(Artificial) Neural Networks with TensorFlow

Industrial AI Lab.

Prof. Seungchul Lee

Training Neural Networks: Deep Learning Libraries

- Caffe
  - Platform: Linux, Mac OS, Windows
  - Written in: C++
  - Interface: Python, MATLAB
- Theano
  - Platform: Cross-platform
  - Written in: Python
  - Interface: Python
- Tensorflow
  - Platform: Linux, Mac OS, Windows
  - Written in: C++, Python
  - Interface: Python, C/C++, Java, Go, R

TensorFlow: Constant

- TensorFlow is an open-source software library for deep learning
  - tf.constant
  - tf.Variable
  - tf.placeholder

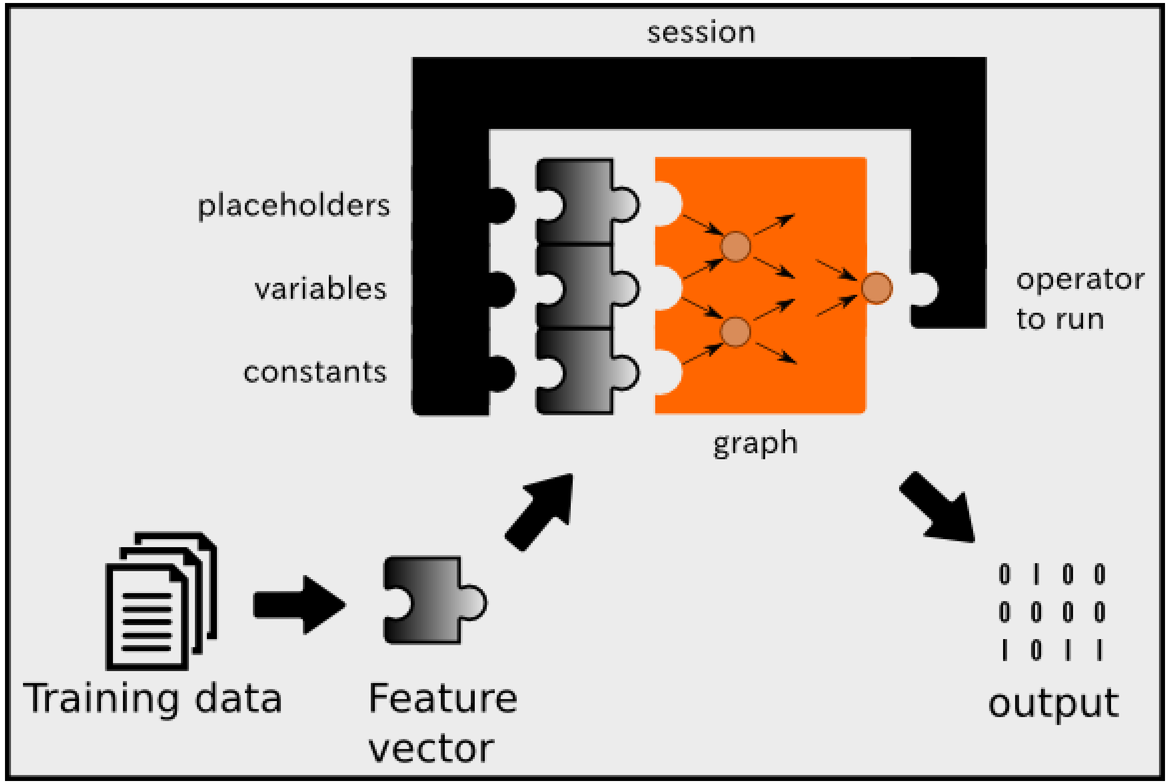
```
import tensorflow as tf

a = tf.constant([1, 2, 3])
b = tf.constant([4, 5, 6])

A = a + b
B = a * b
```

```
A
```

```
<tf.Tensor 'add:0' shape=(3,) dtype=int32>
```

TensorFlow: Session

- To run any of the three defined operations, we need to create a session for that graph. The session will also allocate memory to store the current value of the variable.



TensorFlow

```
sess = tf.Session()
sess.run(A)
```

```
array([5, 7, 9], dtype=int32)
```

```
sess.run(B)
```

```
array([ 4, 10, 18], dtype=int32)
```

TensorFlow: Initialization

- tf.Variable is regarded as the decision variable in optimization.
- We should initialize variables to use tf.Variable.

```
w = tf.Variable([1, 1])
```

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

```
sess.run(w)
```

```
array([1, 1], dtype=int32)
```

TensorFlow: Placeholder

- The value of tf.placeholder must be fed using the feed_dict optional argument to Session.run()

```
x = tf.placeholder(tf.float32, [2, 2])
```

```
sess.run(x, feed_dict={x : [[1,2],[3,4]]})
```

```
array([[ 1.,  2.],
       [ 3.,  4.]], dtype=float32)
```

Training Neural Networks: Optimization

- Learning or estimating weights and biases of multi-layer perceptron from training data

- 3 key components
  - objective function $f(\cdot)$
  - decision variable or unknown θ
  - constraints $g(\cdot)$

- In mathematical expression

$$\begin{aligned} \min_{\theta} \quad & f(\theta) \\ \text{subject to} \quad & g_i(\theta) \leq 0, \qquad i = 1, \cdots, m \end{aligned}$$

Training Neural Networks: Loss Function

- Measures error between target values and predictions

$$\min_{\theta} \sum_{i=1}^{m} \ell\left(h_{\theta}\left(x^{(i)}\right), y^{(i)}\right)$$

- Example
  - Squared loss (for regression):

$$\frac{1}{N} \sum_{i=1}^{N} \left(h_{\theta}\left(x^{(i)}\right) - y^{(i)}\right)^{2}$$

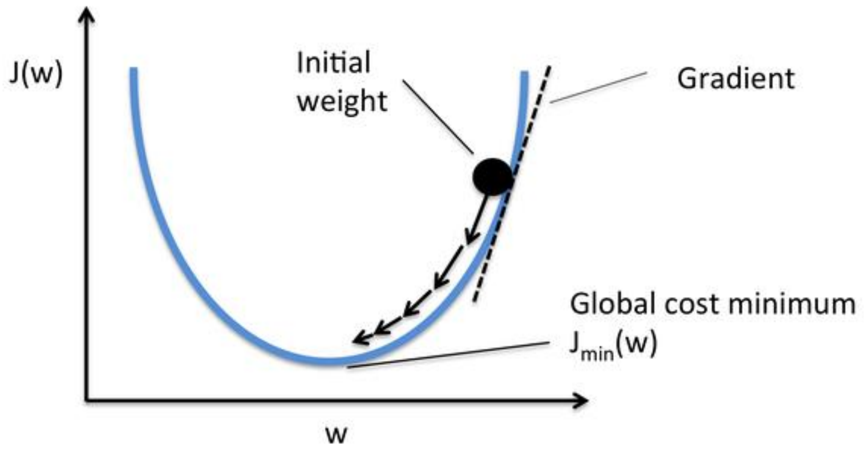
  - Cross entropy (for classification):

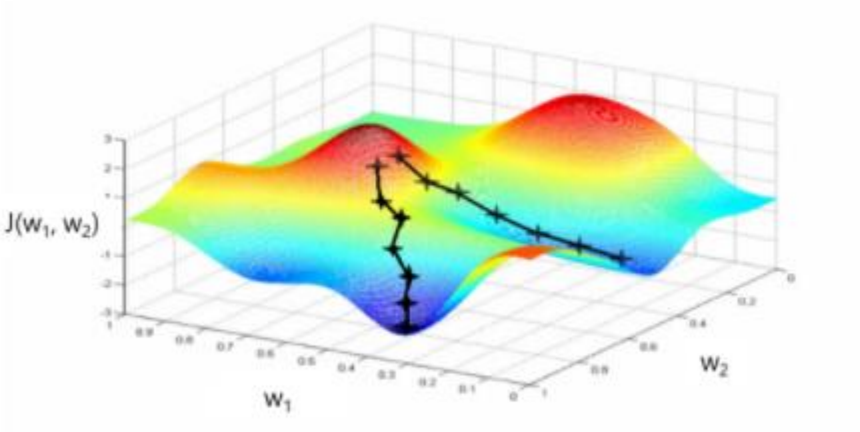
$$-\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log\left(h_{\theta}\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_{\theta}\left(x^{(i)}\right)\right)$$

Training Neural Networks: Gradient Descent

- Negative gradients points directly downhill of the cost function
- We can decrease the cost by moving in the direction of the negative gradient (α is a learning rate)

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

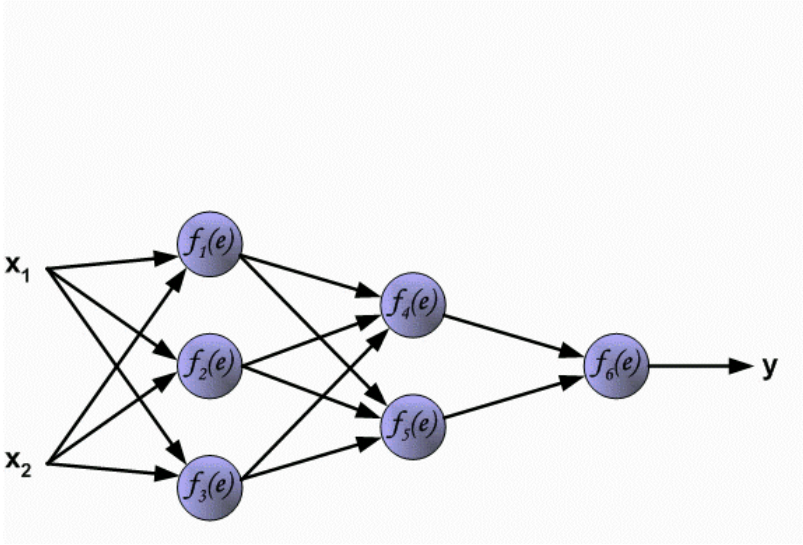




Training Neural Networks: Learning

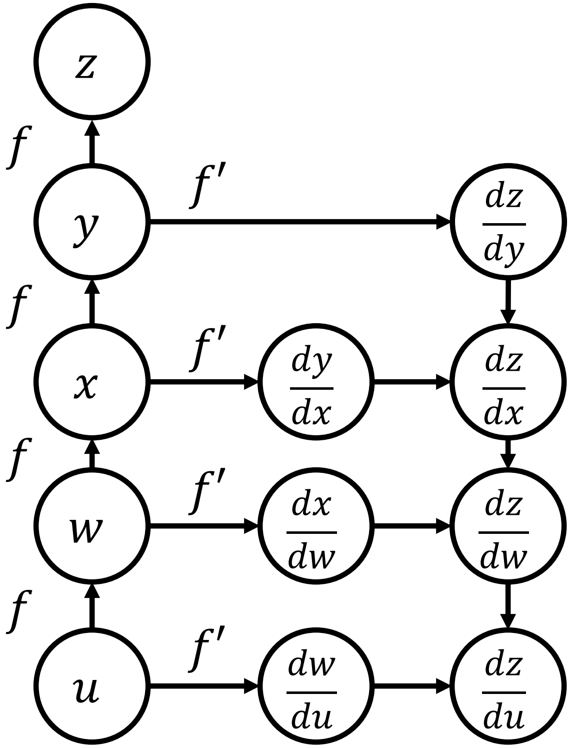
Backpropagation

- Forward propagation
  – the initial information propagates up to the hidden units at each layer and finally produces output
- Backpropagation
  – allows the information from the cost to flow backwards through the network in order to compute the gradients



Backpropagation

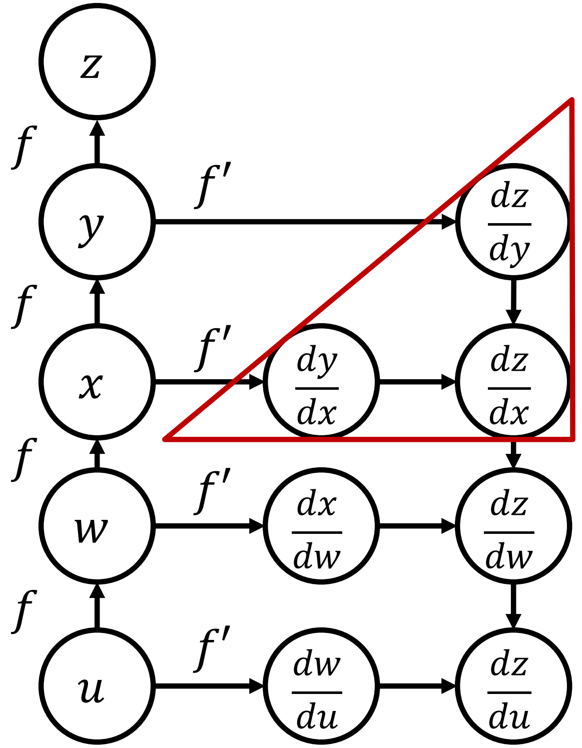
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

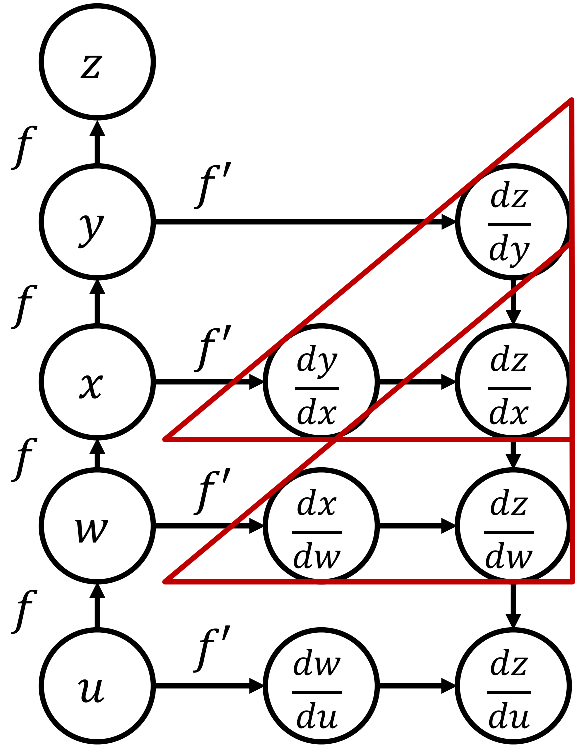
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

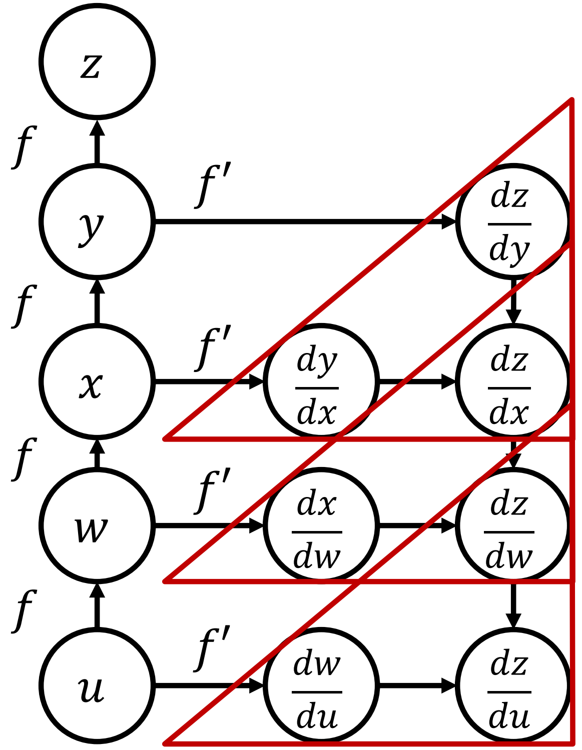
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$

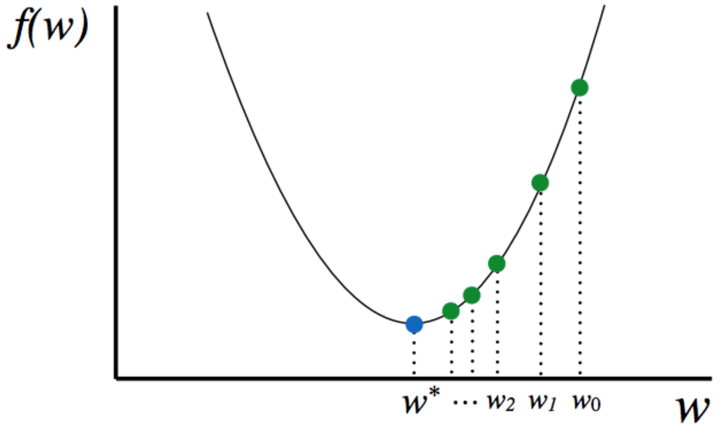


- Backpropagation
  - Update weights recursively

Training Neural Networks

- Optimization procedure

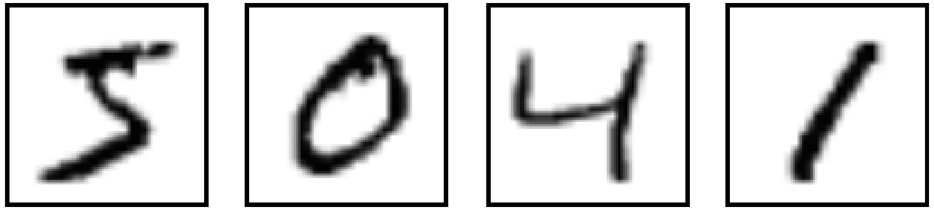
Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied

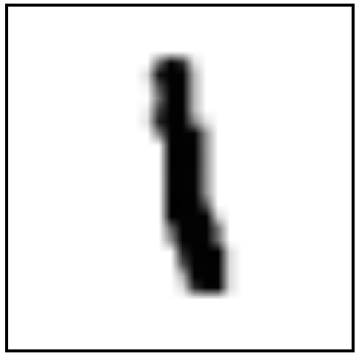


- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools

ANN with MNIST

- MNIST database
  - Mixed National Institute of Standards and Technology database
  - Handwritten digit database
  - 28×28 gray scaled image
  - Flattened matrix into a vector of 28×28=784

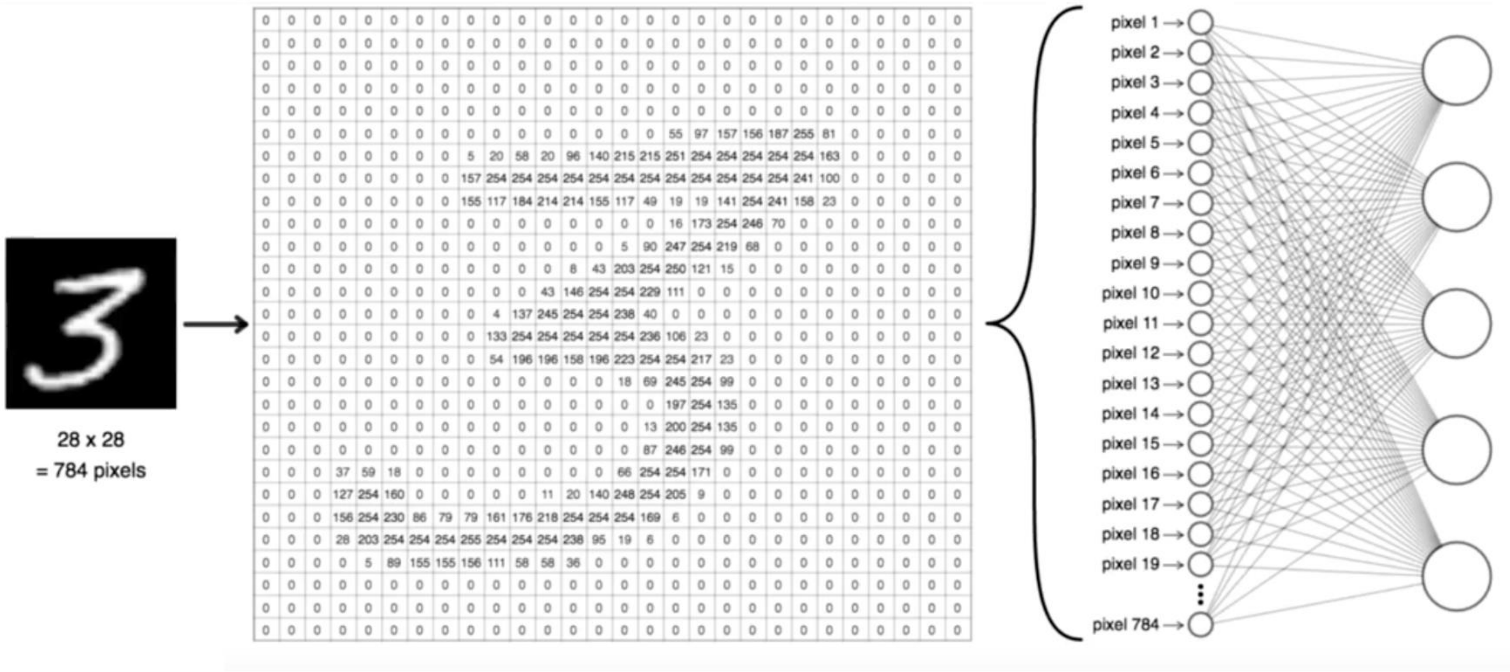




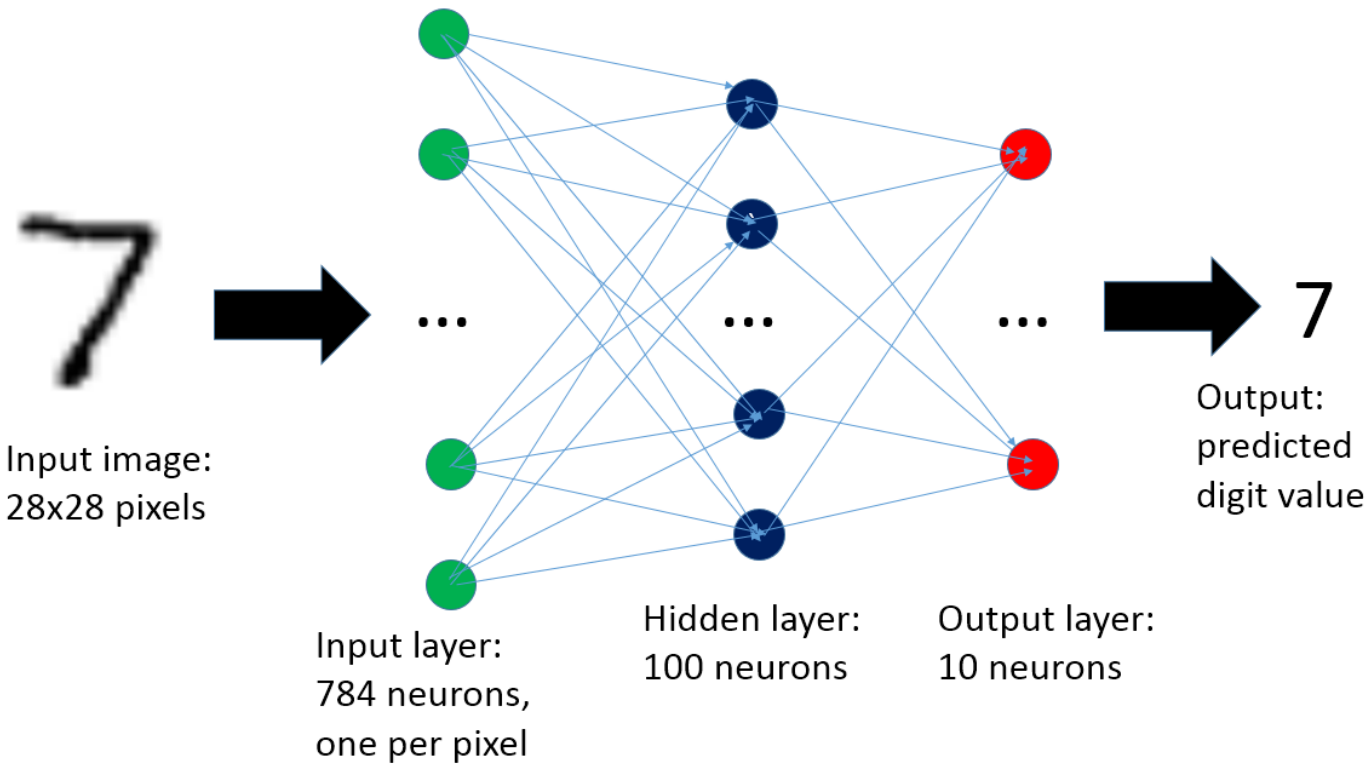
$$\cong \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

ANN with TensorFlow

- Feed a gray image to ANN



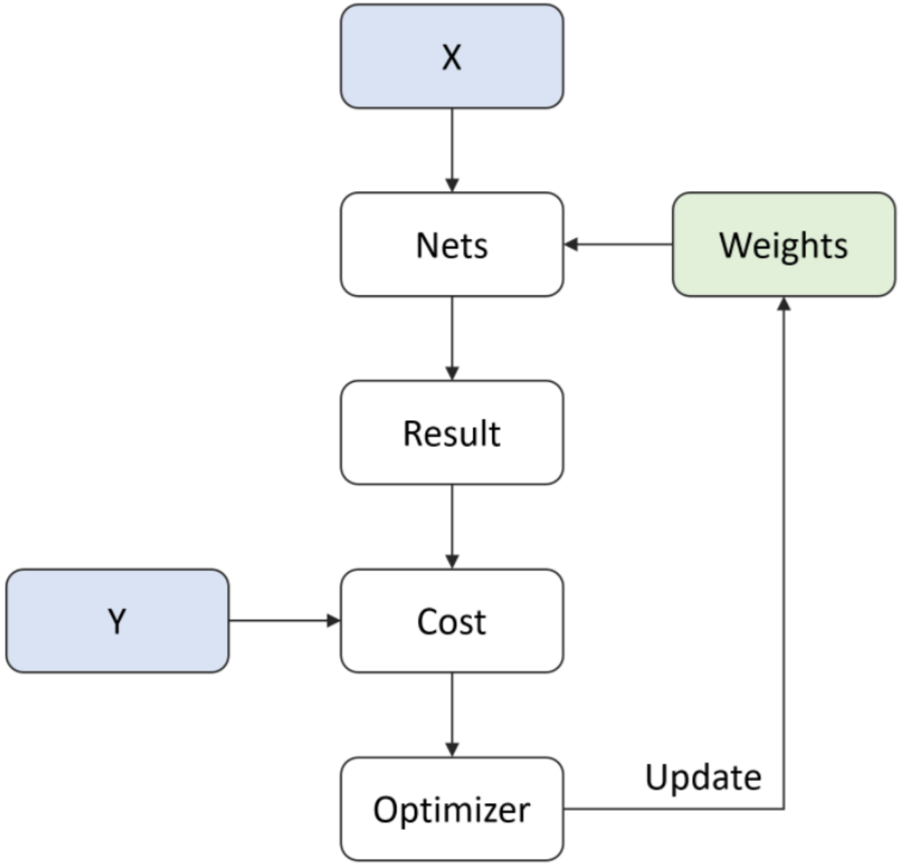
Our Network Model



Iterative Optimization

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

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



Mini-batch Gradient Descent

① Linear Regression Cost function $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^i - y^i)^2$

m training data

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)\cdot x_j^i$$

② Vanilla (Batch) G.D.

$$\theta_j := \theta_j - \alpha\cdot\underbrace{\frac{\partial}{\partial\theta_j}J(\theta)}_{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)x_j^i}$$

③ Stochastic G.D.

for i in range(m):

$$\theta_j := \theta_j - \alpha\cdot\boxed{\text{only one example}}$$

$$(\hat{y}^i - y^i)x_j^i$$

④ Mini-batch gradient descent uses *n* data batch at each iteration

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

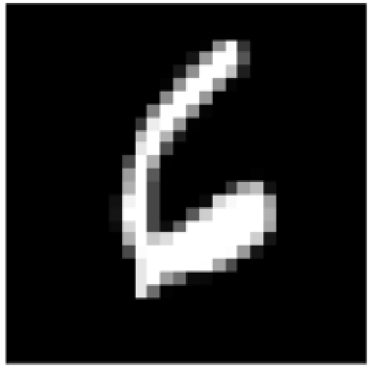
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
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
```

One Hot Encoding

```
train_x, train_y = mnist.train.next_batch(10)
img = train_x[3,:].reshape(28,28)

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

Label : 6



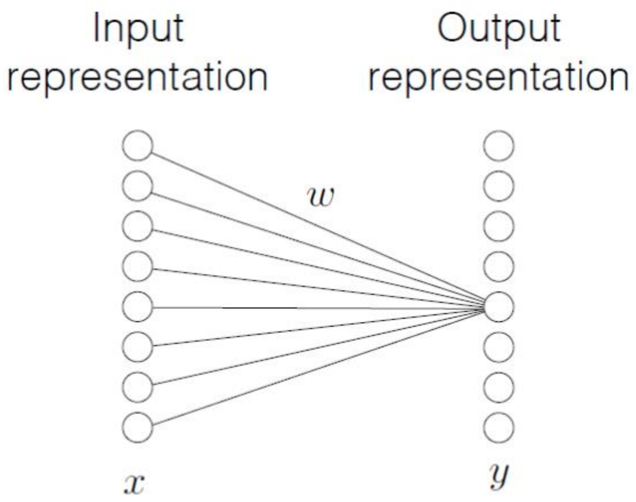
- One hot encoding

```
print ('Train labels : {}'.format(train_y[3, :]))
```

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

Build a Model

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



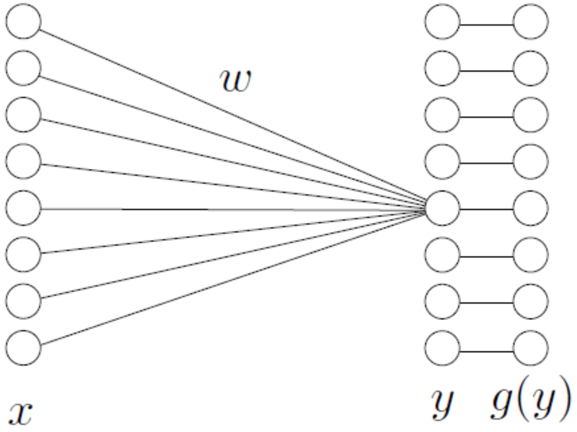
$$y_j = \left(\sum_i \omega_{ij} x_i \right) + b_j$$

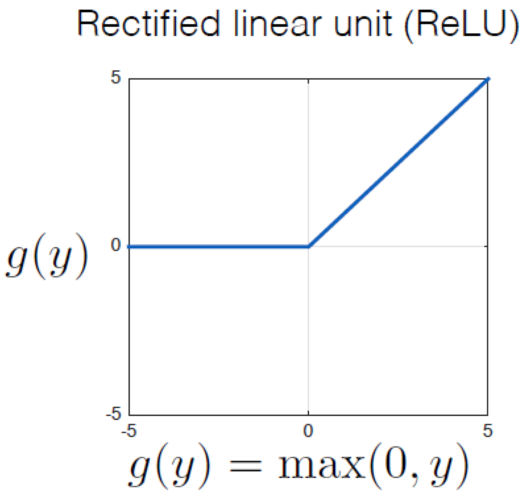
$$y = \omega^T x + b$$

```
# hidden1 = tf.matmul(x, weights['hidden1']) + biases['hidden1']
hidden1 = tf.add(tf.matmul(x, weights['hidden1']), biases['hidden1'])
```

Build a Model

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



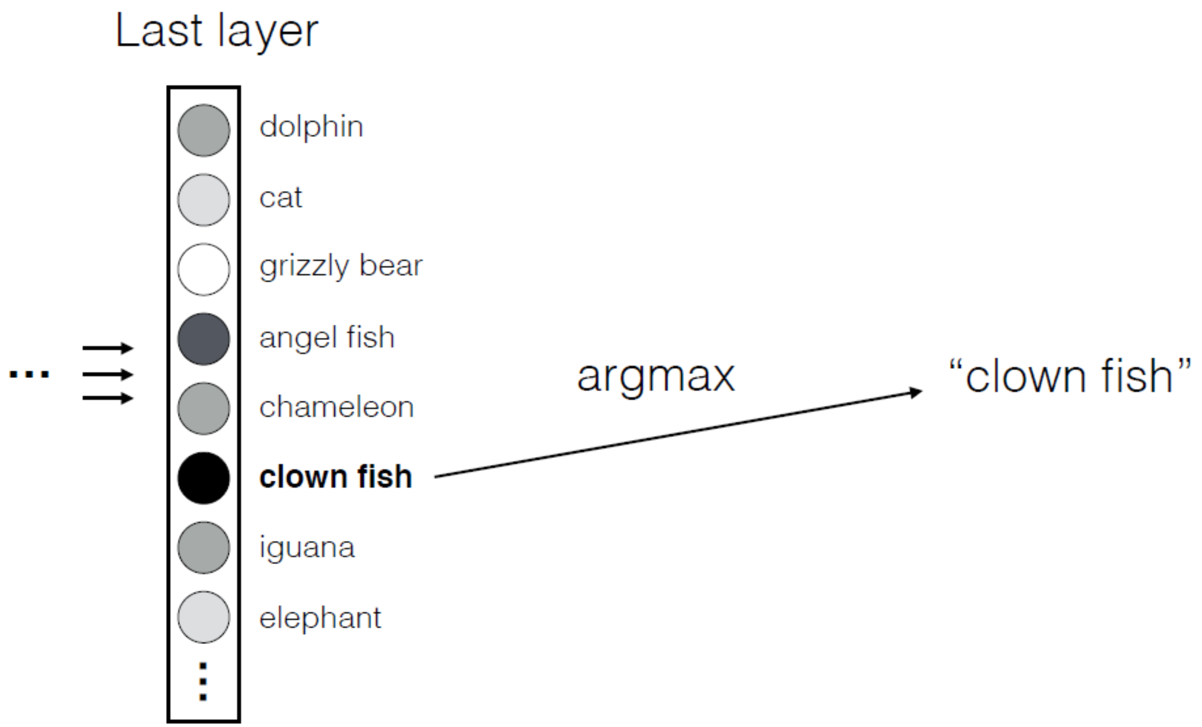



```
hidden1 = tf.nn.relu(hidden1)
```

POSTECH

Build a Model

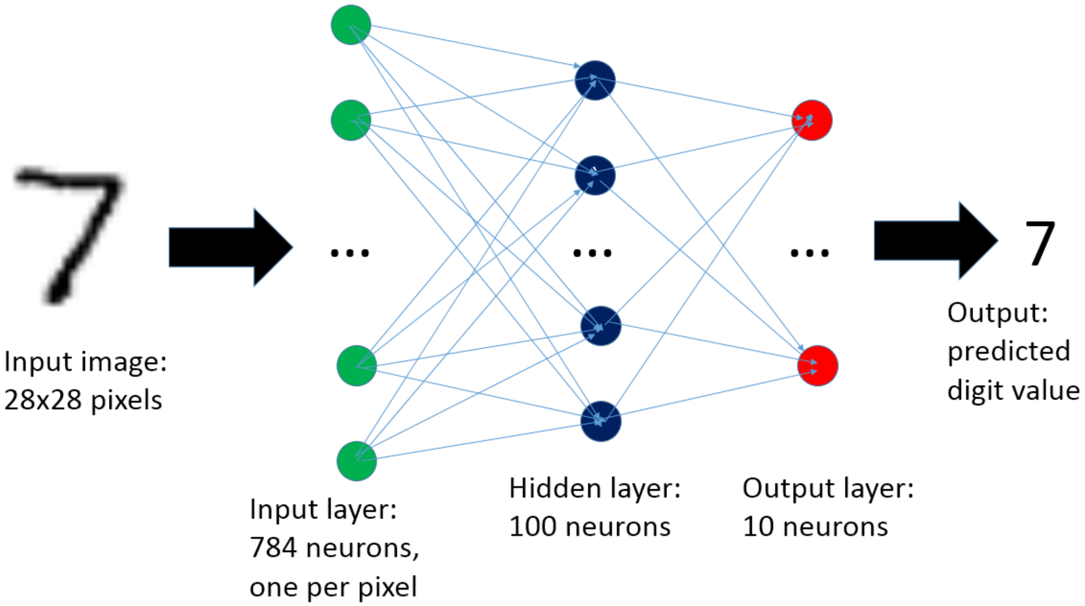
- Third, predict values with an affine transformation




```
# output = tf.matmul(hidden1, weights['output']) + biases['output']
output = tf.add(tf.matmul(hidden1, weights['output']), biases['output'])
```

ANN's Shape

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




```
n_input = 28*28
n_hidden1 = 100
n_output = 10
```

Weights and Biases

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

```
weights = {
    'hidden1' : tf.Variable(tf.random_normal([n_input, n_hidden1], stddev = 0.1)),
    'output' : tf.Variable(tf.random_normal([n_hidden1, n_output], stddev = 0.1)),
}

biases = {
    'hidden1' : tf.Variable(tf.random_normal([n_hidden1], stddev = 0.1)),
    'output' : tf.Variable(tf.random_normal([n_output], stddev = 0.1)),
}

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

Build a Model

```python
# Define Network
def build_model(x, weights, biases):
    # first hidden layer
    hidden1 = tf.add(tf.matmul(x, weights['hidden1']), biases['hidden1'])
    # non linear activate function
    hidden1 = tf.nn.relu(hidden1)

    # Output layer with linear activation
    output = tf.add(tf.matmul(hidden1, weights['output']), biases['output'])
    return output
```

Cost, Initializer and Optimizer

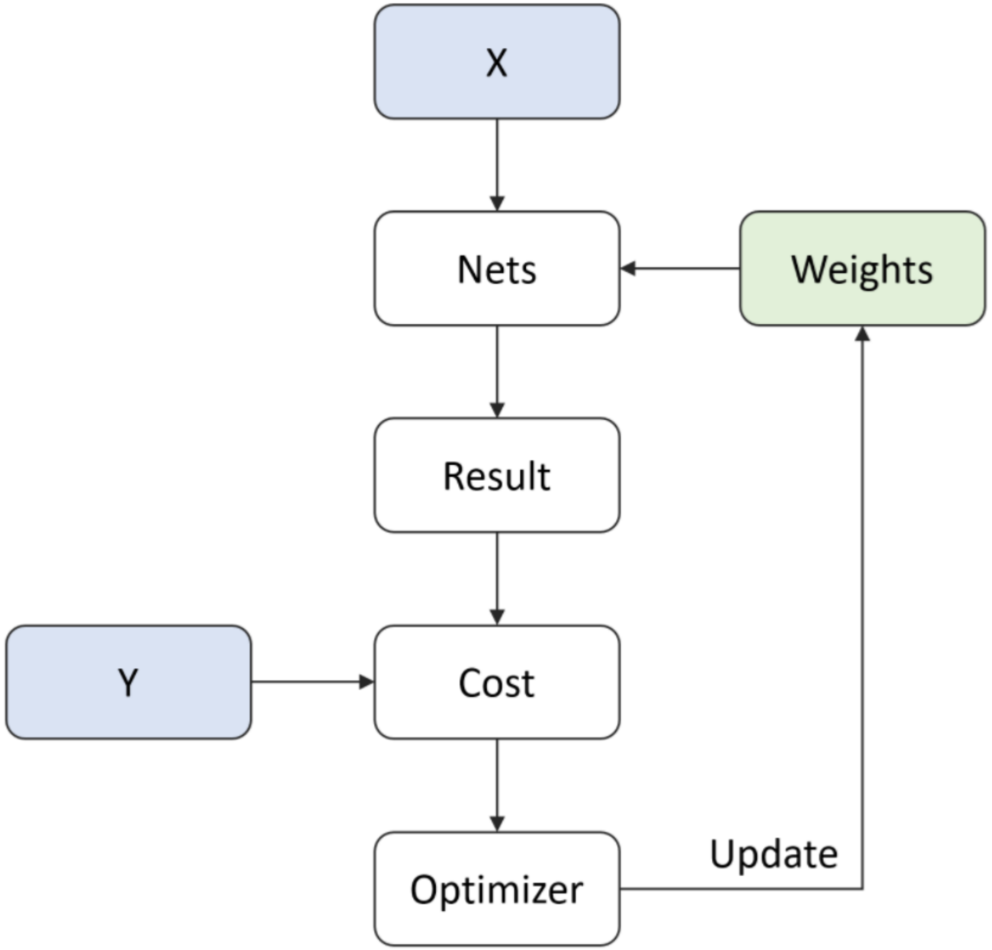
- Loss: cross entropy

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

- Initializer
  – Initialize all the empty variables
- Optimizer
  – AdamOptimizer: the most popular optimizer

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

# optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
LR = 0.0001
optm = tf.train.AdamOptimizer(LR).minimize(loss)

init = tf.global_variables_initializer()
```

Summary of Model



Iteration Configuration

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

```
n_batch = 50      # Batch Size
n_iter = 2500     # Learning Iteration
n_prt = 250       # Print Cycle
```

Optimization

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

# Training cycle
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:
        c = sess.run(loss, feed_dict={x : train_x,  y : train_y})
        print ("Iter : {}".format(epoch))
        print ("Cost : {}".format(c))
```

```
Iter : 0
Cost : 2.4568586349487305
Iter : 250
Cost : 1.4568665027618408
Iter : 500
Cost : 0.7992963194847107
Iter : 750
Cost : 0.6279309988021851
Iter : 1000
Cost : 0.4135037958621979
Iter : 1250
Cost : 0.4587993784046173
```

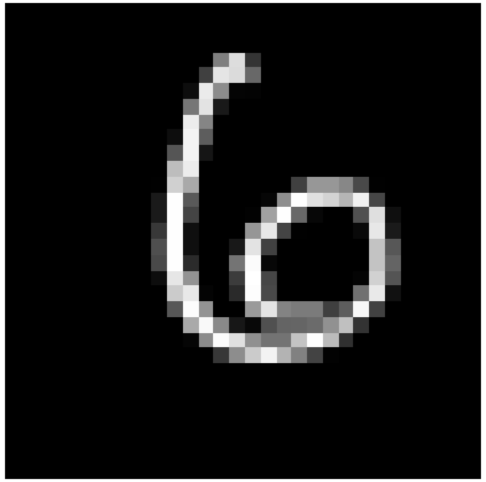
Test or Evaluation

```
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 : 92.0%
```

Test or Evaluation

```
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 : 6
Probability : [ 0.    0.01  0.04  0.    0.01  0.    0.93  0.    0.01  0.  ]
```