# (Artificial) Neural Networks with TensorFlow

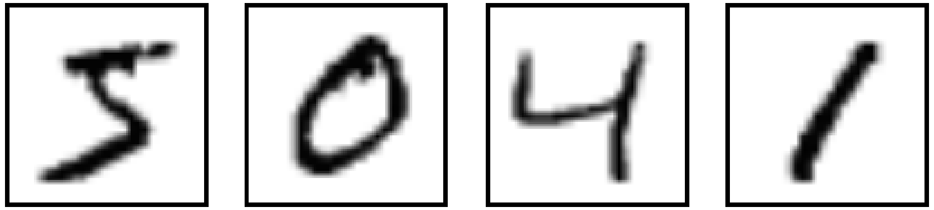
Industrial AI Lab.

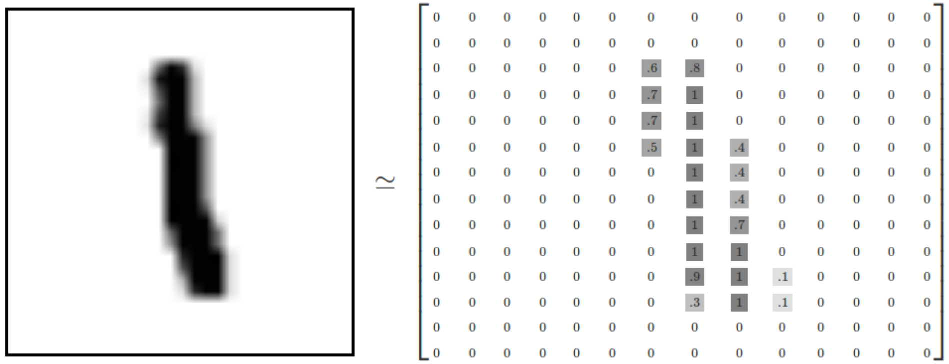
Prof. Seungchul Lee

# ANN in TensorFlow: MNIST

# 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



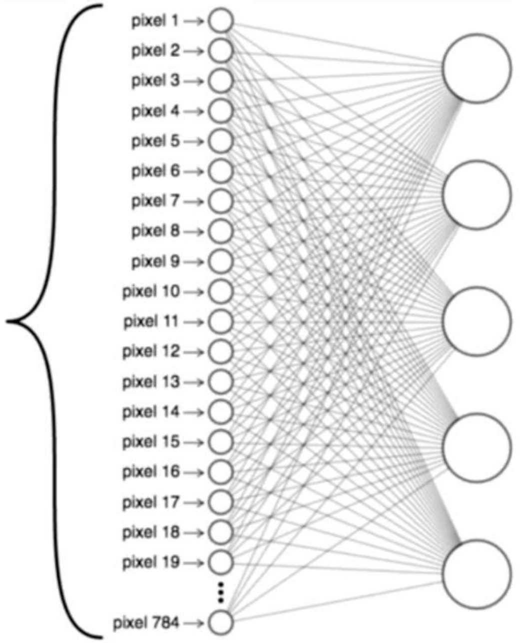


# ANN with TensorFlow

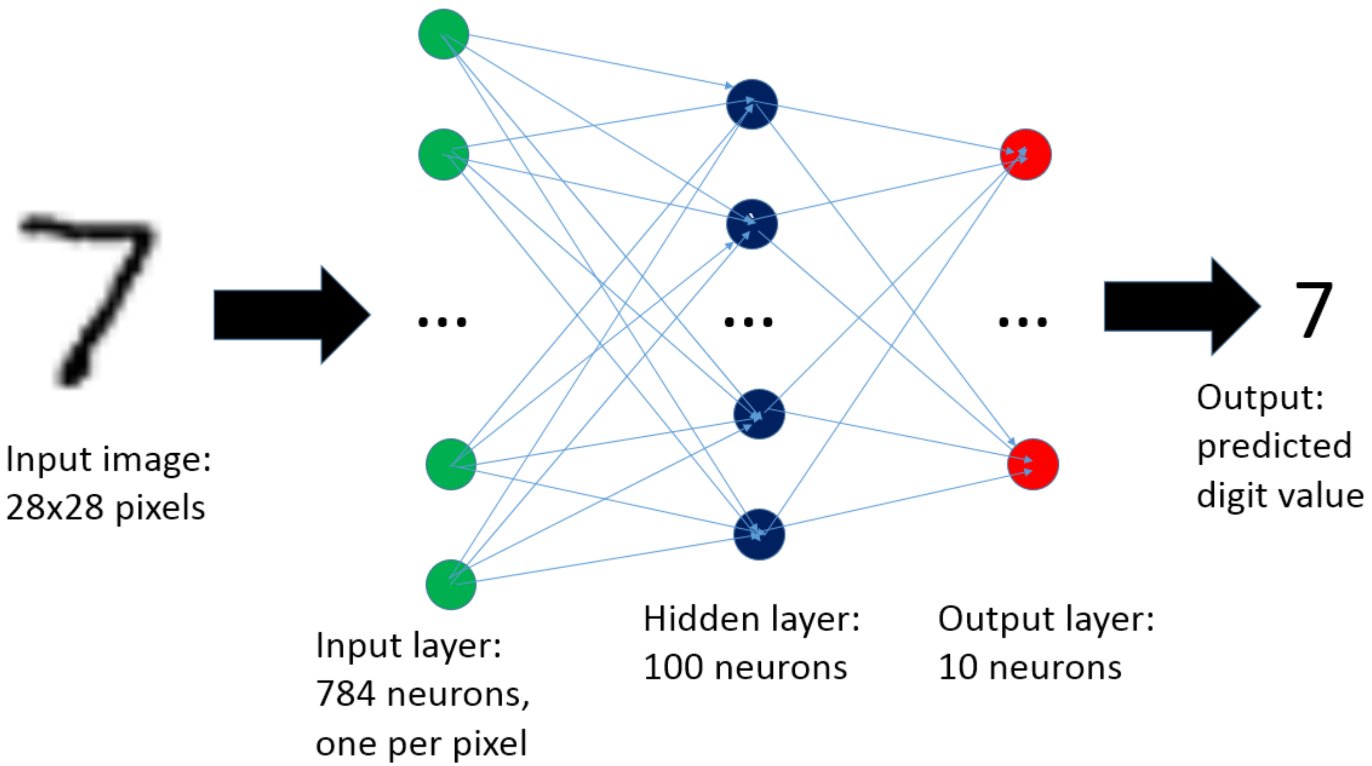
- Feed a gray image to ANN



| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | 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 | 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 | 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 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 55 | 97 | 157 | 156 | 187 | 255 | 81 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 20 | 58 | 20 | 96 | 140 | 215 | 215 | 251 | 254 | 254 | 254 | 254 | 254 | 163 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 157 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 254 | 241 | 100 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 155 | 117 | 184 | 214 | 214 | 155 | 117 | 49 | 19 | 19 | 141 | 254 | 241 | 158 | 23 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 173 | 254 | 246 | 70 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 90 | 247 | 254 | 219 | 68 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 43 | 203 | 254 | 250 | 121 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 43 | 146 | 254 | 254 | 229 | 111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 137 | 245 | 254 | 254 | 238 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 133 | 254 | 254 | 254 | 254 | 254 | 236 | 106 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 54 | 196 | 196 | 158 | 196 | 223 | 254 | 254 | 217 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 18 | 69 | 245 | 254 | 99 | 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 | 197 | 254 | 135 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 200 | 254 | 135 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 87 | 246 | 254 | 99 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 37 | 59 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 66 | 254 | 254 | 171 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 127 | 254 | 160 | 0 | 0 | 0 | 0 | 0 | 11 | 20 | 140 | 248 | 254 | 205 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 156 | 254 | 230 | 86 | 79 | 79 | 161 | 176 | 218 | 254 | 254 | 254 | 169 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 28 | 203 | 254 | 254 | 254 | 255 | 254 | 254 | 254 | 238 | 95 | 19 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 5 | 89 | 155 | 155 | 156 | 111 | 58 | 58 | 36 | 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 | 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 | 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 |



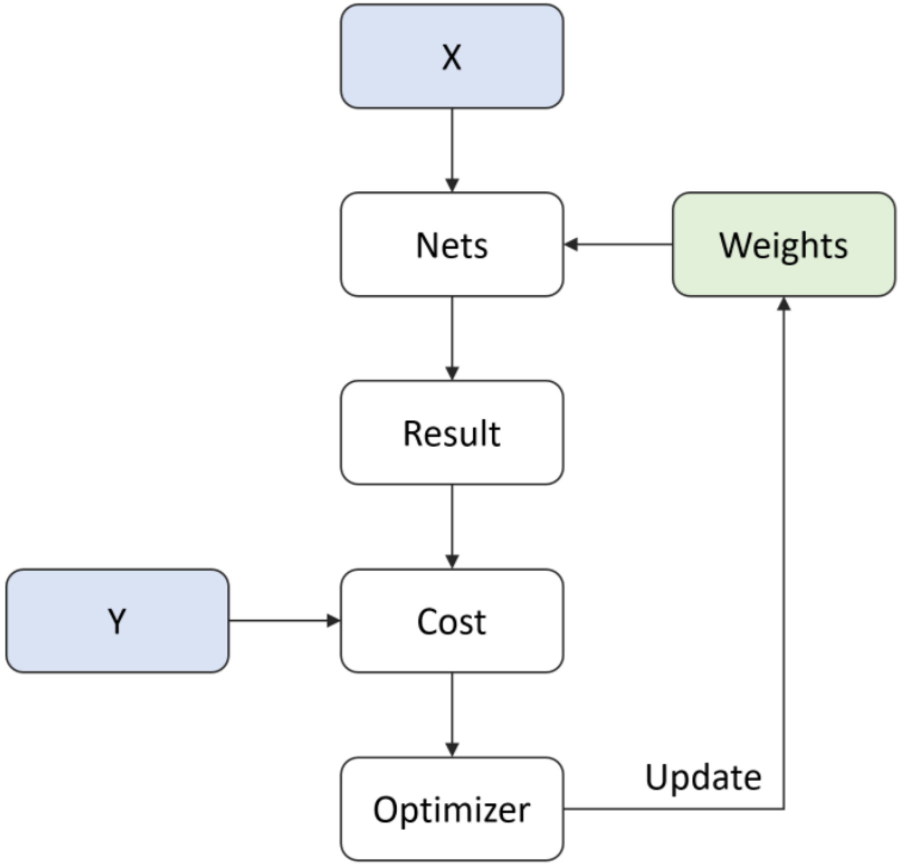
# Our Network Model



# 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):

$$\theta_j := \theta_j - \alpha \cdot \underbrace{\boxed{\text{only one example}}}_{(\hat{y}^i - y^i)x_j^i}$$

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

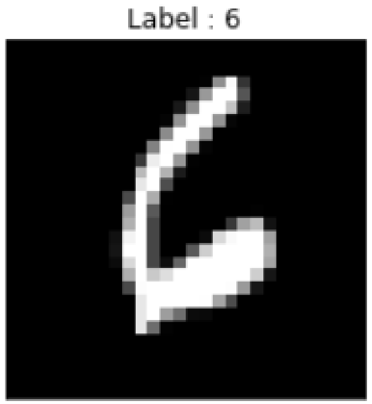
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
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

```python
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()
```



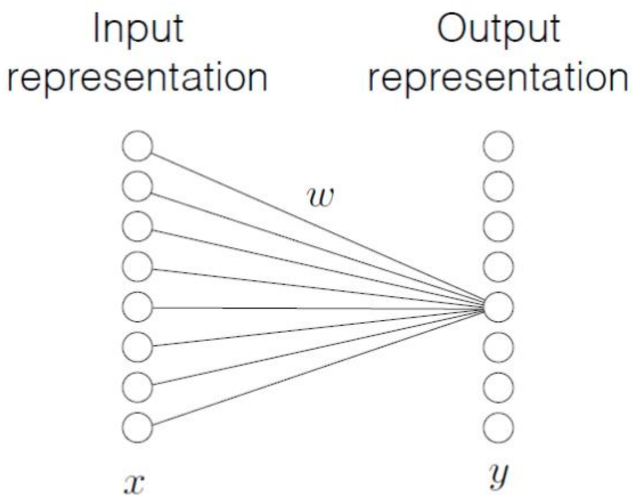

- One hot encoding

```python
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



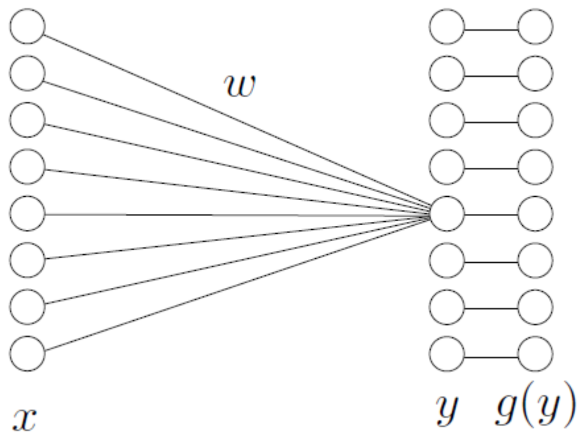

$$y_j = \left( \sum_i \omega_{ij} x_i \right) + b_j$$

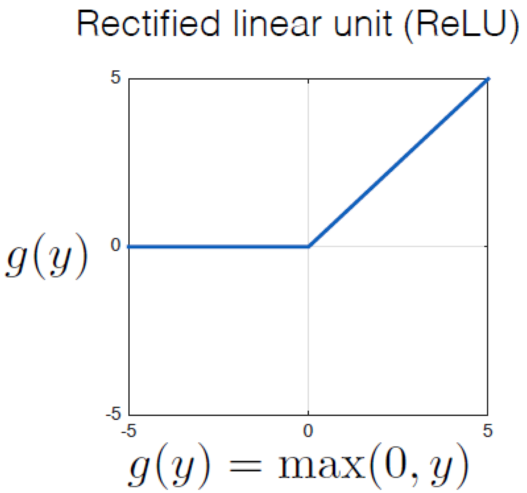
$$y = \omega^T x + b$$

```
# 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



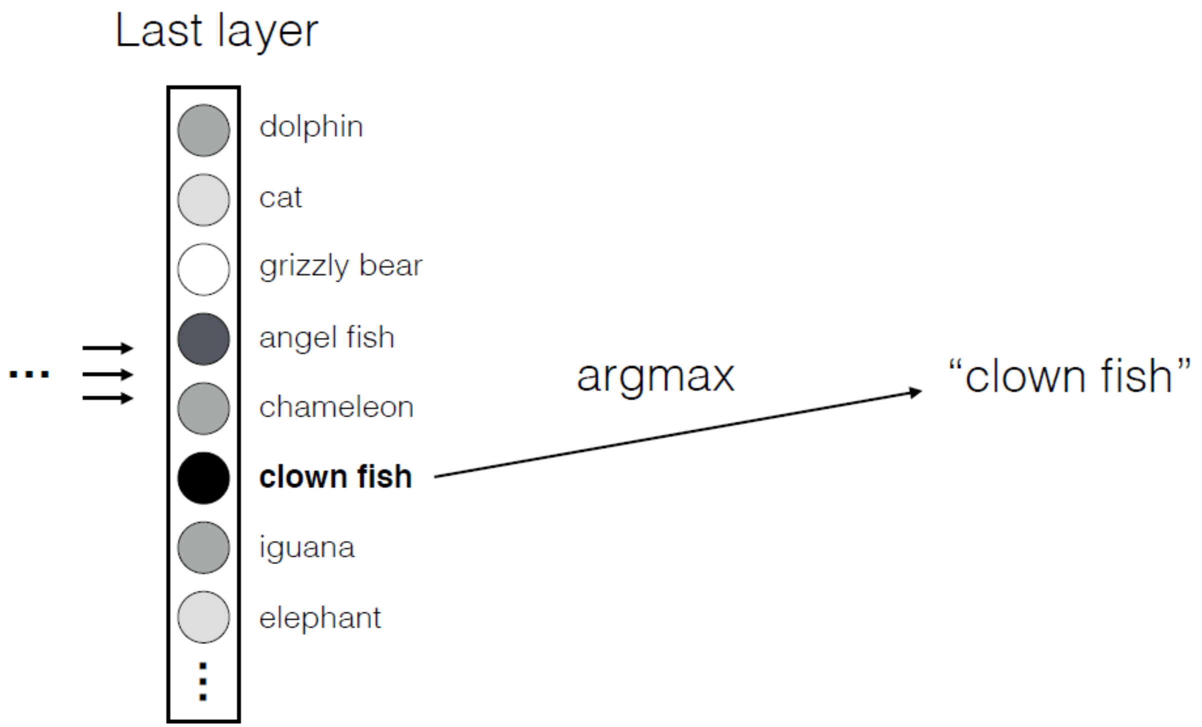




```
hidden1 = tf.nn.relu(hidden1)
```

POSTECH

# Build a Model

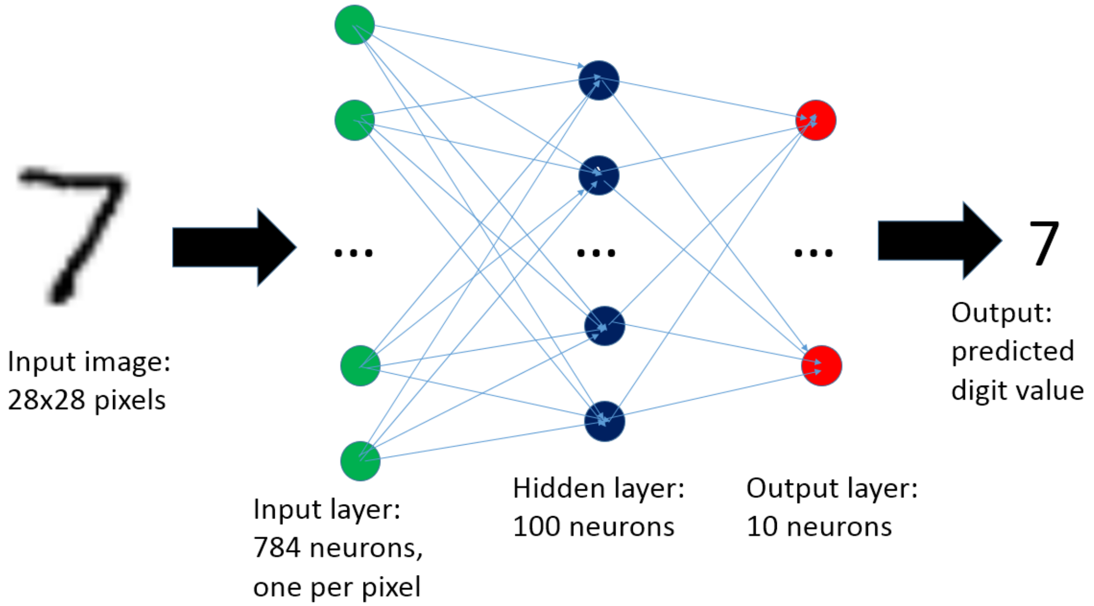
- Third, predict values with an affine transformation




```
# 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

```
# 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

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

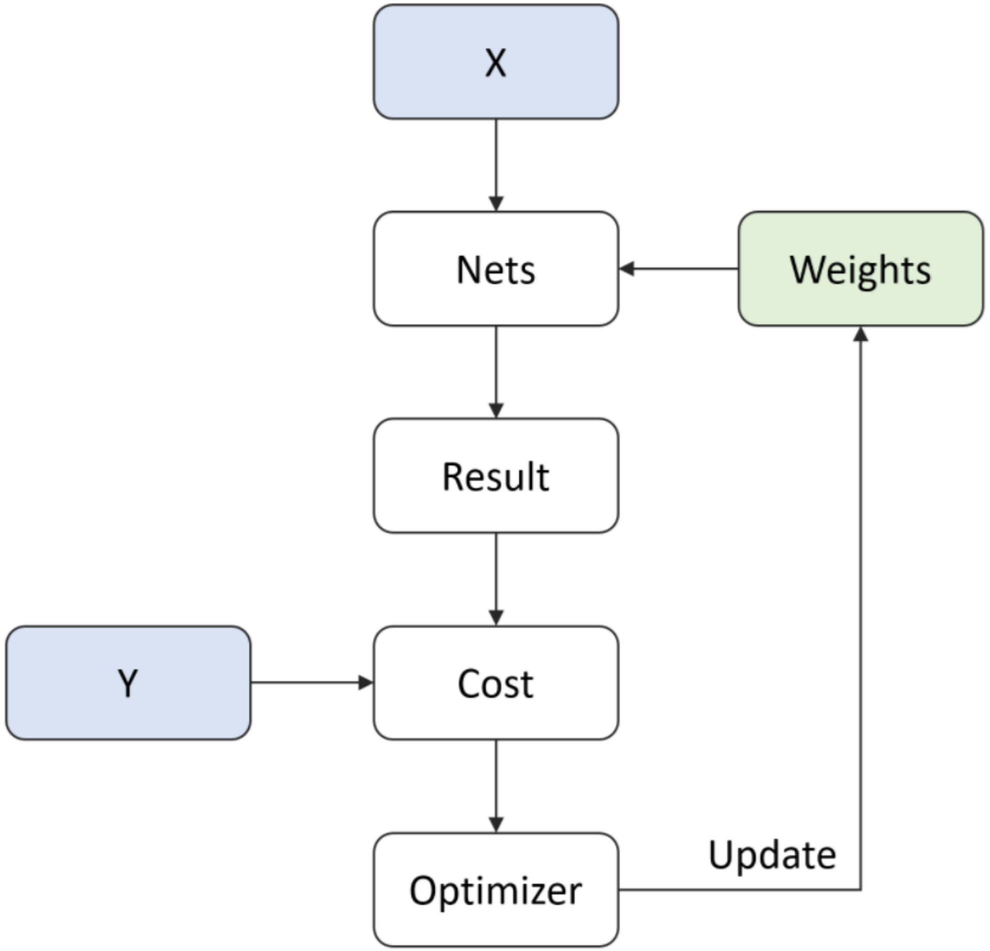
- Initializer
  - Initialize all the empty variables
- Optimizer
  - AdamOptimizer: the most popular optimizer

```
# 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 : 0.4587993794046173
```

# 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))
```

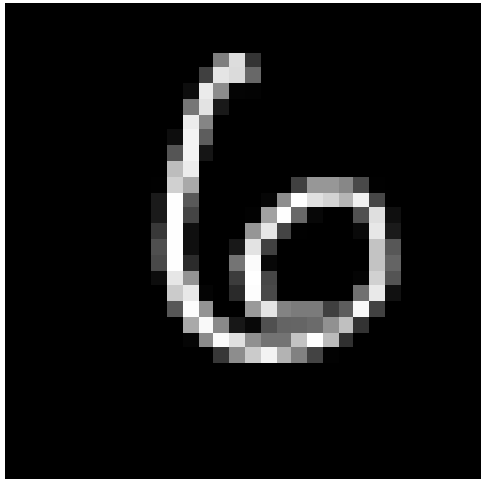
```
Accuracy : 92.0%
```

# 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.   ]
```