Recurrent Neural Network

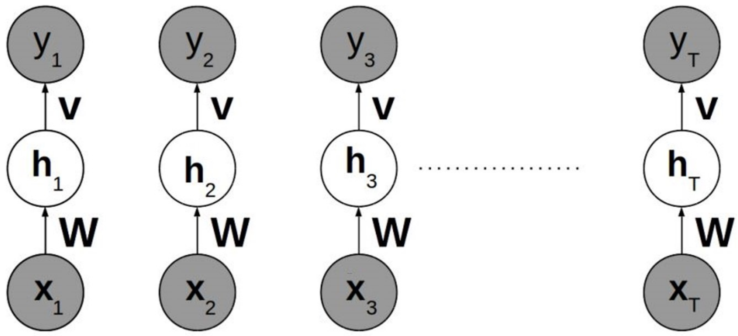
By Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

1. Recurrent Neural Network (RNN)

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

1.1. Feedforward Network and Sequential Data



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

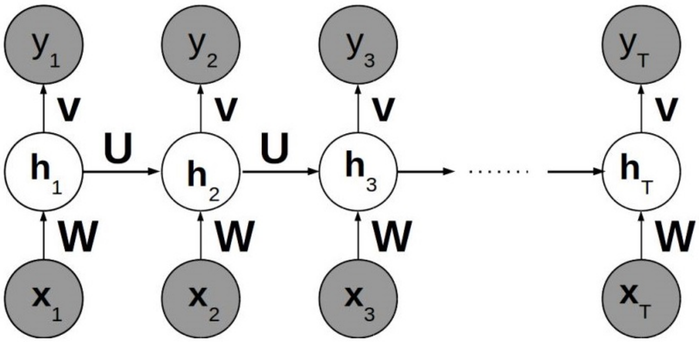
In [1]:

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

1.2. Structure of RNN

Recurrence

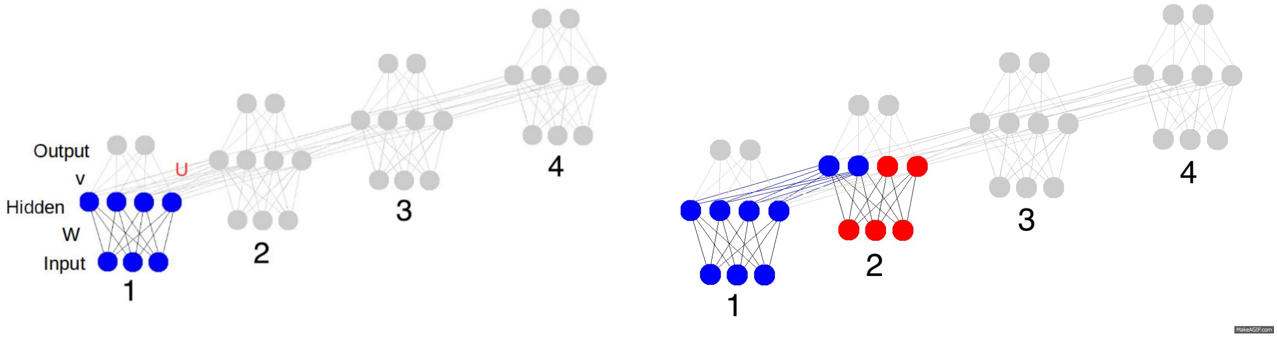
- It is possible to use the **same** transition function f with the same parameters at every time step



Hidden State

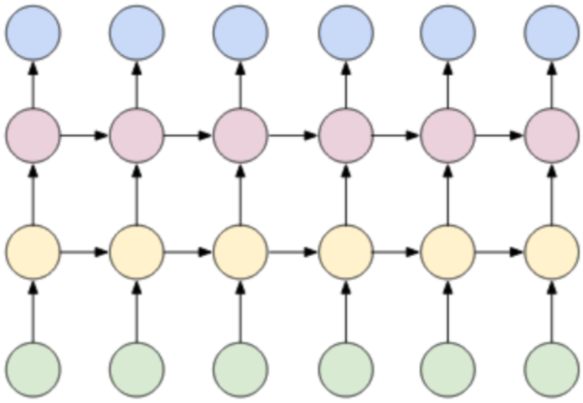
- Lossy summary of the the past sequence of inputs up to t
- Keep some aspects of the past sequence with more precision than other aspects
- Network learns the function f

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}\right)$$
$$f\left(h^{(t-1)}, x^{(t)}\right) = g\left(W x_t + U h_{t-1}\right)$$



Deep Recurrent Networks

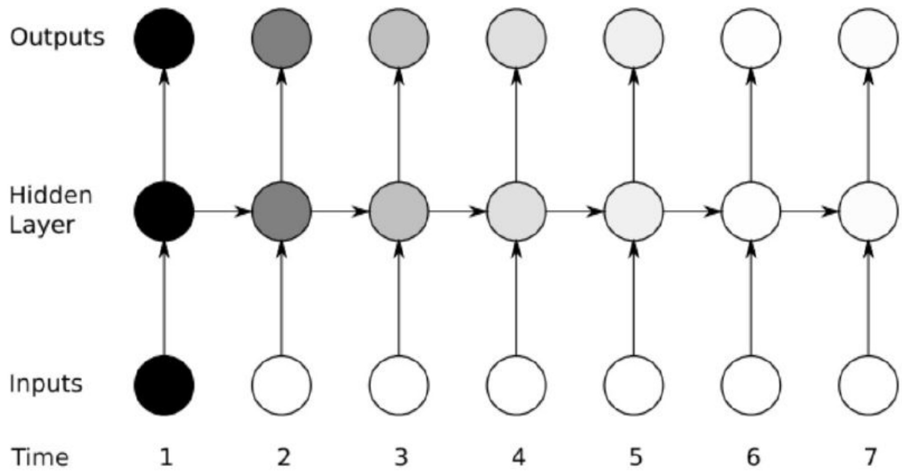
- Three blocks of parameters and associated transformation
  1. From the input to the hidden state (from green to yellow)
  2. From the previous hidden state to the next hidden state (from yellow to red)
  3. From the hidden state to the output (from red to blue)



1.3. RNN with LSTM

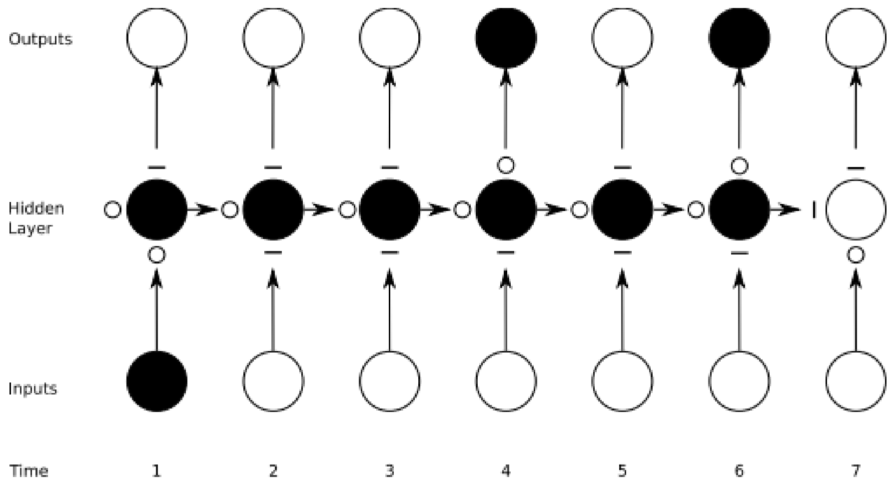
Long-Term Dependencies

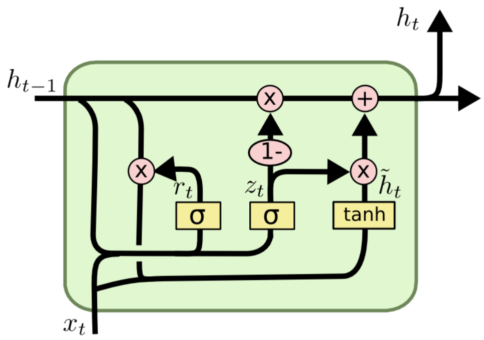
- Gradients propagated over many stages tend to either **vanish** or **explode**
- Difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions



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





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

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

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

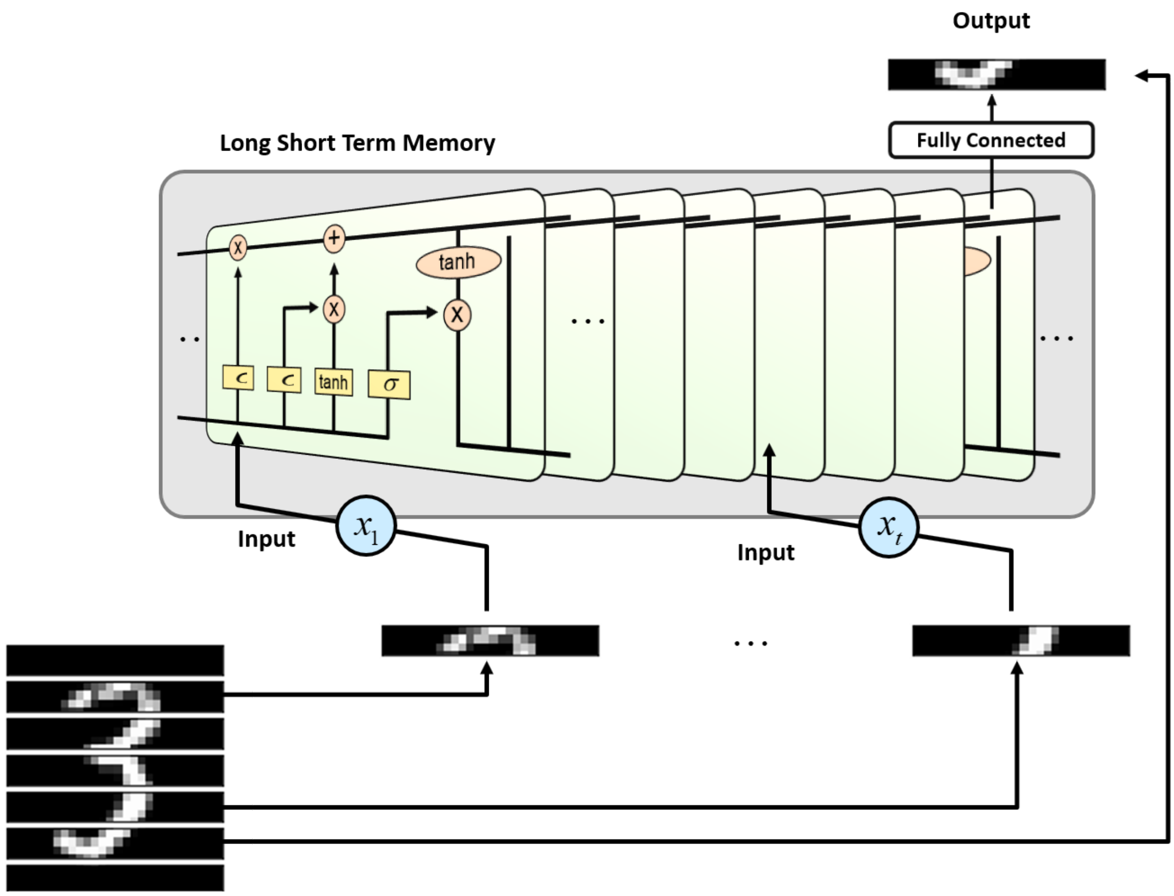
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Summary

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

1.4. RNN and Sequential Data

Series Data **Prediction**



2. RNN with Tensorflow

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

2.1. Import Library

In [2]:

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

2.2. Load MNIST Data

- Download MNIST data from the tensorflow tutorial example

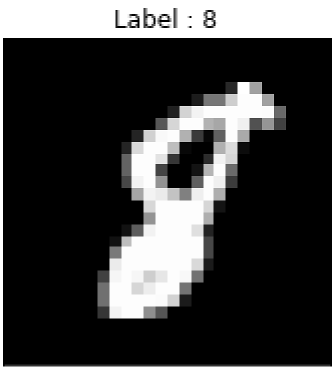
In [3]:

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

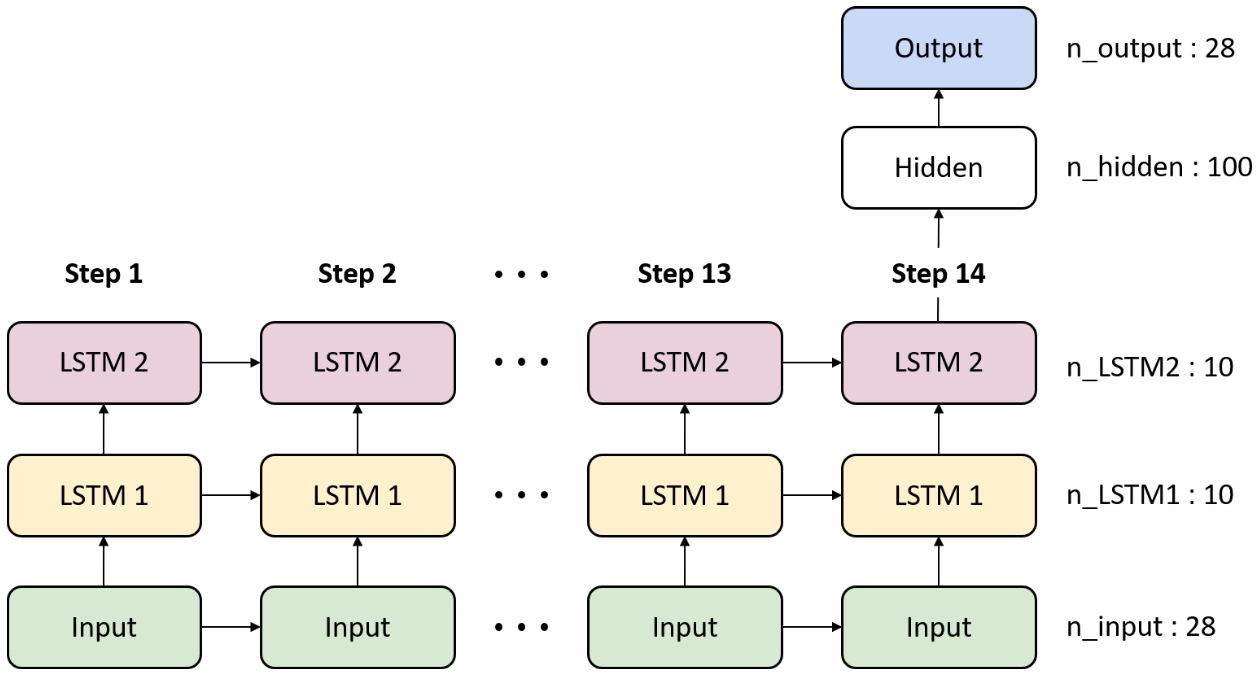
```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

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()
```



2.3. 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
```

2.4. 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])
```

2.5. 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
```

2.6. 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

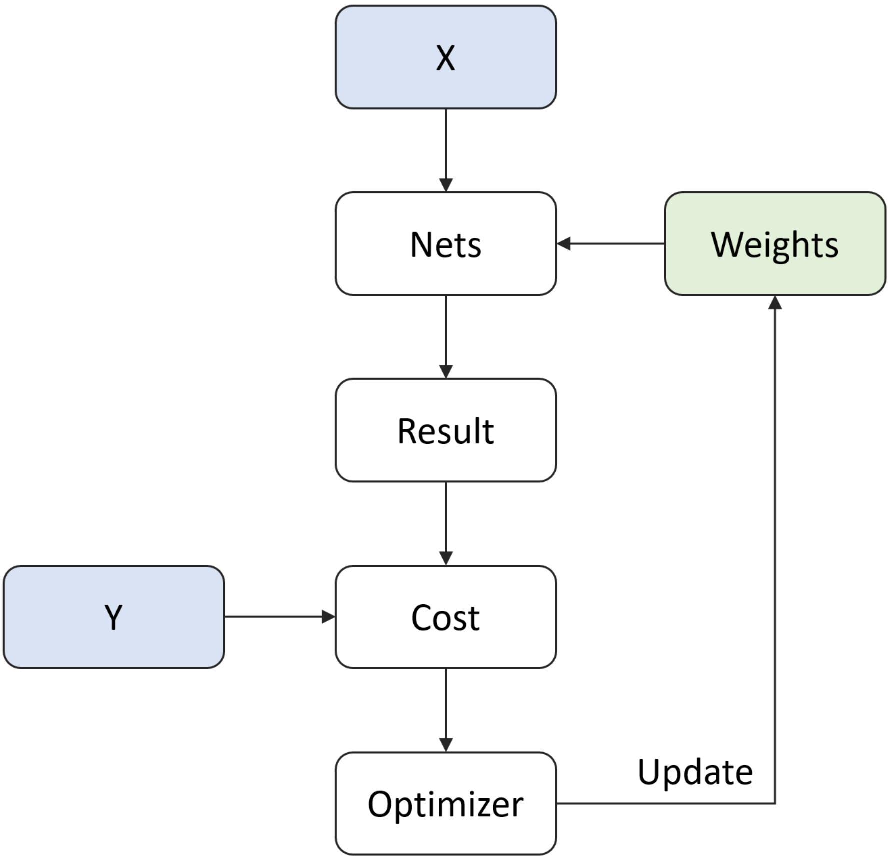
Optimizer

- 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()
```

2.7. Summary of Model



2.8. 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
```

2.9. Optimization

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

In [10]:

```
# 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))
```

```
Iter : 0
Cost : 0.00017322145868092775
Iter : 100
Cost : 0.0027603446505963802
Iter : 200
Cost : 0.0017704330384731293
Iter : 300
Cost : 0.0018281807424500585
Iter : 400
Cost : 0.0022316621616482735
Iter : 500
Cost : 0.0019235319923609495
Iter : 600
Cost : 0.0029685343615710735
Iter : 700
Cost : 0.00260598654858768
Iter : 800
Cost : 0.002004891401156783
Iter : 900
Cost : 0.00437586847692728
Iter : 1000
Cost : 0.0031971693970263004
Iter : 1100
Cost : 0.0011580168502405286
Iter : 1200
Cost : 0.0010057692416012287
Iter : 1300
Cost : 0.0005786378751508892
Iter : 1400
Cost : 0.000733629975002259
Iter : 1500
Cost : 0.0027604512870311737
Iter : 1600
Cost : 0.0014676948776468635
Iter : 1700
Cost : 0.0013189016608521342
Iter : 1800
Cost : 0.002196046058088541
Iter : 1900
Cost : 0.0012356654042378068
Iter : 2000
Cost : 0.0031192346941679716
Iter : 2100
Cost : 0.0004458320909179747
Iter : 2200
Cost : 0.00024697737535461783
Iter : 2300
Cost : 0.0025314786471426487
Iter : 2400
Cost : 0.001291859894990921
```

2.10. Test

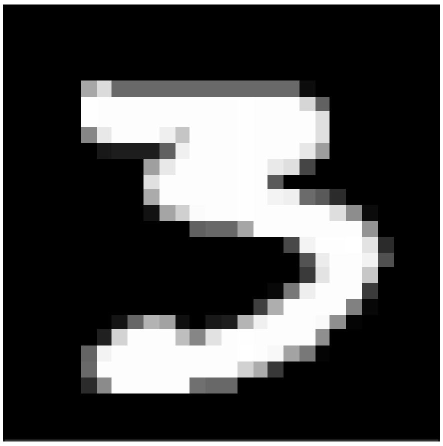
- Do not run on CPU. It will take quite a while.

- 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.

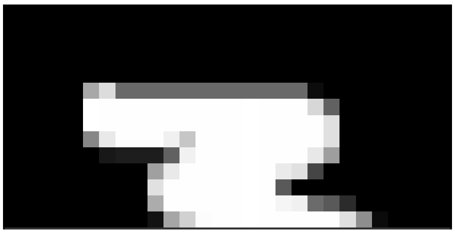
In [11]:

```
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()
```

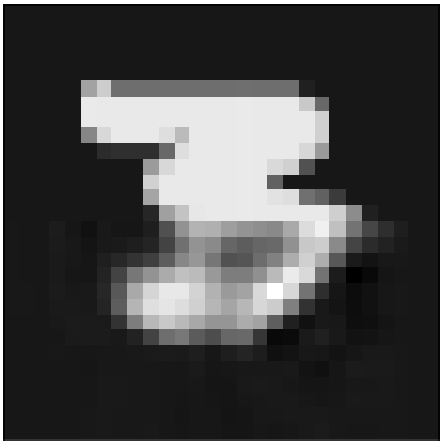
Original Img



Input



Generated Img



3. 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 [12]:

```
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
```

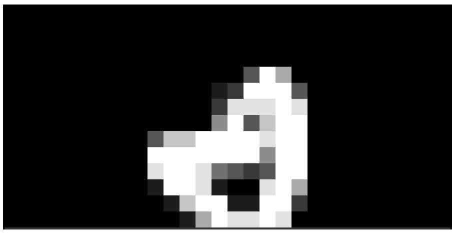
- Test with the pre-trained Model

In [13]:

```
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 [14]:

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