Recurrent Neural Network (RNN)

Industrial AI Lab.

Prof. Seungchul Lee

Recurrence

- Consider the classical form of a dynamical system:

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

- This is recurrent because the definition of s at time t refers back to the same definition at time $t - 1$
- For some finite number of time steps τ, the graph represented by this recurrence can be unfolded by using the definition $\tau - 1$ times. For example, when $\tau = 3$

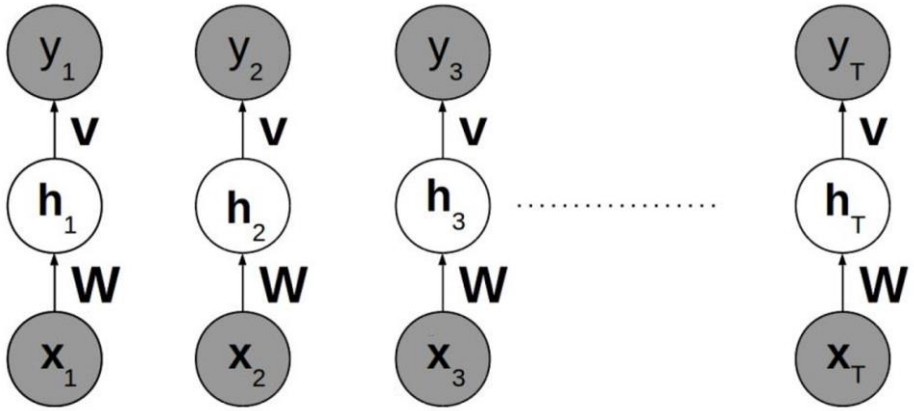
$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

- This expression does not involve any recurrence and can be represented by a traditional directed acyclic computational graph

Recurrent Neural Network (RNN)

- RNNs are a family of neural networks for processing sequential data

- Feedforward Network and Sequential Data
  - Separate parameters for each value of the time index
  - Cannot share statistical strength across different time indices



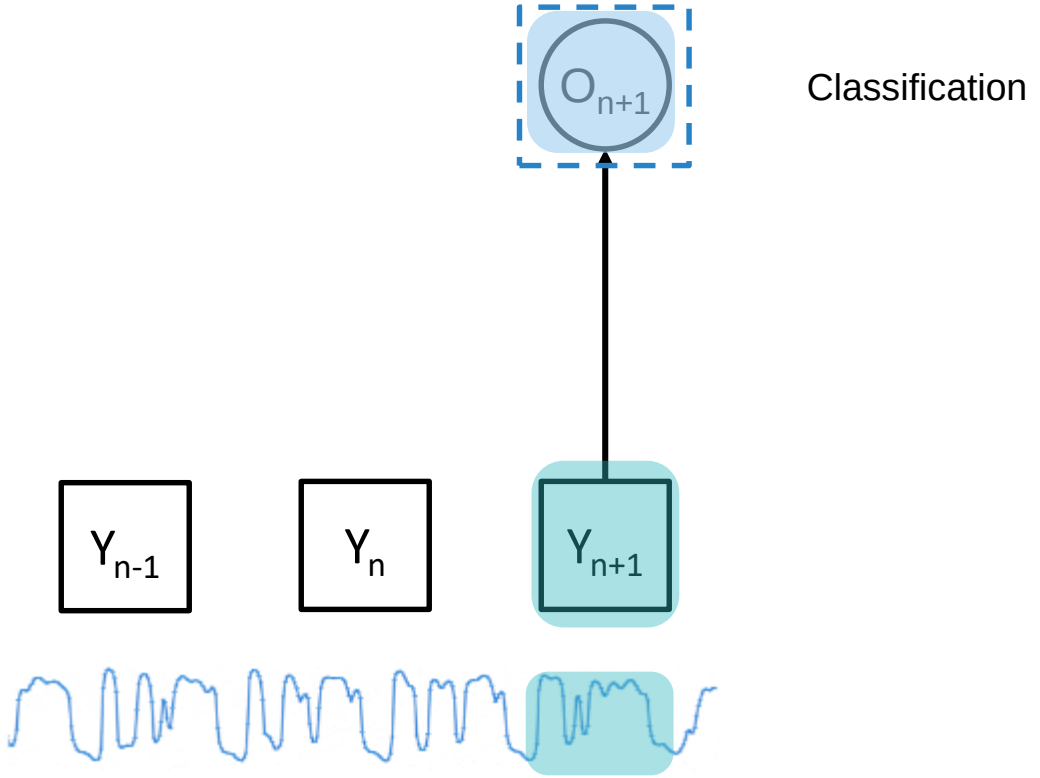
Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN)

- In many situations one must consider a series of inputs to produce an output
  - Outputs to may be a series

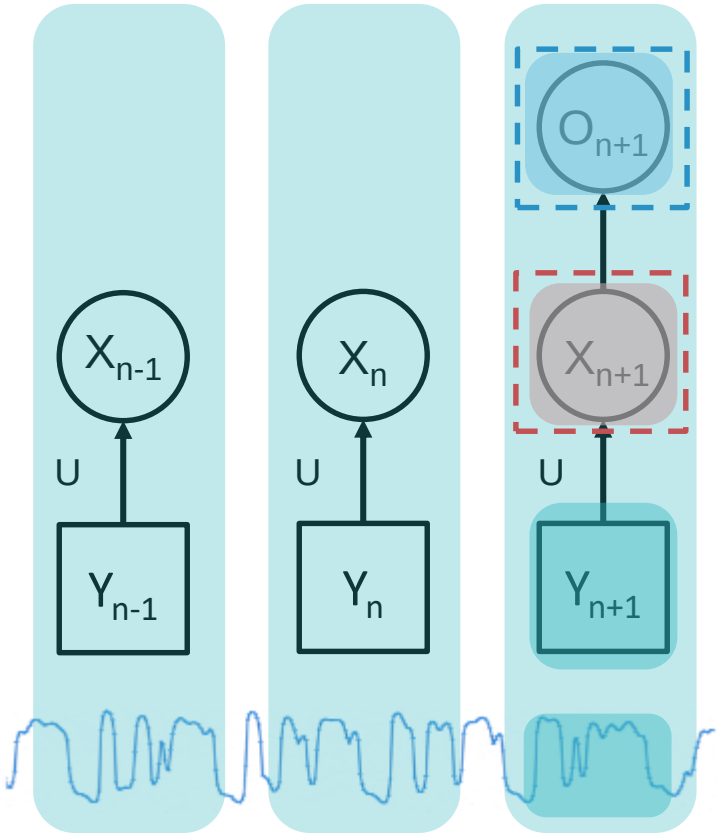
Recurrent NN (RNN)

- Hidden state extraction and transformation



Recurrent NN (RNN)

- Hidden state extraction and transformation

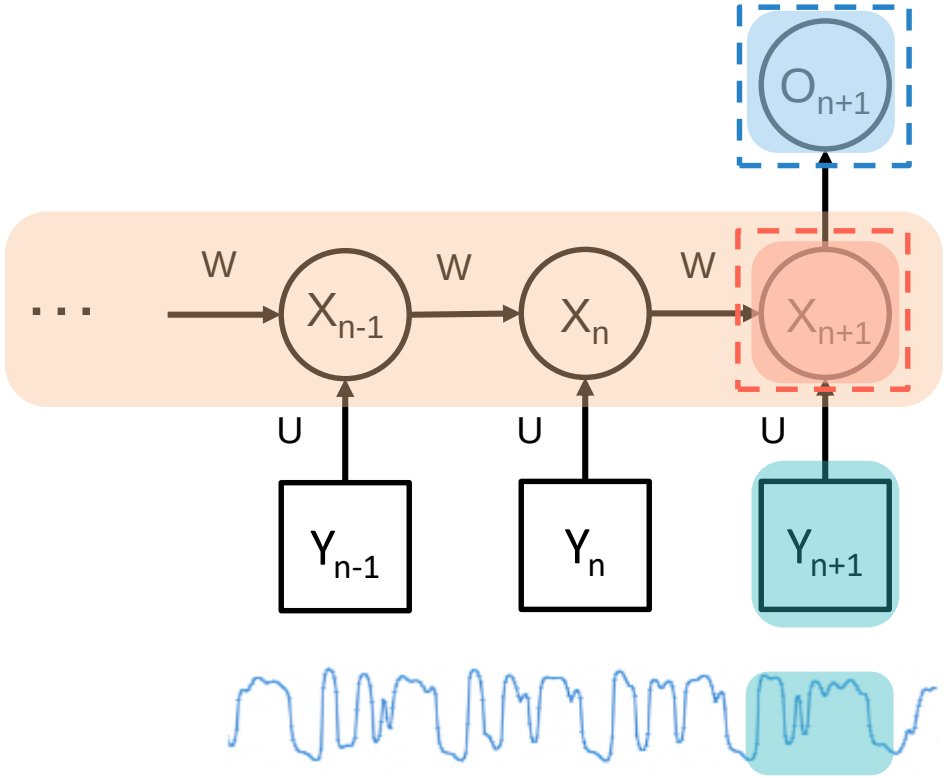


Classification based on states

Learned latent state

Recurrent NN (RNN)

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

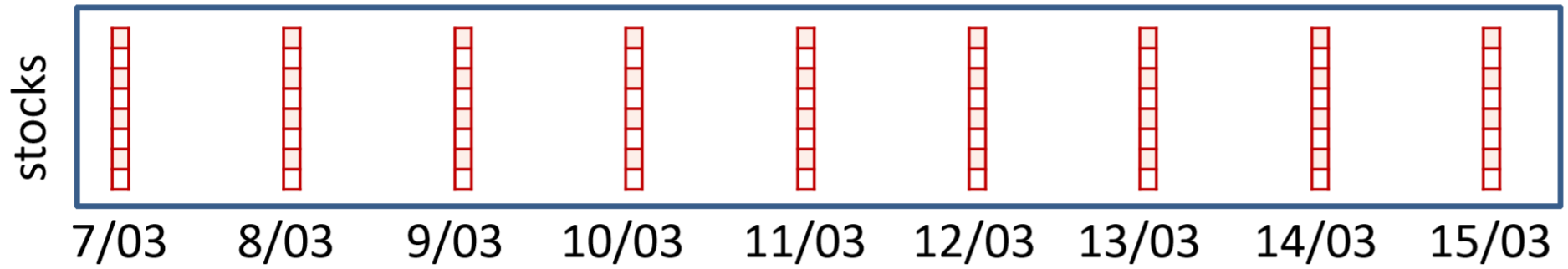


Classification based on states

Learned latent state and its dynamics

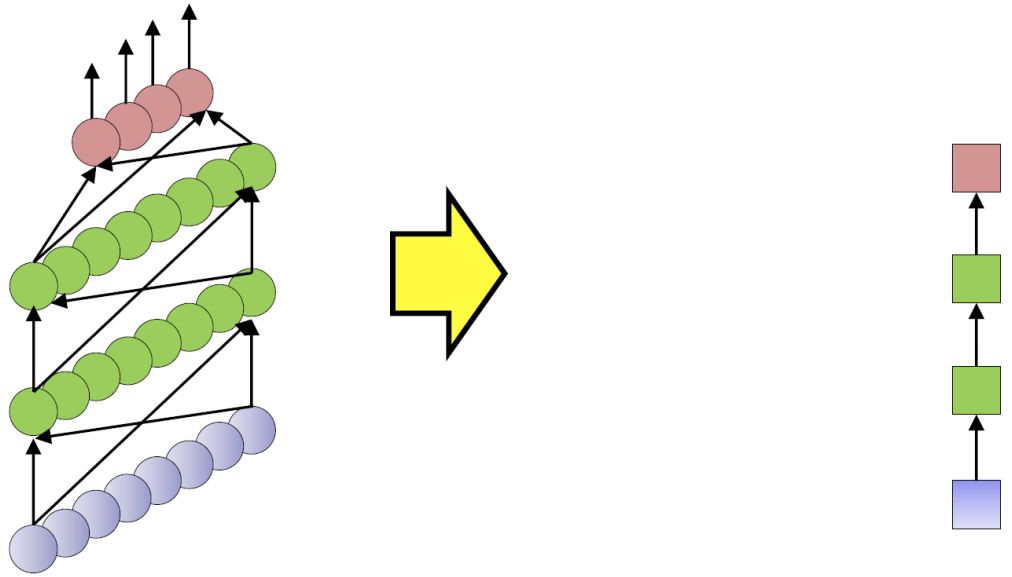
Should I Invest…

- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today
  - Ideally consider all of history



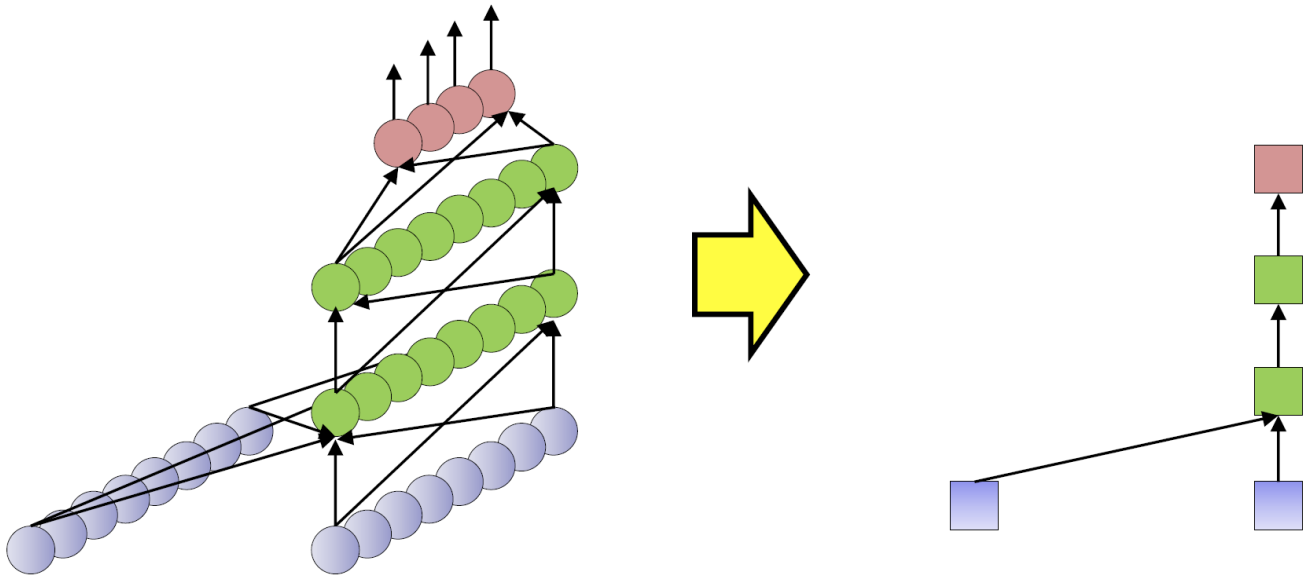
- Note: Inputs are vectors. Output may be scalar or vector

Representation Shortcut



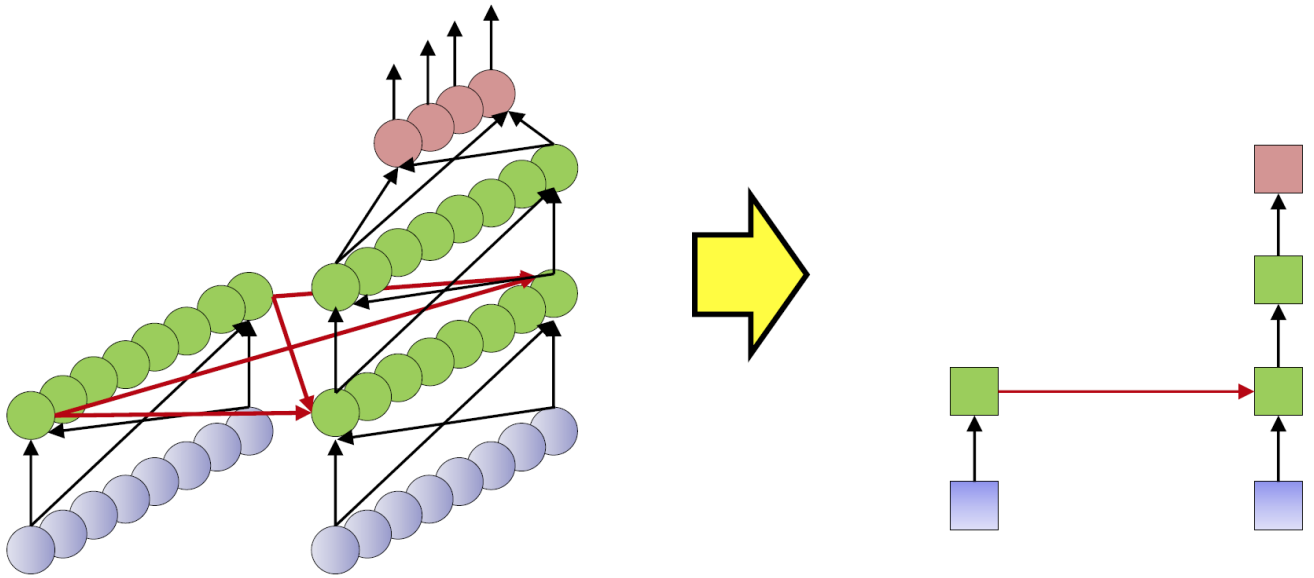
- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

Representation Shortcut



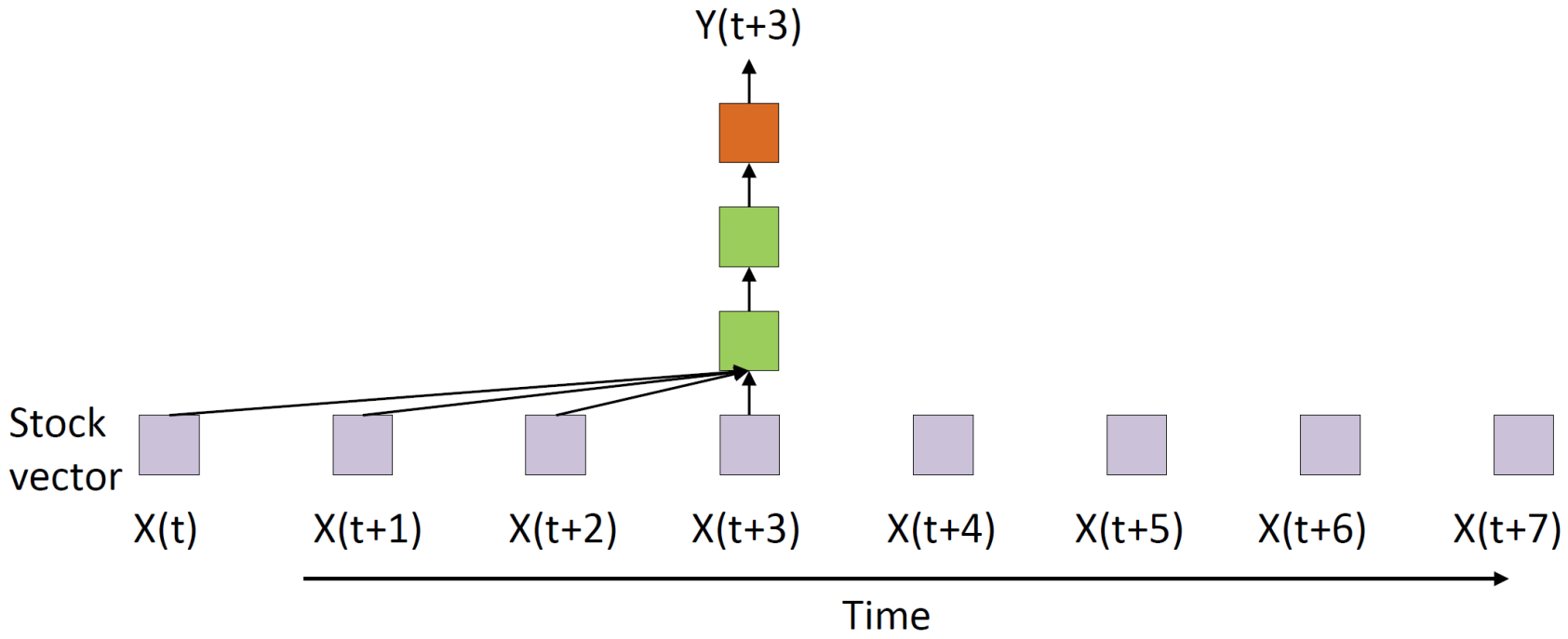
- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

Representation Shortcut



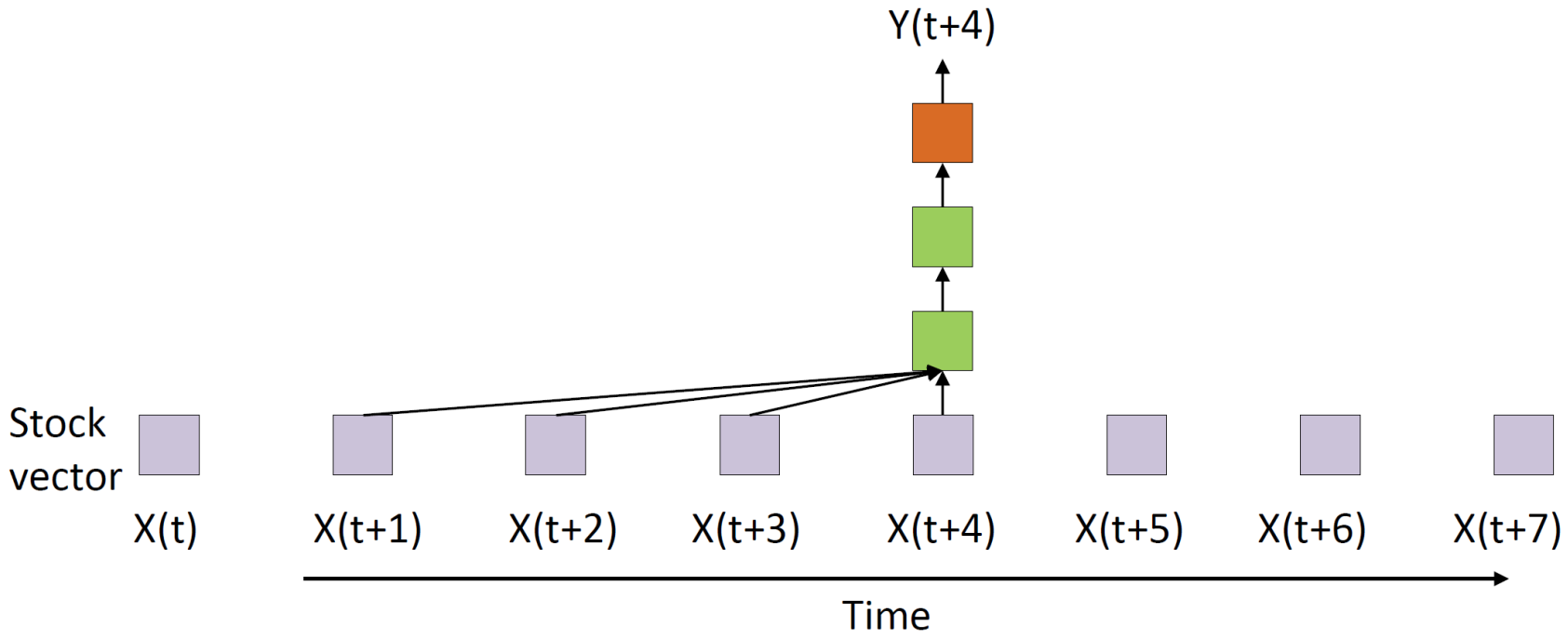
- Input at each time is a vector
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything simple boxes
  - Each box actually represents an entire layer with many units

Stock Predictor



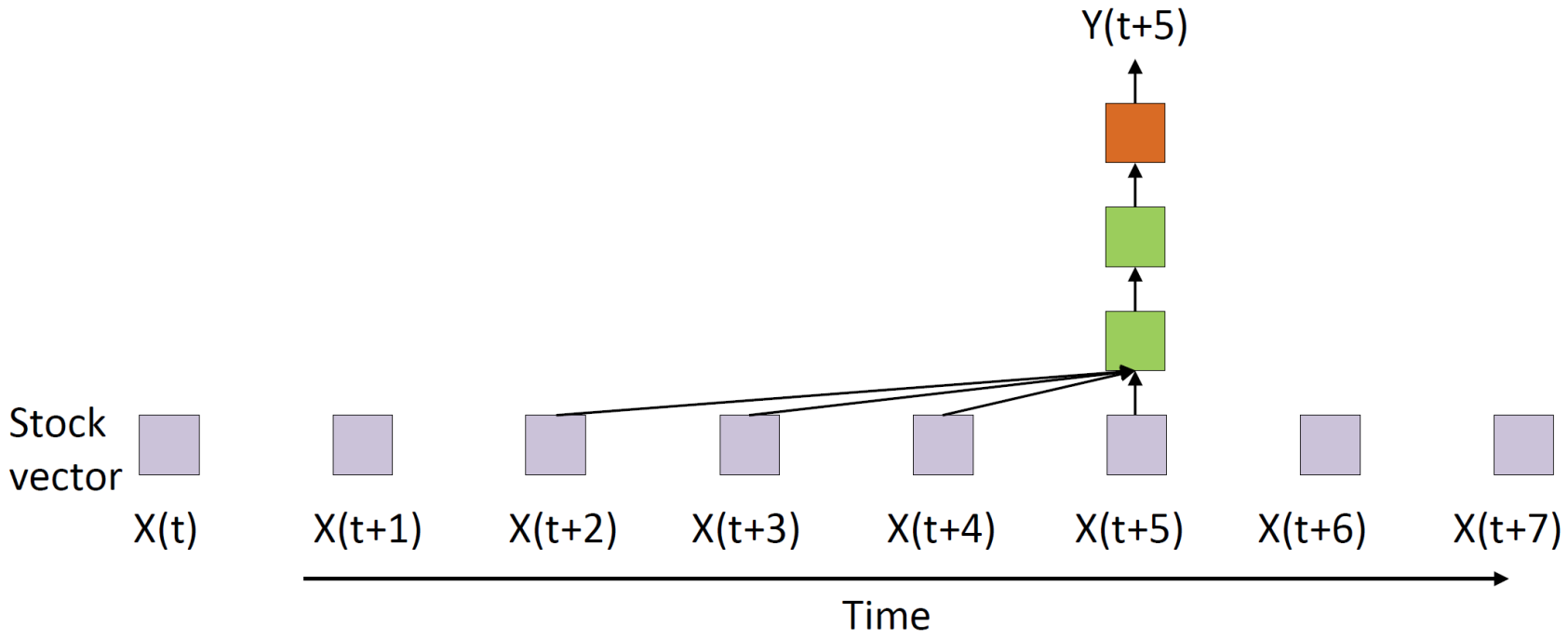
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

Stock Predictor



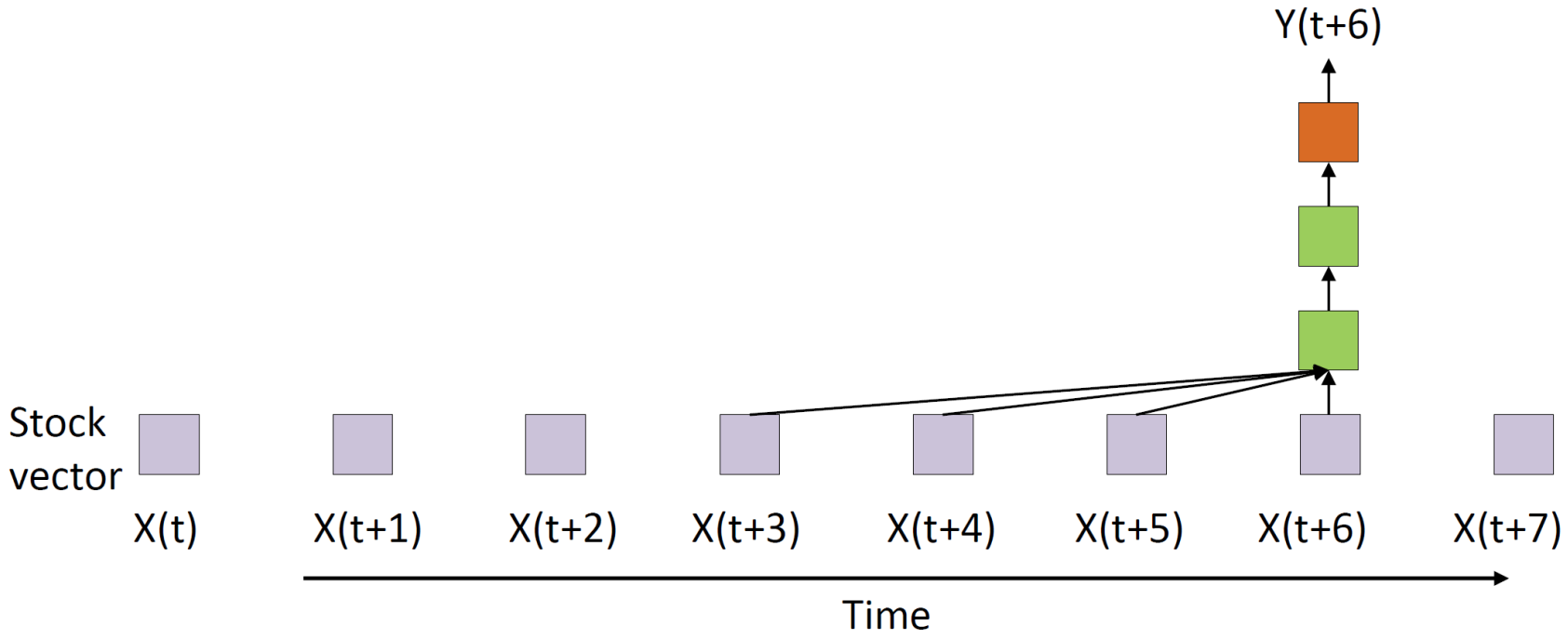
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

Stock Predictor



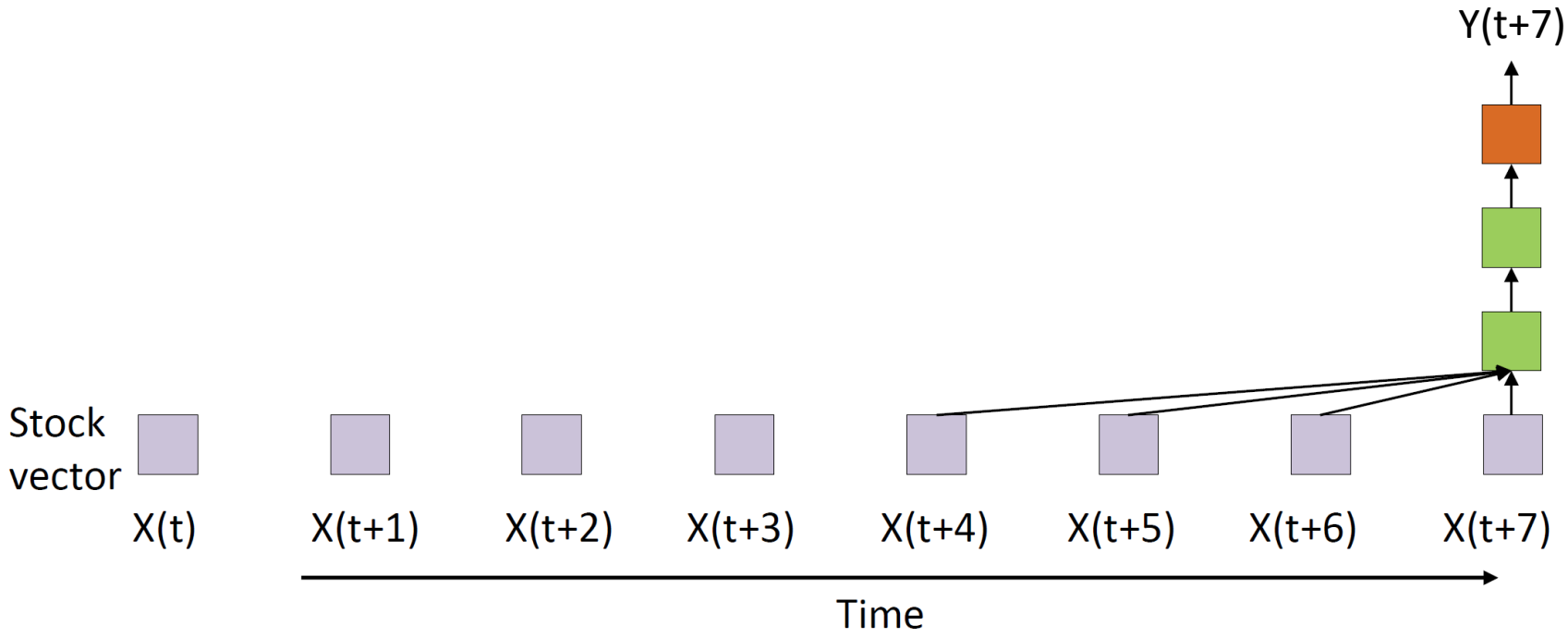
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

Stock Predictor



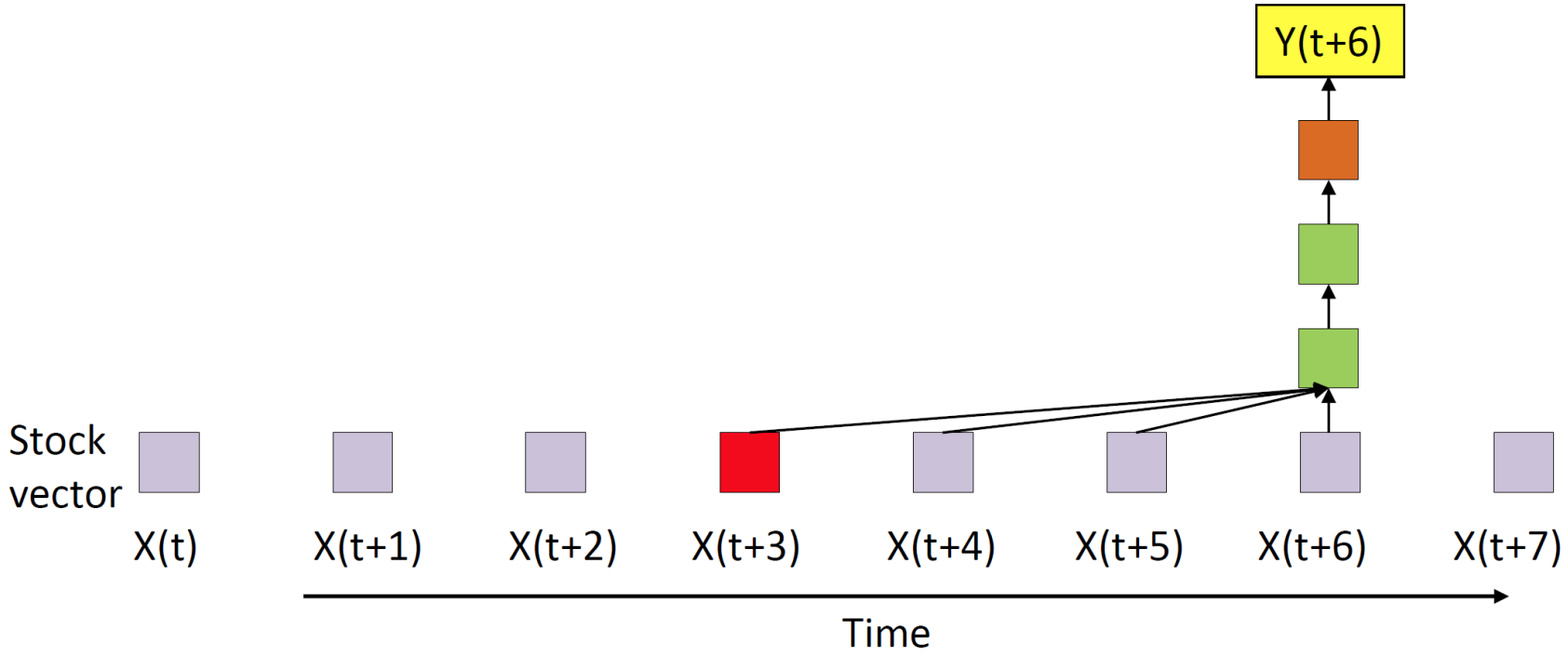
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

Stock Predictor



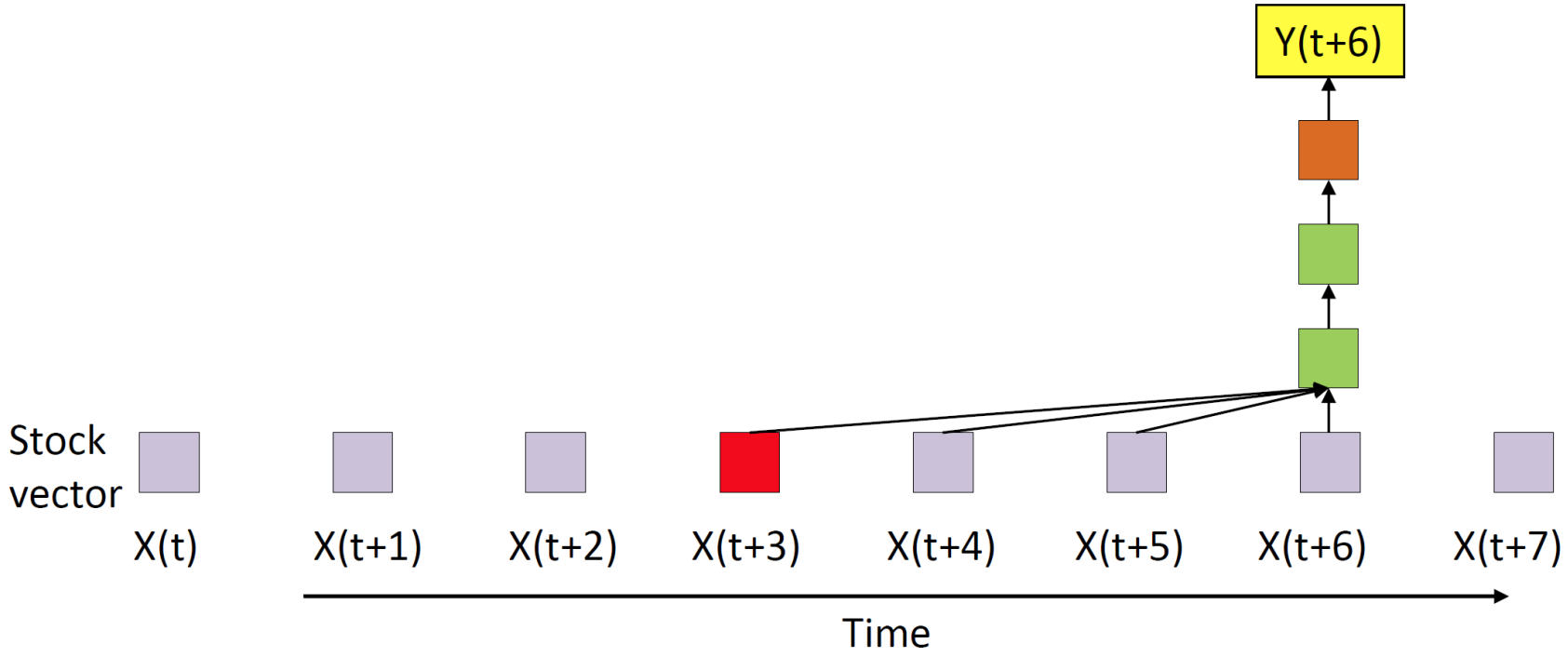
- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to sequential data

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

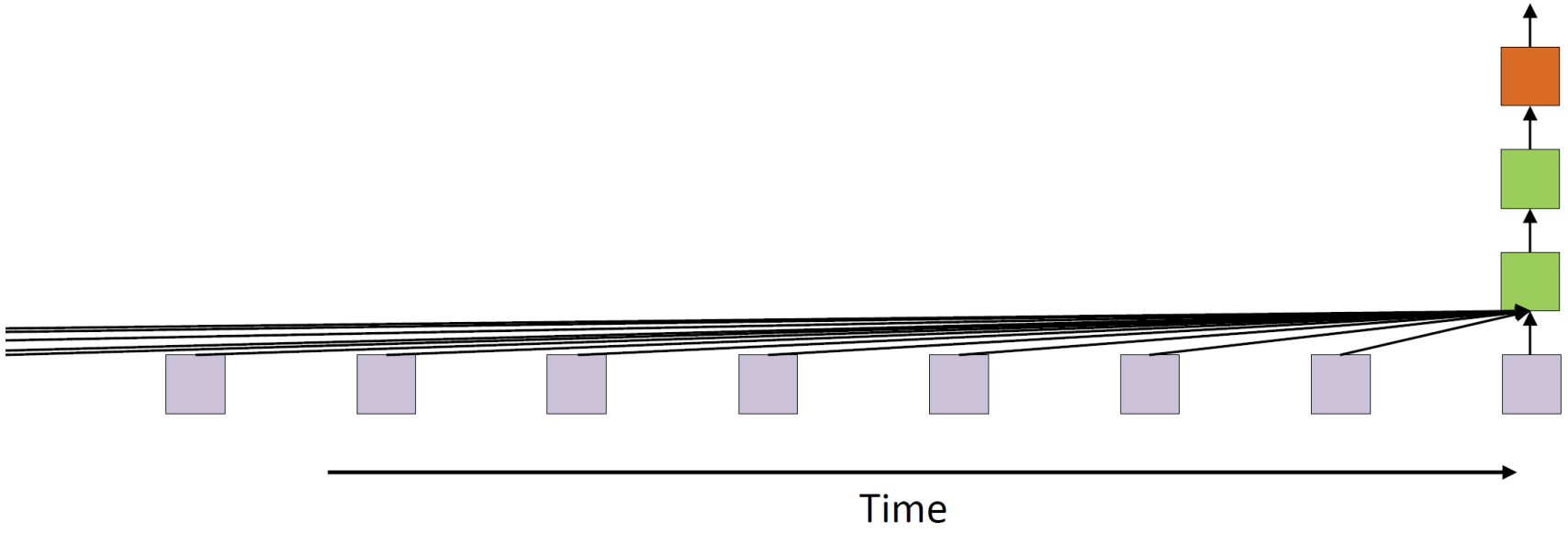


- Problem: Increasing the “history” makes the network more complex
- No worries, we have the CPU and memory

Systems often Have Long-term Dependencies

- Longer-term trends
  - Weekly trends in the market
  - Monthly trends in the market
  - Annual trends
  - Though longer history tends to affect us less than more recent events..

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

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

An Example of Infinite Response Systems

$$Y_t = f(X_t, Y_{t-1})$$

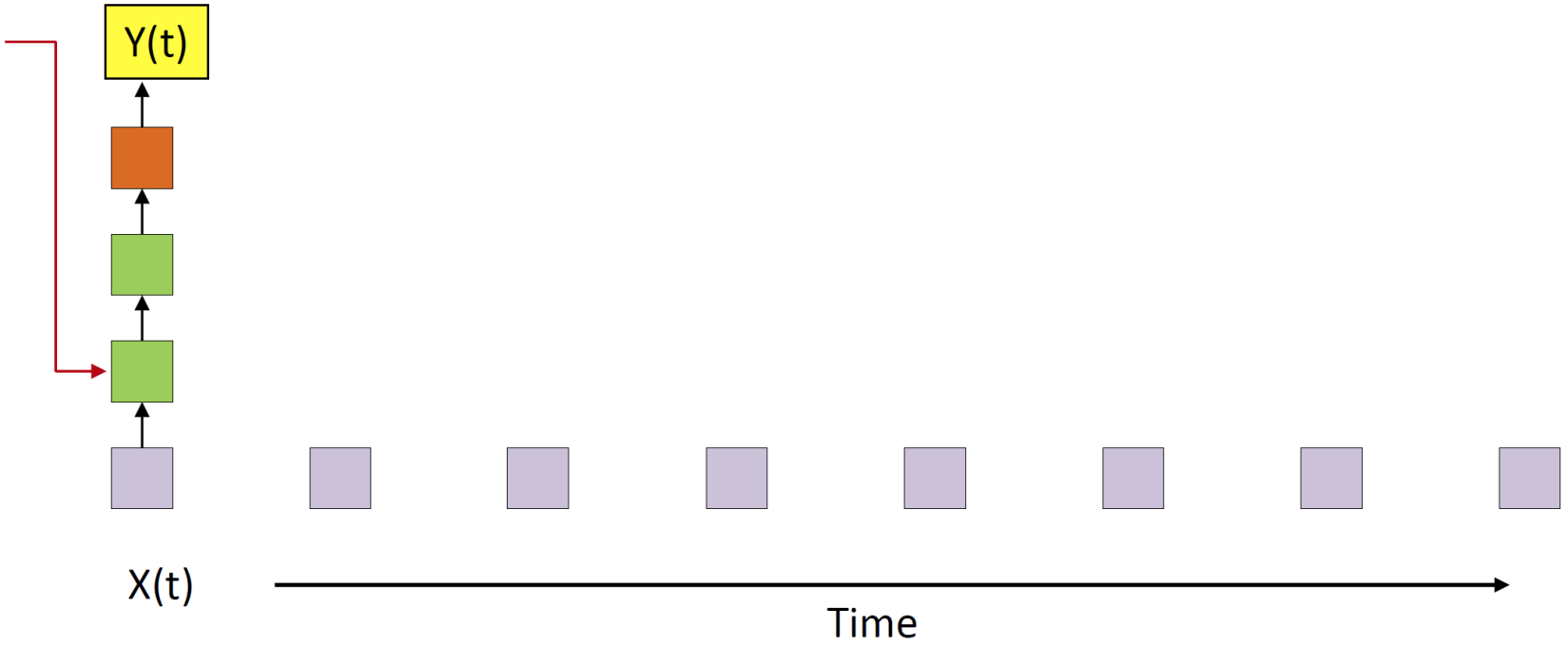
- Required: Define initial output: Y_{t-1} for $t = 0$
- An input at X_0 at $t = 0$ produces Y_0
- Y_0 produces Y_1 which produces Y_2 and so on until Y_∞ even if $X_1, \cdots, X_\infty$ are 0

- This is an instance of a NARX network
  - "nonlinear autoregressive network with exogenous inputs"

$$Y_t = f(X_{0:t}, Y_{0:t-1})$$

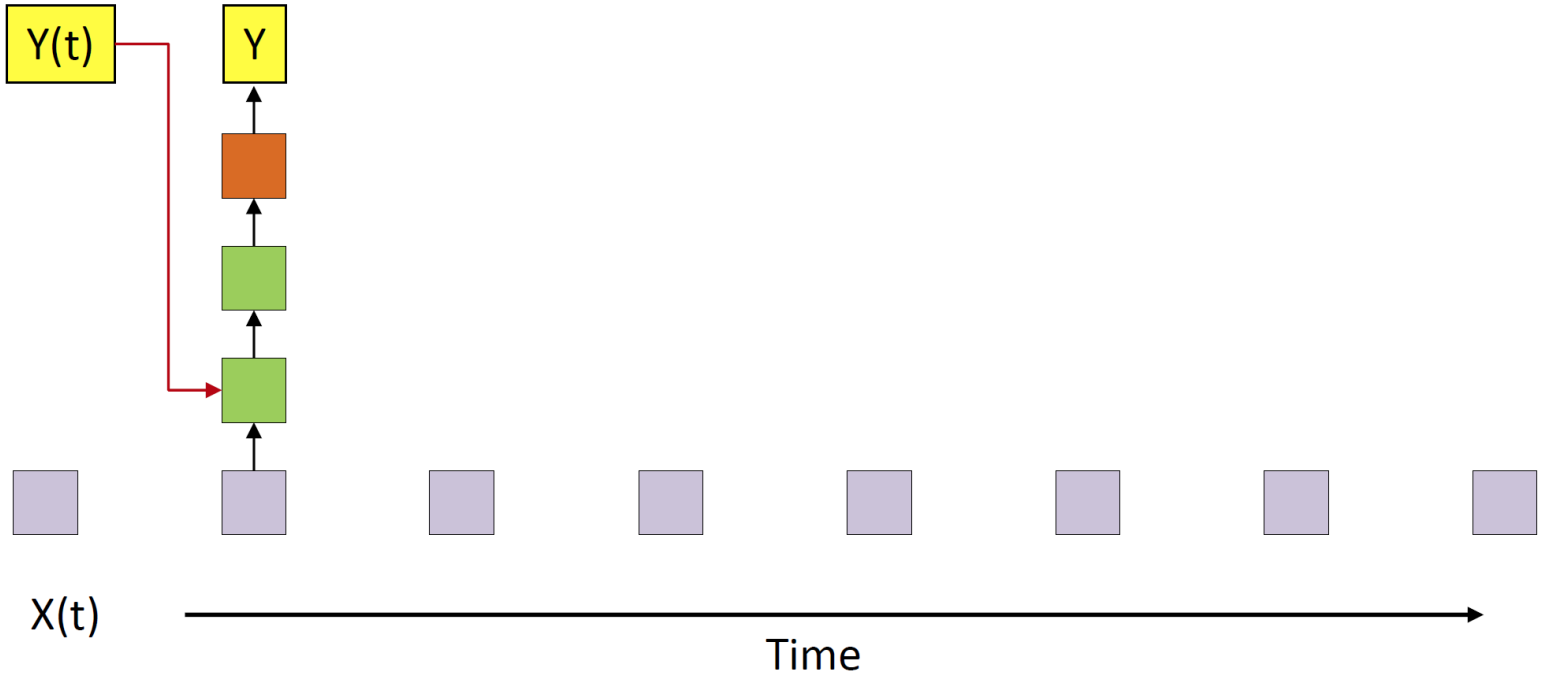
- Output contains information about the entire past

One-tap NARX Network



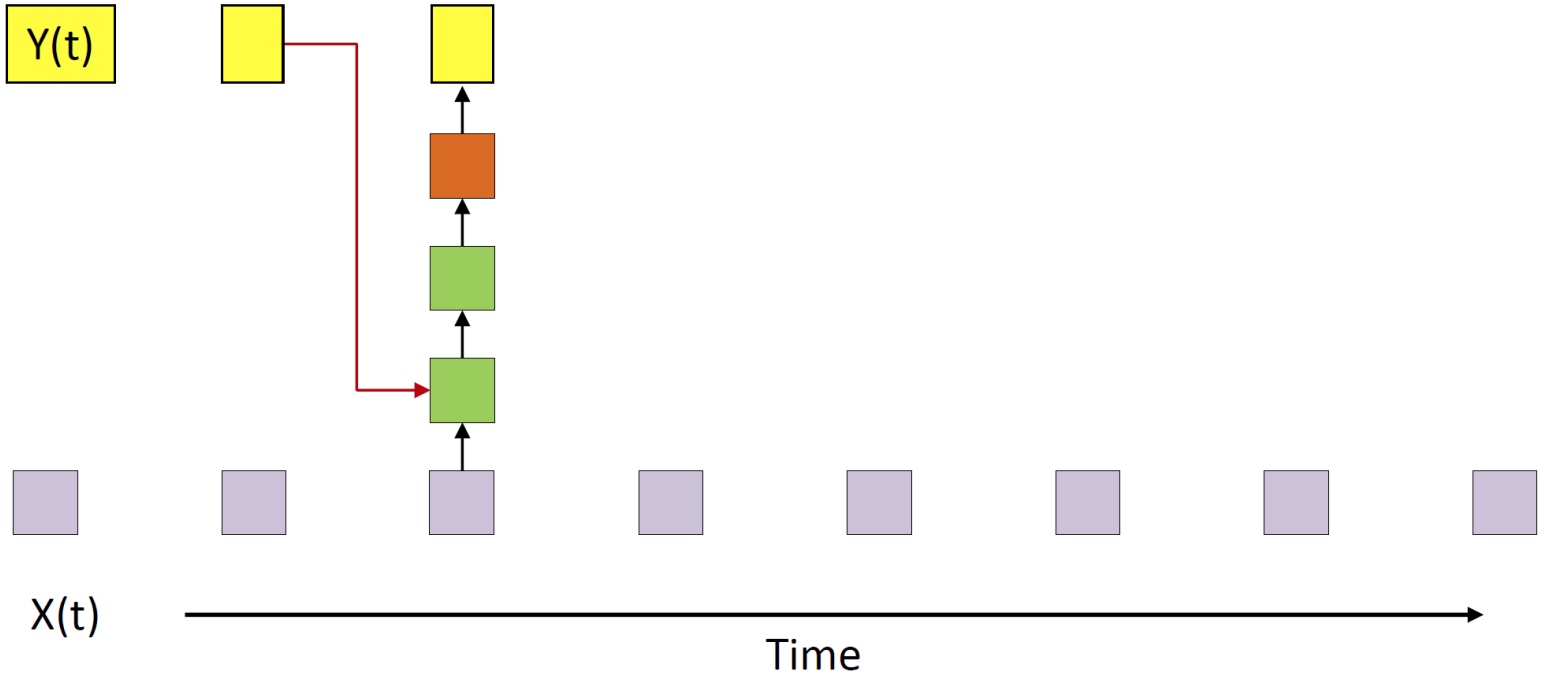
- A NARX net with recursion from the output

One-tap NARX Network



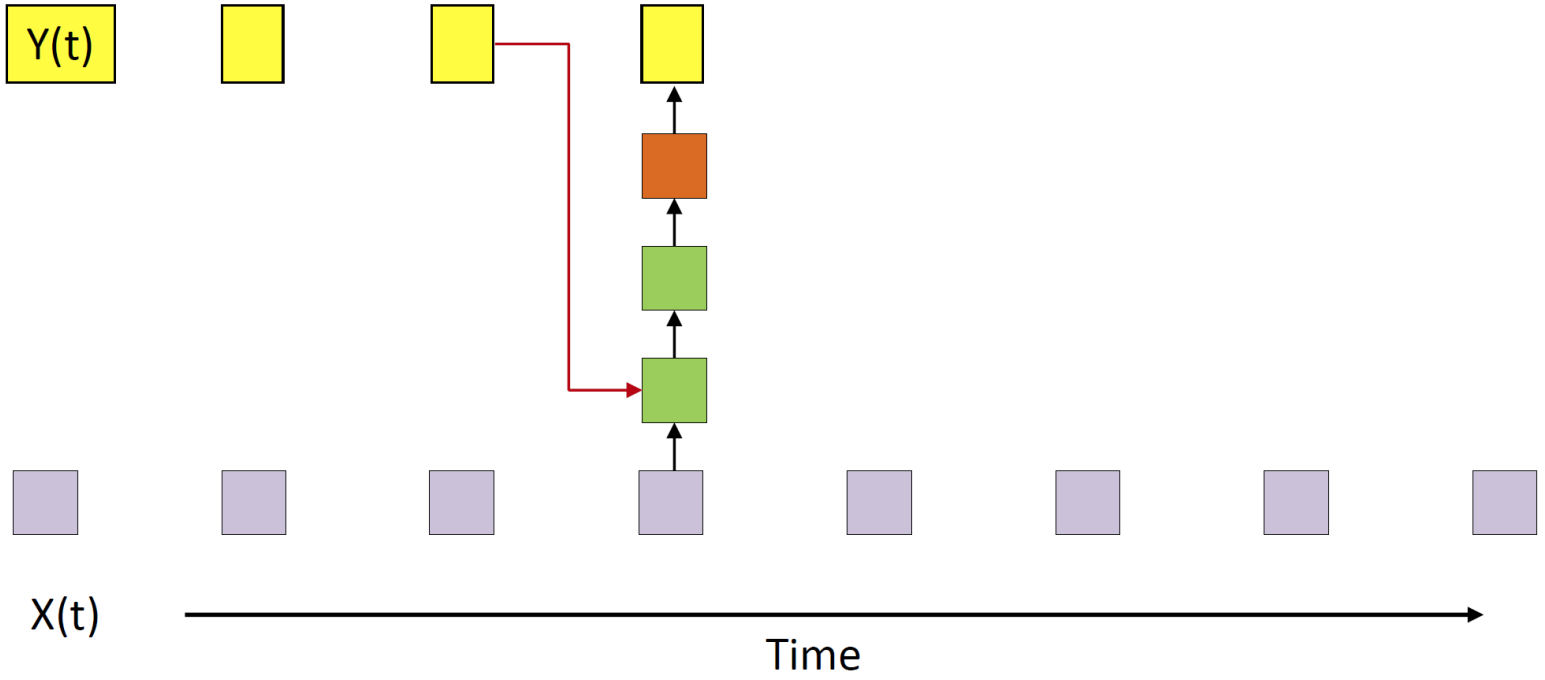
- A NARX net with recursion from the output

One-tap NARX Network



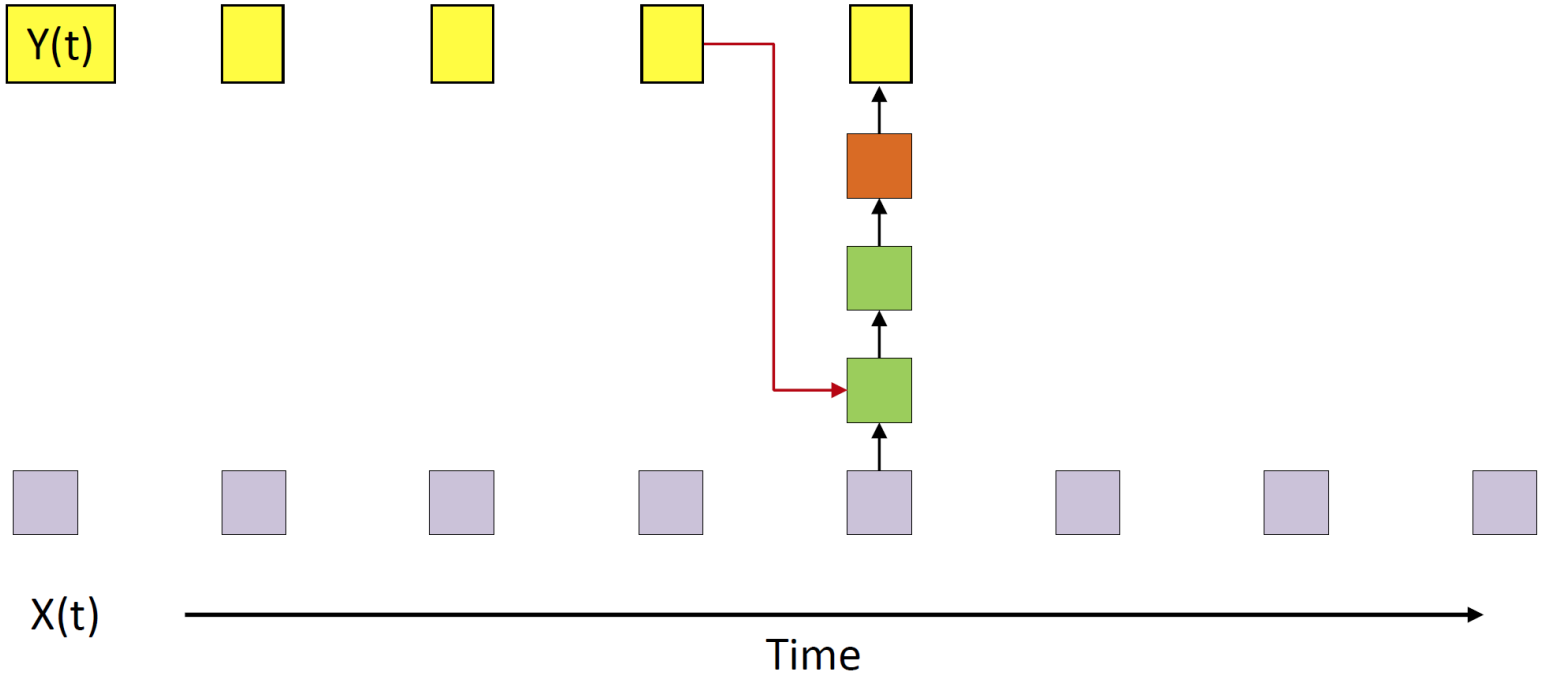
- A NARX net with recursion from the output

One-tap NARX Network



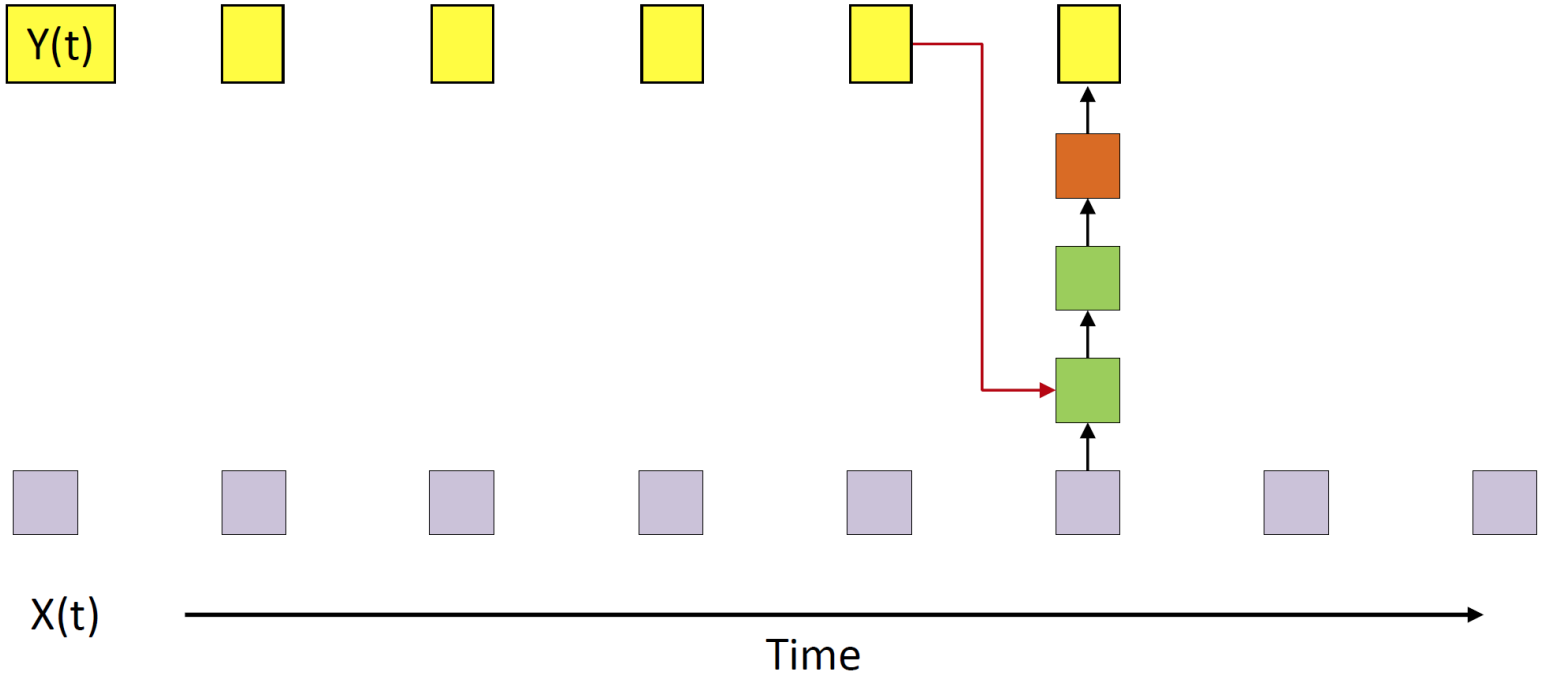
- A NARX net with recursion from the output

One-tap NARX Network



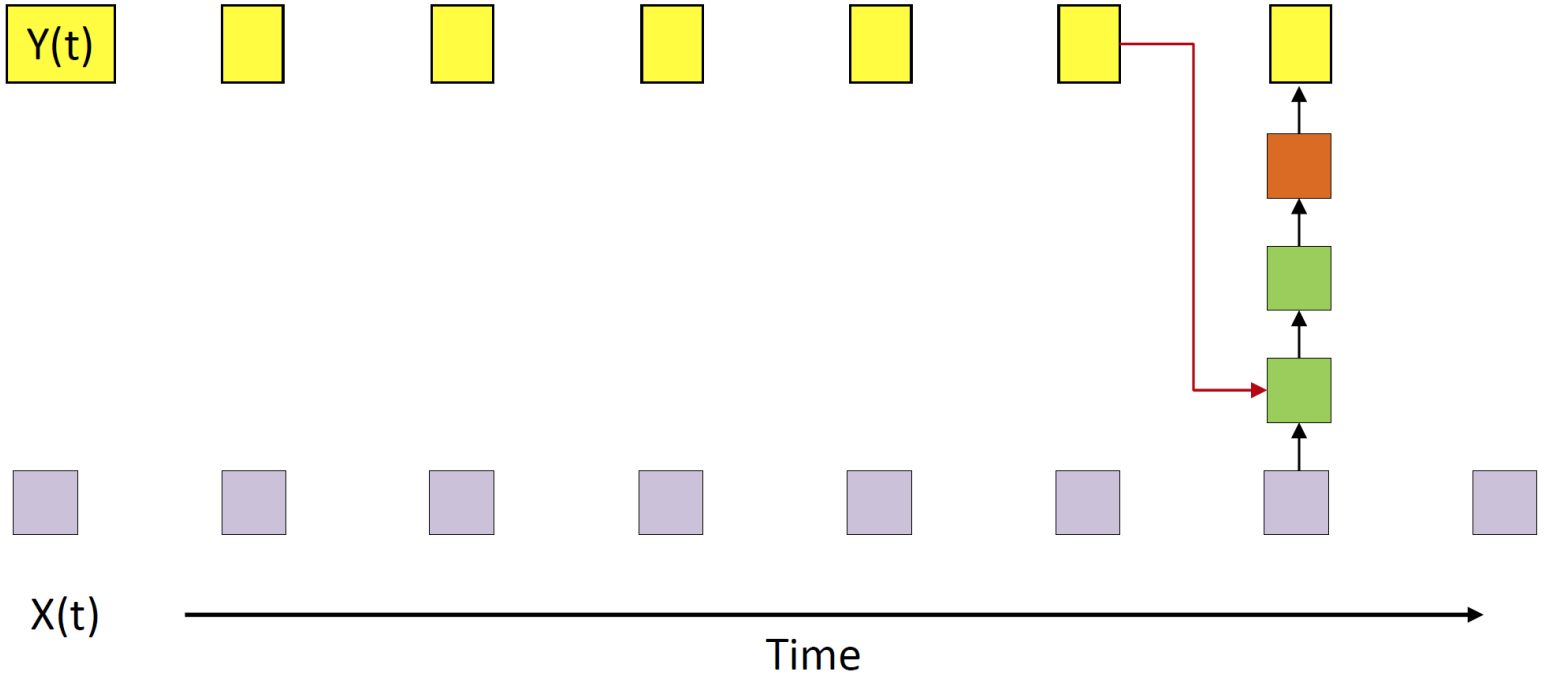
- A NARX net with recursion from the output

One-tap NARX Network



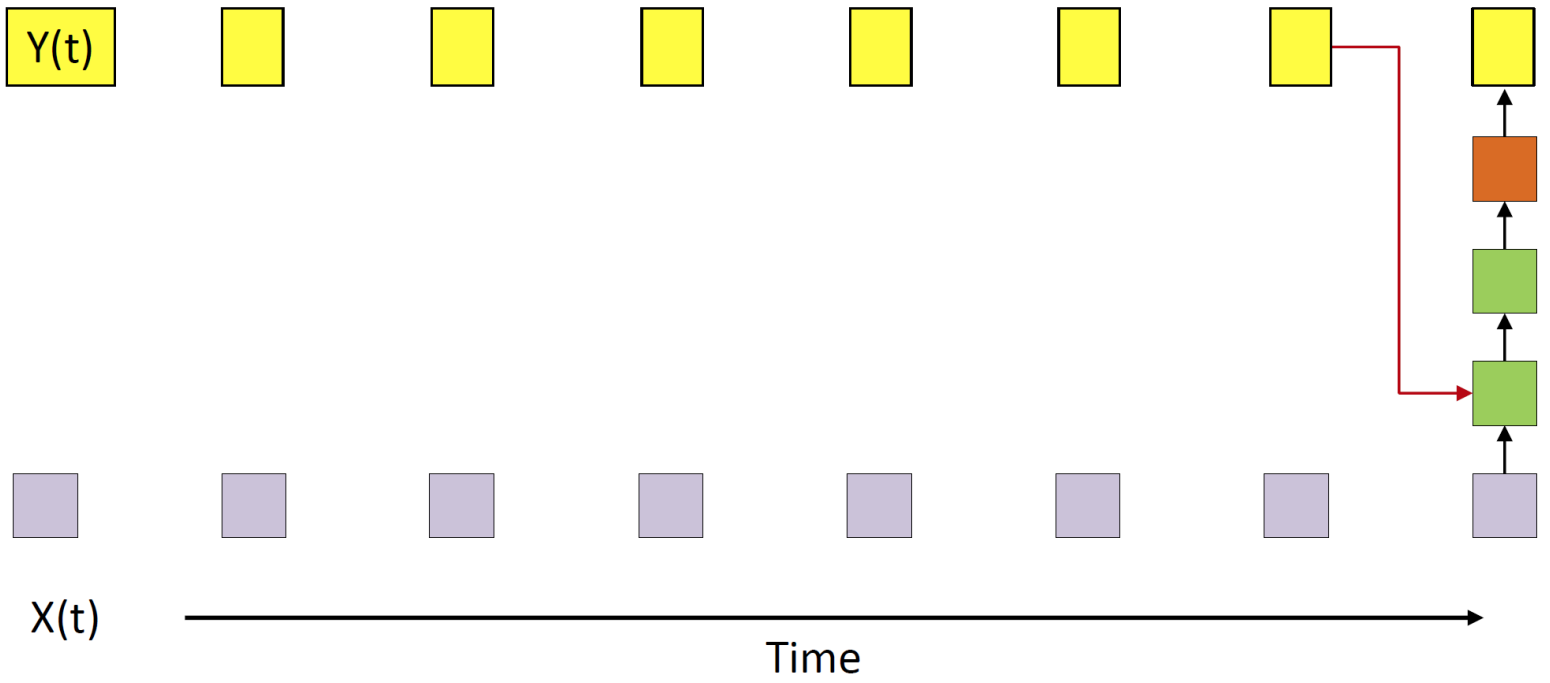
- A NARX net with recursion from the output

One-tap NARX Network



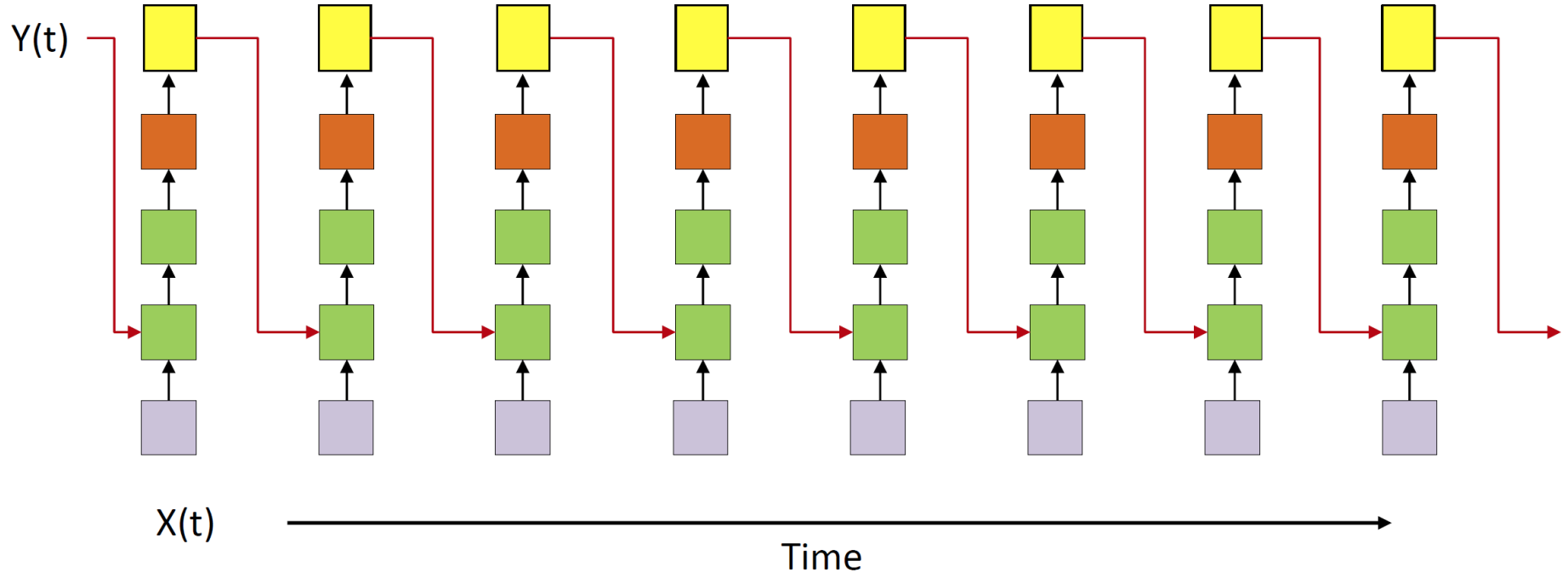
- A NARX net with recursion from the output

One-tap NARX Network



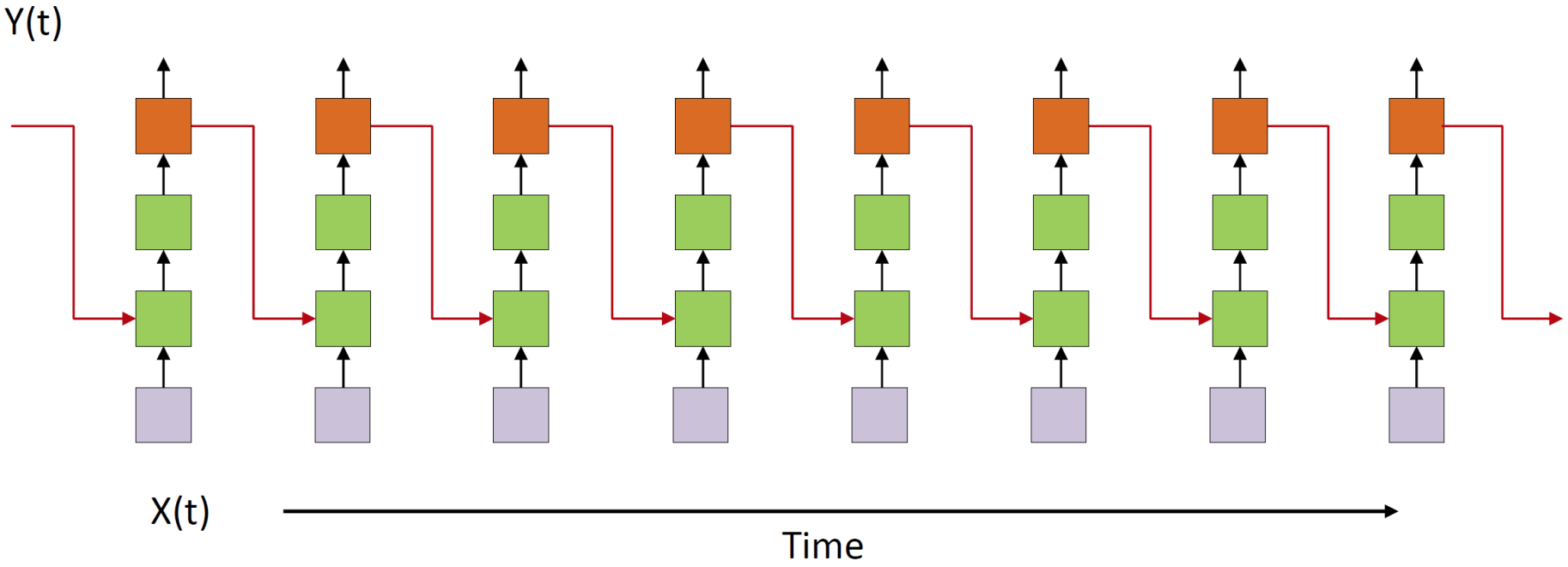
- A NARX net with recursion from the output

More Complete Representation



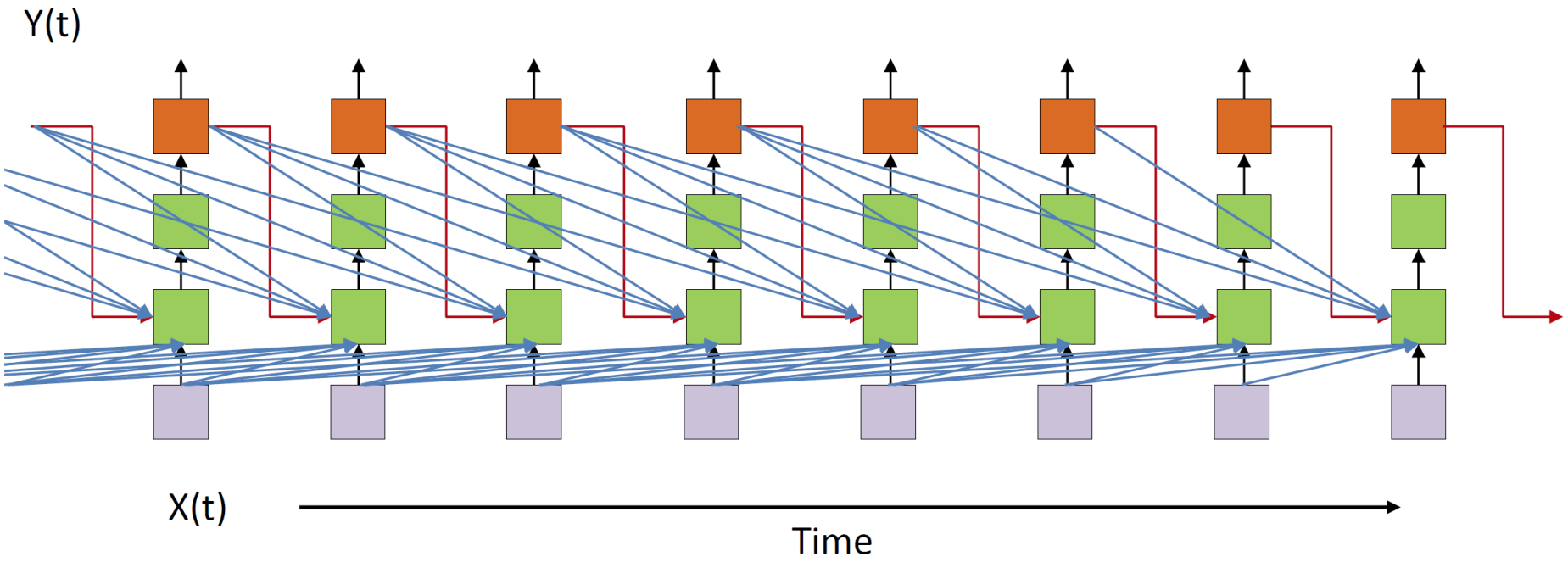
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- An input at $t = 0$ affects outputs forever

Same Figure Redrawn



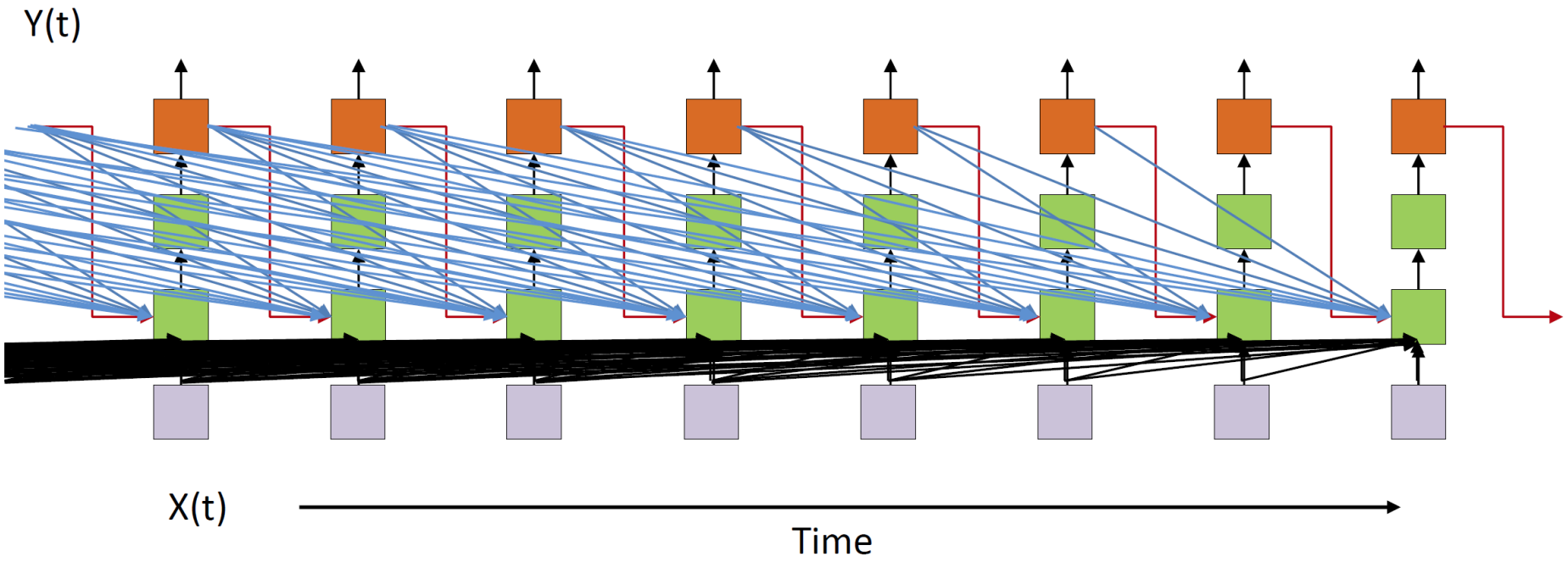
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- An input at $t = 0$ affects outputs forever

A More Generic NARX Network



- The output Y_t at time t is computed from the past K outputs $Y_{t-1}, \cdots Y_{t-K}$ and the current and past L inputs $X_t, \cdots X_{t-K}$

A “Complete” NARX Network



- The output Y_t at time t is computed from all the past outputs and all the inputs until time t
  - Not really a practical model

NARX Networks

- Very popular for time-series prediction
  - Weather
  - Stock markets
  - As alternate system models in tracking systems

- Any phenomena with distinct “innovations” that “drive” an output

An Alternate Model for Infinite Response Systems

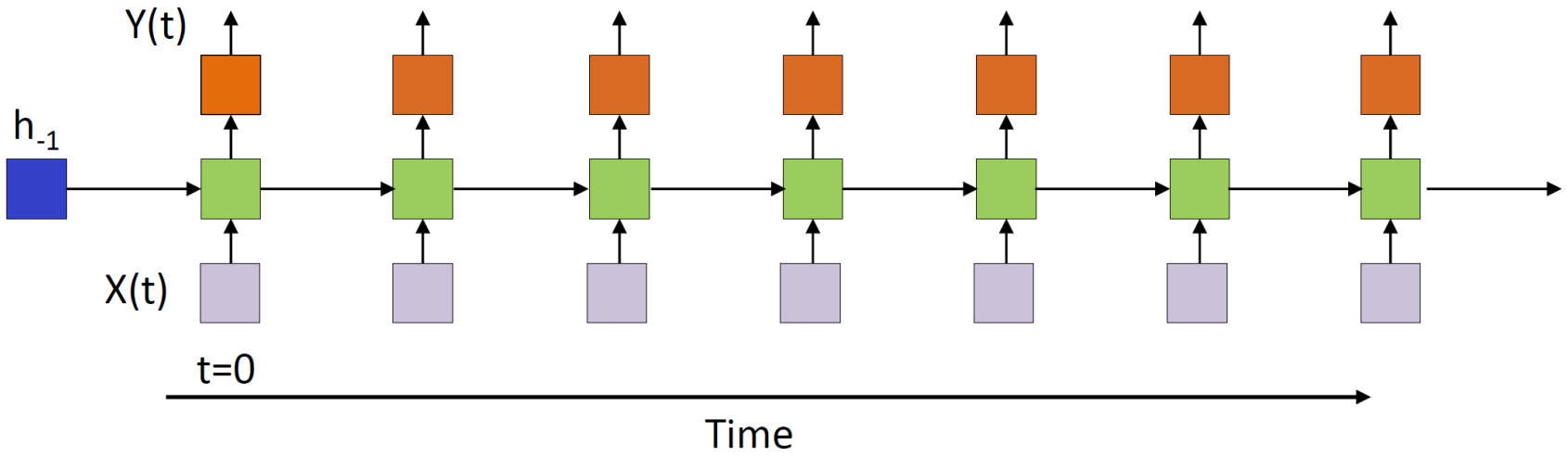
- the state-space model

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

- h_t is the state of the network
- Need to define initial state h_{-1}

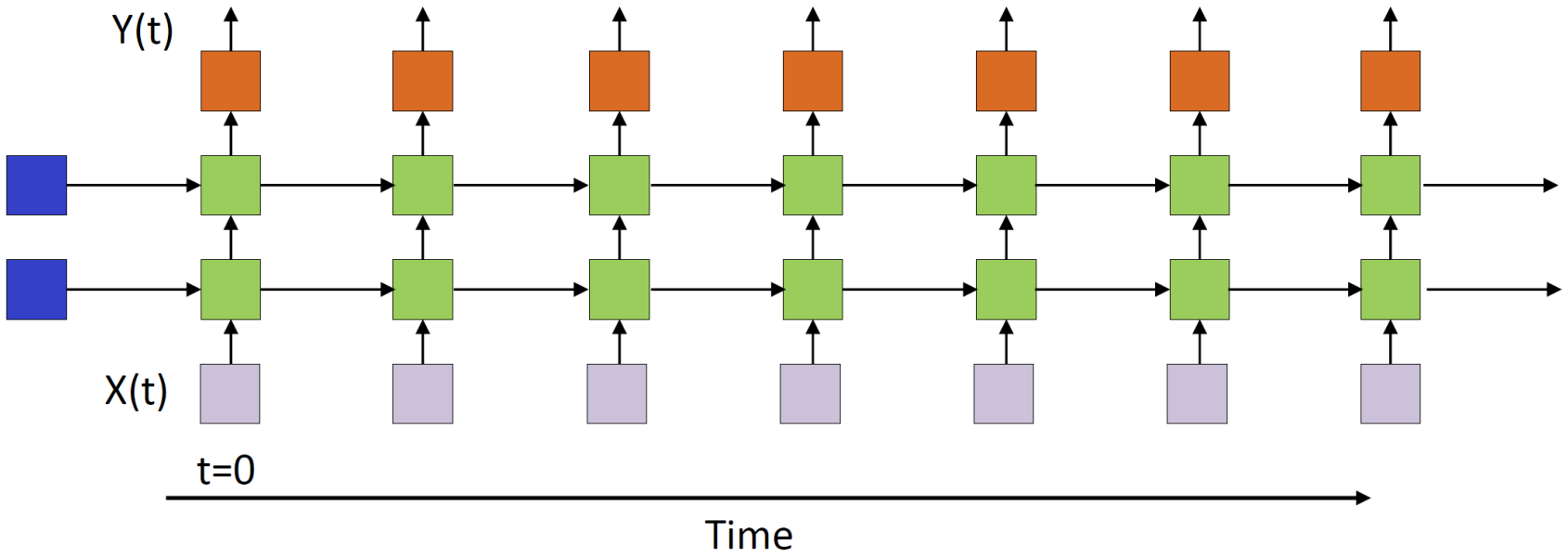
- This is a recurrent neural network
- State summarizes information about the entire past

Single Hidden Layer RNN (Simplest State-Space Model)



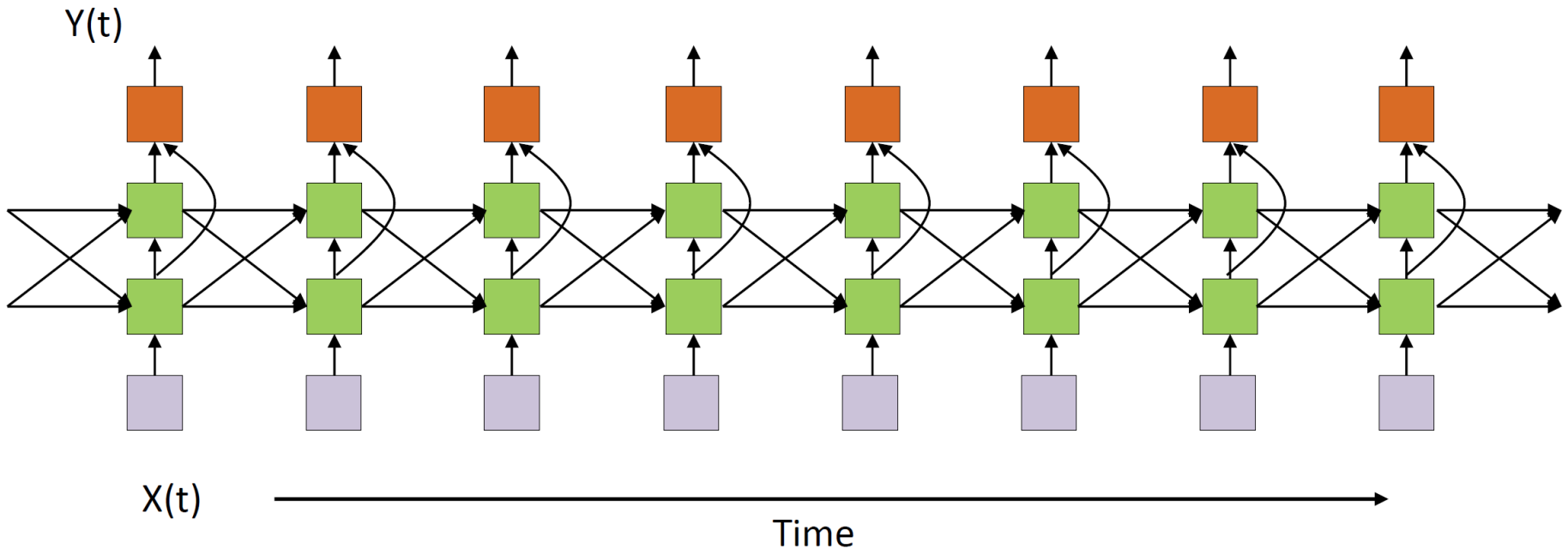
- The state (green) at any time is determined by the input at that time, and the state at the previous time
- All columns are identical
- An input at $t = 0$ affects outputs forever
- Also known as a recurrent neural net

Multiple Recurrent Layer RNN



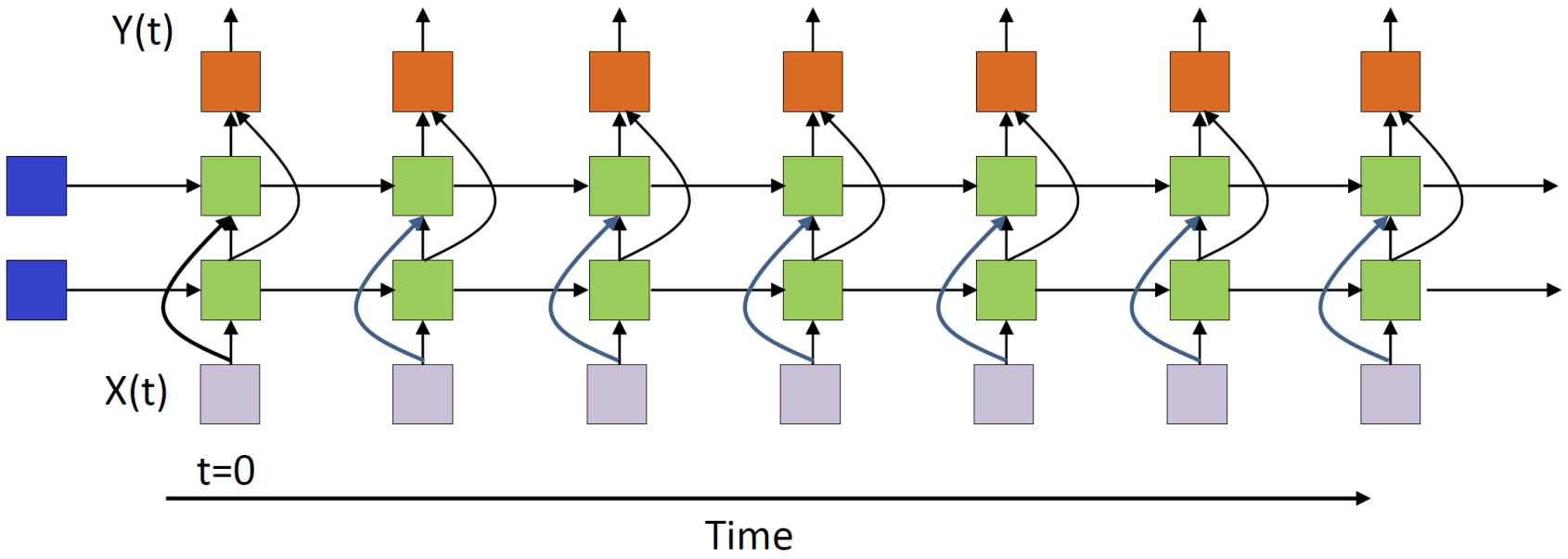
- The state (green) at any time is determined by the input at that time, and the state at the previous time
- All columns are identical
- An input at $t = 0$ affects outputs forever
- Also known as a recurrent neural net

A More Complex State



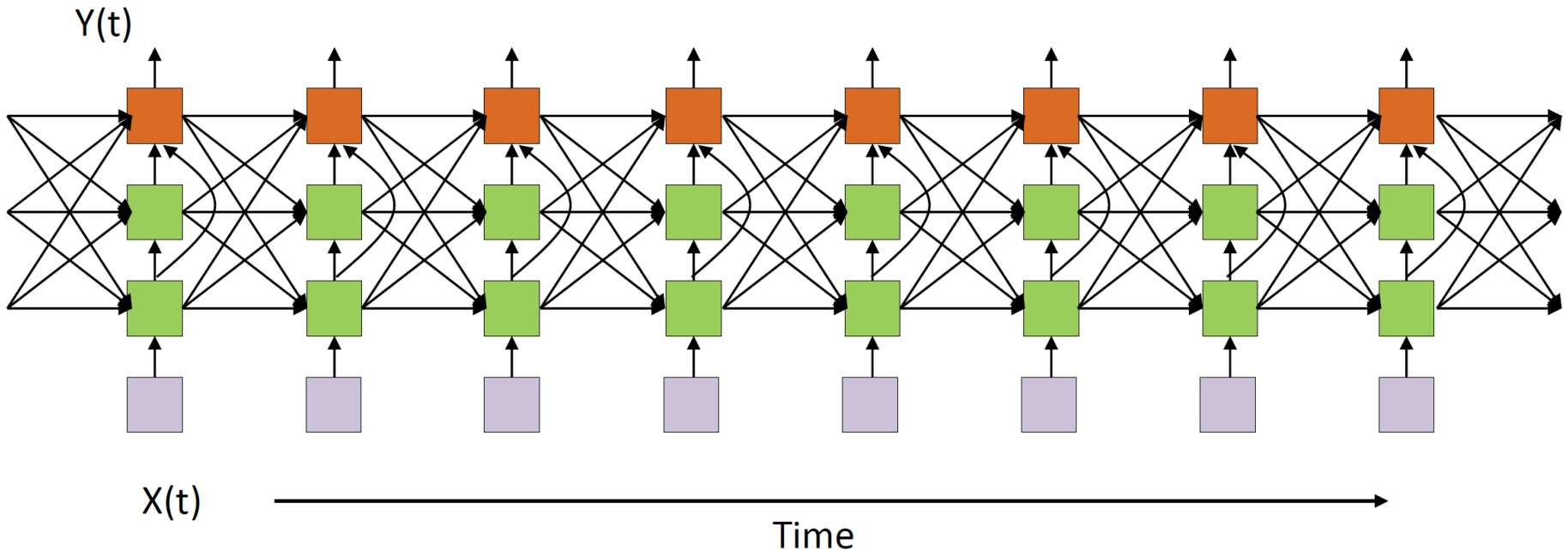
- All columns are identical
- An input at $t = 0$ affects outputs forever

A More Complex State



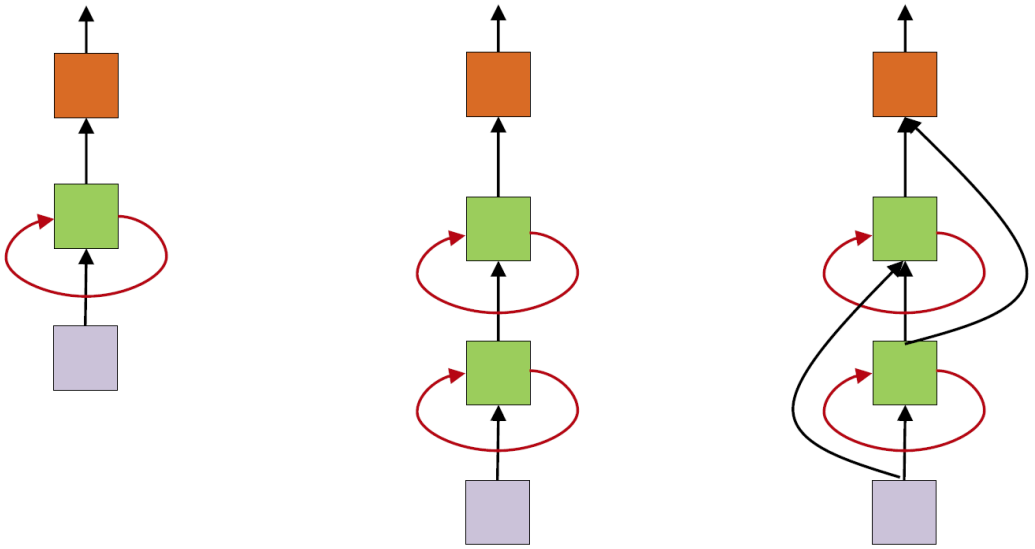
- All columns are identical
- An input at $t = 0$ affects outputs forever

Even More Complicated Network



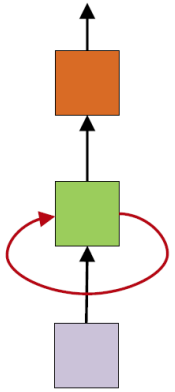
- All columns are identical
- An input at $t = 0$ affects outputs forever

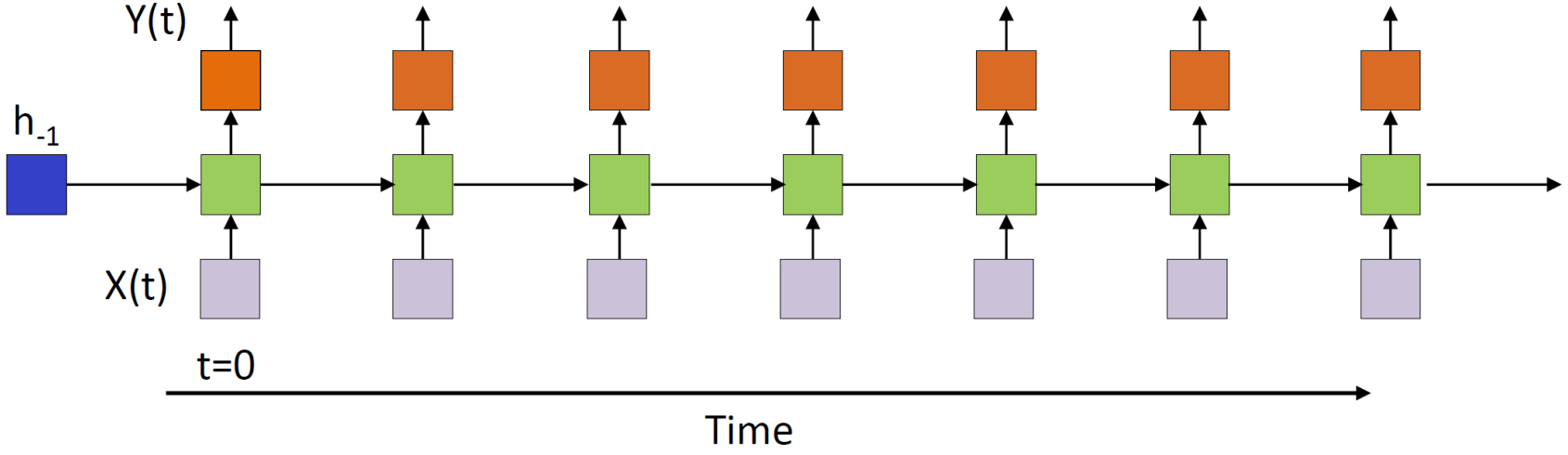
Recurrent Neural Network



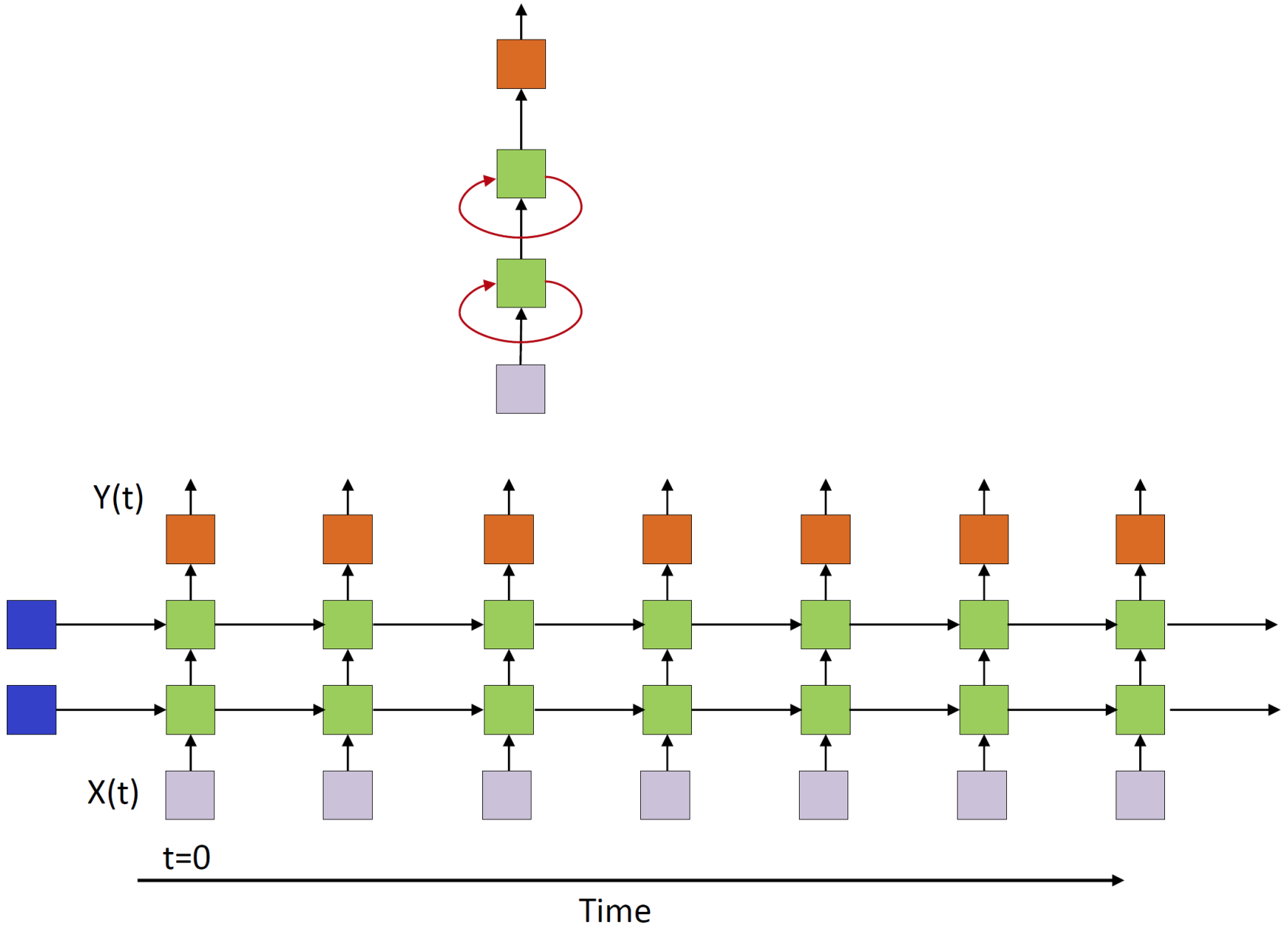
- Simplified models often drawn
- The loops imply recurrence

The Detailed Version of RNN

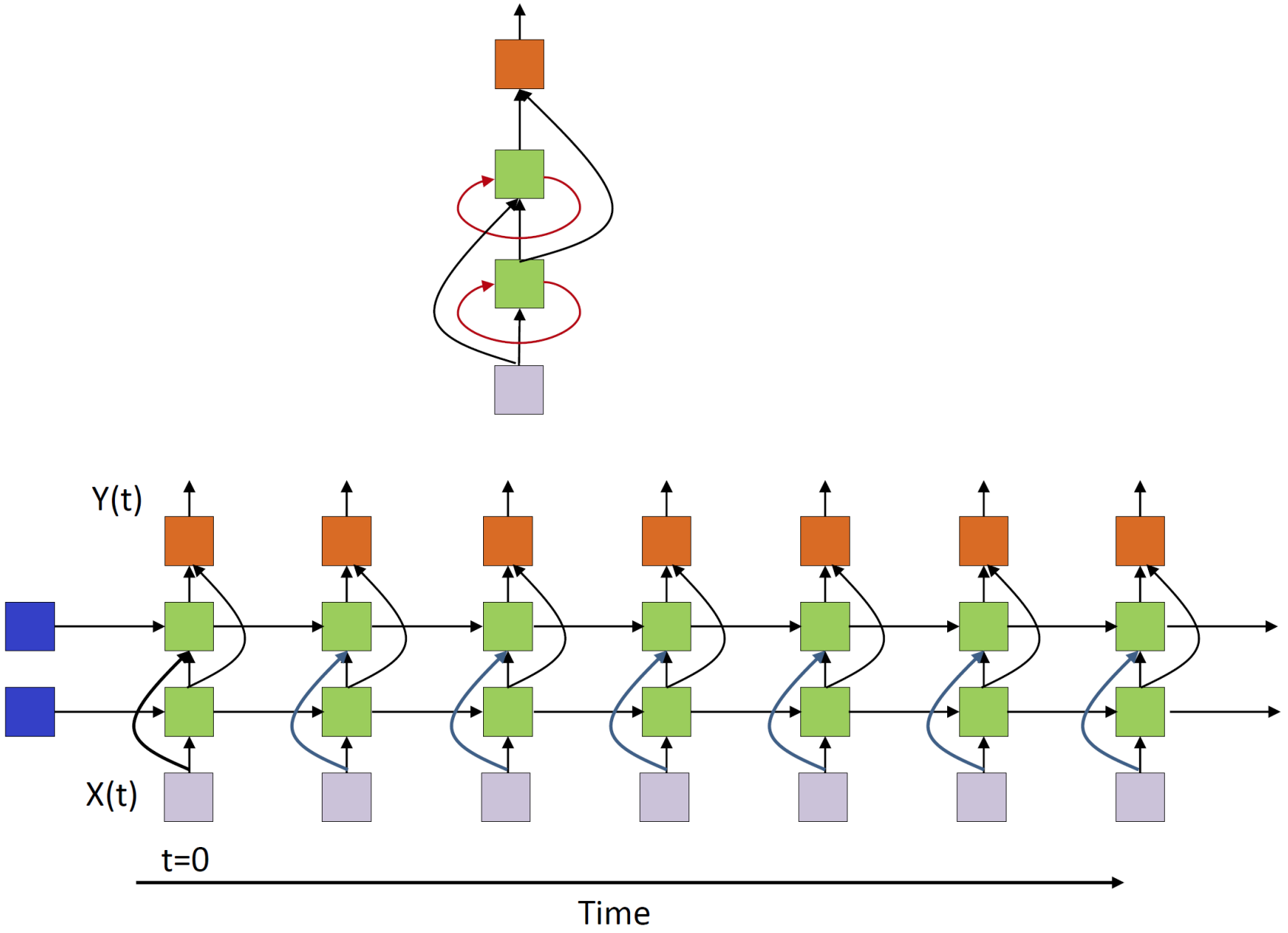




The Detailed Version of RNN

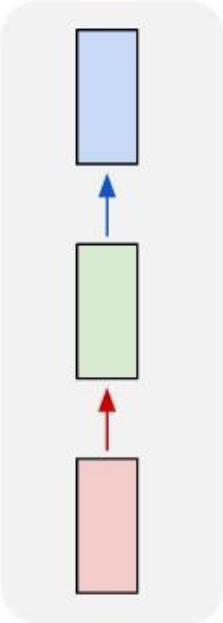


The Detailed Version of RNN



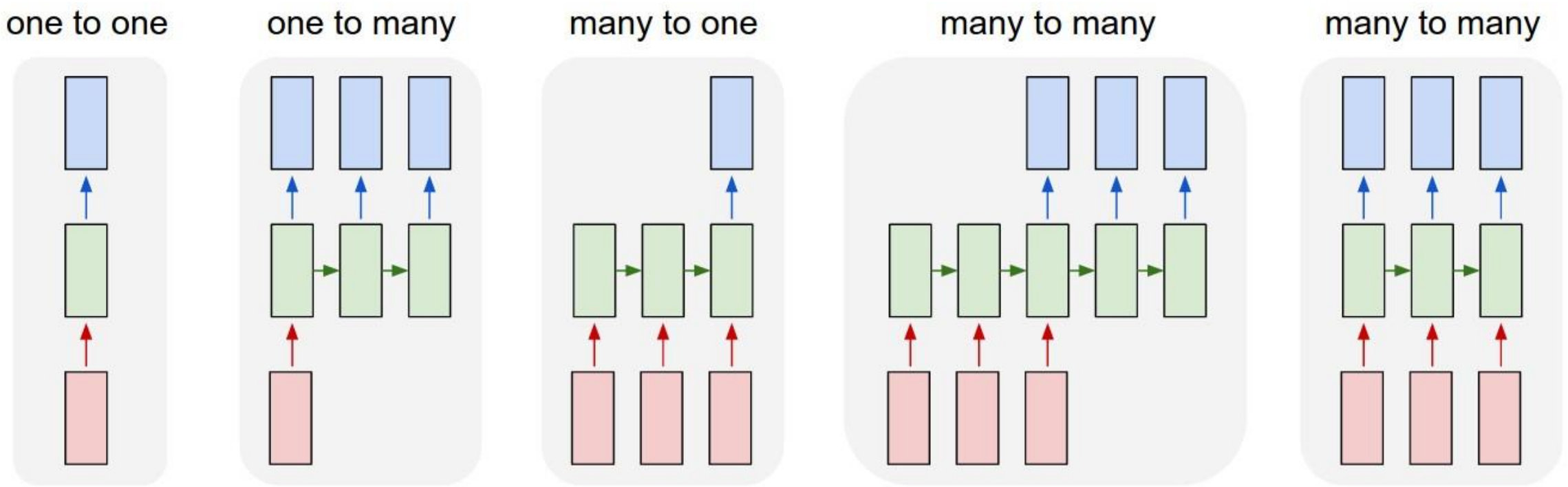
“Vanilla” Neural Network

one to one



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

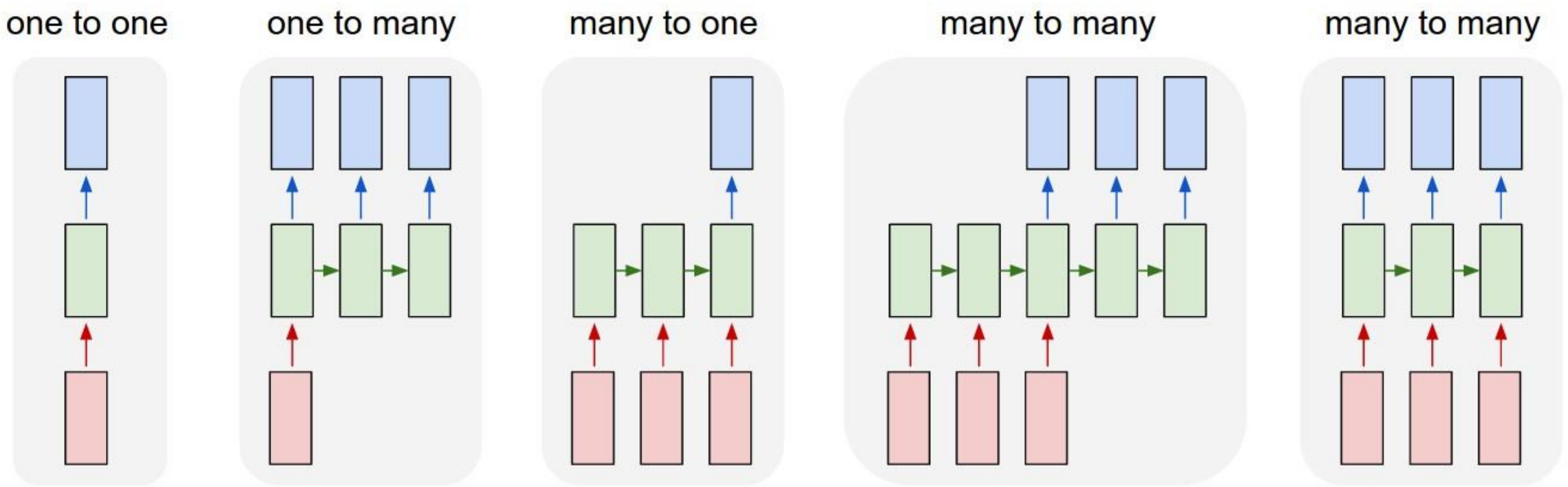
Recurrent Neural Network: Process Sequences



e.g. Image Captioning
image → sequence of words

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

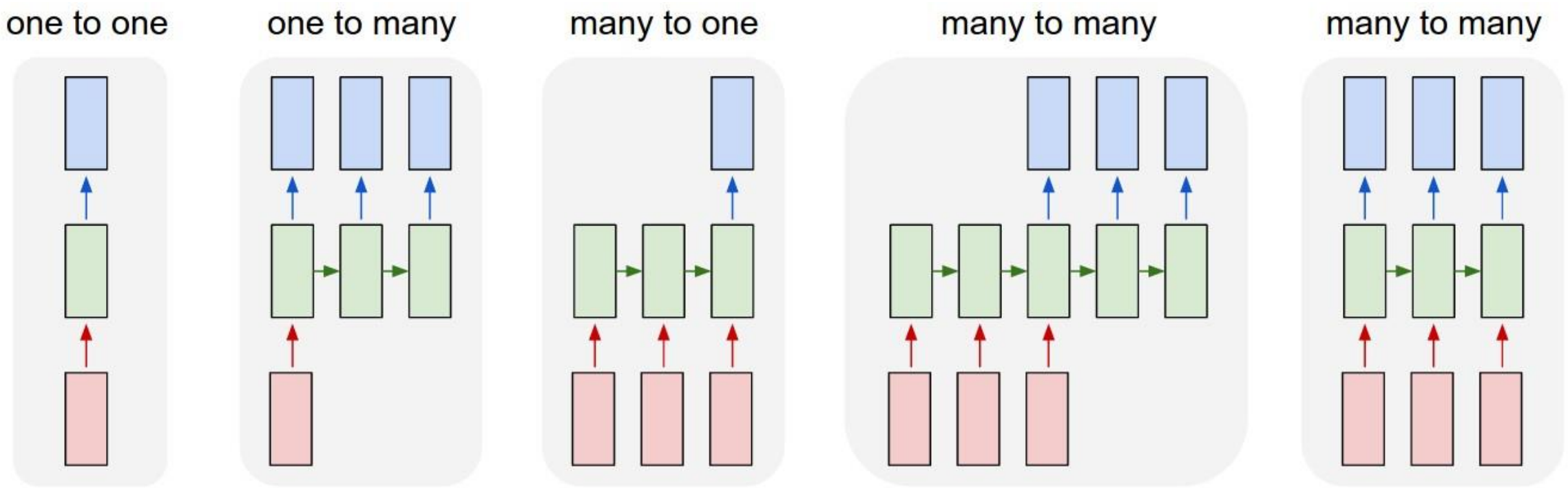
Recurrent Neural Network: Process Sequences



e.g. Sentiment Classification
sequence of words → sentiment

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

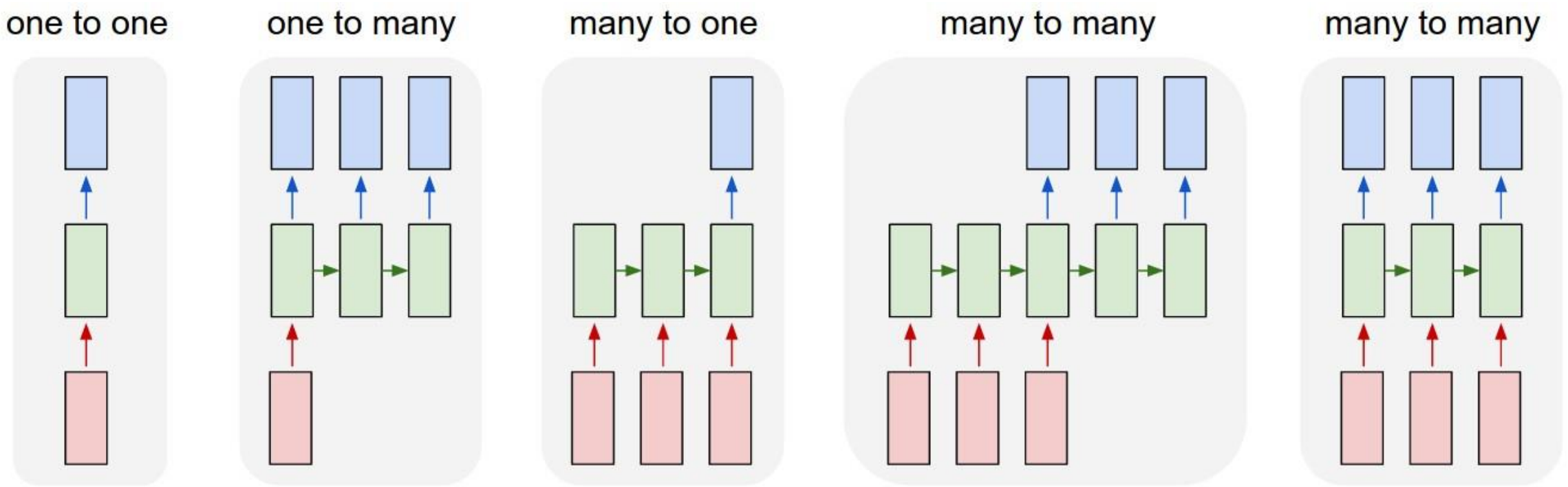
Recurrent Neural Network: Process Sequences



e.g. Machine Translation
Seq. of words → seq. of words

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

Recurrent Neural Network: Process Sequences



e.g. Video classification on frame level

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

How to Train the Network

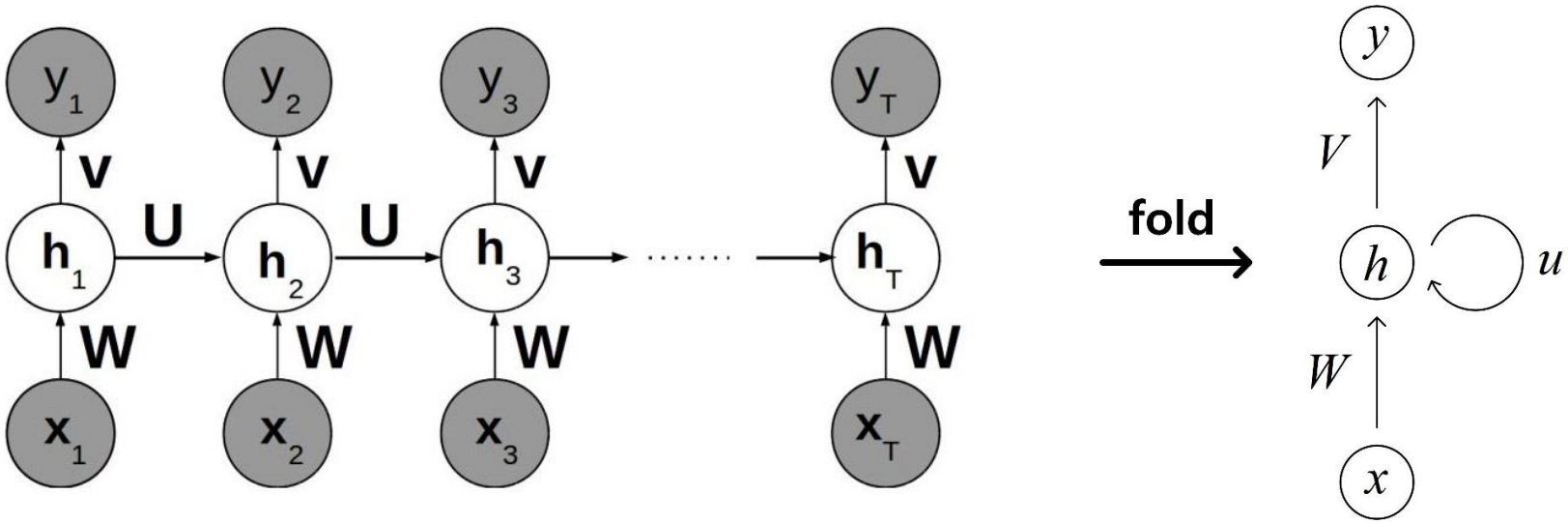
- Back propagation through time (BPTT)

RNN

- Excellent models for time-series analysis tasks
  - Time-series prediction
  - Time-series classification
  - Sequence prediction..
- They can even simplify some problems that are difficult for MLPs
- But first - a problem..

Structure of RNN

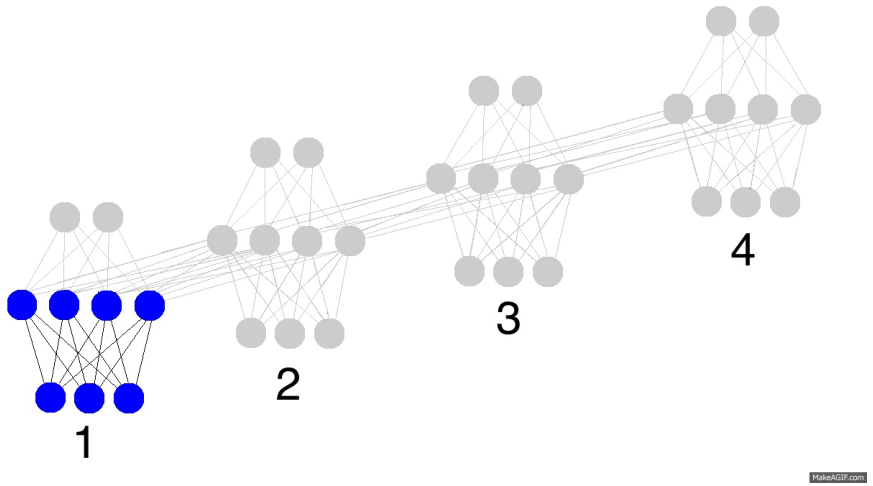
- Order matters
- Recurrence
  - It is possible to use the same transition function f with the same parameters at every time step



Hidden State

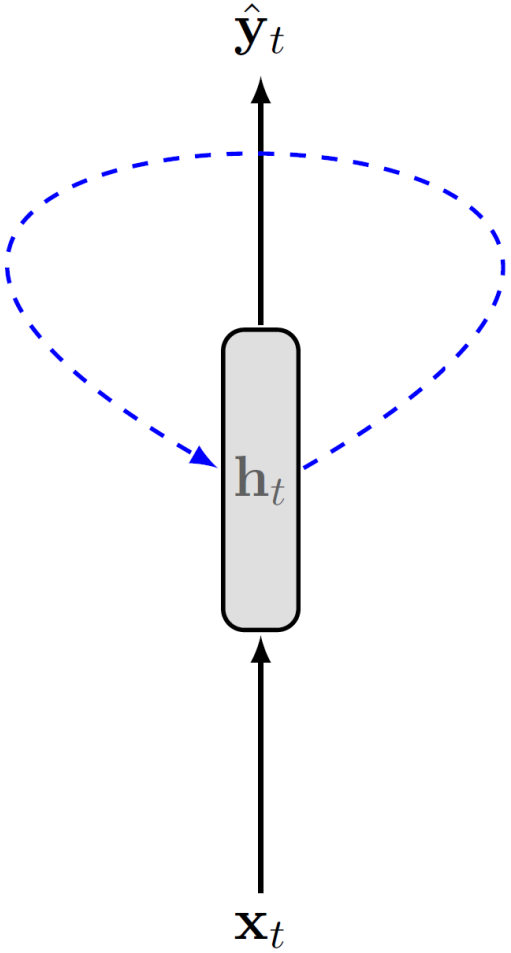
- Summary of the past sequence of inputs up to t
- Keep some aspects of the past sequence with more precision than other aspects
- Network learns the function f

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}\right)$$
$$f\left(h^{(t-1)}, x^{(t)}\right) = g\left(Wx_t + Uh_{t-1}\right)$$

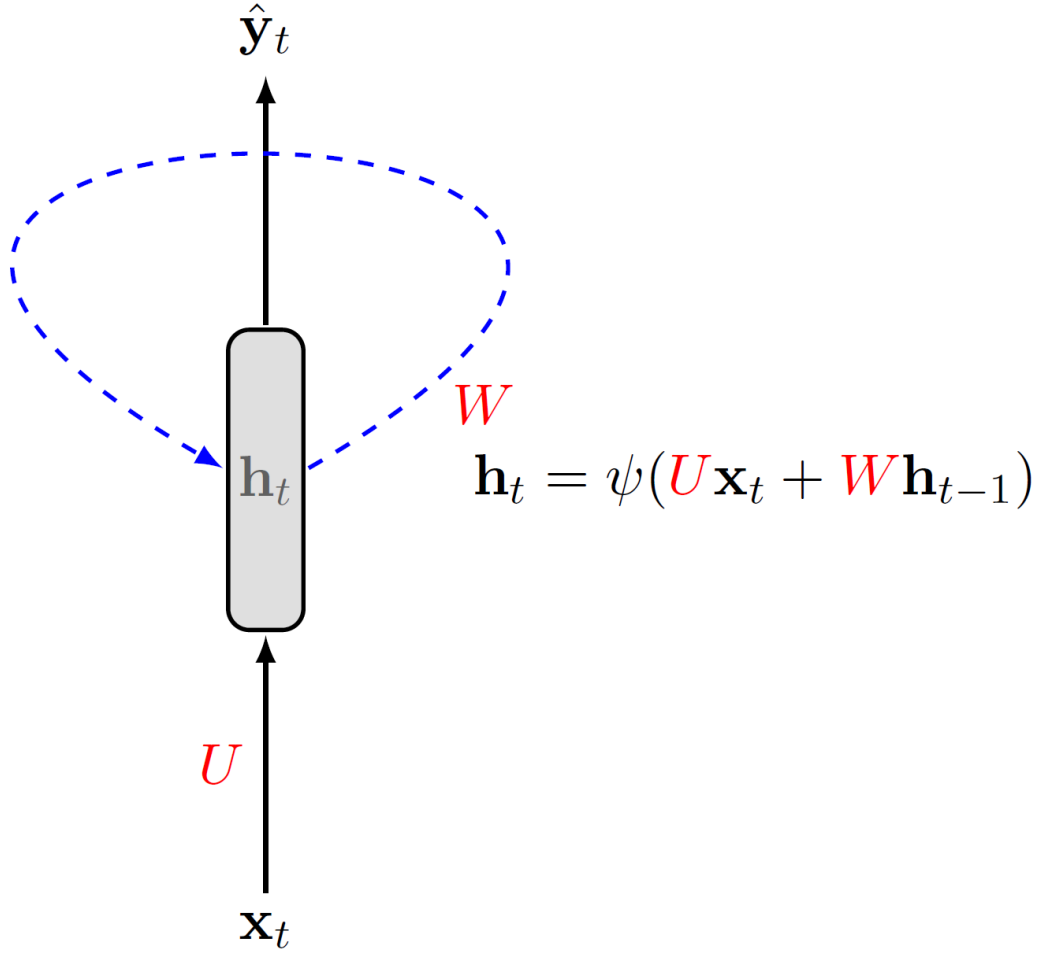


Design Patterns of Recurrent Networks

Vanilla Recurrent Network

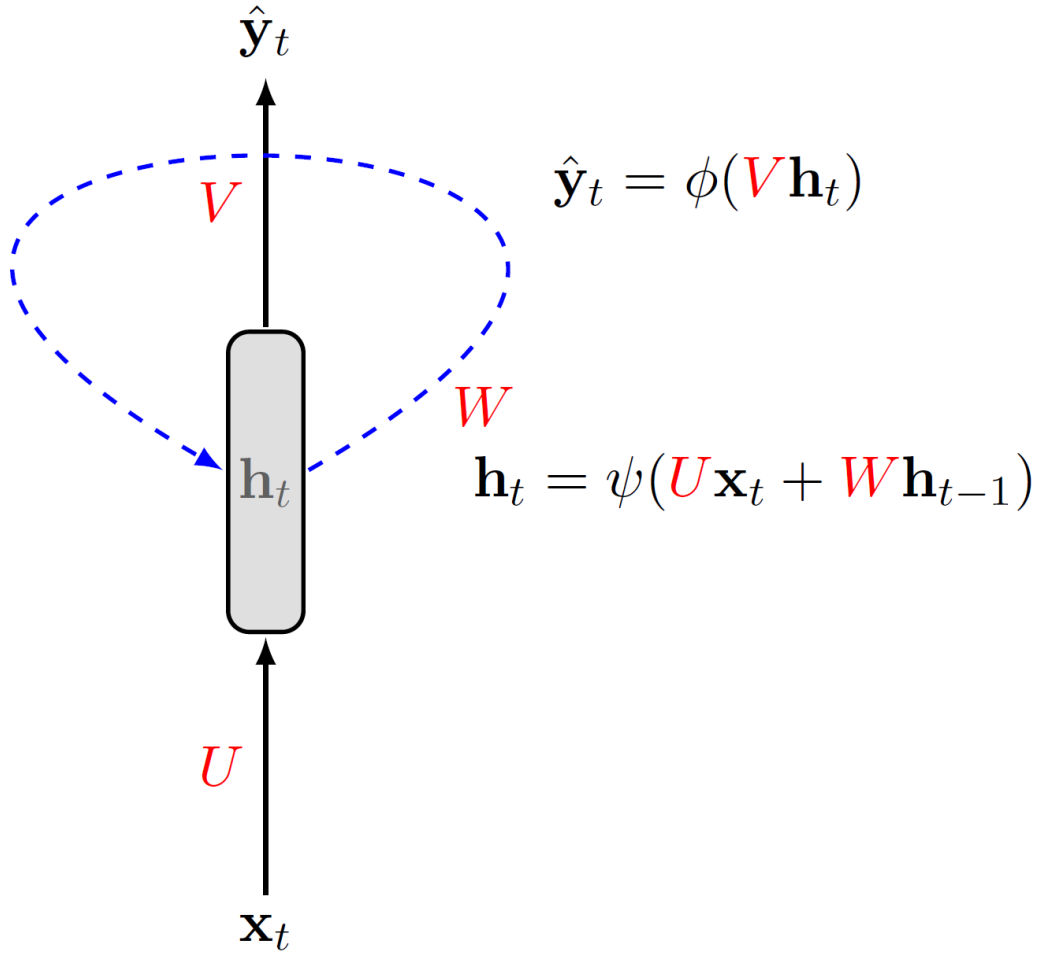


Recurrent Connections



Recurrent Connections

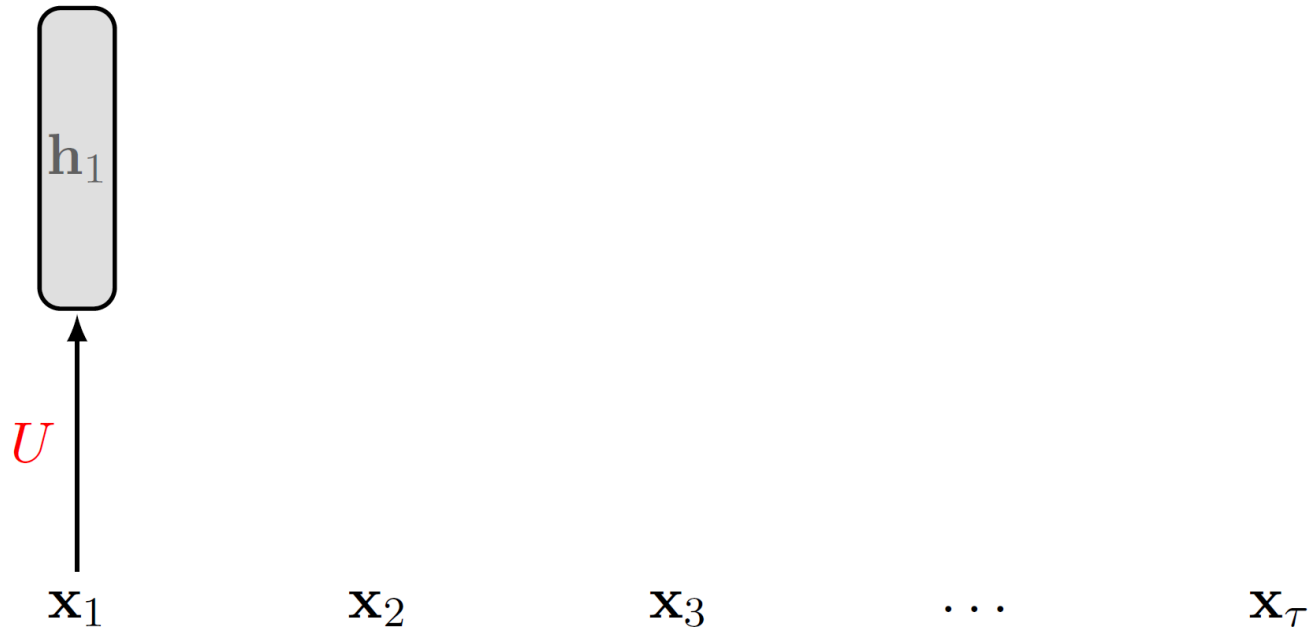
- For instance, ψ can be $tanh$ and ϕ can be softmax



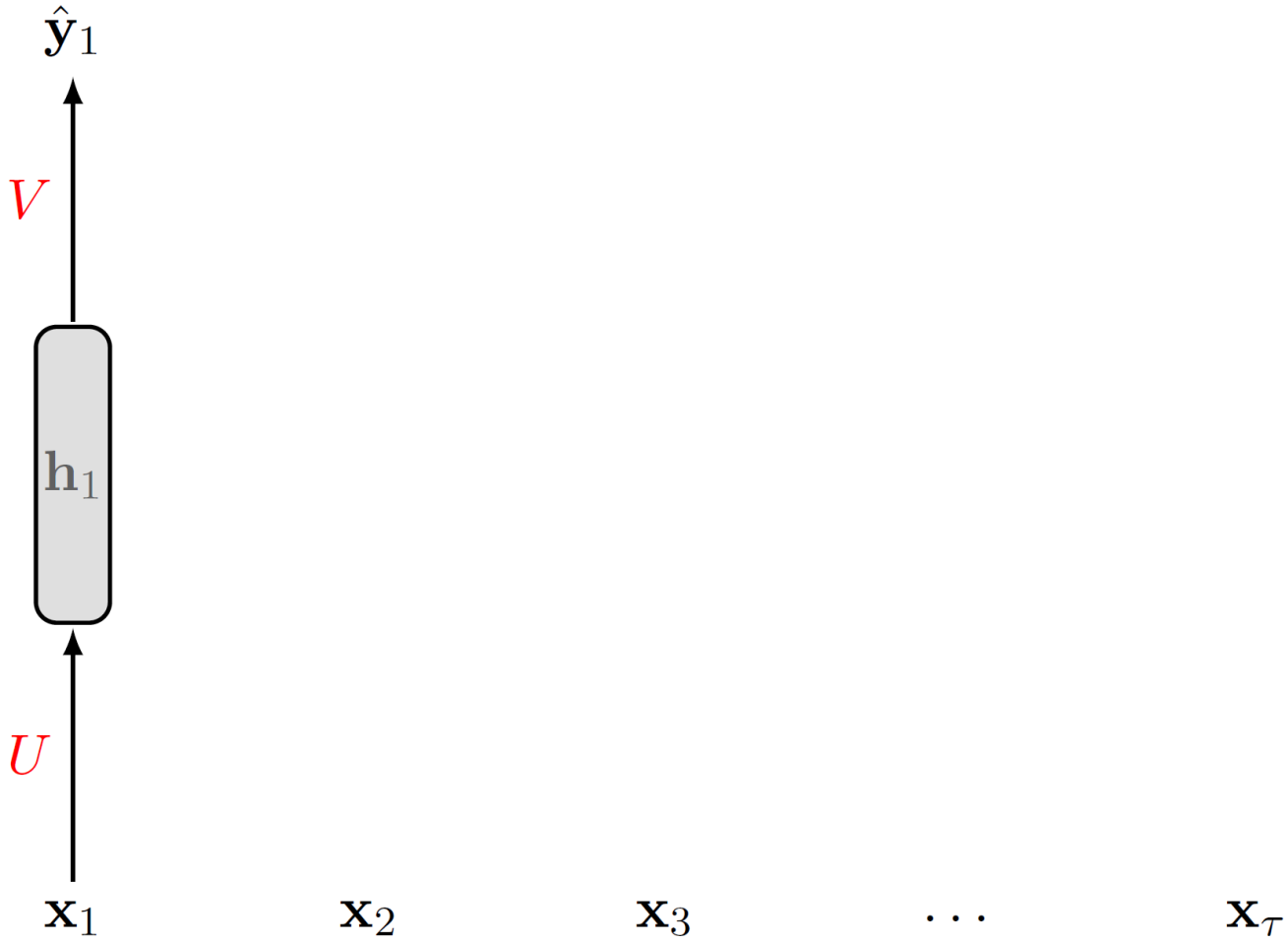
Unrolling the Recurrence

$$\mathbf{x}_1 \qquad \mathbf{x}_2 \qquad \mathbf{x}_3 \qquad \cdots \qquad \mathbf{x}_\tau$$

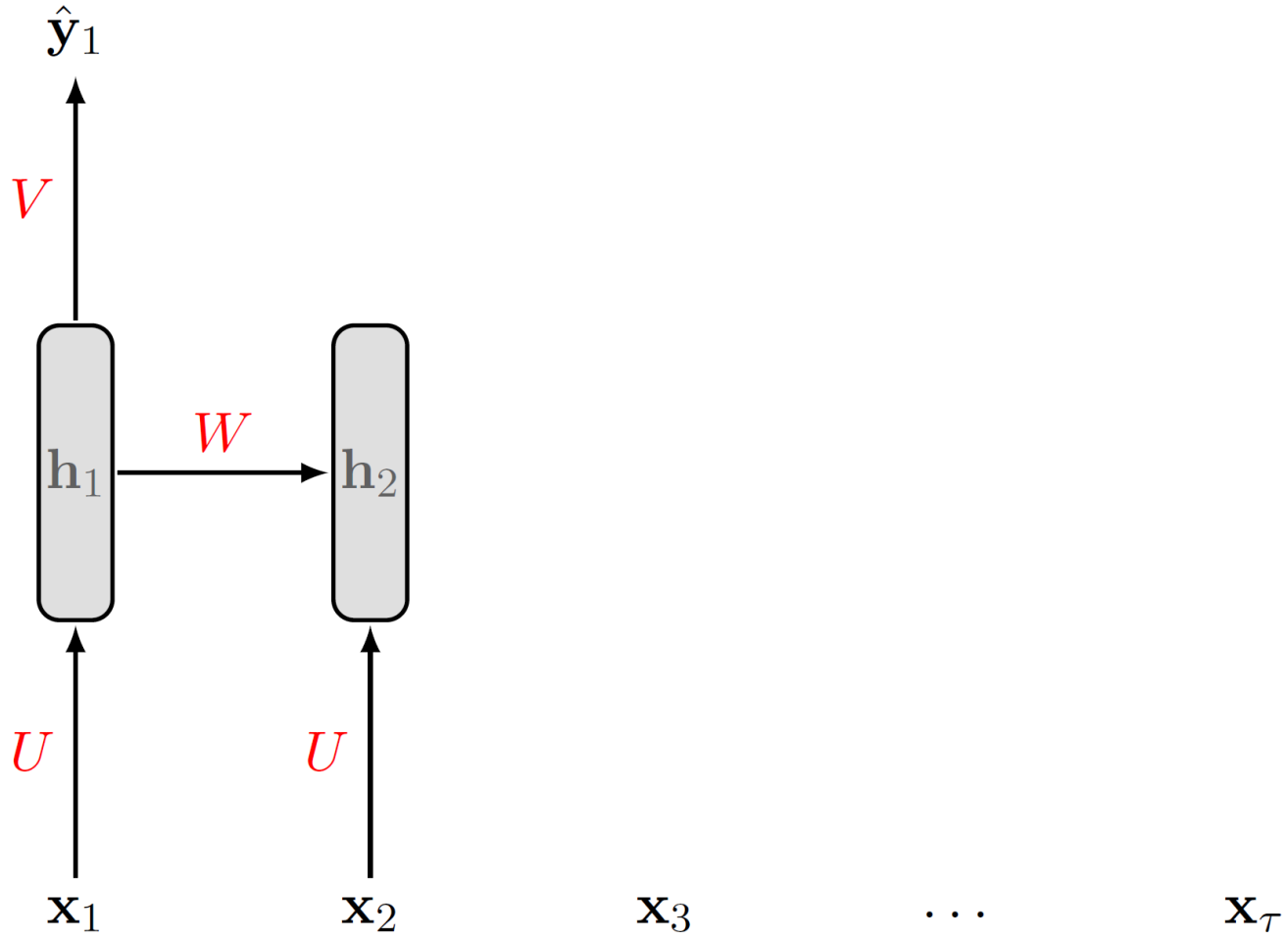
Unrolling the Recurrence



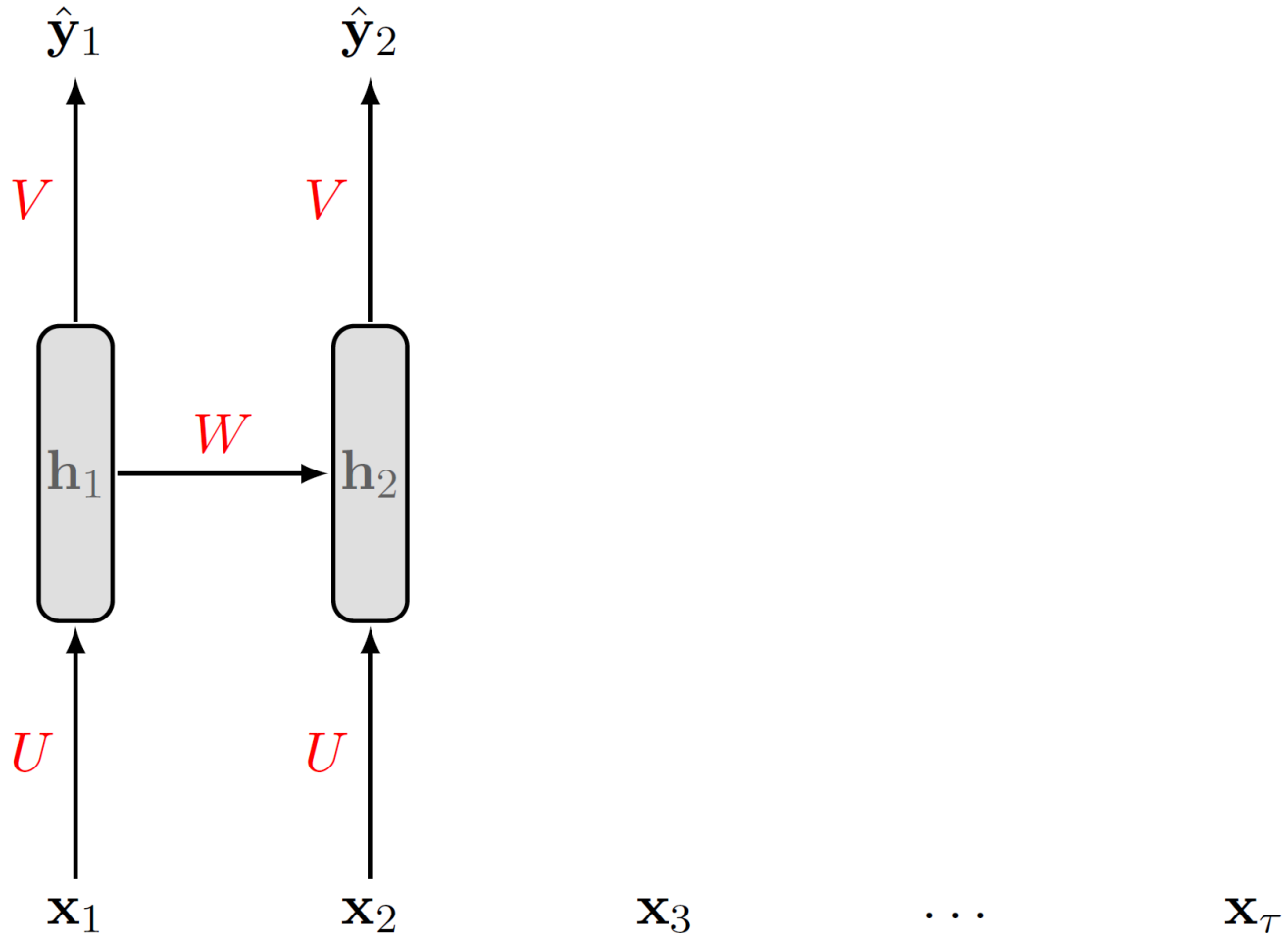
Unrolling the Recurrence



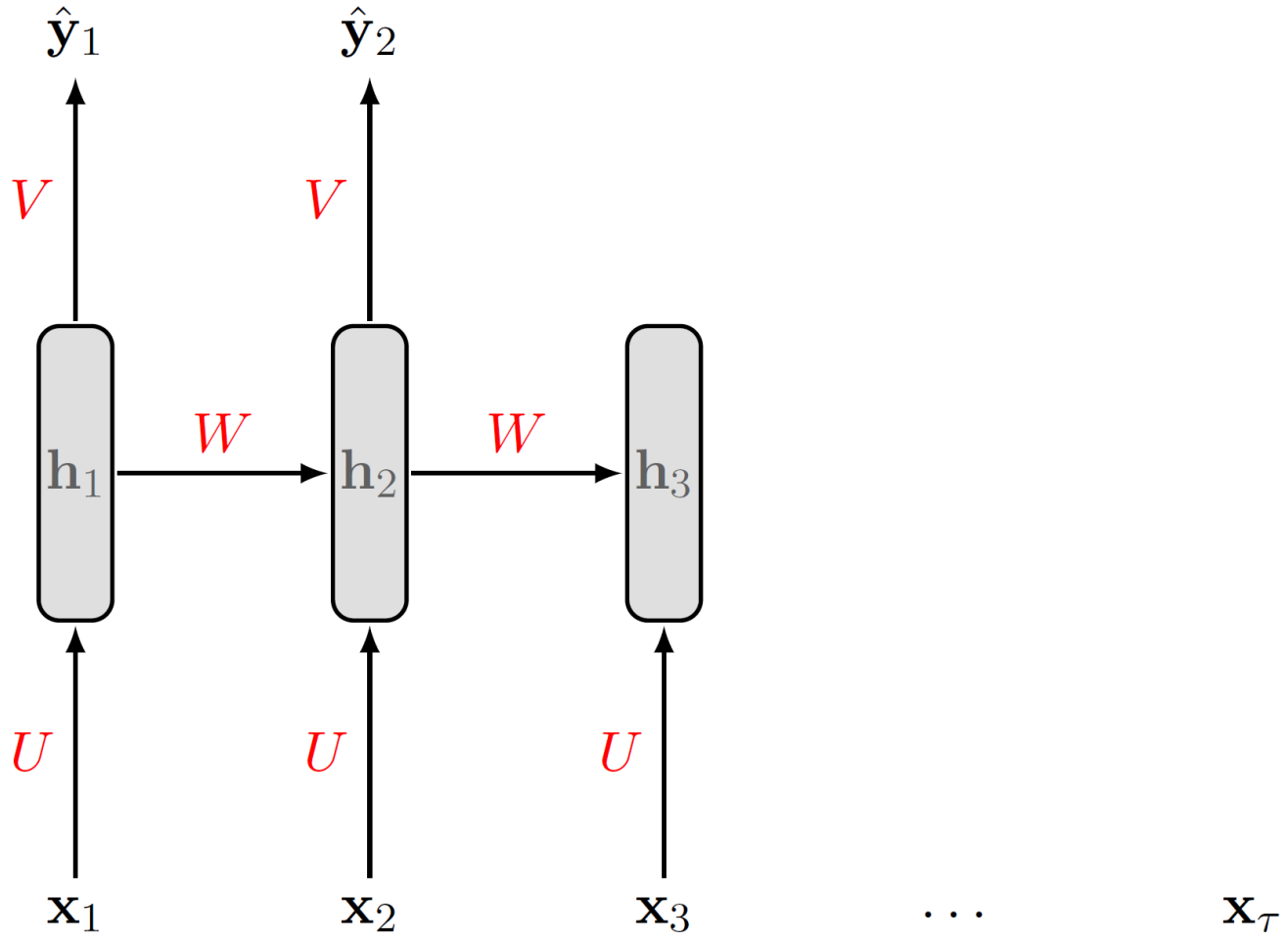
Unrolling the Recurrence



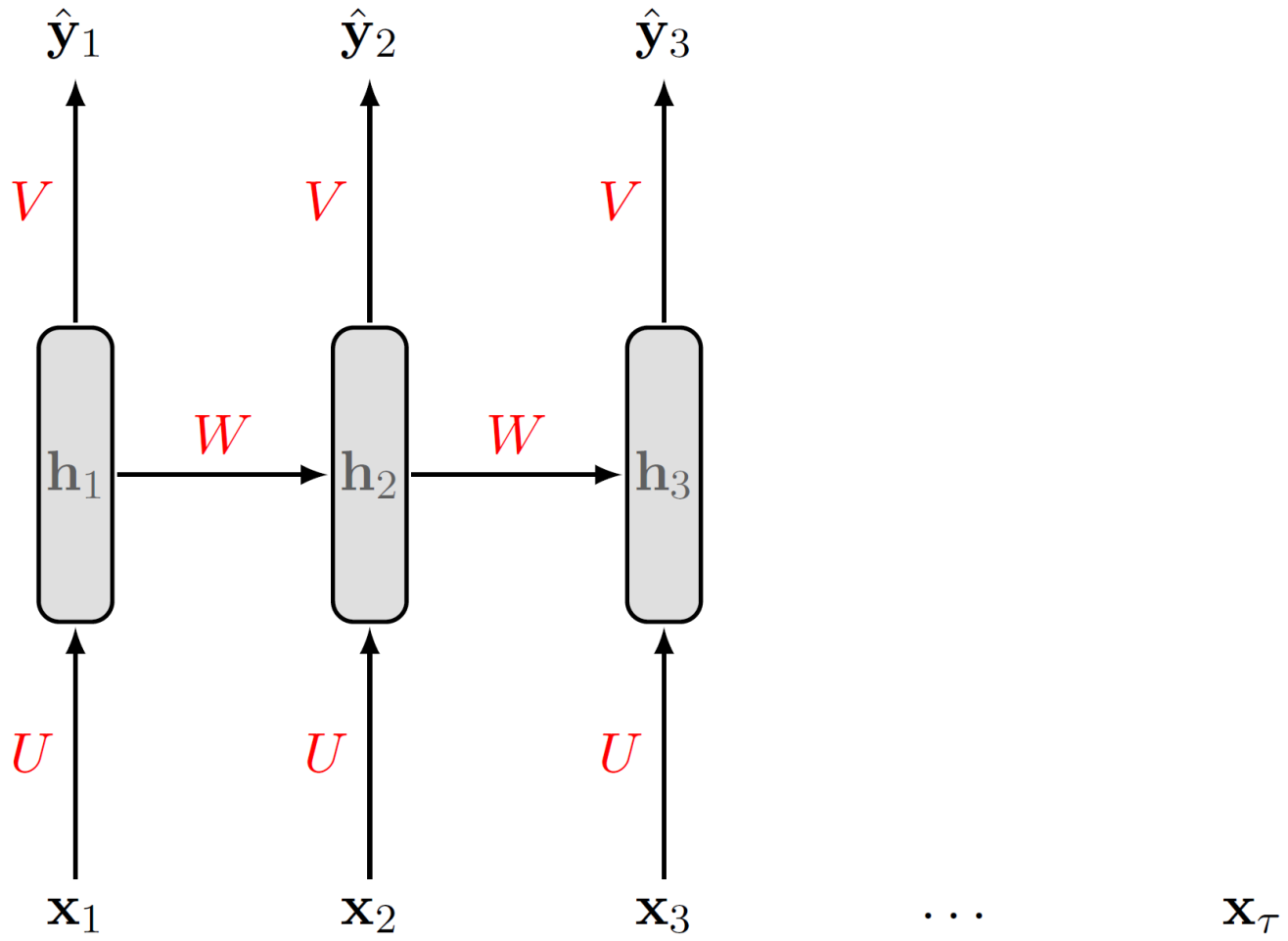
Unrolling the Recurrence



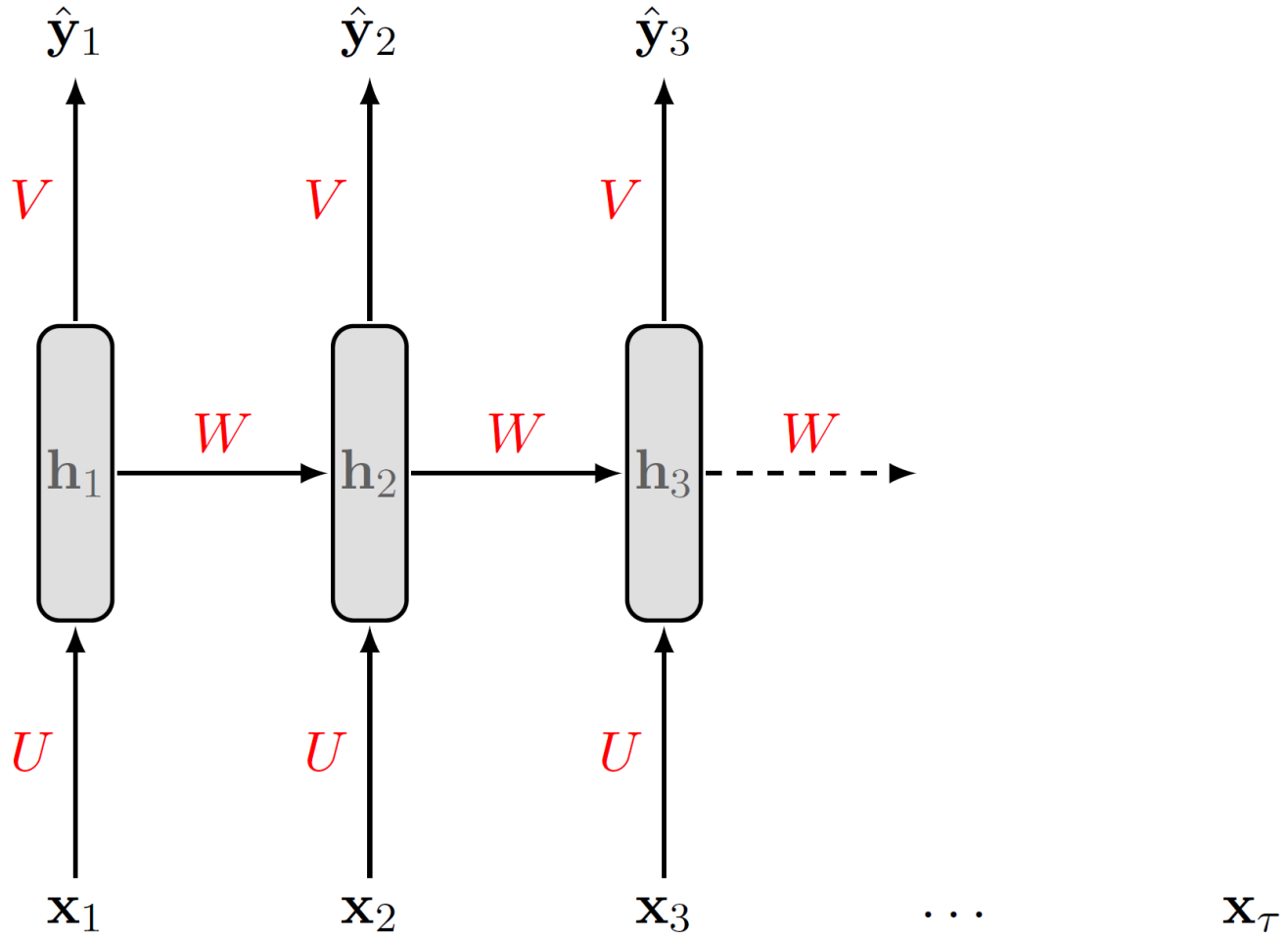
Unrolling the Recurrence



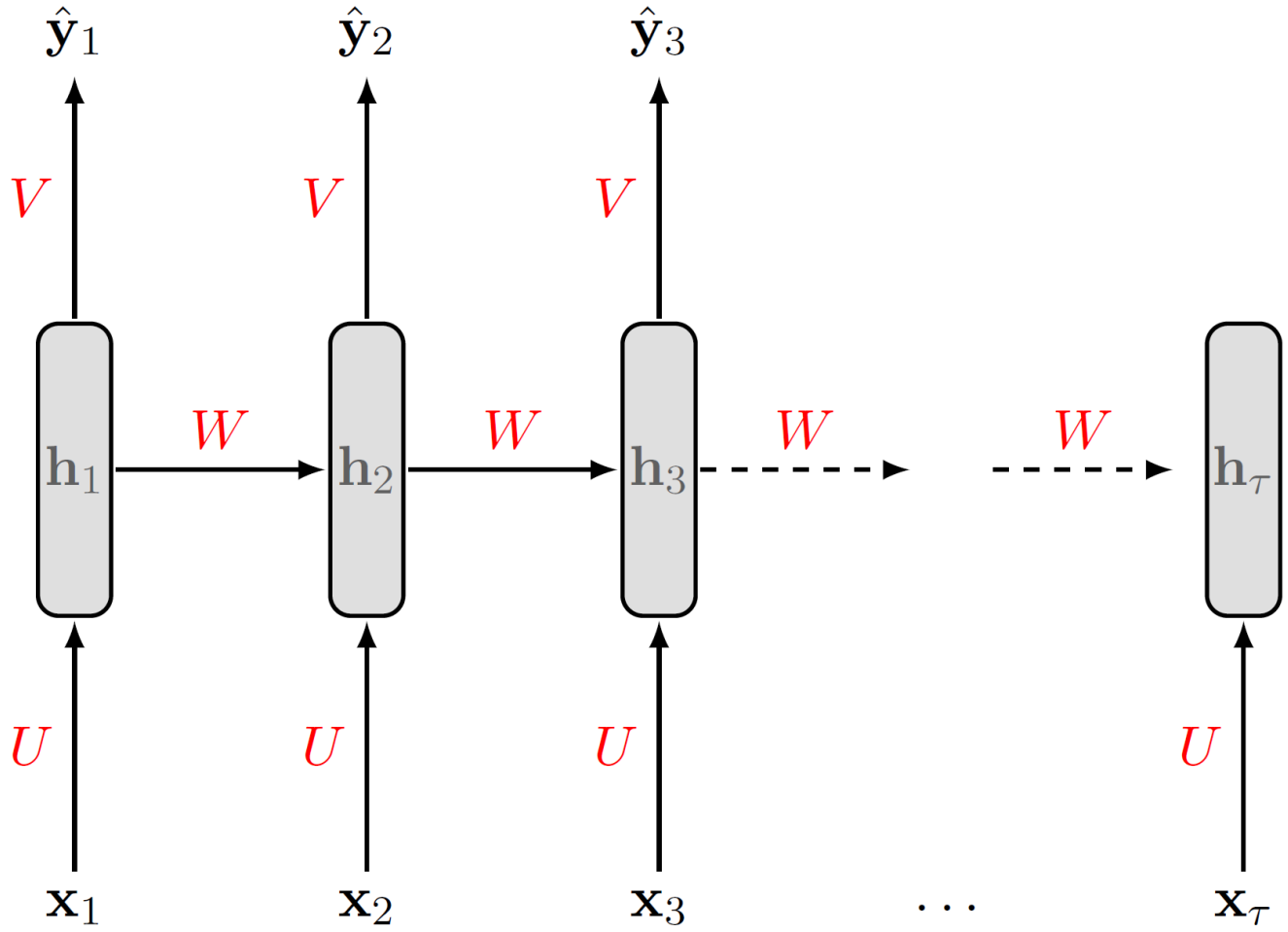
Unrolling the Recurrence



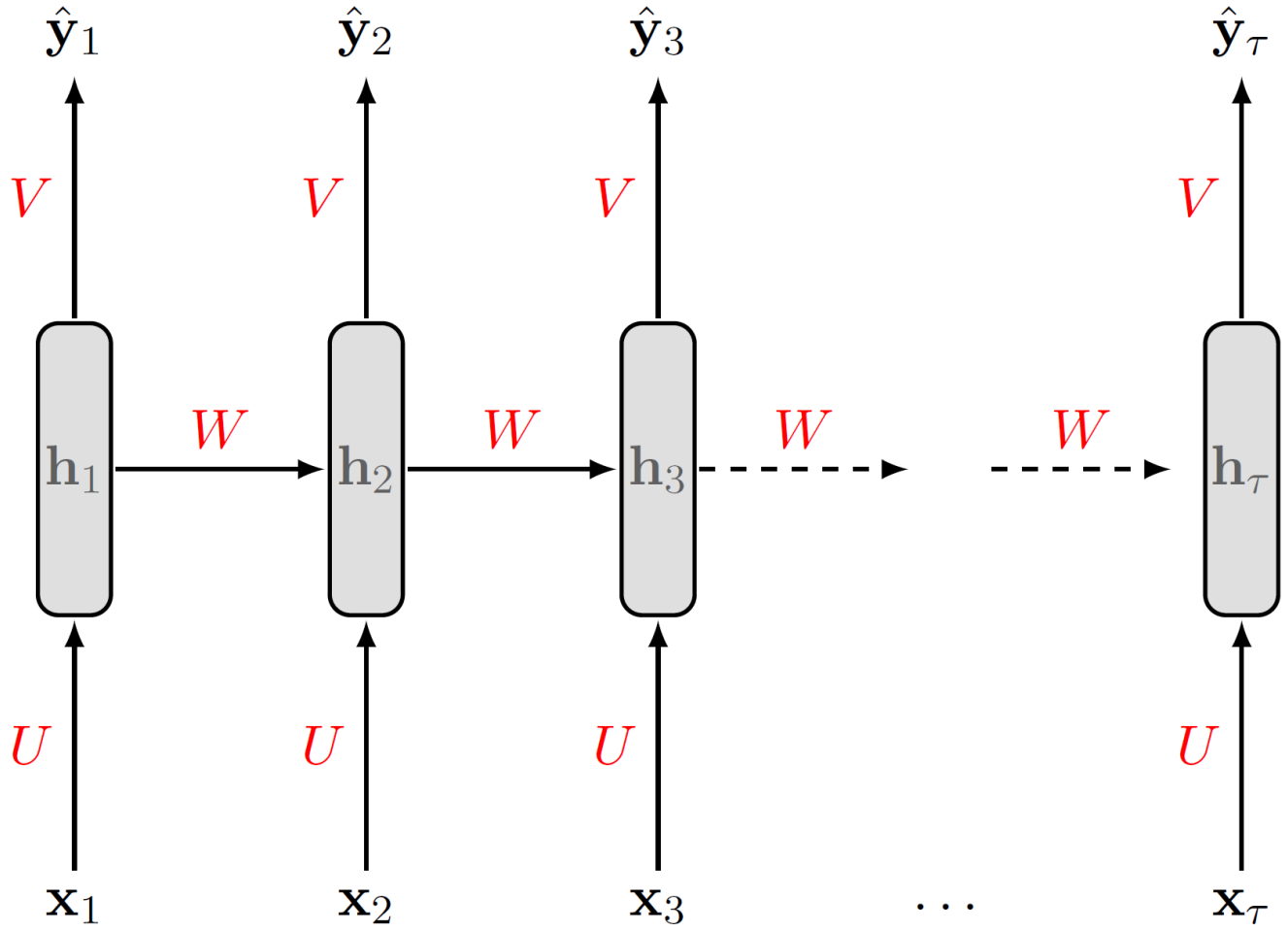
Unrolling the Recurrence



Unrolling the Recurrence



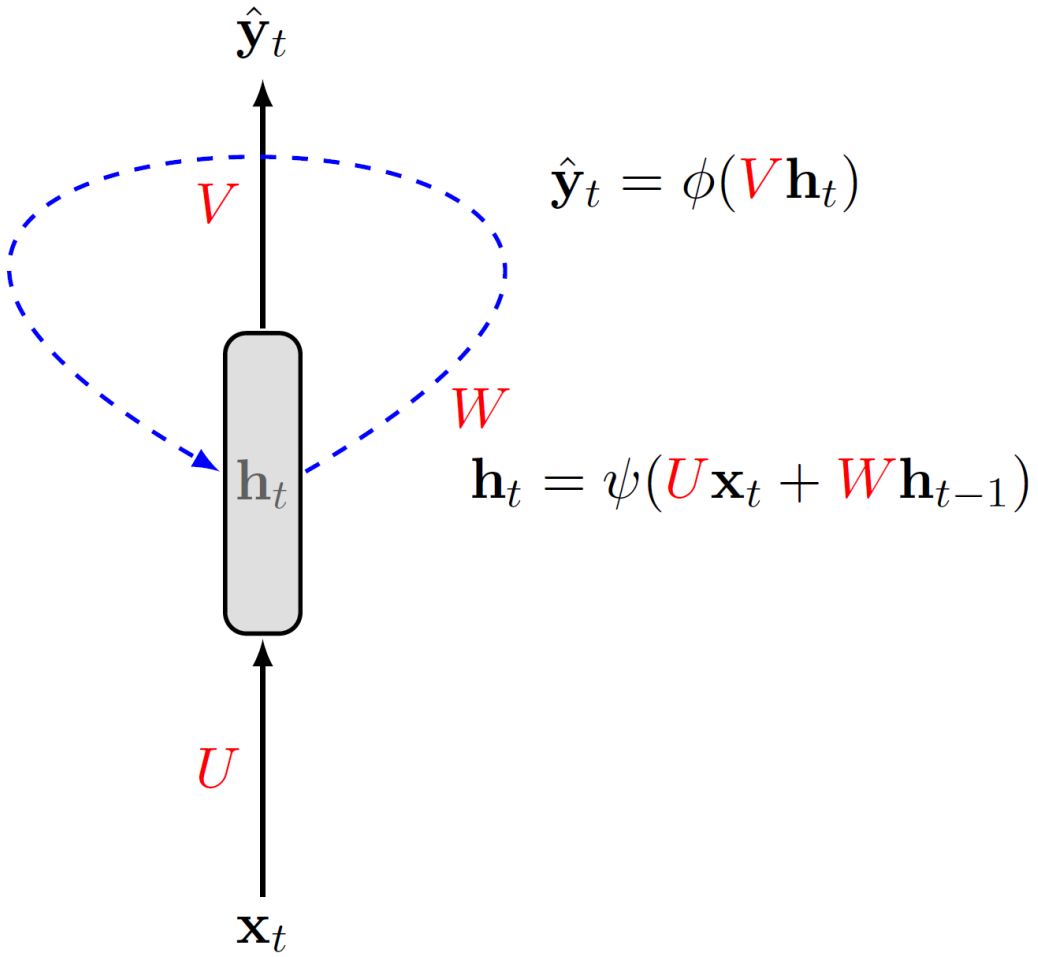
Unrolling the Recurrence



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:

$$\mathbf{a}_t = b + W\mathbf{h}_{t-1} + U\mathbf{x}_t$$
$$\mathbf{h}_t = \tanh \mathbf{a}_t$$
$$\mathbf{o}_t = c + V\mathbf{h}_t$$
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{o}_t)$$



Backward Propagation

- Loss would just be the sum of losses over time steps

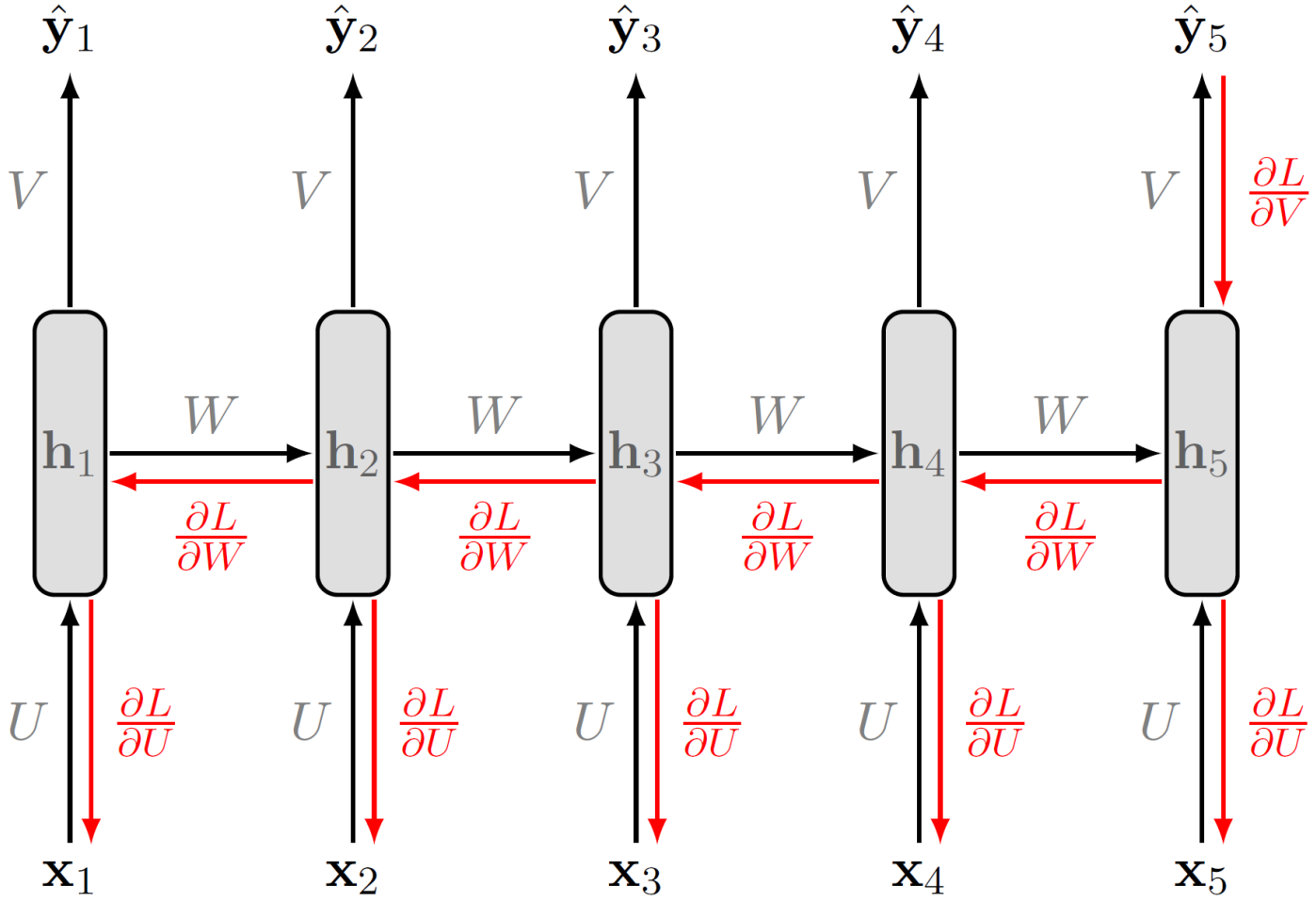
- Need to find: $\nabla_V L, \nabla_W L, \nabla_U L$
- And the gradients w.r.t biases: $\nabla_c L$ and $\nabla_b L$

- Treat the recurrent network as a usual multilayer network and apply backpropagation on the unrolled network

- We move from the right to left:

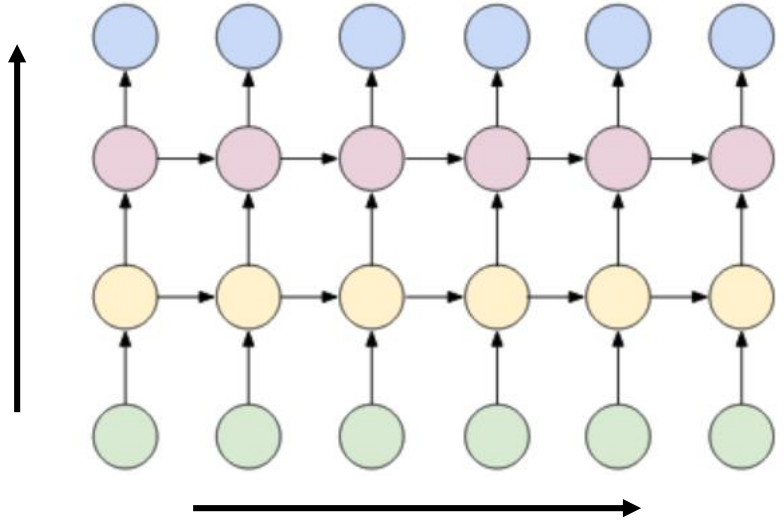
- This is called Backpropagation through time

Backpropagation Through Time (BPTT)



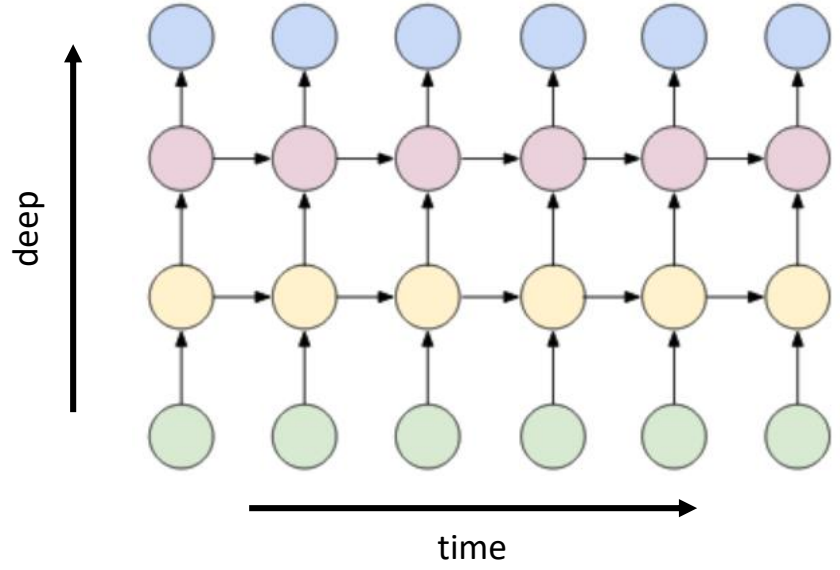
Deep Recurrent Networks

- Three blocks of parameters and associated transformation
  - From the input to the hidden state (from green to yellow)
  - From the previous hidden state to the next hidden state (from yellow to red)
  - From the hidden state to the output (from red to blue)

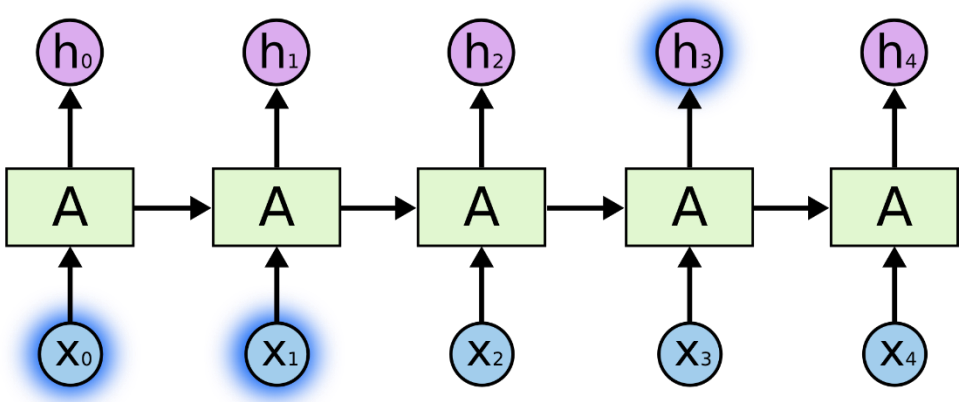


Deep Recurrent Networks

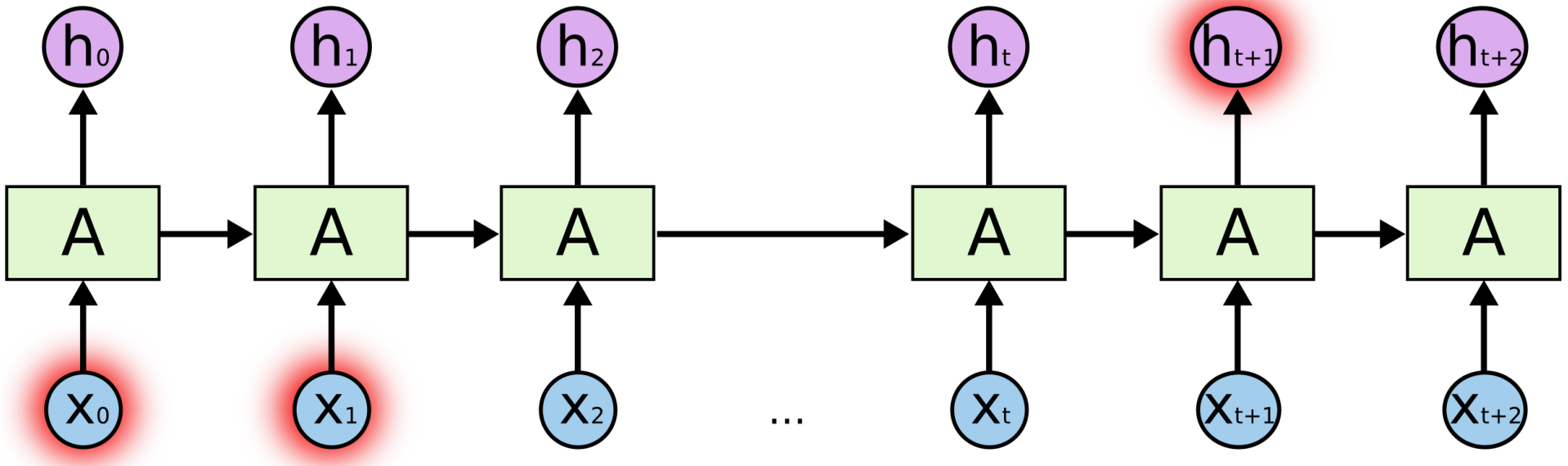
- Three blocks of parameters and associated transformation
  - From the input to the hidden state (from green to yellow)
  - From the previous hidden state to the next hidden state (from yellow to red)
  - From the hidden state to the output (from red to blue)



- “the clouds are in the *sky*,”

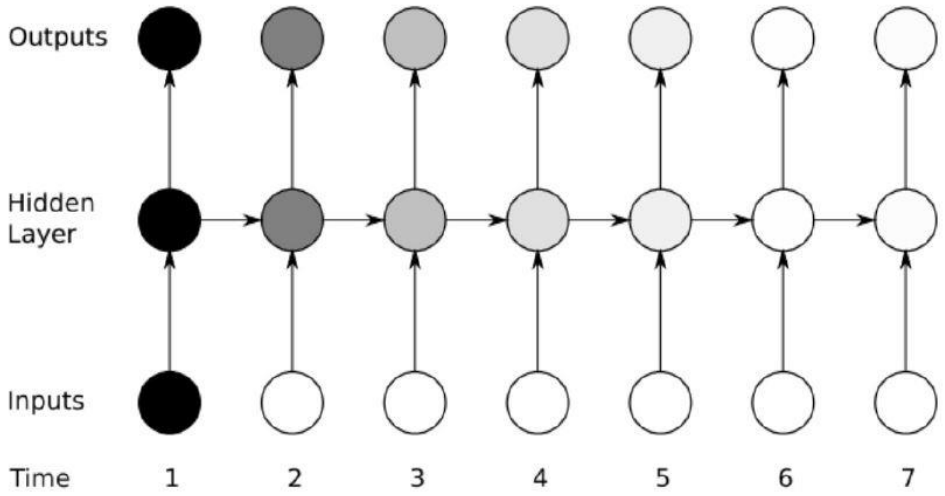


- “I grew up in France… I speak fluent *French*.”



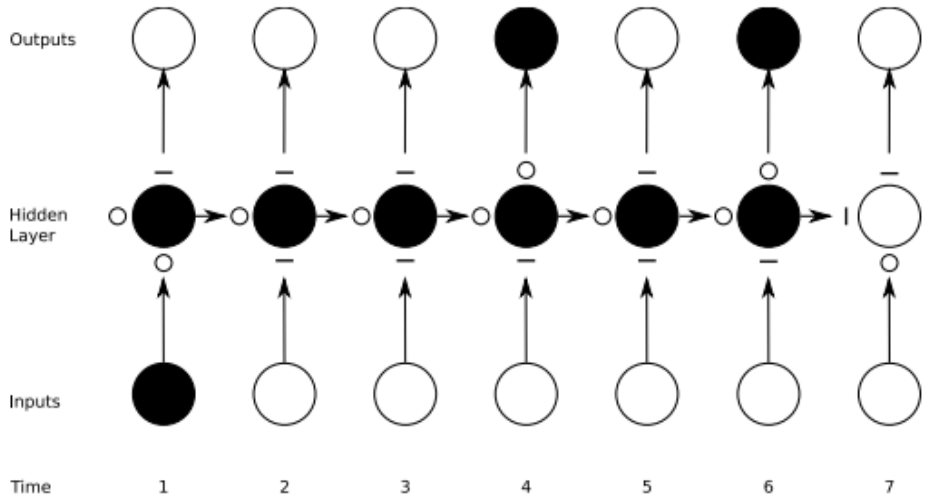
RNN with LSTM

- 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

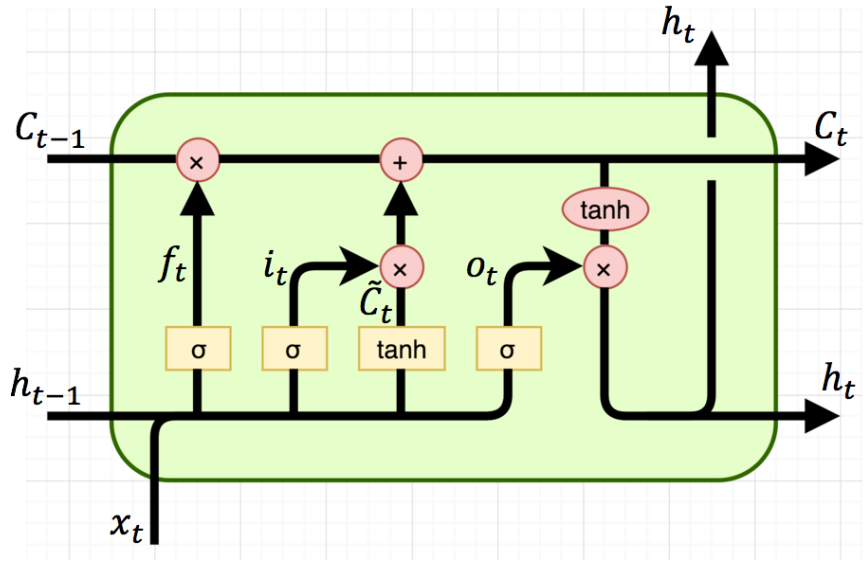


Long Short-Term Memory (LSTM)

- Allow the network to **accumulate** information over a long duration
- Once that information has been used, it might be used for the neural network to **forget** the old state



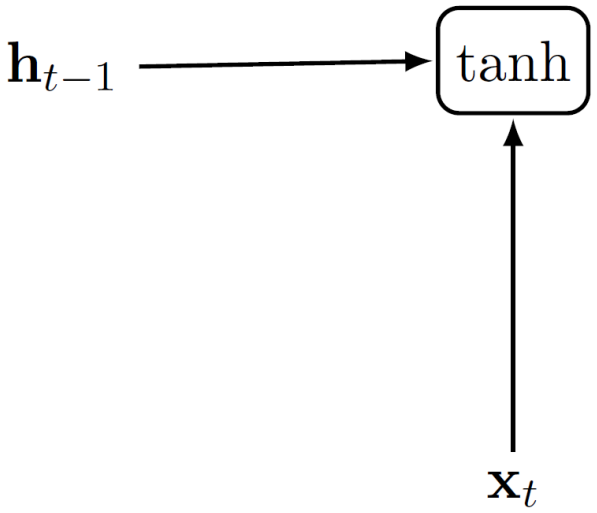
Long Short-Term Memory (LSTM)



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

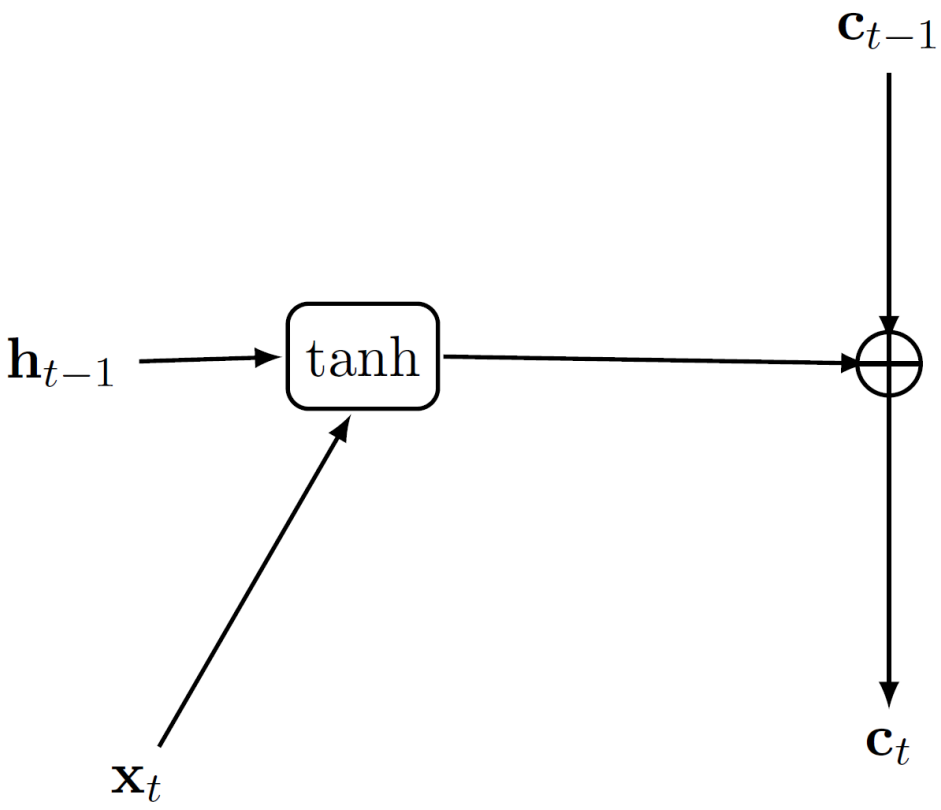
- Summary
  - Connect LSTM cells in a recurrent manner
  - Train parameters in LSTM cells

Long Short Term Memory



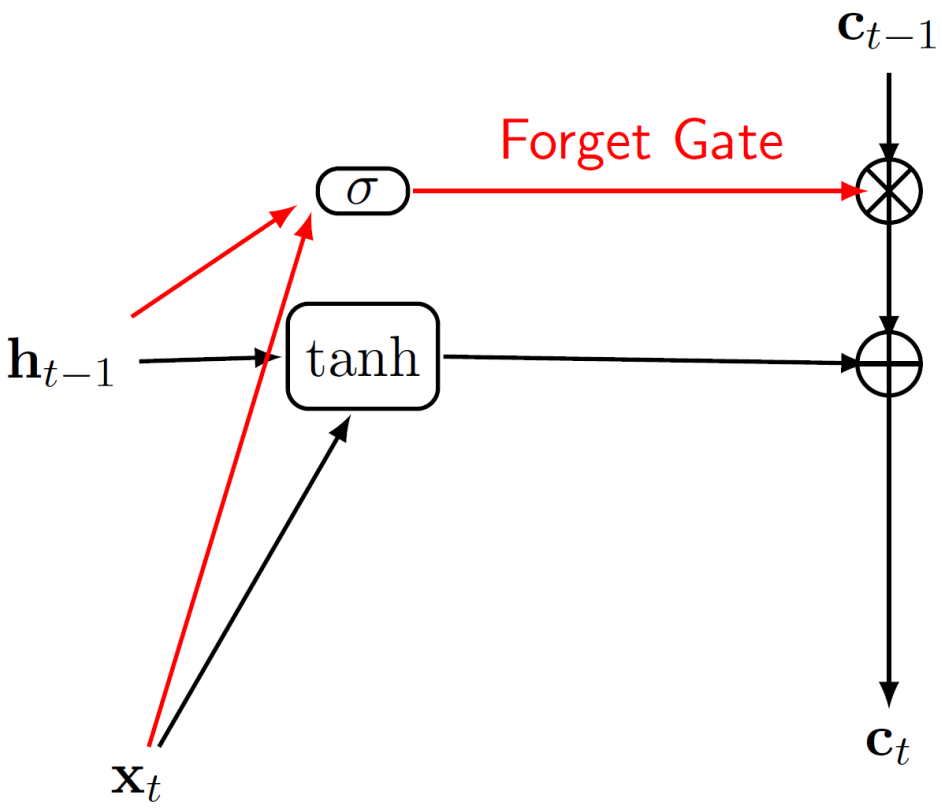
$$\mathbf{h}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$

Long Short Term Memory



$$\tilde{\mathbf{c}}_t = \tanh(W\mathbf{h}_{t-1} + U\mathbf{x}_t)$$
$$\mathbf{c}_t = \mathbf{c}_{t-1} + \tilde{c}_t$$

Long Short Term Memory

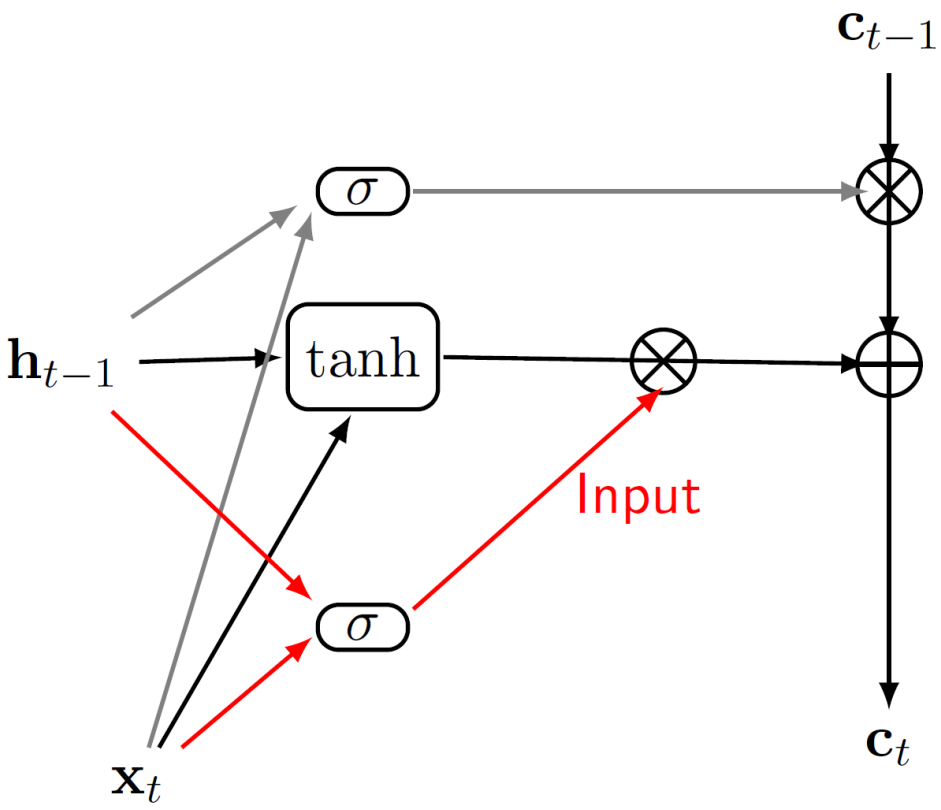


$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + \tilde{c}_t$$

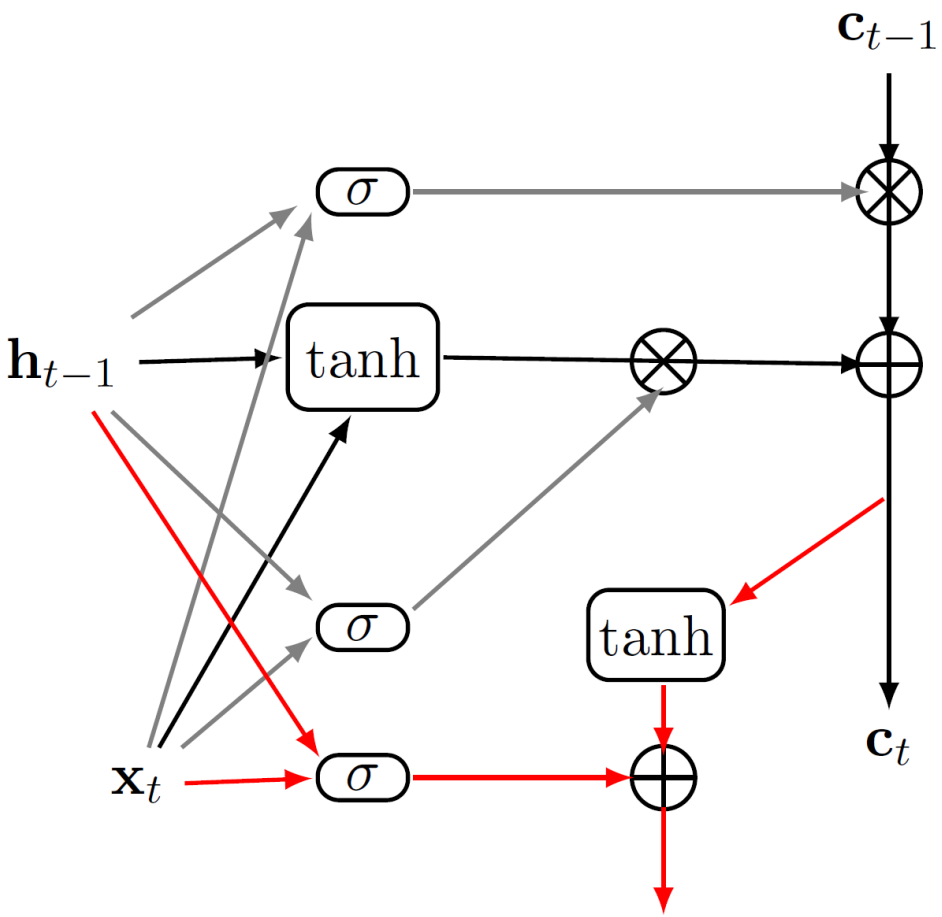
Long Short Term Memory



$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$

$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{c}_t$$

Long Short Term Memory

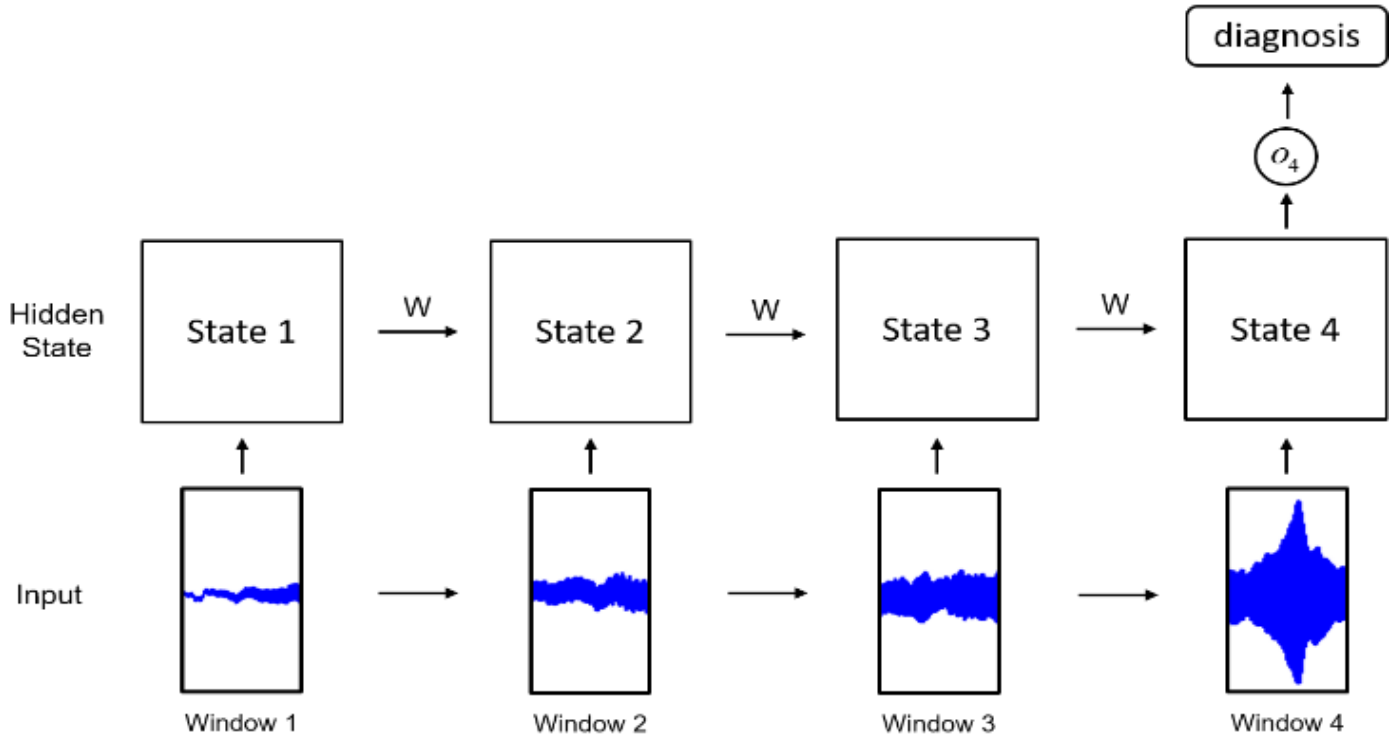


$$f_t = \sigma(W_f \mathbf{h}_{t-1} + U_f \mathbf{x}_t)$$
$$i_t = \sigma(W_i \mathbf{h}_{t-1} + U_i \mathbf{x}_t)$$
$$o_t = \sigma(W_o \mathbf{h}_{t-1} + U_o \mathbf{x}_t)$$

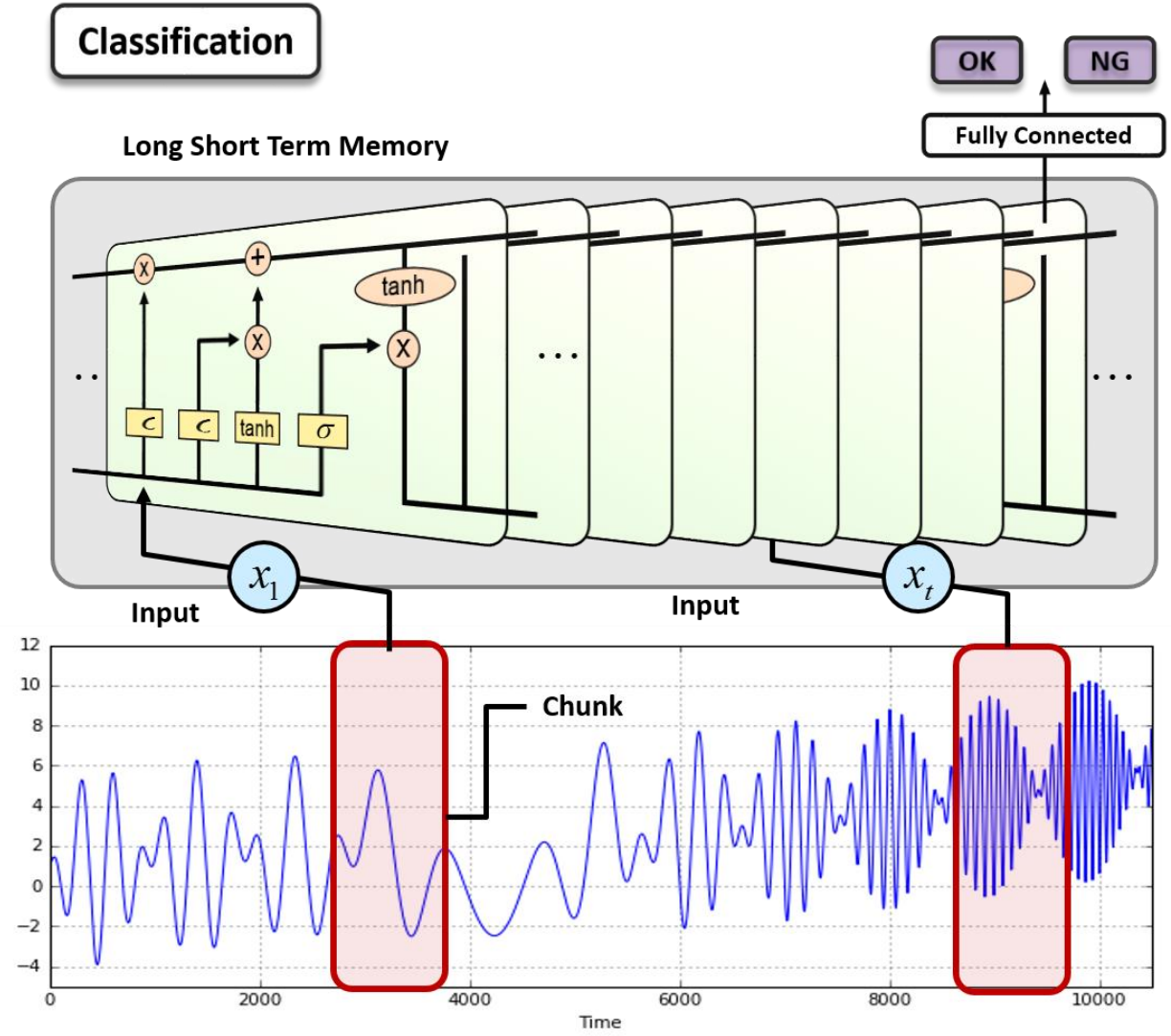
$$\tilde{\mathbf{c}}_t = \tanh(W \mathbf{h}_{t-1} + U \mathbf{x}_t)$$
$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{c}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

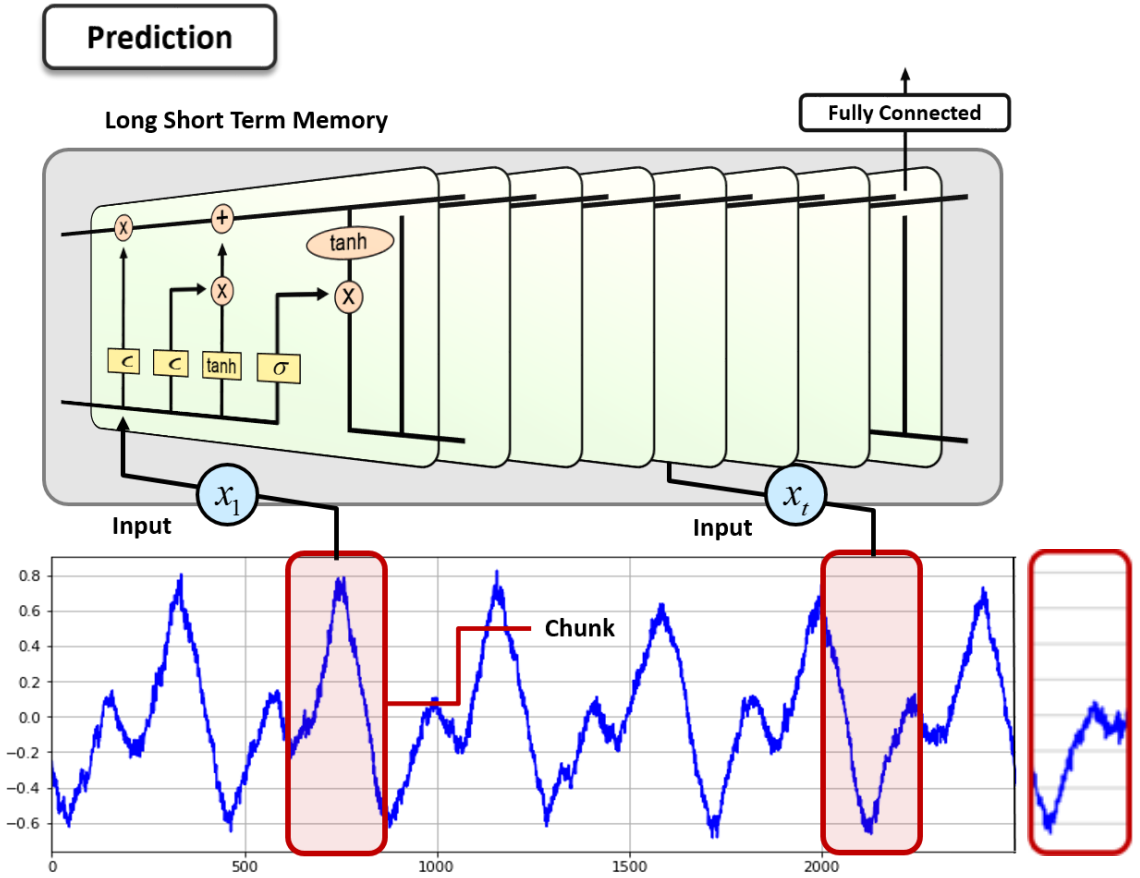
Time Series Data and RNN



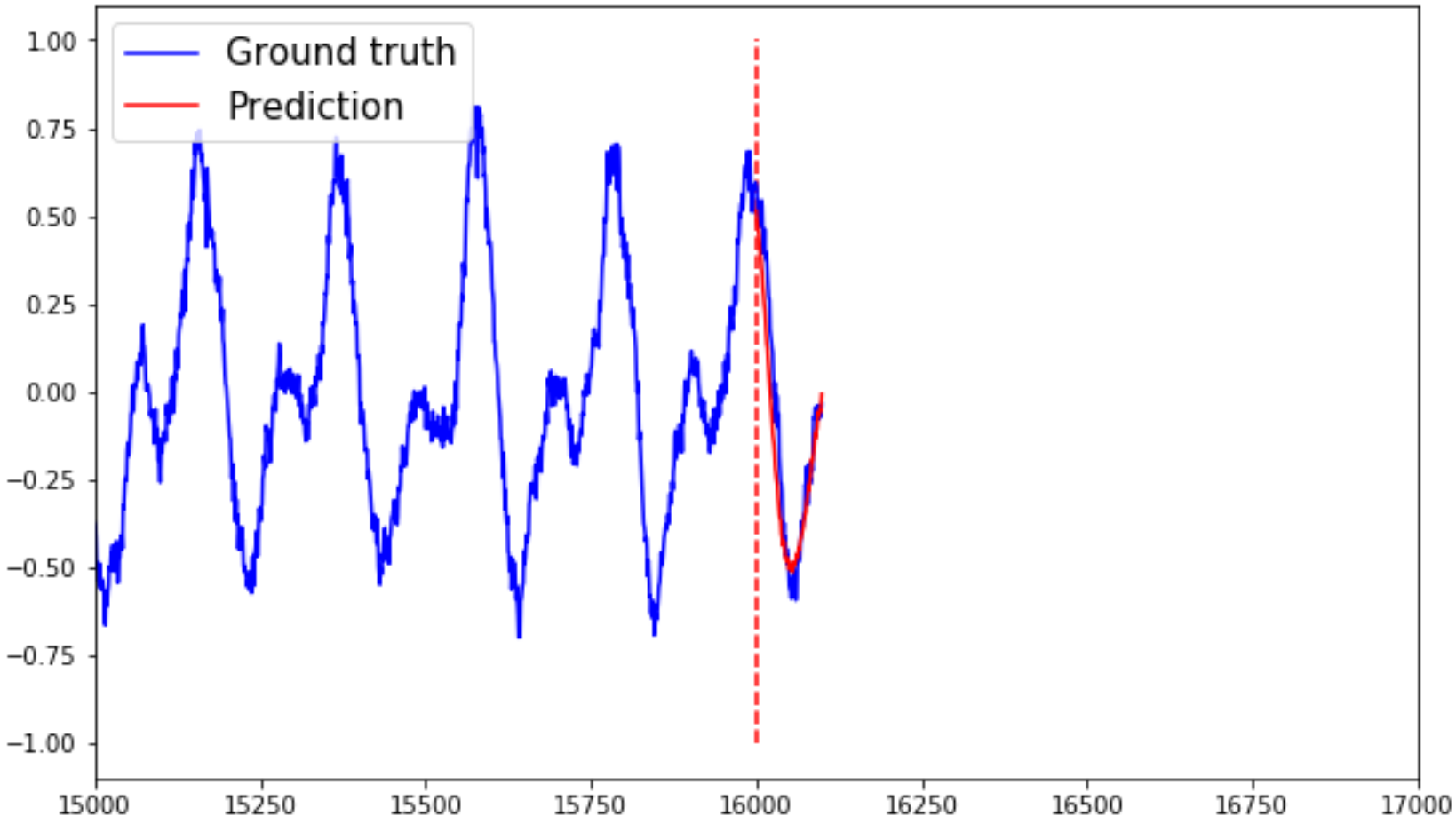
RNN for Classification



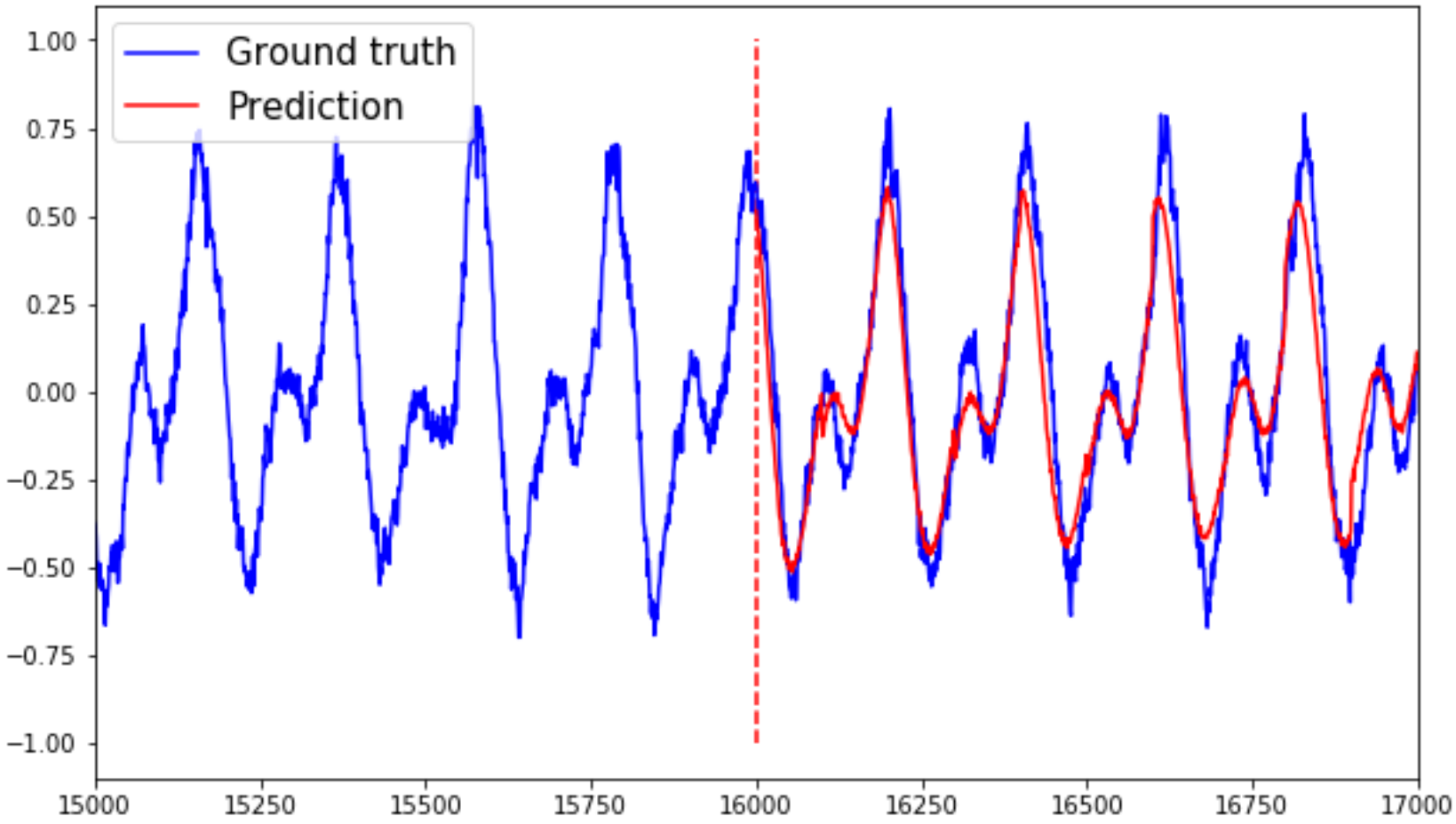
RNN for Prediction



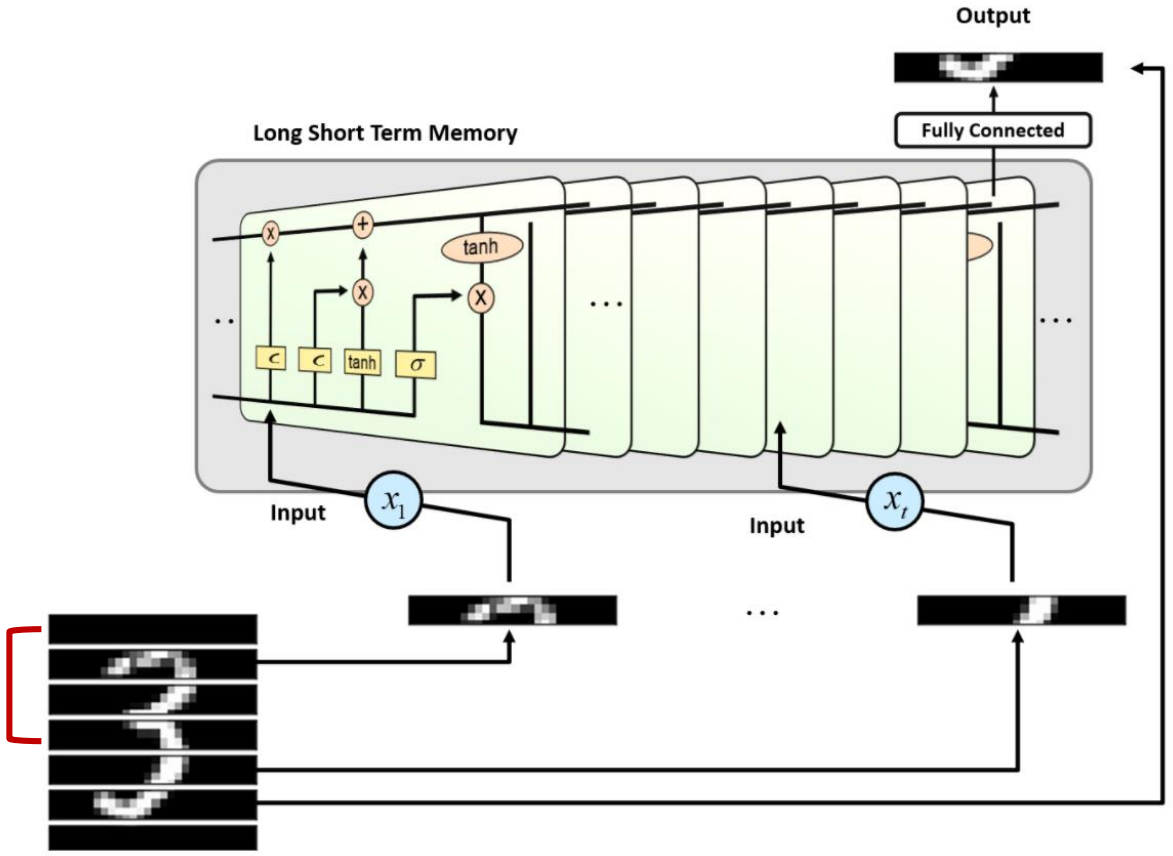
Prediction Example



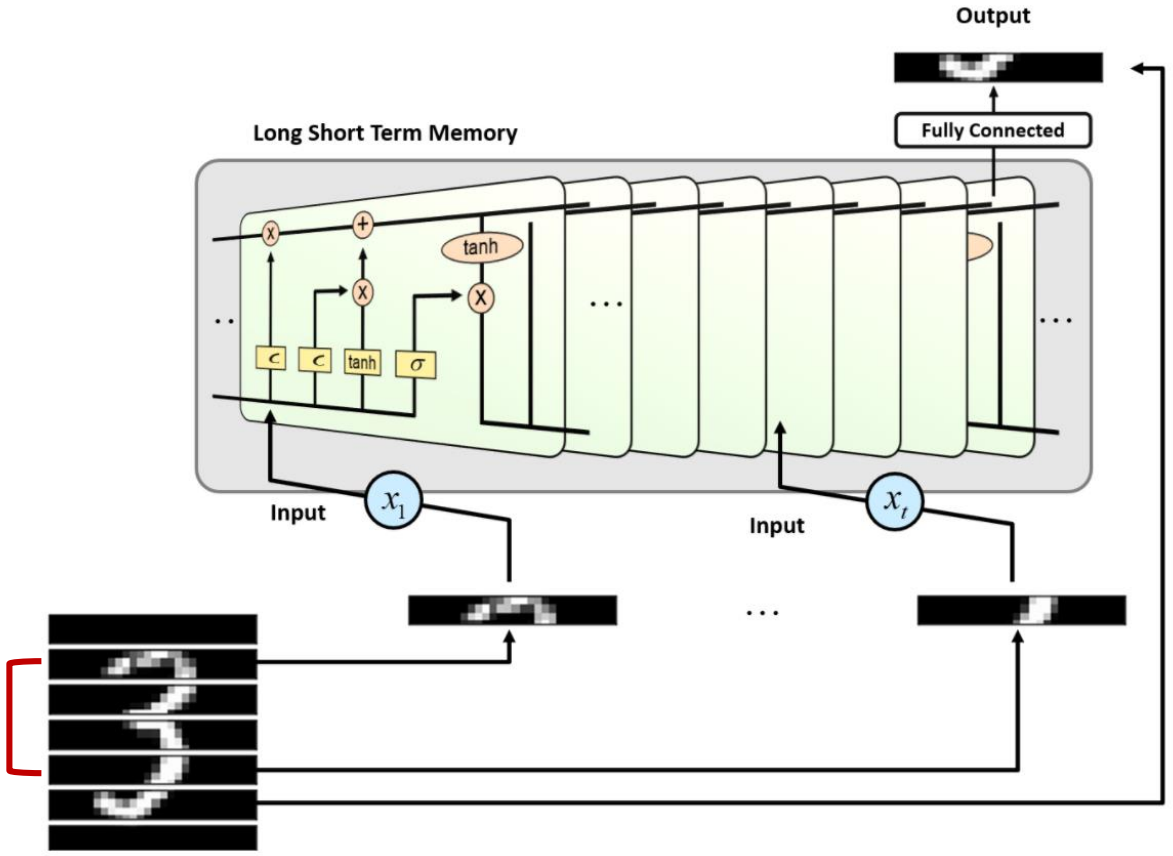
Prediction Example



Prediction of MNIST



Prediction of MNIST



RNN with TensorFlow

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

- Import Library

```
import tensorflow as tf
from six.moves import cPickle
import numpy as np
import matplotlib.pyplot as plt
```

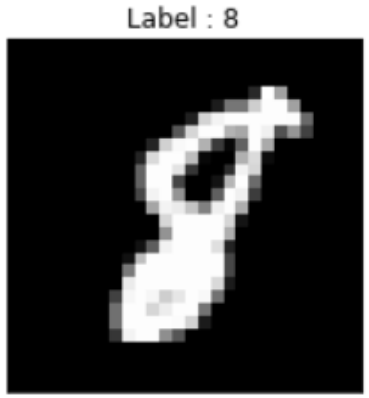
- Load MNIST Data
  - Download MNIST data from the tensorflow tutorial example

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

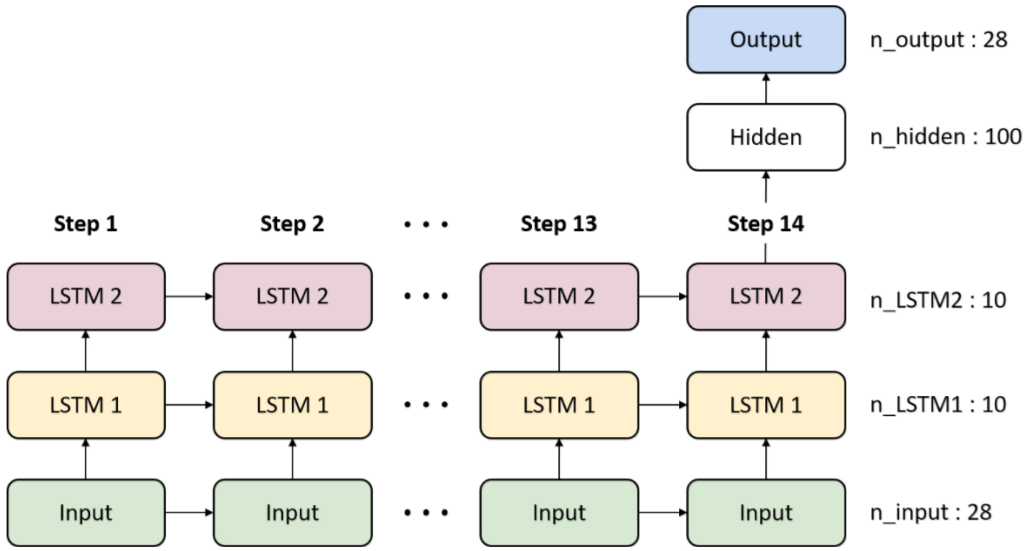
```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

RNN with TensorFlow

```
# Check data
train_x, train_y = mnist.train.next_batch(10)
img = train_x[9,:].reshape(28, 28)

plt.figure(figsize=(5, 3))
plt.imshow(img,'gray')
plt.title("Label : {}".format(np.argmax(train_y[9])))
plt.xticks([])
plt.yticks([])
plt.show()
```



RNN Structure




```
n_step = 14
n_input = 28

## LSTM shape
n_lstm1 = 10
n_lstm2 = 10

## Fully connected
n_hidden = 100
n_output = 28
```

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

```
lstm = tf.contrib.rnn.BasicLSTMCell(n_lstm)
```

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

```
h, c = tf.nn.dynamic_rnn(lstm, input_tensor, dtype=tf.float32)
```

```
def build_model(x, weights, biases):
    with tf.variable_scope('rnn'):
        # Build RNN network
        with tf.variable_scope('lstm1'):
            lstm1 = tf.contrib.rnn.BasicLSTMCell(n_lstm1)
            h1, c1 = tf.nn.dynamic_rnn(lstm1, x, dtype=tf.float32)
        with tf.variable_scope('lstm2'):
            lstm2 = tf.contrib.rnn.BasicLSTMCell(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

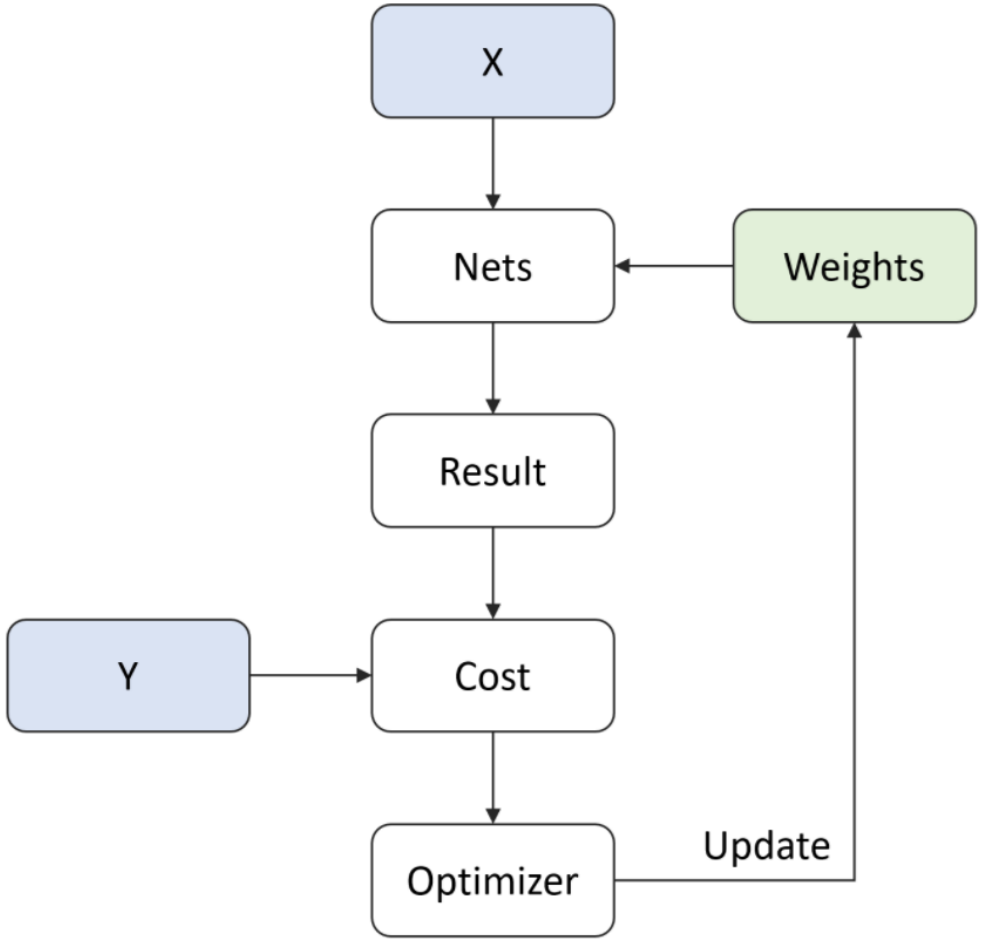
$$\frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})^2$$

- Initializer
  - Initialize all the empty variables

- Optimizer
  - AdamOptimizer: the most popular optimize

```
LR = 0.0005

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()
```

Summary of Model



Iteration Configuration

- Define parameters for training RNN
  - n_iter: the number of training steps
  - n_prt: check loss for every n_prt iteration

```
n_iter = 2500
n_prt = 100
```

Optimization

- Do not run on CPU. It will take quite a while.

```
# Run initialize
# config = tf.ConfigProto(allow_soft_placement=True)  # GPU Allocating policy
# sess = tf.Session(config=config)
sess = tf.Session()
sess.run(init)

for i in range(n_iter):
    train_x, train_y = mnist.train.next_batch(50)
    train_x = train_x.reshape(-1, 28, 28)

    for j in range(n_step):
        sess.run(optm, feed_dict={x: train_x[:,j:j+n_step,:],  y: train_x[:,j+n_step]})
    if i % n_prt == 0:
        c = sess.run(loss, feed_dict={x: train_x[:,13:13+n_step,:],  y: train_x[:,13+n_step]})
        print ("Iter : {}".format(i))
        print ("Cost : {}".format(c))
```

Test or Evaluation

- Do not run on CPU. It will take quite a while.
- Predict the MNIST image
- MNIST is 28 x 28 image.
  - The model predicts a piece of 1 x 28 image.
  - First, 14 x 28 image will be fed into a model, then the model predict the last 14 x 28 image, recursively.

```
test_x, test_y = mnist.test.next_batch(10)
test_x = test_x.reshape(-1, 28, 28)

idx = 0
gen_img = []

sample = test_x[idx, 0:14, :]
input_img = sample.copy()

feeding_img = test_x[idx, 0:0+n_step, :]

for i in range(n_step):
    test_pred = sess.run(pred, feed_dict={x: feeding_img.reshape(1, 14, 28)})
    feeding_img = np.delete(feeding_img, 0, 0)
    feeding_img = np.vstack([feeding_img, test_pred])
    gen_img.append(test_pred)
```

Test or Evaluation

Original Img

Input

Generated Img

Load pre-trained Model

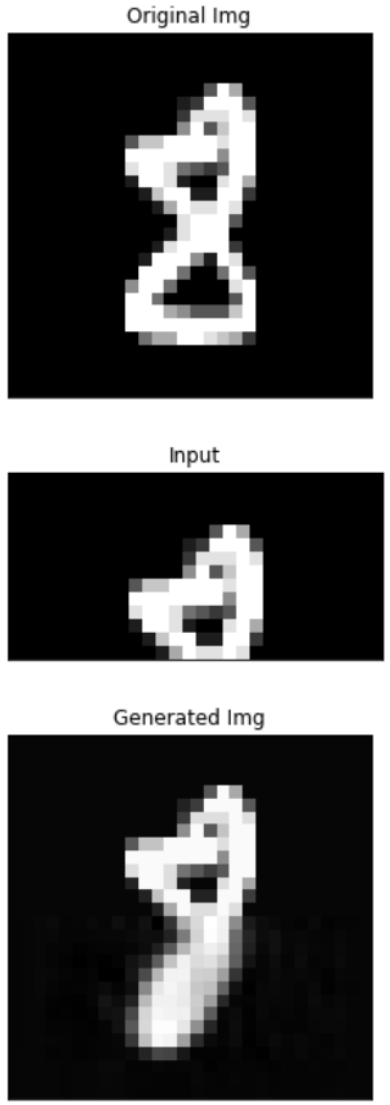
- We trained the model on GPU for you.
- You can load the pre-trained model to see RNN MNIST results
- LSTM size
  - n_lstm1 = 128
  - n_lstm2 = 256

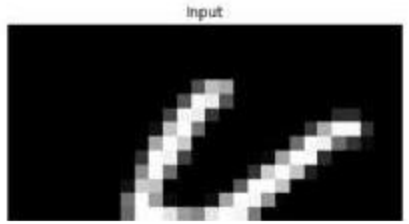
```
from RNN import RNN
my_rnn = RNN()
my_rnn.load('./data_files/RNN_mnist/checkpoint/RNN_5000')
```

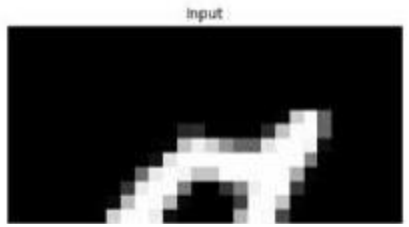
```
INFO:tensorflow:Restoring parameters from ./data_files/RNN_mnist/checkpoint/RNN_5000
Model loaded from file : ./data_files/RNN_mnist/checkpoint/RNN_5000
```

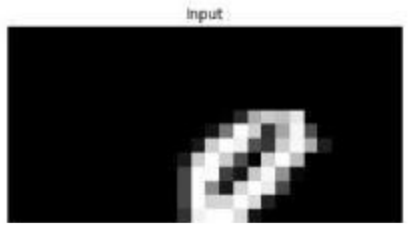
Test with pre-trained Model

```
test_x, test_y = mnist.test.next_batch(10)
test_x = test_x.reshape(-1, 28, 28)

sample = test_x[0, 0:14,:]

gen_img = my_rnn.predict(sample)
```




Input


Input


Input