\f_t &= \sigma(x_t U^f + h_{t-1} W^f) \\o_t &= \sigma(x_t U^o + h_{t-1} W^o) \\\tilde{C}_t &= \tanh(x_t U^g + h_{t-1} W^g) \\C_t &= \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t) \\h_t &= \tanh(C_t) * o_t\end{aligned}$$

# Element-by-Element

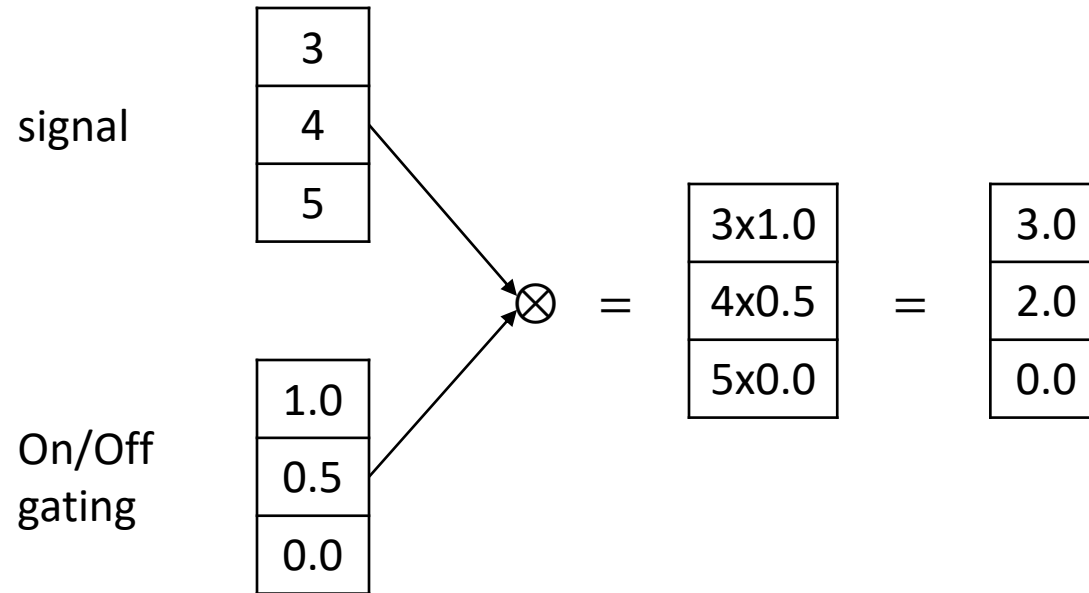
Element-by-Element Addition



Element-by-Element Multiplication

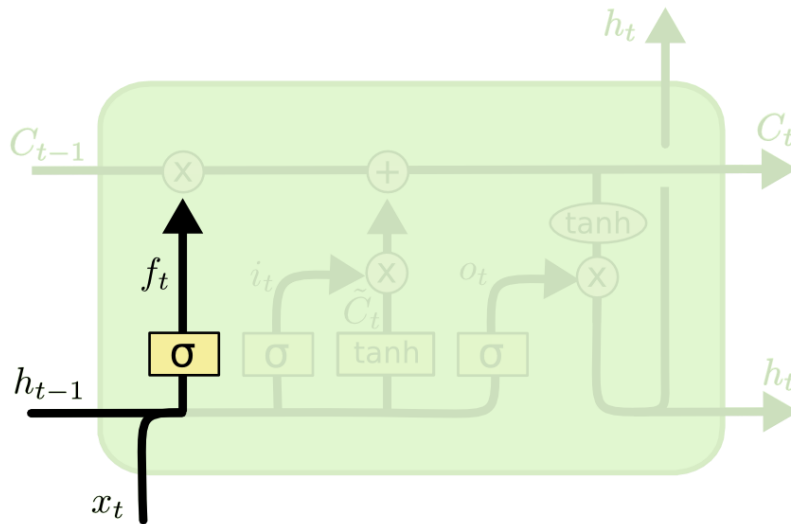


# Gating



# Long Short-Term Memory (LSTM)

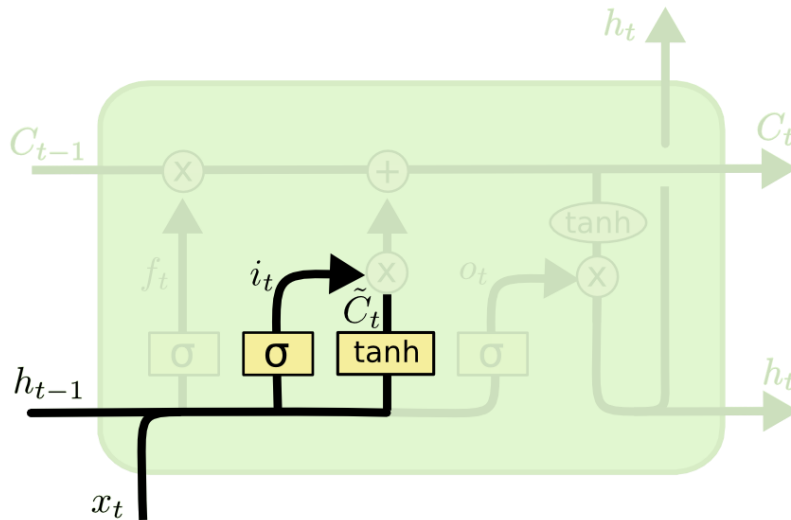
- Forget gate controls erase operation
- Input gate controls write operation
- Output gate controls the read operation



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

# Long Short-Term Memory (LSTM)

- Forget gate controls erase operation
- Input gate controls write operation
- Output gate controls the read operation

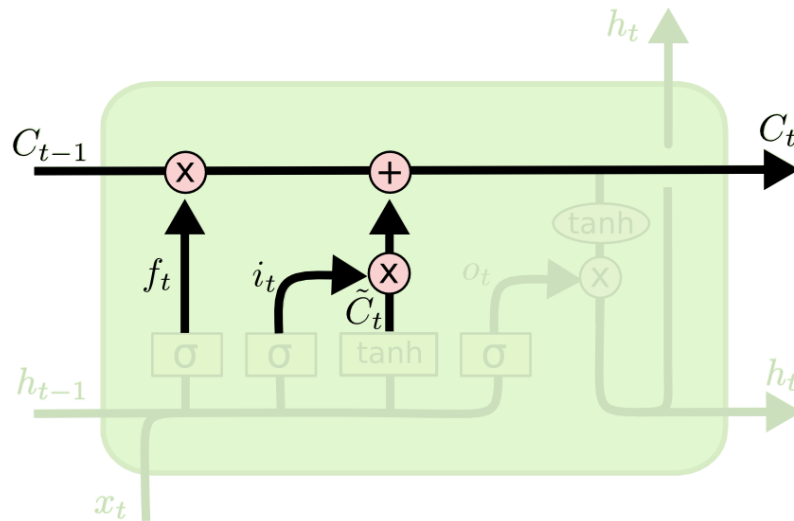


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

# Long Short-Term Memory (LSTM)

- Forget gate controls erase operation
- Input gate controls write operation
- Output gate controls the read operation

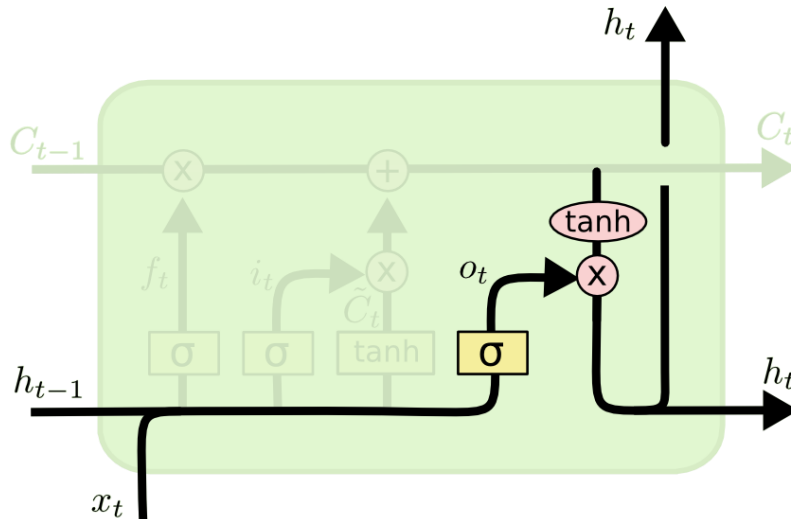


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



# Long Short-Term Memory (LSTM)

- Forget gate controls erase operation
- Input gate controls write operation
- Output gate controls the read operation

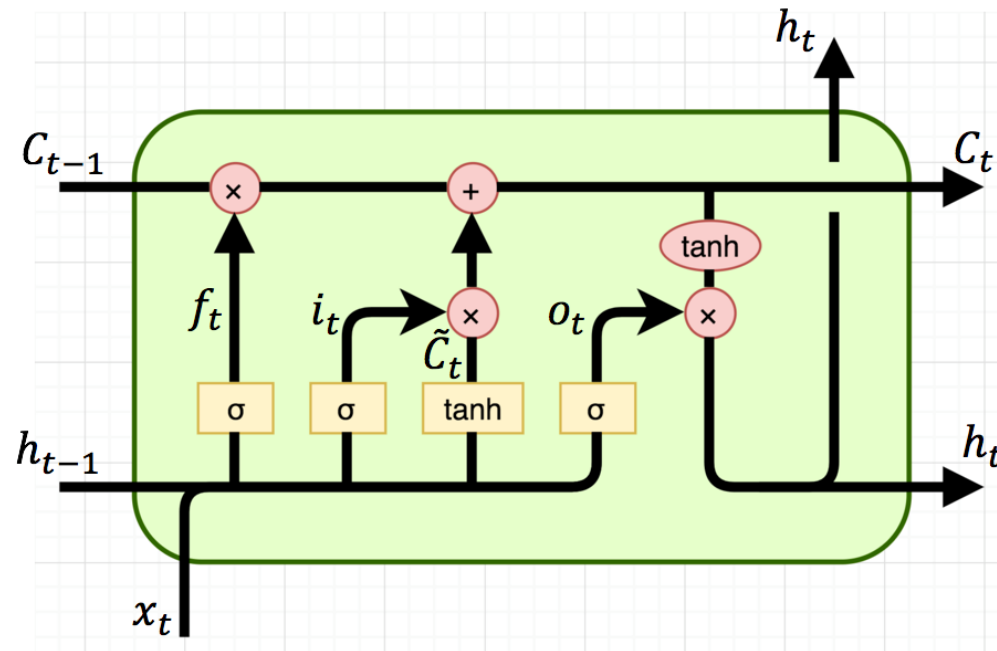


$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

# Long Short-Term Memory (LSTM)

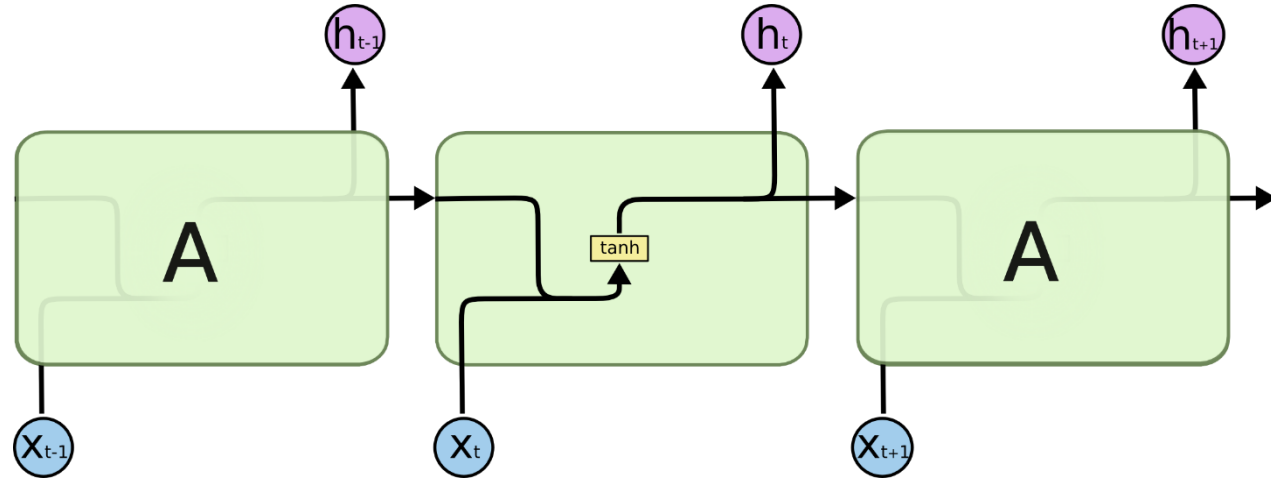
- Forget gate controls erase operation
- Input gate controls write operation
- Output gate controls the read operation



# Comparison RNN and LSTM

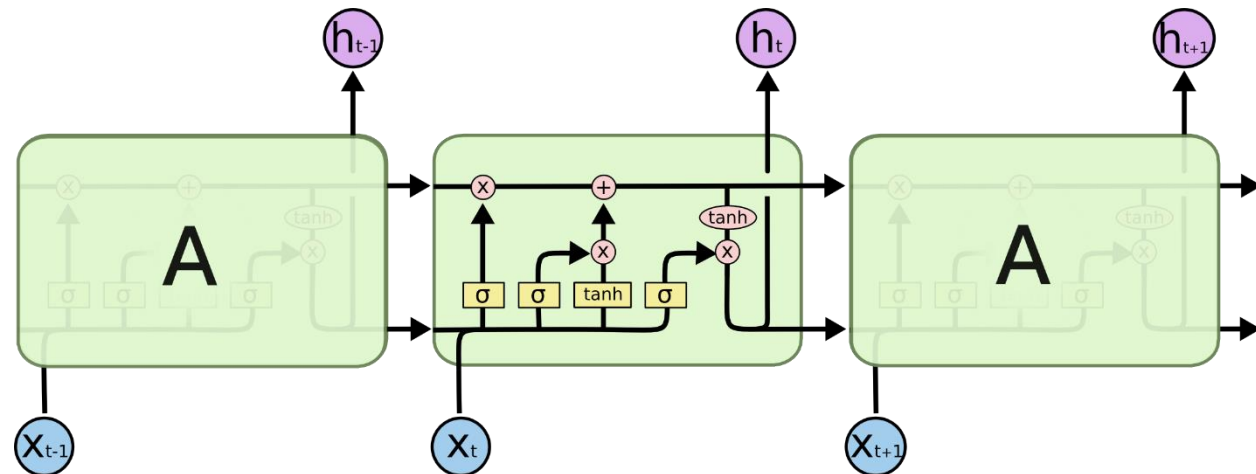
## Vanilla RNN

$$h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$



## LSTM

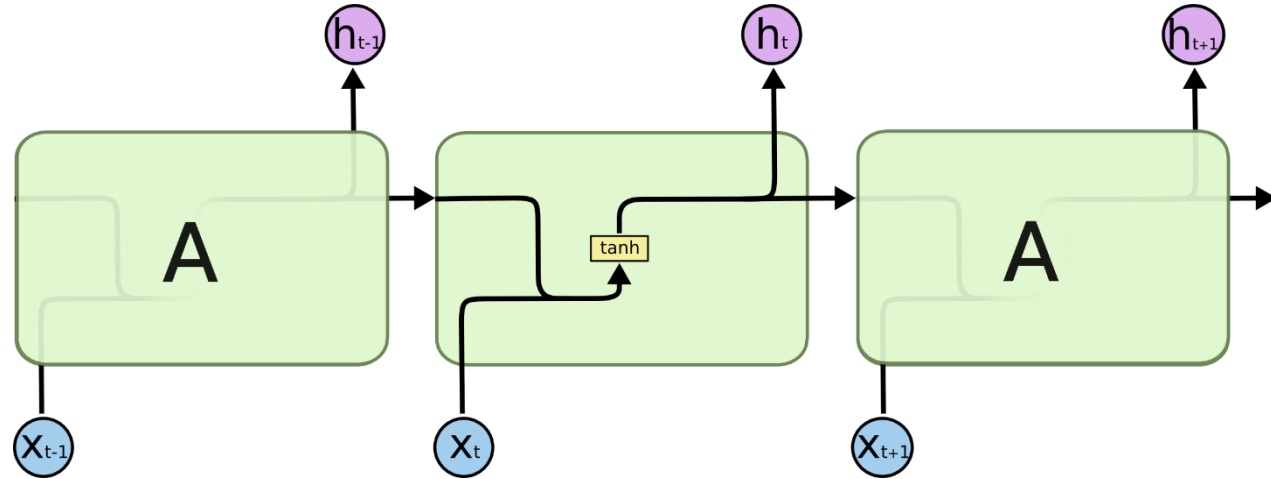
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$



# Comparison RNN and LSTM

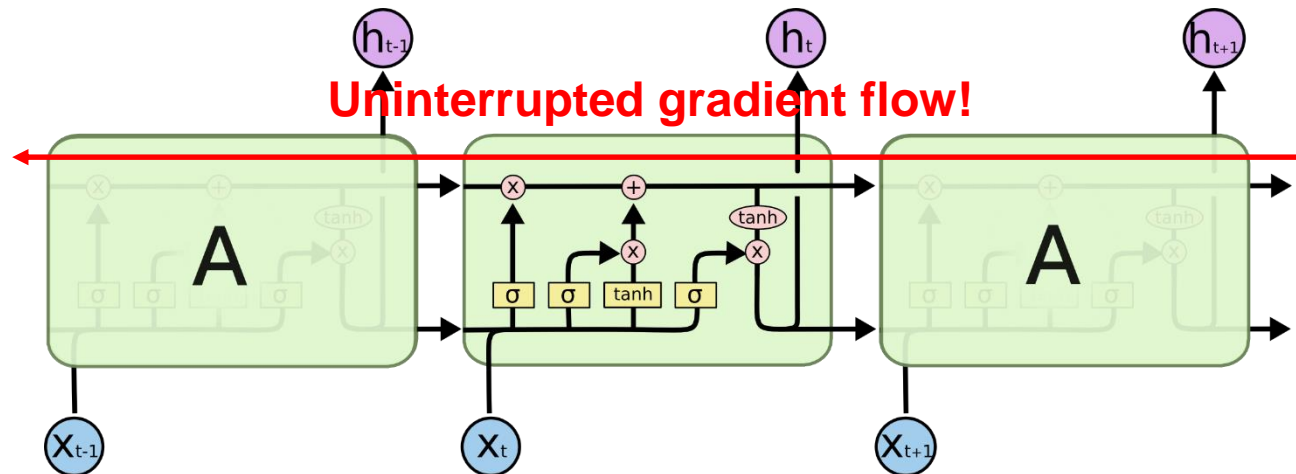
## Vanilla RNN

$$h_t = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$



## LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

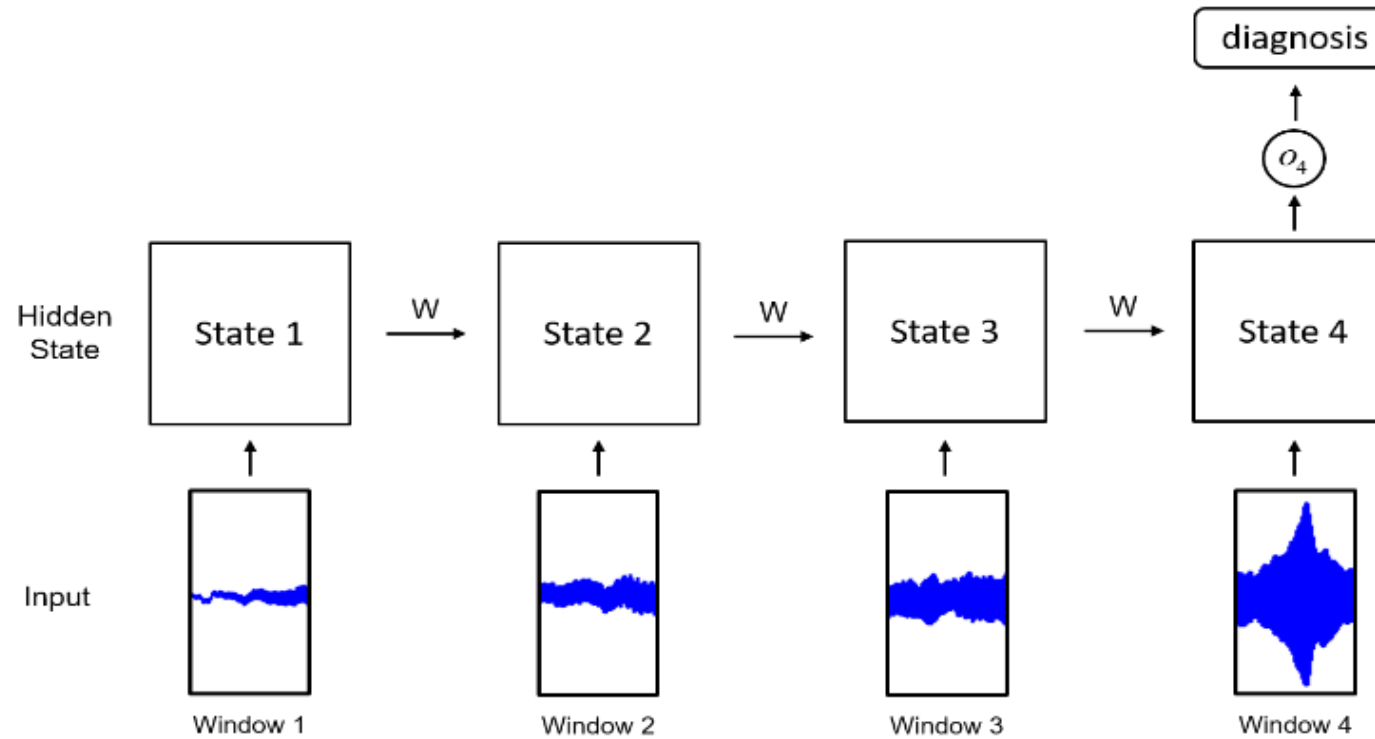


# Weakness of RNN and LSTM

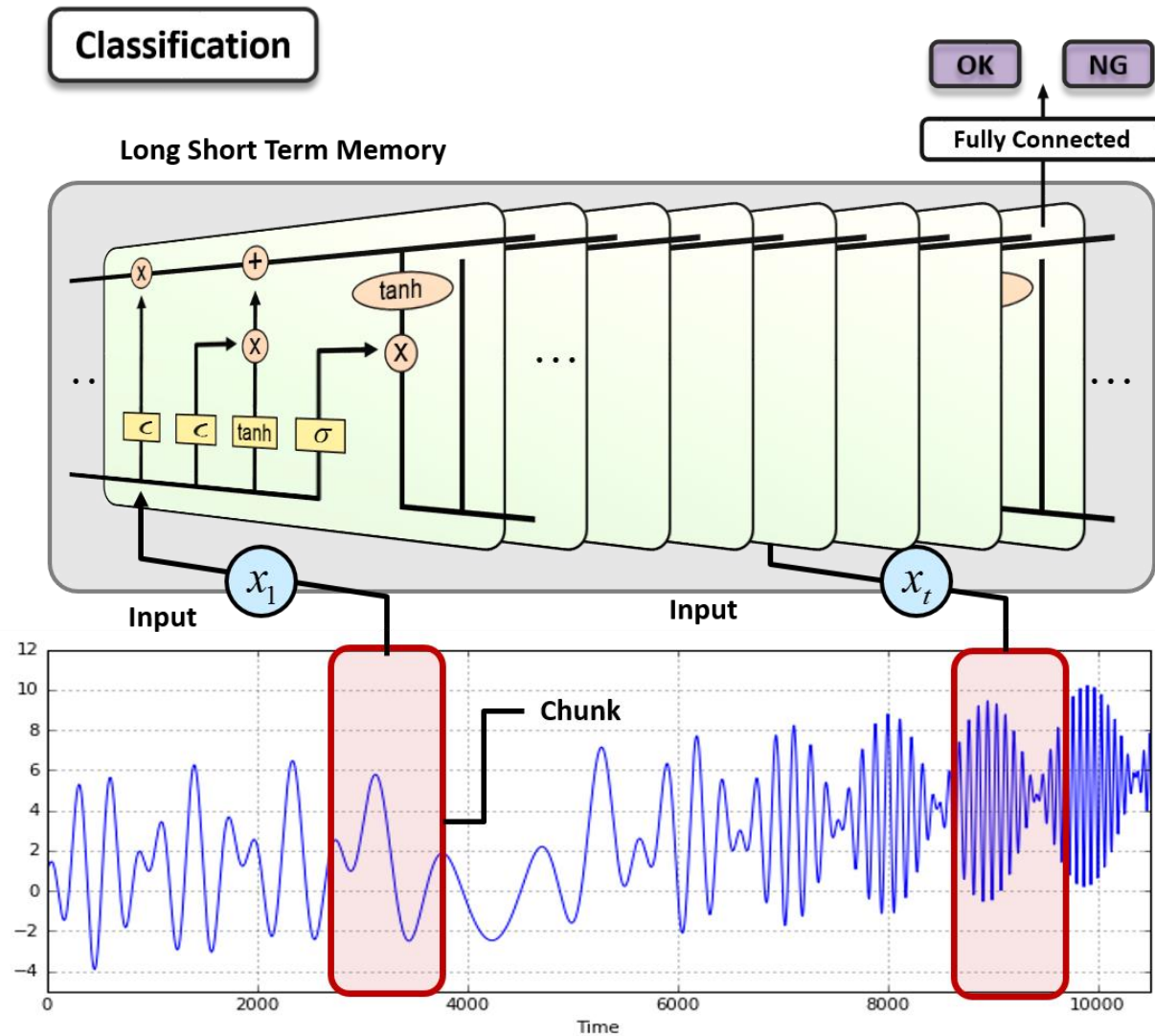
- Sequential computation is slow
- Vanishing and exploding gradients are still problematic
- Long-term credit assignment is difficult

# LSTM Implementation

# Time Series Data and LSTM

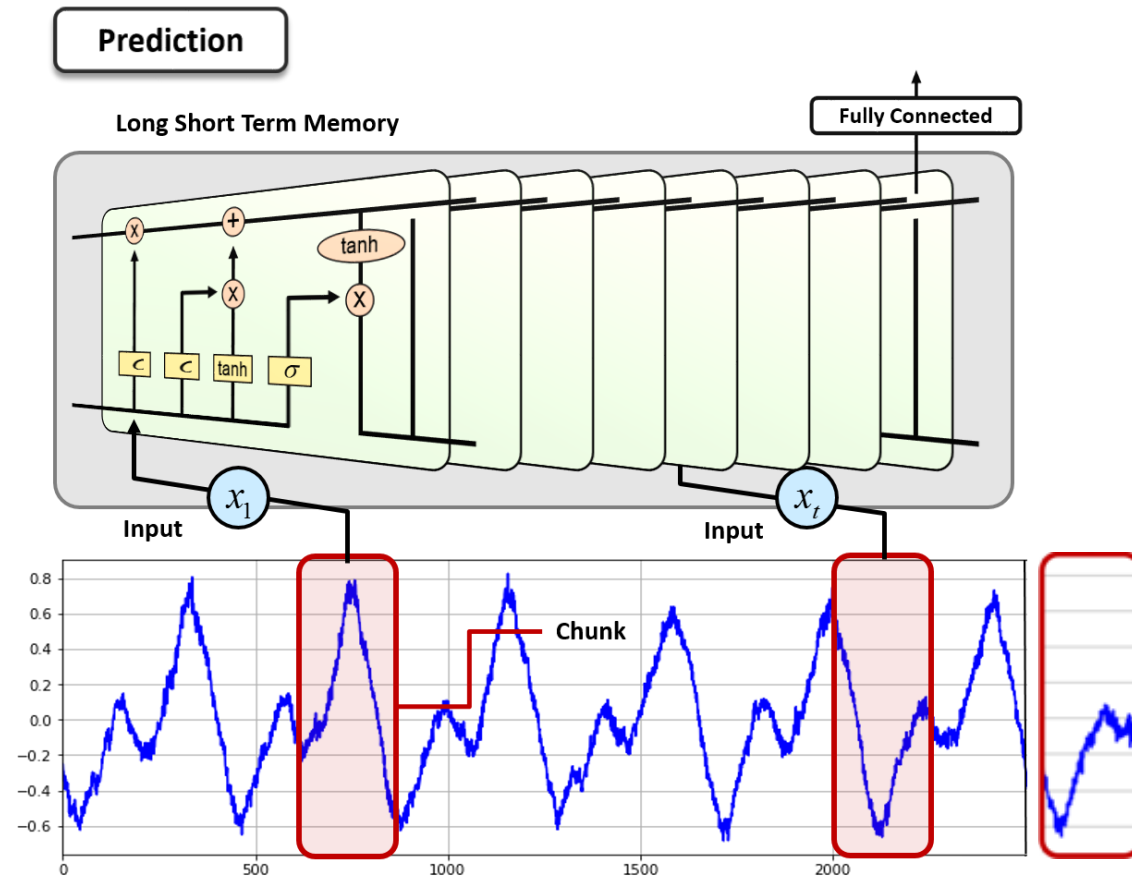


# LSTM for Classification



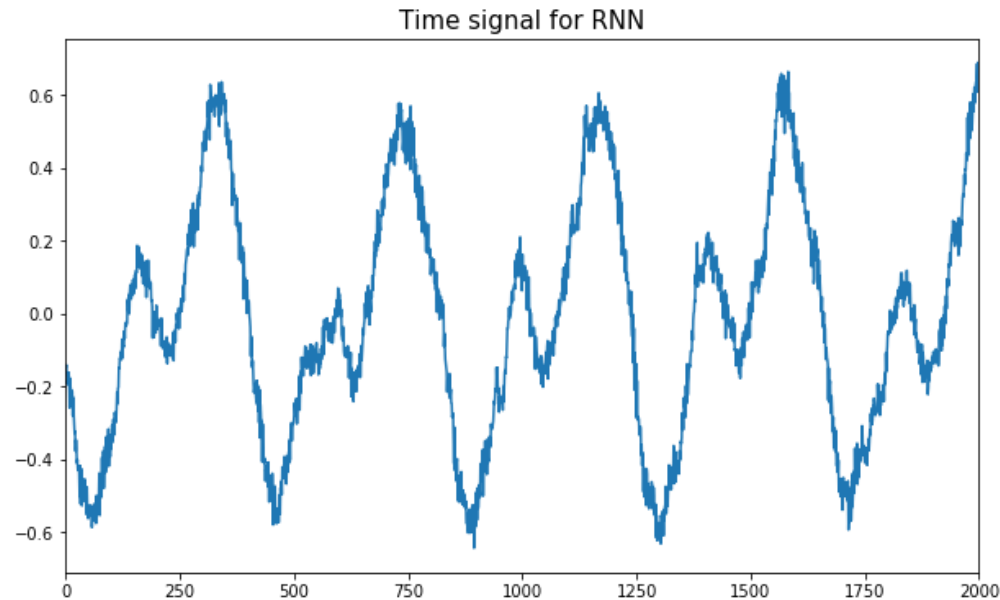


# LSTM for Prediction



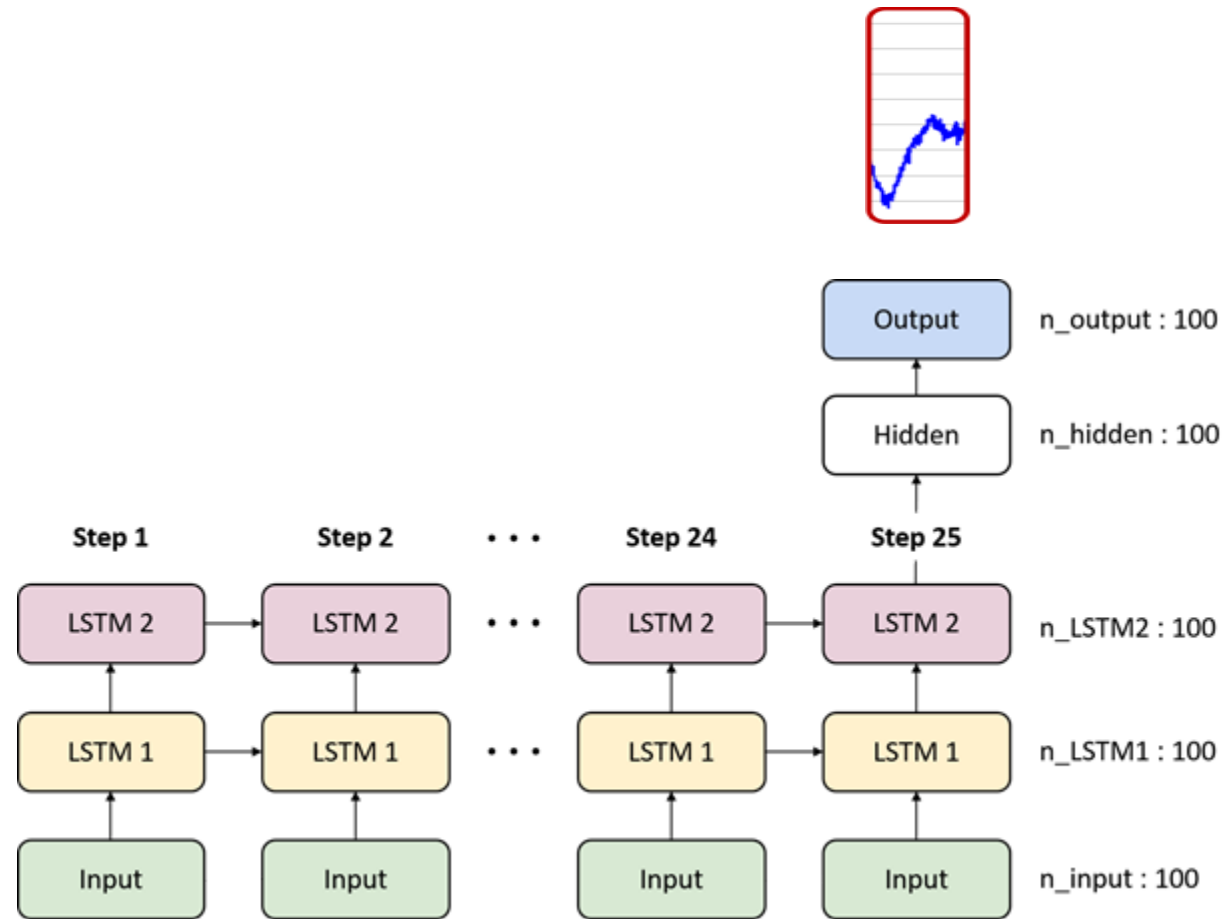
# LSTM with TensorFlow

- An example for predicting a next piece of an acceleration signal
- Regression problem



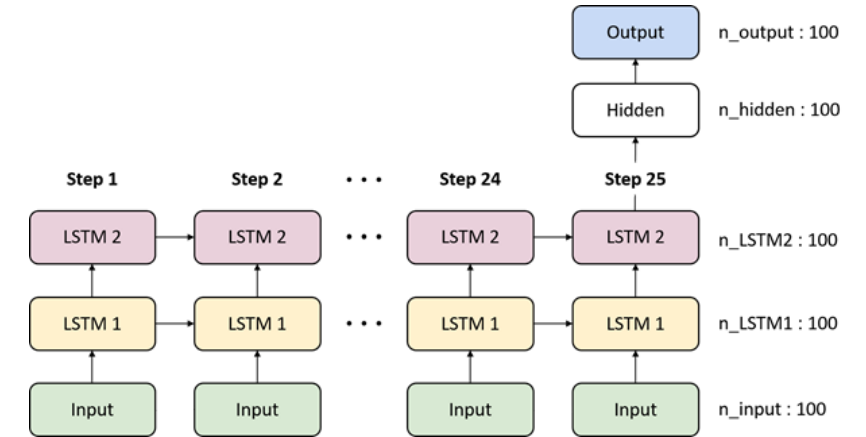
# LSTM Structure

```
n_step = 25  
n_input = 100  
  
## LSTM shape  
n_lstm1 = 100  
n_lstm2 = 100  
  
## Fully connected  
n_hidden = 100  
n_output = 100
```



# LSTM, Weights and Biases

- LSTM Cell
  - Do not need to define weights and biases of LSTM cells
- Fully connected
  - Define parameters based on the predefined layer size
  - Initialize with a normal distribution with  $\mu = 0$  and  $\sigma = 0.01$

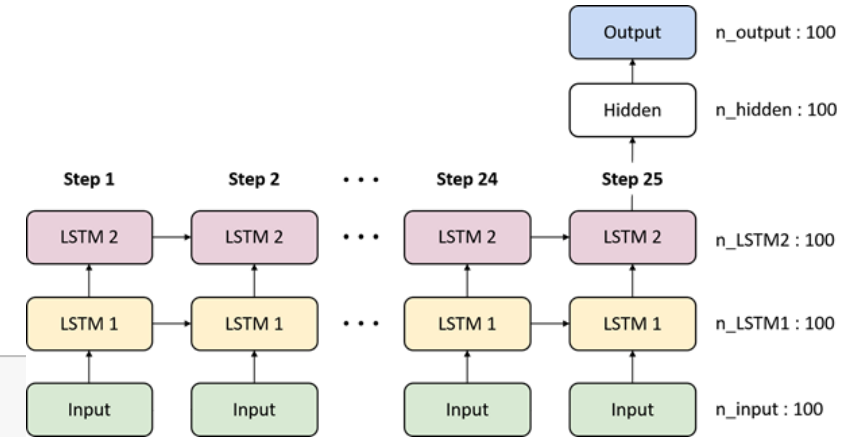


```
weights = {  
    'hidden' : tf.Variable(tf.random_normal([n_lstm2, n_hidden], stddev = 0.01)),  
    'output' : tf.Variable(tf.random_normal([n_hidden, n_output], stddev = 0.01))  
}  
  
biases = {  
    'hidden' : tf.Variable(tf.random_normal([n_hidden], stddev = 0.01)),  
    'output' : tf.Variable(tf.random_normal([n_output], stddev = 0.01))  
}  
  
x = tf.placeholder(tf.float32, [None, n_step, n_input])  
y = tf.placeholder(tf.float32, [None, n_output])
```

# Build a Model

- First, define the LSTM cells
- Second, compute hidden state (h) and LSTM cell (c) with the predefined LSTM cell and input

```
def build_model(x, weights, biases):  
    with tf.variable_scope('rnn'):  
        # Build RNN network  
        with tf.variable_scope('lstm1'):  
            lstm1 = tf.nn.rnn_cell.LSTMCell(n_lstm1)  
            h1, c1 = tf.nn.dynamic_rnn(lstm1, x, dtype = tf.float32)  
        with tf.variable_scope('lstm2'):  
            lstm2 = tf.nn.rnn_cell.LSTMCell(n_lstm2)  
            h2, c2 = tf.nn.dynamic_rnn(lstm2, h1, dtype = tf.float32)  
  
        # Build classifier  
        hidden = tf.add(tf.matmul(h2[:, -1, :], weights['hidden']), biases['hidden'])  
        hidden = tf.nn.relu(hidden)  
        output = tf.add(tf.matmul(hidden, weights['output']), biases['output'])  
    return output
```



# Cost, Initializer and Optimizer

- Loss
  - Regression: Squared loss
- Initializer
  - Initialize all the empty variables
- Optimizer
  - AdamOptimizer: the most popular optimizer

$$\frac{1}{N} \sum_{i=1}^N (\hat{y}^{(i)} - y^{(i)})^2$$

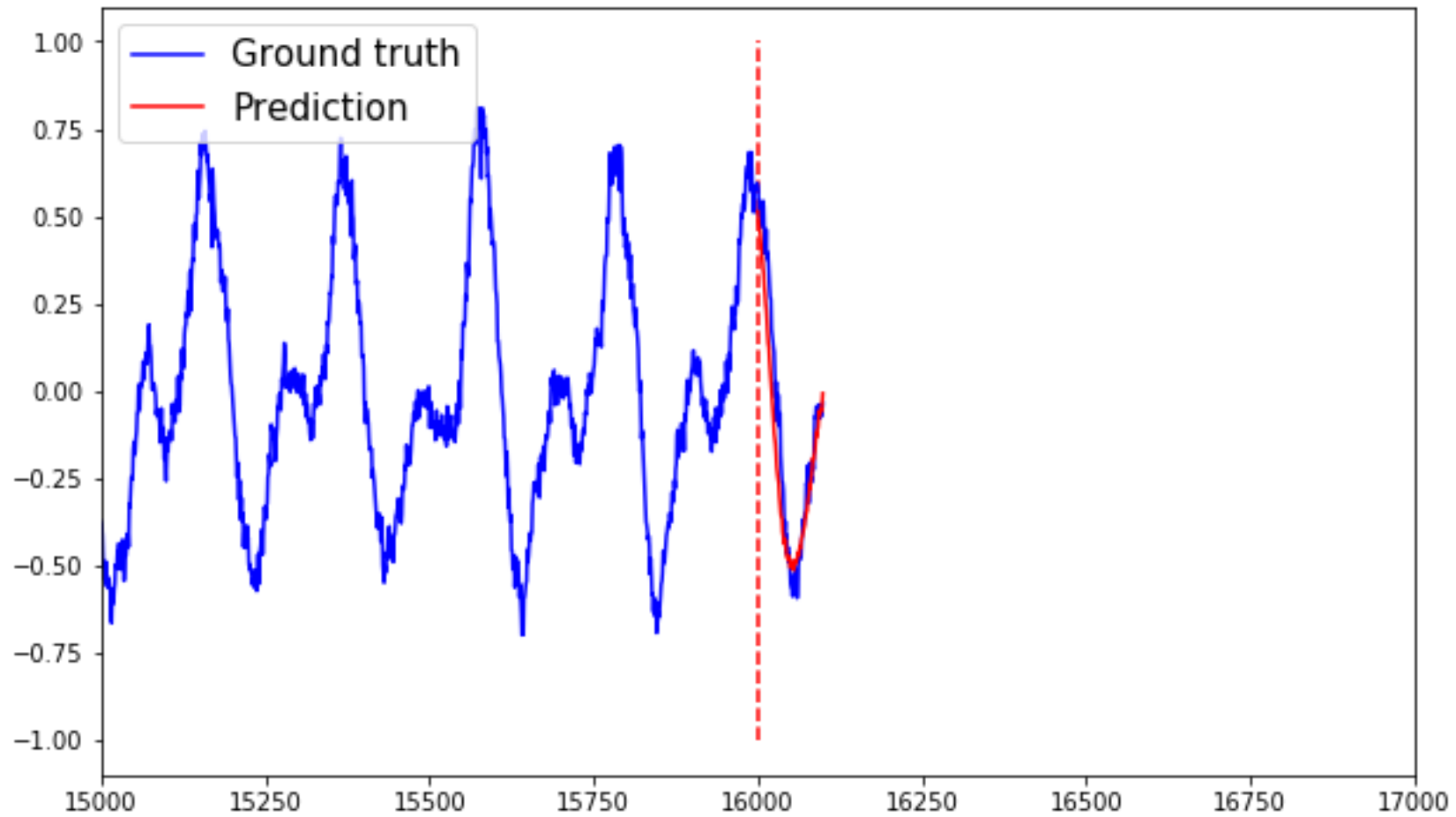
```
LR = 0.0001

pred = build_model(x, weights, biases)
loss = tf.square(tf.subtract(y, pred))
loss = tf.reduce_mean(loss)

optm = tf.train.AdamOptimizer(LR).minimize(loss)
init = tf.global_variables_initializer()

sess = tf.Session()
```

# Prediction Example



# Prediction Example

