

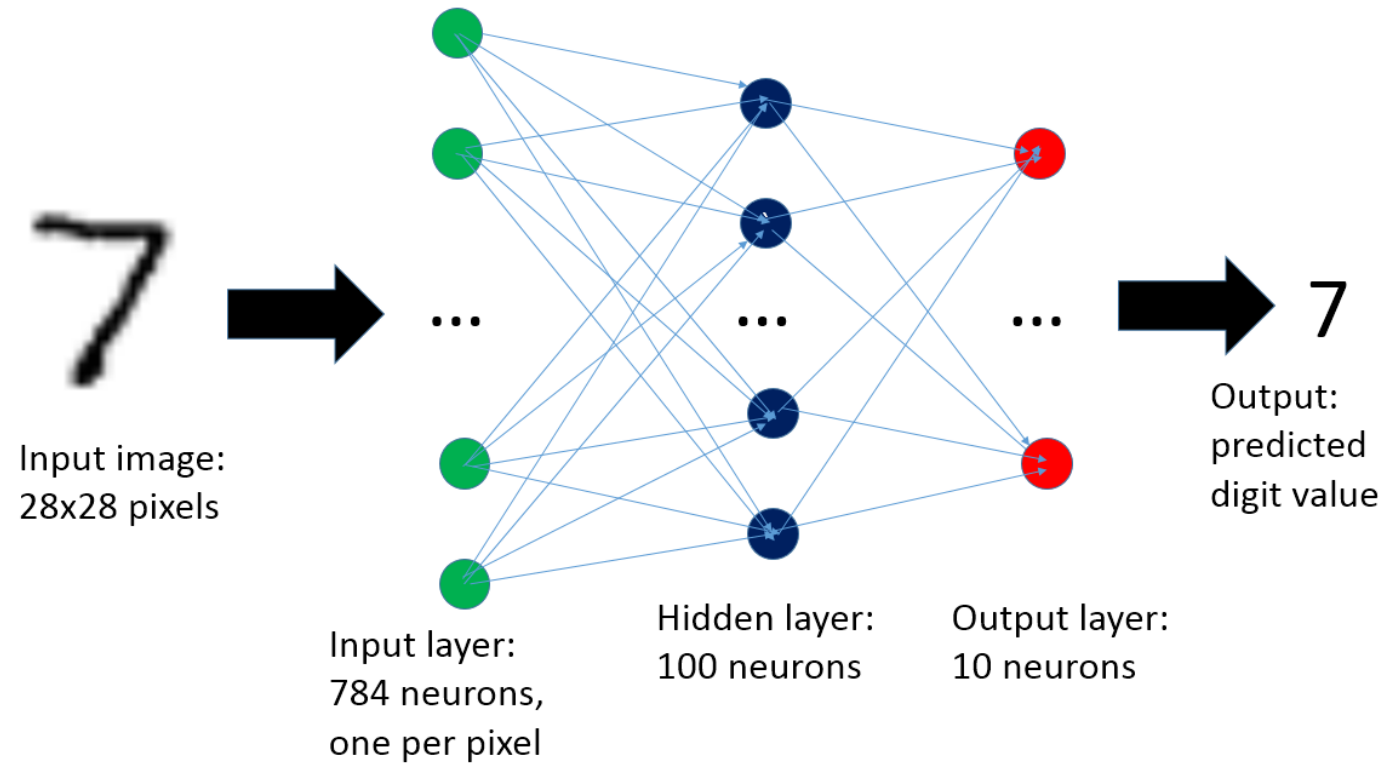


(Artificial) Neural Networks with TensorFlow

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

ANN in TensorFlow: MNIST

Our Network Model

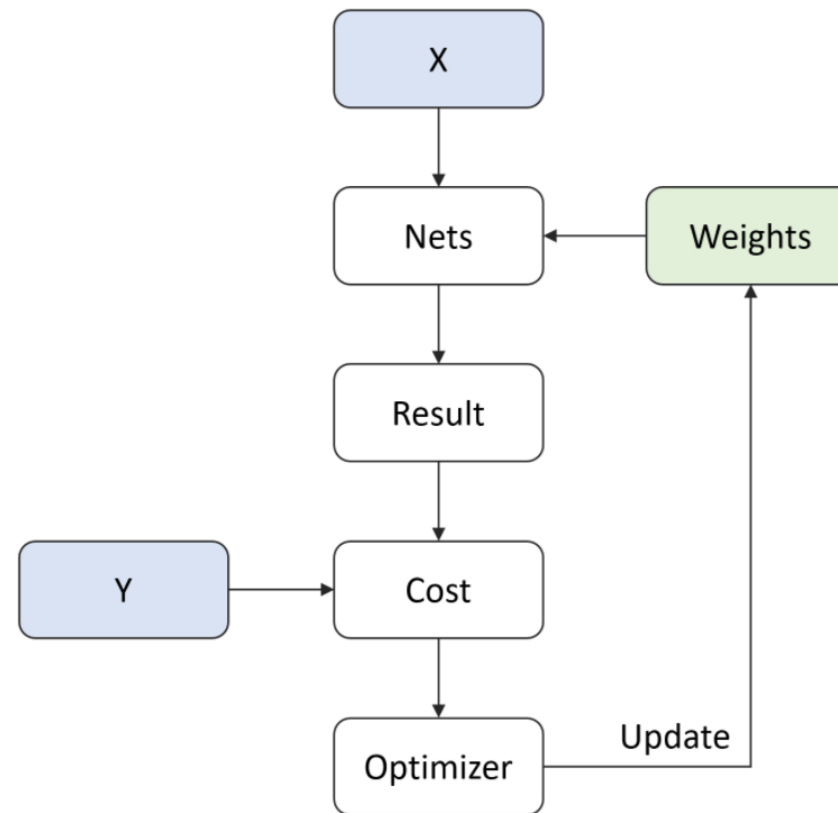


Iterative Optimization

- We will use
 - Mini-batch gradient descent
 - Adam optimizer

$$\begin{array}{ll} \min_{\theta} & f(\theta) \\ \text{subject to} & g_i(\theta) \leq 0 \end{array}$$

$$\theta := \theta - \alpha \nabla_{\theta} \left(h_{\theta} \left(x^{(i)} \right), y^{(i)} \right)$$



ANN with TensorFlow

- Import Library

```
# Import Library
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

- Load MNIST Data

- Download MNIST data from TensorFlow tutorial example

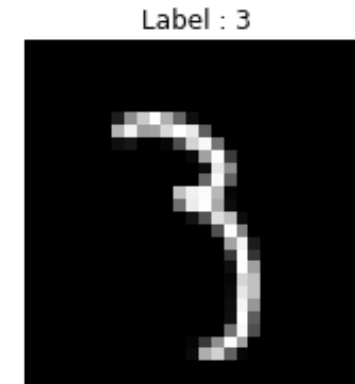
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

One Hot Encoding

- Batch maker

```
train_x, train_y = mnist.train.next_batch(1)
img = train_x[0,:].reshape(28,28)

plt.figure(figsize=(5,3))
plt.imshow(img, 'gray')
plt.title("Label : {}".format(np.argmax(train_y[0,:])))
plt.xticks([])
plt.yticks([])
plt.show()
```



- One hot encoding

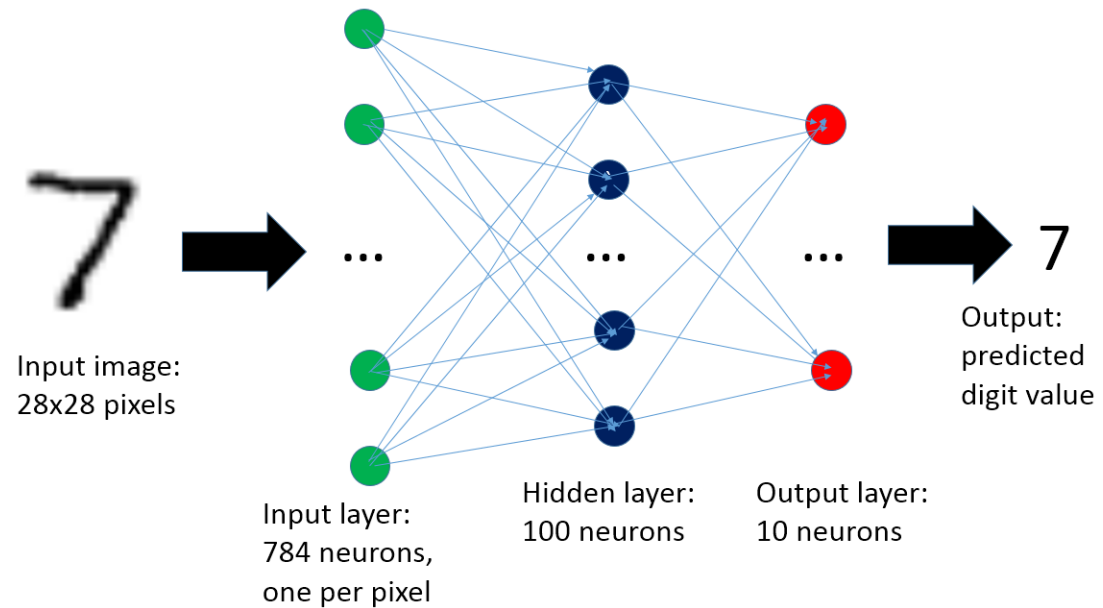
```
print ('Train labels : {}'.format(train_y[0, :]))
```

Train labels : [0. 0. 0. 1. 0. 0. 0. 0. 0.]

ANN Structure

- Input size
- Hidden layer size
- The number of classes

```
n_input = 28*28  
n_hidden = 100  
n_output = 10
```



Weights & Biases and Placeholder

- Define parameters based on predefined layer size
- Initialize with normal distribution with $\mu = 0$ and $\sigma = 0.1$

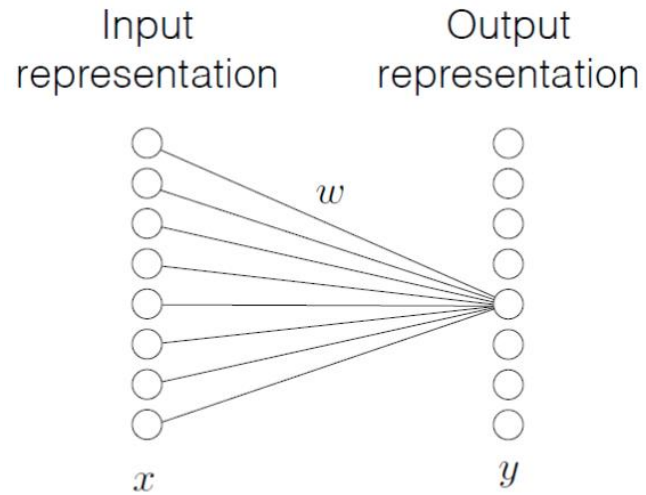
```
weights = {  
    'hidden' : tf.Variable(tf.random_normal([n_input, n_hidden], stddev = 0.1)),  
    'output' : tf.Variable(tf.random_normal([n_hidden, n_output], stddev = 0.1))  
}  
  
biases = {  
    'hidden' : tf.Variable(tf.random_normal([n_hidden], stddev = 0.1)),  
    'output' : tf.Variable(tf.random_normal([n_output], stddev = 0.1))  
}
```

- Placeholder

```
x = tf.placeholder(tf.float32, [None, n_input])  
y = tf.placeholder(tf.float32, [None, n_output])
```

Build a Model

- First, the layer performs several matrix multiplication to produce a set of linear activations



$$y_j = \left(\sum_i \omega_{ij} x_i \right) + b_j$$
$$y = \omega^T x + b$$

```
# Define Network
def build_model(x, weights, biases):

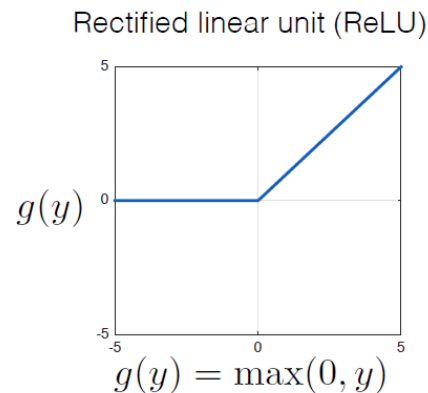
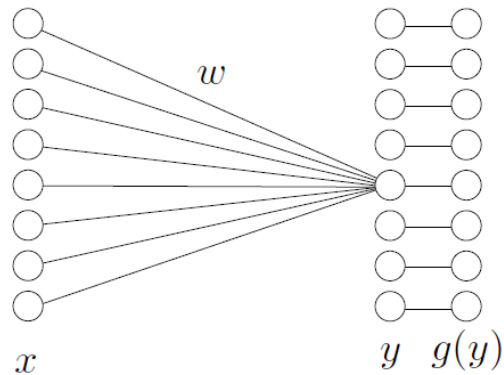
    # first hidden layer
    hidden = tf.add(tf.matmul(x, weights['hidden']), biases['hidden'])
    # non-linear activate function
    hidden = tf.nn.relu(hidden)

    # Output layer
    output = tf.add(tf.matmul(hidden, weights['output']), biases['output'])

    return output
```

Build a Model

- Second, each linear activation is running through a nonlinear activation function



```
# Define Network
def build_model(x, weights, biases):

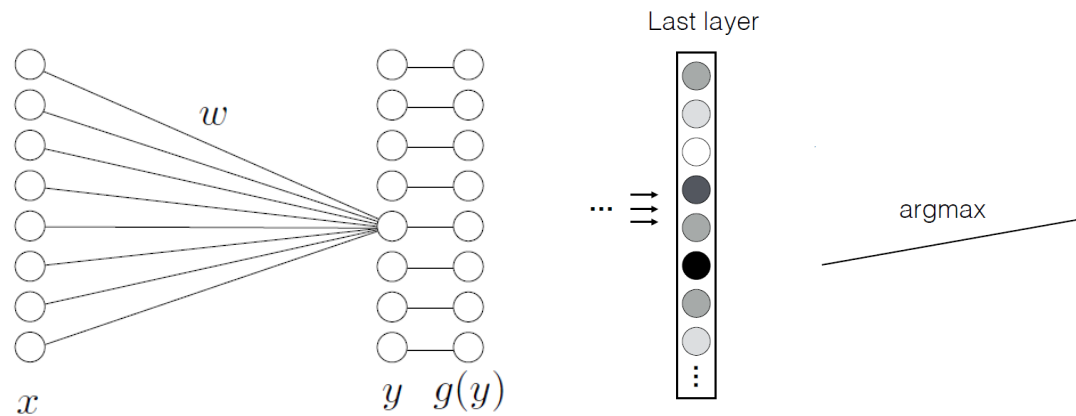
    # first hidden layer
    hidden = tf.add(tf.matmul(x, weights['hidden']), biases['hidden'])
    # non-linear activate function
    hidden = tf.nn.relu(hidden)

    # Output Layer
    output = tf.add(tf.matmul(hidden, weights['output']), biases['output'])

    return output
```

Build a Model

- Third, predict values with an affine transformation



```
# Define Network
def build_model(x, weights, biases):

    # first hidden layer
    hidden = tf.add(tf.matmul(x, weights['hidden']), biases['hidden'])
    # non-linear activate function
    hidden = tf.nn.relu(hidden)

    # Output layer
    output = tf.add(tf.matmul(hidden, weights['output']), biases['output'])

    return output
```

Loss and Optimizer

- Loss: softmax cross entropy

$$-\frac{1}{N} \sum_{i=1}^N y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))$$

- Optimizer
 - AdamOptimizer: the most popular optimizer

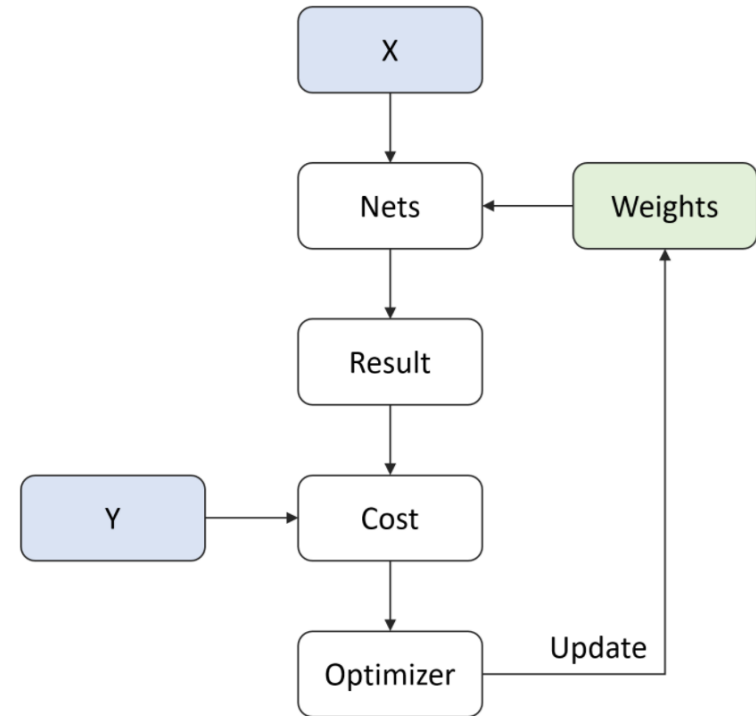
```
# Define Loss
pred = build_model(x, weights, biases)
loss = tf.nn.softmax_cross_entropy_with_logits(logits = pred, labels = y)
loss = tf.reduce_mean(loss)

LR = 0.0001
optm = tf.train.AdamOptimizer(LR).minimize(loss)
```

Iteration Configuration

- Define parameters for training ANN
 - n_batch: batch size for mini-batch gradient descent
 - n_iter: the number of iteration steps
 - n_prt: check loss for every n_prt iteration

```
n_batch = 50      # Batch Size  
n_iter = 5000     # Learning Iteration  
n_prt = 250       # Print Cycle
```



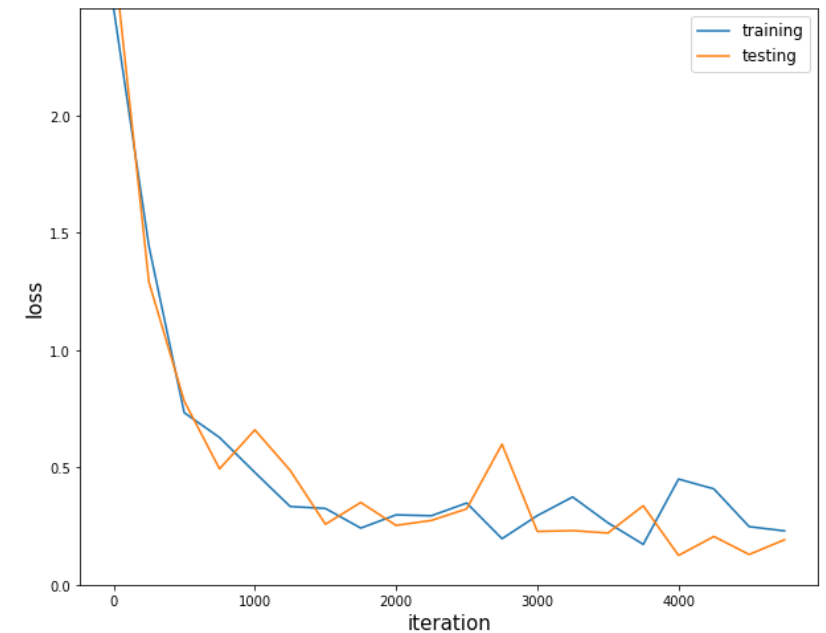
Optimization

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

loss_record_training = []
loss_record_testing = []
for epoch in range(n_iter):
    train_x, train_y = mnist.train.next_batch(n_batch)
    sess.run(optm, feed_dict = {x: train_x, y: train_y})

    if epoch % n_prt == 0:
        test_x, test_y = mnist.test.next_batch(n_batch)
        c1 = sess.run(loss, feed_dict = {x: train_x, y: train_y})
        c2 = sess.run(loss, feed_dict = {x: test_x, y: test_y})
        loss_record_training.append(c1)
        loss_record_testing.append(c2)
        print ("Iter : {}".format(epoch))
        print ("Cost : {}".format(c1))

plt.figure(figsize=(10,8))
plt.plot(np.arange(len(loss_record_training))*n_prt,
         loss_record_training, label = 'training')
plt.plot(np.arange(len(loss_record_testing))*n_prt,
         loss_record_testing, label = 'testing')
plt.xlabel('iteration', fontsize = 15)
plt.ylabel('loss', fontsize = 15)
plt.legend(fontsize = 12)
plt.ylim([0,np.max(loss_record_training)])
plt.show()
```



Test or Evaluation

```
test_x, test_y = mnist.test.next_batch(100)

my_pred = sess.run(pred, feed_dict = {x : test_x})
my_pred = np.argmax(my_pred, axis = 1)

labels = np.argmax(test_y, axis = 1)

accr = np.mean(np.equal(my_pred, labels))
print("Accuracy : {}".format(accr*100))
```

Accuracy : 96.0%

```
test_x, test_y = mnist.test.next_batch(1)
logits = sess.run(tf.nn.softmax(pred), feed_dict = {x : test_x})
predict = np.argmax(logits)

plt.imshow(test_x.reshape(28,28), 'gray')
plt.xticks([])
plt.yticks([])
plt.show()

print('Prediction : {}'.format(predict))
np.set_printoptions(precision=2, suppress=True)
print('Probability : {}'.format(logits.ravel()))
```

```
Prediction : 4
Probability : [0.  0.  0.  0.  0.9  0.  0.  0.01 0.  0.09]
```

