Recurrent Neural Network

By Prof. Seungchul Lee
Industrial AI Lab
http://isystems.unist.ac.kr/
POSTECH

Table of Contents

1. Time Series Data

1.1. Deterministic

- For example

$$y[0] = 1, y[1] = \frac{1}{2}, y[2] = \frac{1}{4}$$

- Closed form

$$y[n] = \left(\frac{1}{2}\right)^n$$

- Linear difference equation (LDE) and initial condition

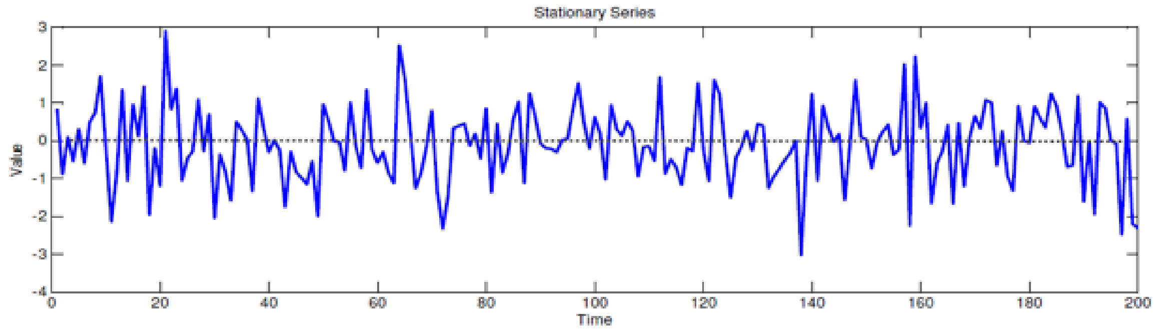
$$y[n] = \frac{1}{2}y[n-1], y[0] = 1$$

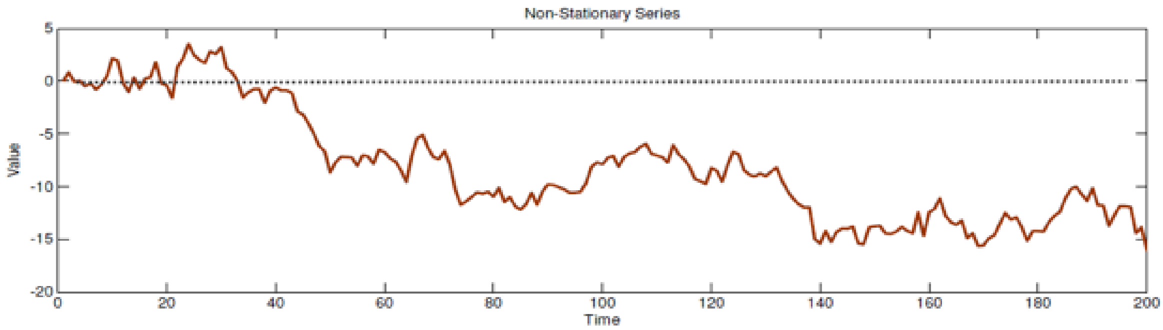
- High order LDEs

$$y[n] = \alpha_1 y[n-1] + \alpha_2 y[n-2]$$
$$y[n] = \alpha_1 y[n-1] + \alpha_2 y[n-2] + \cdots + \alpha_k y[n-k]$$

1.2. Stochastic

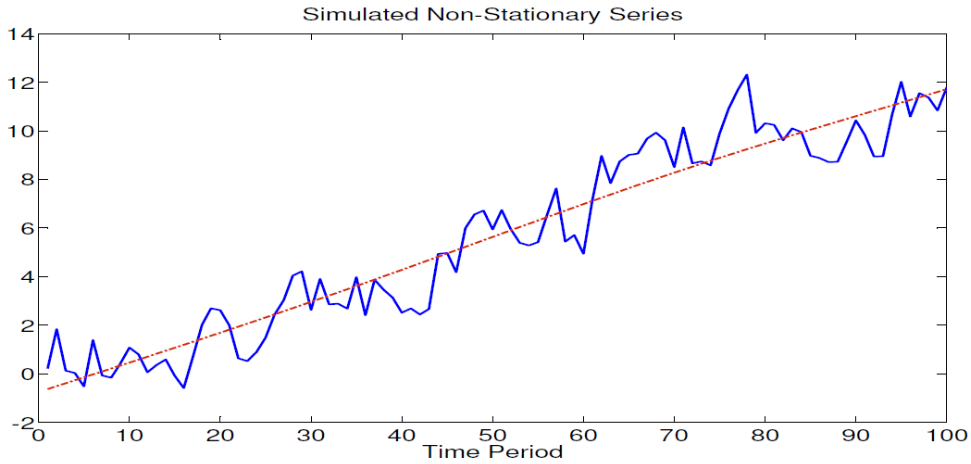
- Stationary
- Non-stationary



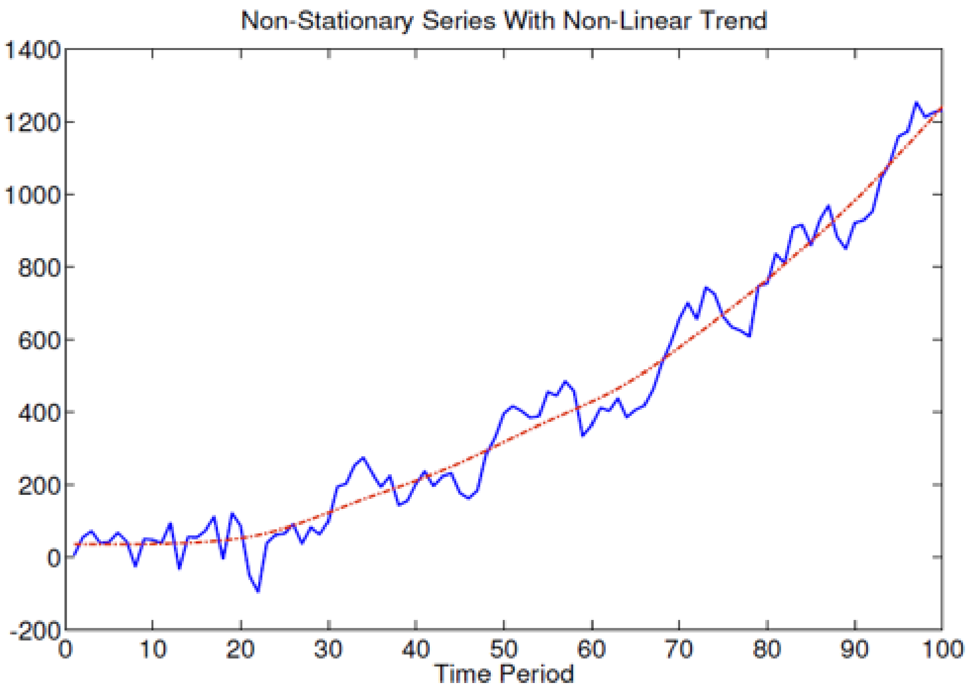


1.3. Dealing with Non-stationary

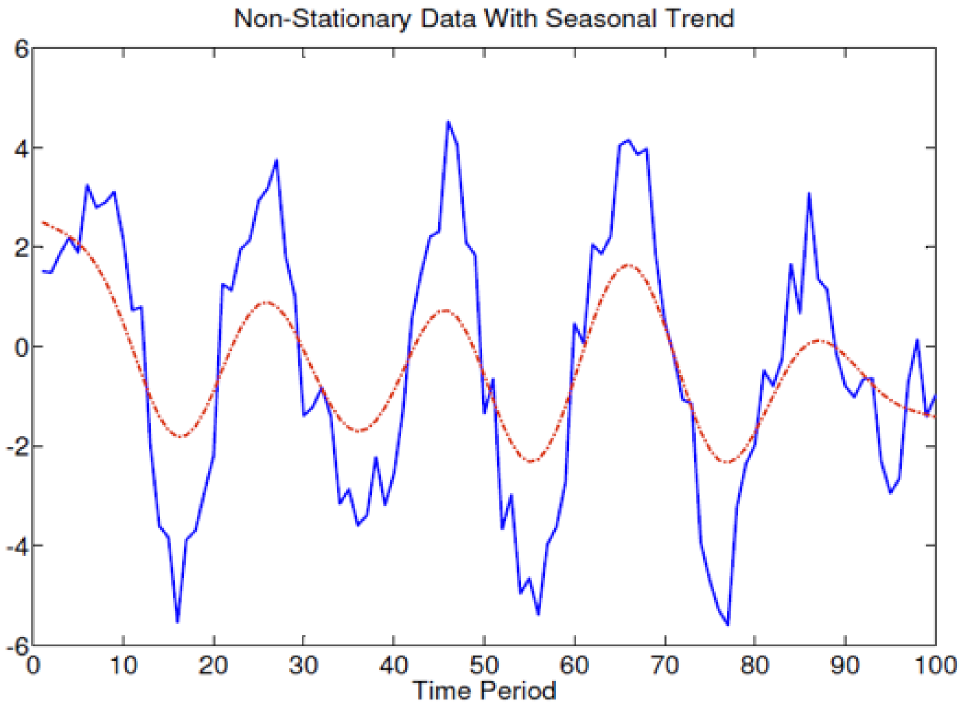
- Linear trends



- Non-linear trends



- Seasonal trends



- Model assumption

$$
\begin{aligned}
Y_t = \beta_1 &+ \beta_2 Y_{t-1} \\
&+ \beta_3 t + \beta_4 t^{\beta_5} \\
&+ \beta_6 sin\frac{2\pi}{s}t + \beta_7 cos\frac{2\pi}{s}t \\
&+ u_t
\end{aligned}
$$

2. Markov Process

- Joint distr4ibution can be factored into a series of conditional distributions

$$p(q_0, q_1, \ldots, q_T) = p(q_0)p(q_1 \mid q_0)p(q_2 \mid q_1, q_0) \cdots$$

- Markovian property

$$p(q_{t+1} \mid q_t, \cdots, q_0) = p(q_{t+1} \mid q_t)$$

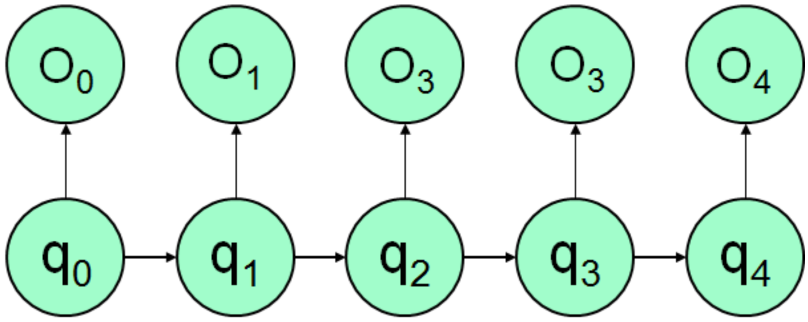
- Tractable in computation of joint distribution $$ \begin{align} p(q_0, q_1, \cdots, q_T) &= p(q_0)p(q_1 \mid q_0)p(q_2 \mid q_1,q_0) \cdots \

```
                    &= p(q_0)p(q_1 \mid q_0)p(q_2 \mid q_1)p(q_2 \mid q_2)
     \cdots
```

\end{align}$$

2.1. Hidden Markov Model (HMM)

- True state (or hidden variable) follows Markov chain
- Observation emitted from state



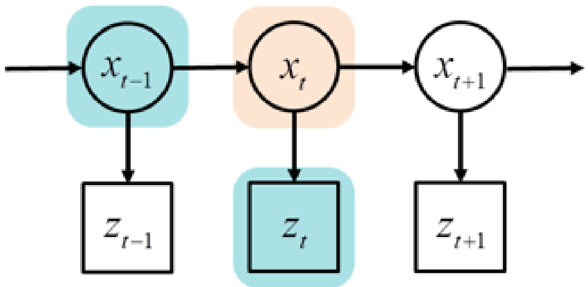
- Question : state estimation

$$\text{What is } p(q_t = s_i \mid O_1, O_2, \ldots, O_T)?$$

- HMM can do this, but with many difficulties

2.2. Kalman Filter

- Linear dynamicl system of motion



$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ z_t &= Cx_t \end{aligned}$$

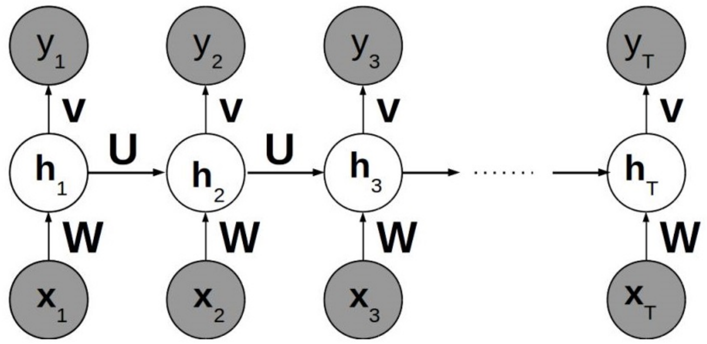
- A, B, C?

3. Recurrent Neural Network (RNN)

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

3.1. Feedforward Network and Sequential Data

- Separate parameters for each value of the time index
- Cannot share statistical strength across different time indices



In [1]:

```
%%html
<center><iframe src="https://www.youtube.com/embed/oYglxfBtSQk?rel=0"
width="560" height="315" frameborder="0" allowfullscreen></iframe></center>
```

3.2. Structure of RNN

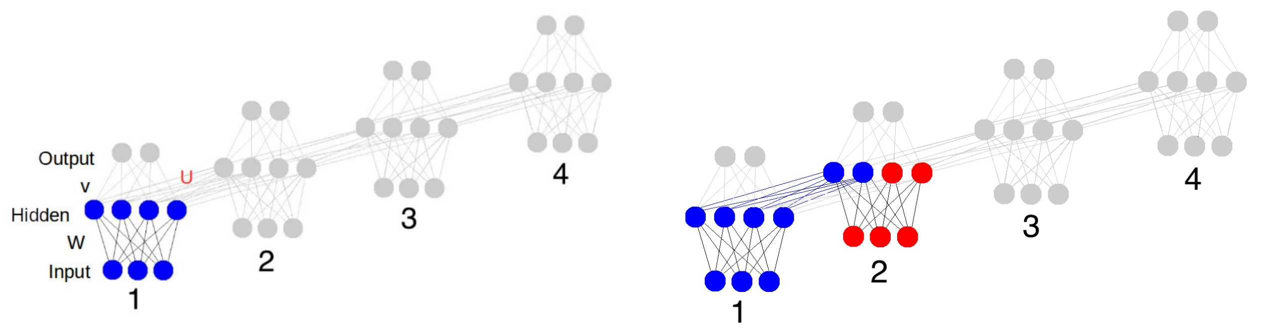
Recurrence

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

Hidden State

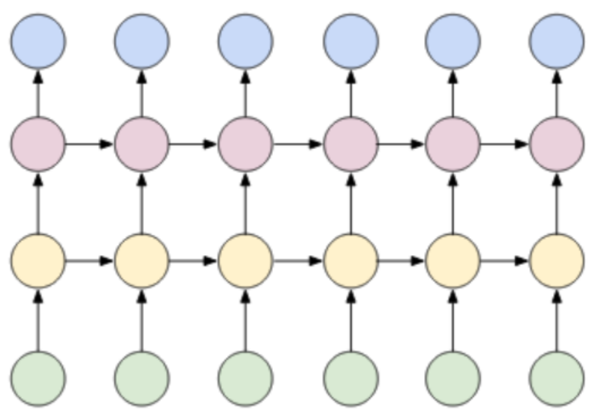
- Summary of the the past sequence of inputs up to t
- Keep some aspects of the past sequence with more precision than other aspects
- Network learns the function f

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



Deep Recurrent Networks

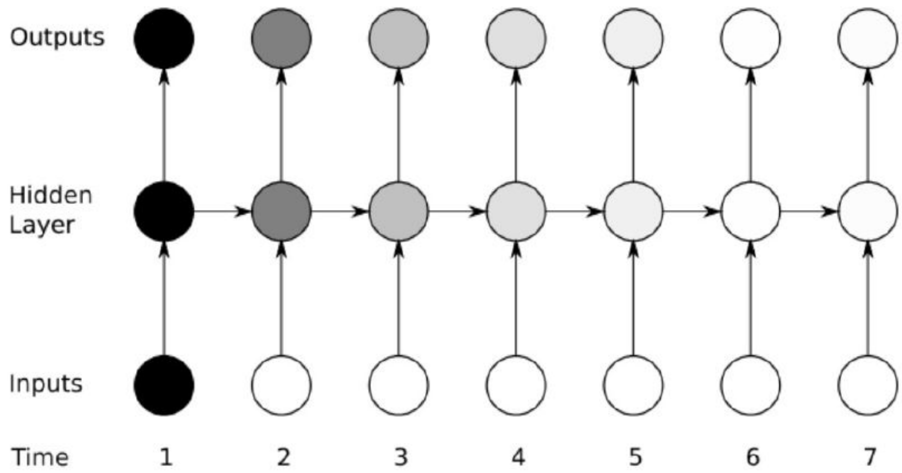
- Three blocks of parameters and associated transformation
    1. From the input to the hidden state (from green to yellow)
    2. From the previous hidden state to the next hidden state (from yellow to red)
    3. From the hidden state to the output (from red to blue)



3.3. RNN with LSTM

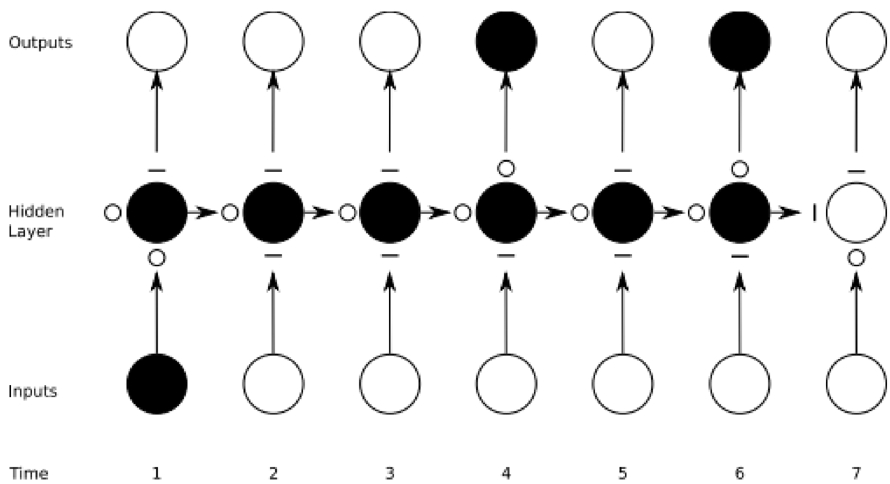
Long-Term Dependencies

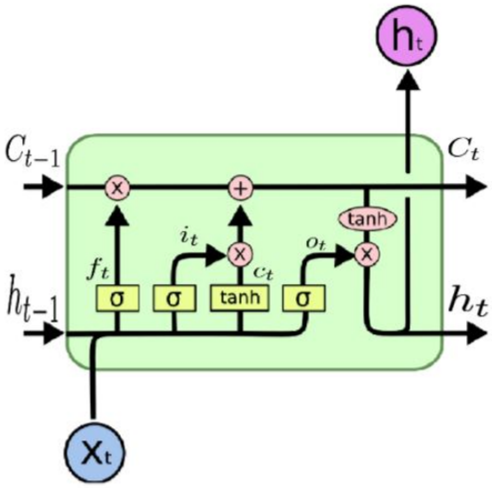
- Gradients propagated over many stages tend to either **vanish** or **explode**
- Difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions



Long Short-Term Memory (LSTM)

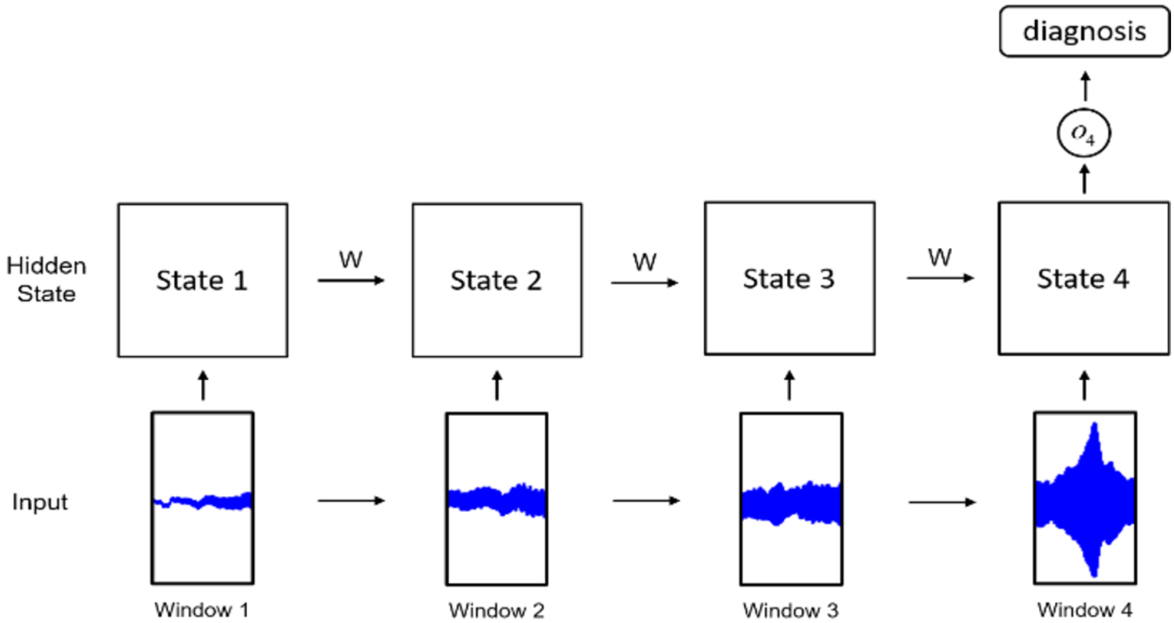
- Allow the network to **accumulate** information over a long duration
- Once that information has been used, it might be used for the neural network to **forget** the old state





$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Time series data and RNN

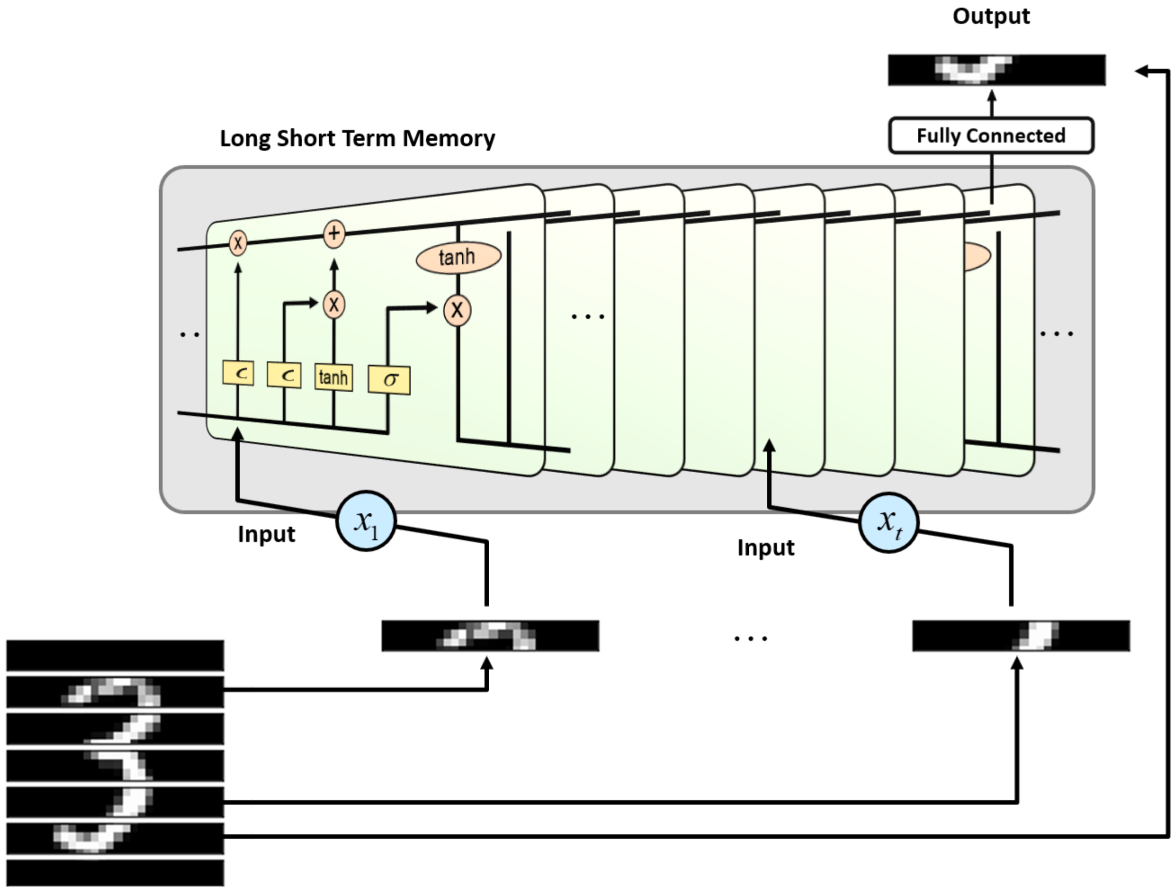


Summary

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

3.4. RNN and Sequential Data

Series Data **Prediction**



4. RNN with Tensorflow

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

4.1. Import Library

In [2]:

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

4.2. Load MNIST Data

- Download MNIST data from the tensorflow tutorial example

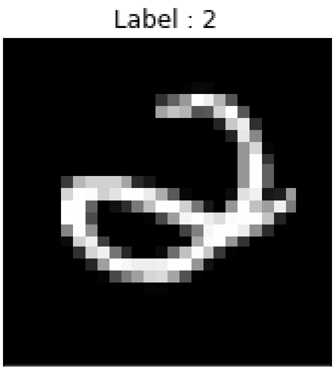
In [3]:

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

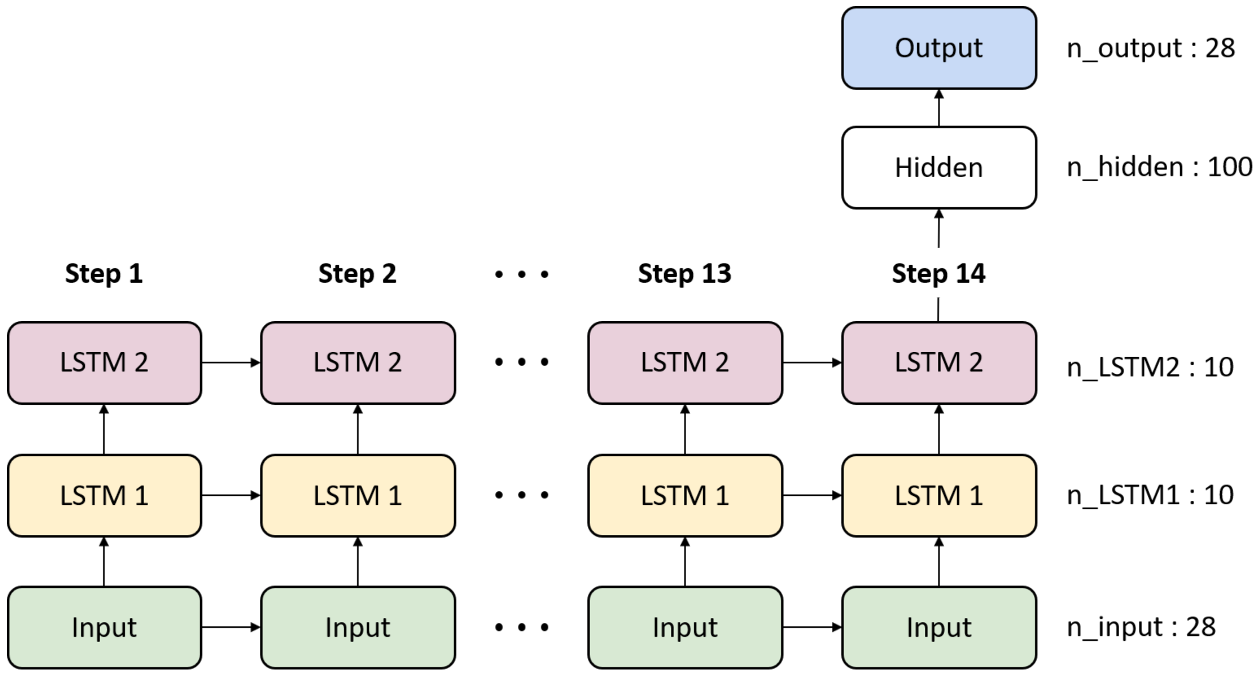
```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

In [4]:

```
# 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()
```



4.3. Define RNN Structure



In [5]:

```
n_step = 14
n_input = 28

## LSTM shape
n_lstm1 = 10
n_lstm2 = 10

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

4.4. Define 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 $\sigma = 0.01$

In [6]:

```
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])
```

4.5. Build a Model

Build the RNN Network

- 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)
```

In [7]:

```
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
```

4.6. Define 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

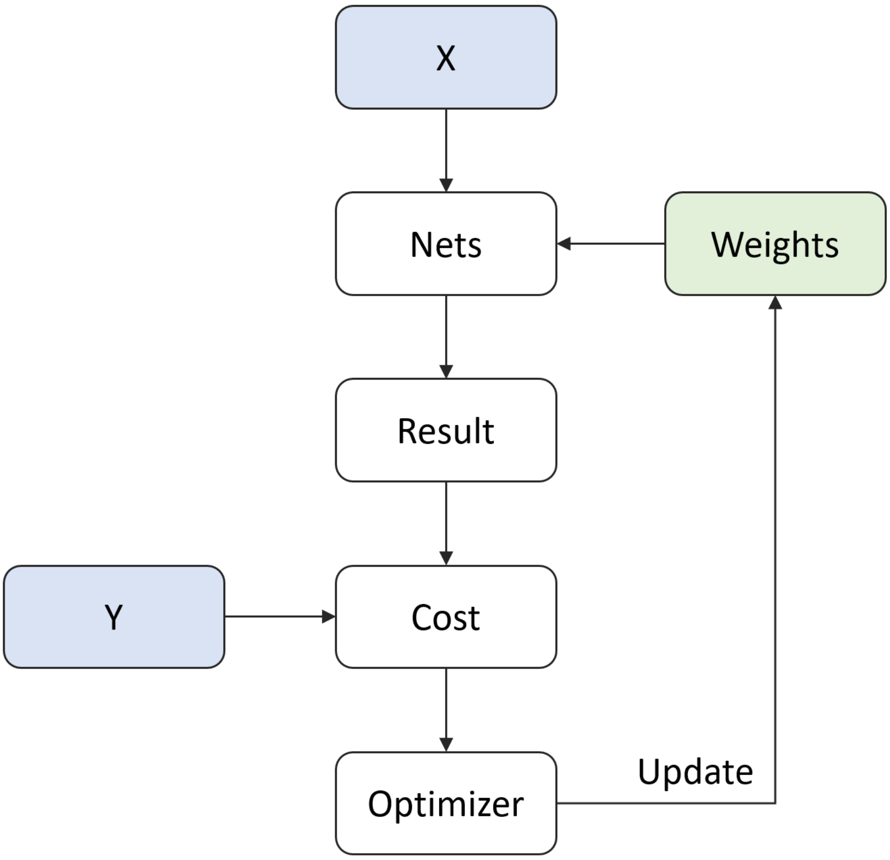
Optimizer

- AdamOptimizer: the most popular optimizer

In [8]:

```
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()
```

4.7. Summary of Model



4.8. Define Configuration

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

In [9]:

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

4.9. Optimization

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

In [ ]:

```
# 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))
```

4.10. Test

- 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 feeded into a model, then the model predict the last 14 x 28 image, recursively.

In [ ]:

```
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)

for i in range(n_step):
    sample = np.vstack([sample, gen_img[i]])

plt.imshow(test_x[idx], 'gray')
plt.title('Original Img')
plt.xticks([])
plt.yticks([])
plt.show()

plt.figure(figsize=(4,3))
plt.imshow(input_img, 'gray')
plt.title('Input')
plt.xticks([])
plt.yticks([])
plt.show()

plt.imshow(sample, 'gray')
plt.title('Generated Img')
plt.xticks([])
plt.yticks([])
plt.show()
```

5. 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`

In [10]:

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

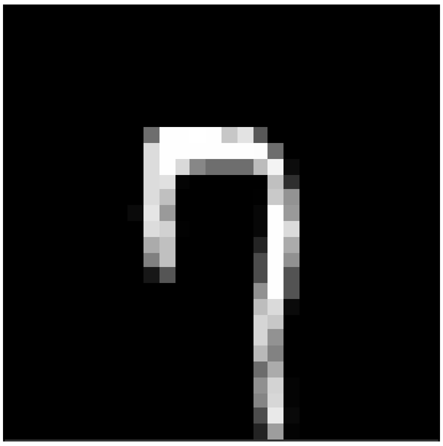
```
INFO:tensorflow:Restoring parameters from ./data_files/RNN_mnist/checkpoin
t/RNN_5000
Model loaded from file : ./data_files/RNN_mnist/checkpoint/RNN_5000
```

- Test with the pre-trained Model

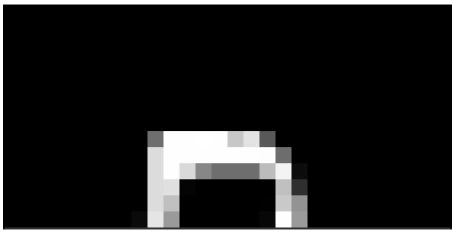
In [11]:

```
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)

plt.imshow(test_x[0], 'gray')
plt.title('Original Img')
plt.xticks([])
plt.yticks([])
plt.show()

plt.figure(figsize=(4,3))
plt.imshow(sample, 'gray')
plt.title('Input')
plt.xticks([])
plt.yticks([])
plt.show()

plt.imshow(gen_img, 'gray')
plt.title('Generated Img')
plt.xticks([])
plt.yticks([])
plt.show()
```

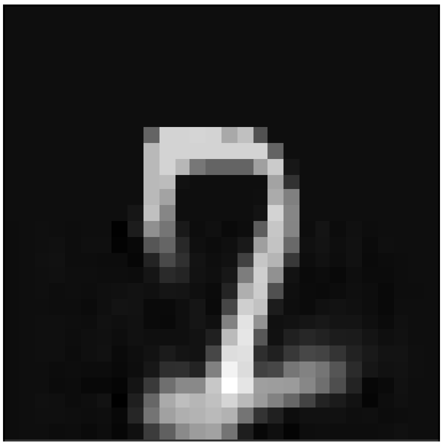
Original Img



Input



Generated Img



In [12]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc.js')
```