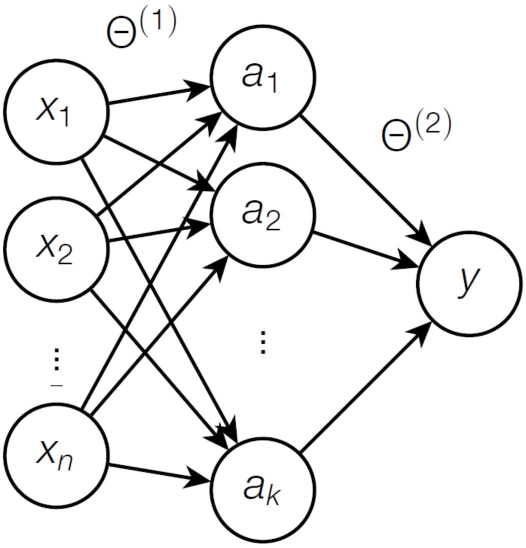
(Artificial) Neural Networks in TensorFlow

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

Table of Contents

1. Artificial Neural Networks (ANN)

1.1 Structure



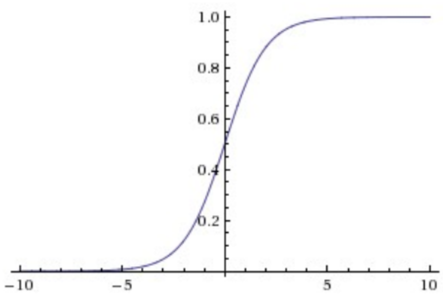
Transformation

- Affine (or linear) transformation and nonlinear activation (layer)

$$f(x) = g\left(\theta^T x + b\right)$$

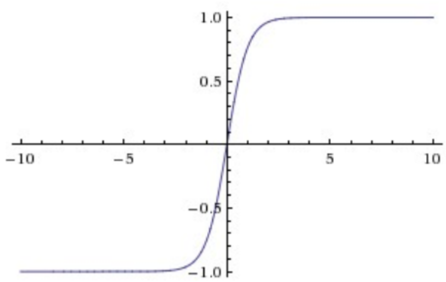
- Nonlinear activation function

Sigmoid



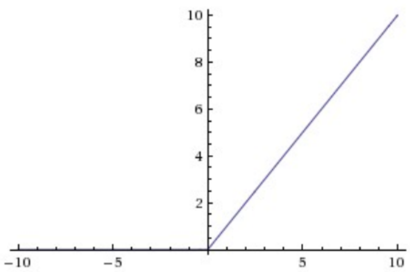
$$g(x) = \frac{1}{1 + e^{-x}}$$

tanh



$$g(x) = \tanh(x)$$

Rectified Linear Unit



$$g(x) = \max(0, x)$$

1.2. Training Neural Networks

Loss Function

- Measures error between target values and predictions
- More or less the same as those for other parametric models, such as linear models

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

- Example
  - Cross entropy:

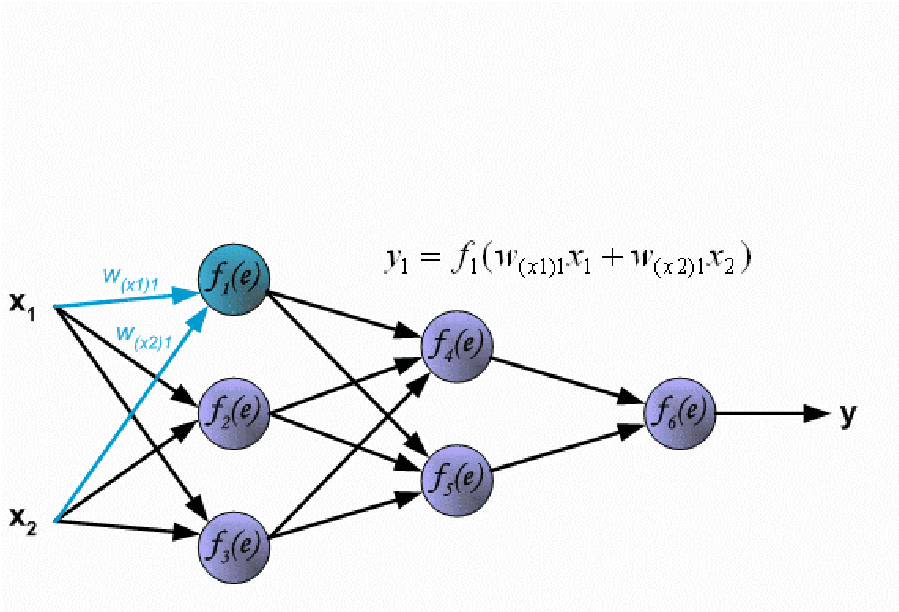
$$-\frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)$$

  - Squared loss:

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

Backpropagation

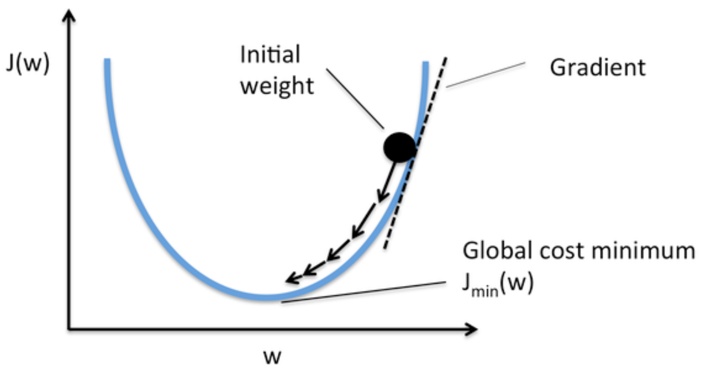
- Forward propagation
  - the initial information propagates up to the hidden units at each layer and finally produces output
- Backpropagation
  - allows the information from the cost to flow backwards through the network in order to compute the gradients

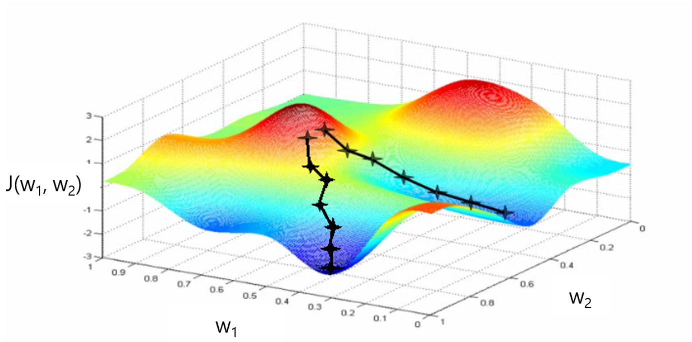


(Stochastic) Gradient Descent

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

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





2. 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

3. TensorFlow

- `tensorflow` is an open-source software library for deep learning.

3.1. Computational Graph

- `tf.constant`
- `tf.Variable`
- `tf.placeholder`

In [1]:
```
import tensorflow as tf

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

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

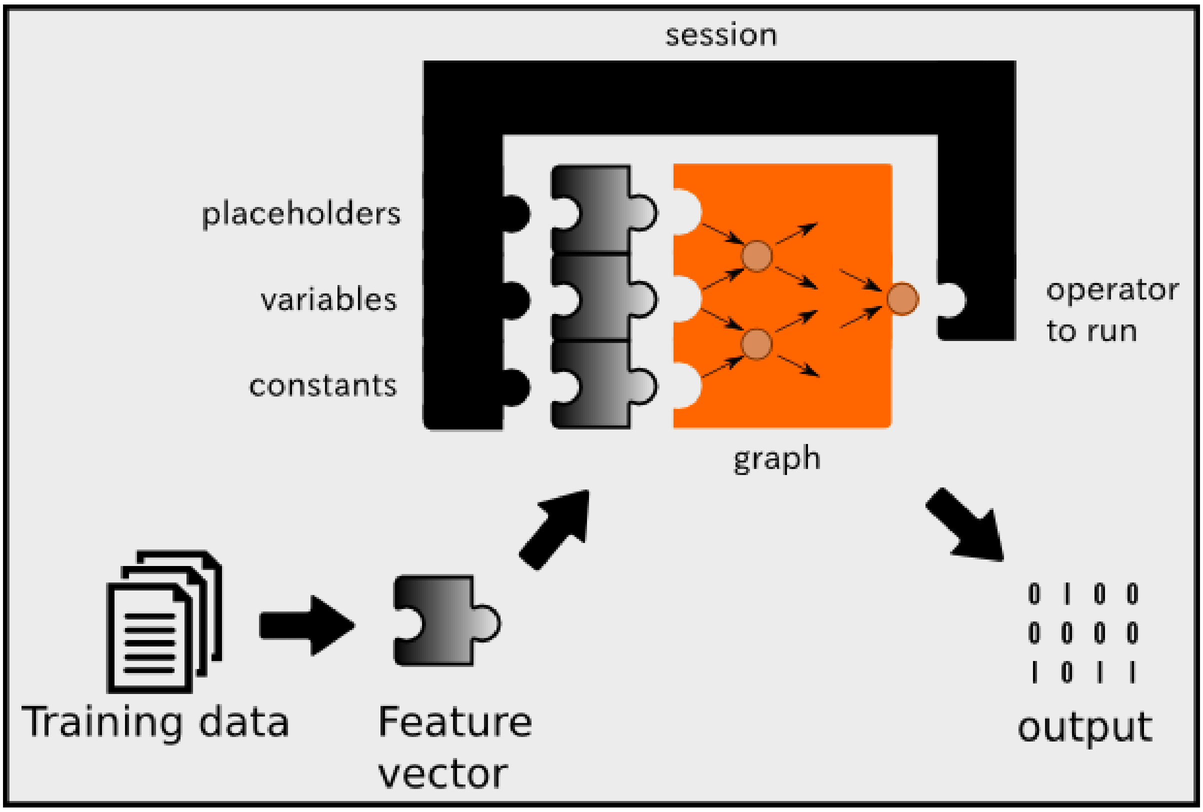
In [2]:
```
A
```

Out[2]:
```
<tf.Tensor 'add:0' shape=(3,) dtype=int32>
```

In [3]:
```
B
```

Out[3]:
```
<tf.Tensor 'mul:0' shape=(3,) dtype=int32>
```

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.



In [4]:
```
sess = tf.Session()
sess.run(A)
```

Out[4]:
```
array([5, 7, 9])
```

In [5]:
```
sess.run(B)
```

Out[5]:
```
array([ 4, 10, 18])
```

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

In [6]:
```
w = tf.Variable([1, 1])
```

In [7]:
```
init = tf.global_variables_initializer()
sess.run(init)
```

```
In [8]: sess.run(w)
```

```
Out[8]: array([1, 1])
```

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

```
In [9]: x = tf.placeholder(tf.float32, [2, 2])
```

```
In [10]: sess.run(x, feed_dict={x : [[1,2],[3,4]]})
```

```
Out[10]: array([[ 1.,  2.],
                [ 3.,  4.]], dtype=float32)
```

3.2. Example: Linear Regression using TensorFlow

$\text{Given} \begin{cases} x_i : \text{inputs} \\ y_i : \text{outputs} \end{cases}$, Find ω_1 and ω_2

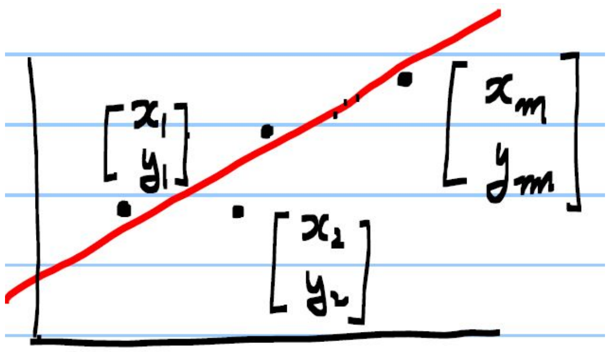
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \omega_1 x_i + \omega_2$$

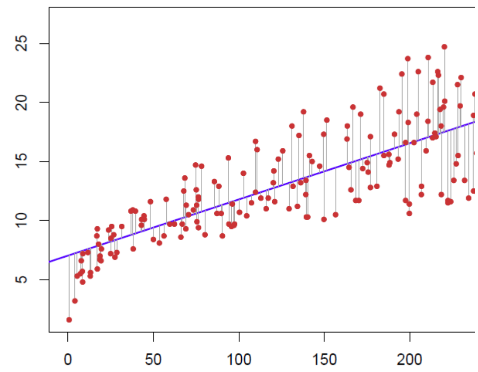
- $\hat{y}_i$: predicted output
- $\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$: Model parameters

$$\hat{y}_i = f(x_i, \omega) \;\; \text{in general}$$

- in many cases, a linear model to predict y_i used

$$\hat{y}_i = \omega_1 x_i + \omega_2 \;\; \text{such that} \; \min_{\omega_1, \omega_2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$



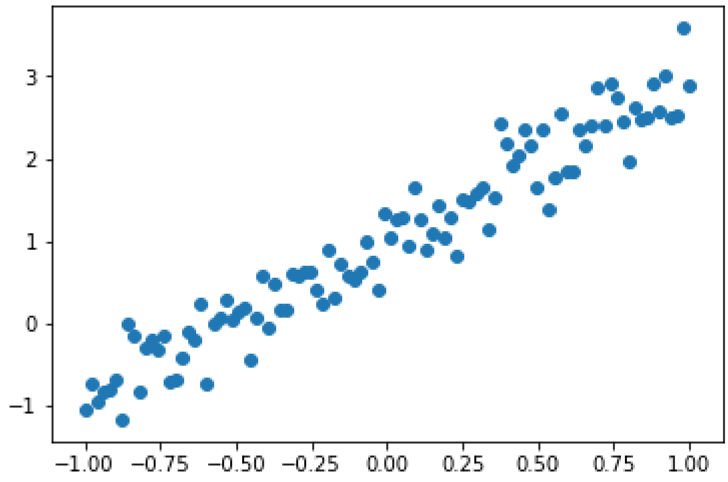


Data Generation

```
In [11]: import numpy as np
         print(np.random.rand(10))
         print(np.random.randint(0,10,size=10))
```

```
[ 0.82055906  0.8822898   0.36515203  0.9781535   0.17633944  0.20166983
  0.89753462  0.54911309  0.27023013  0.22494141]
[7 9 5 9 9 6 3 0 3 0]
```

In [12]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

data_x = np.linspace(-1, 1, 100)
data_y = 2 * data_x + 1 + np.random.randn(*data_x.shape) * 0.3

plt.scatter(data_x, data_y)
plt.show()
```



Prameter Learning (or Estimation) by using TensorFlow

In [13]:
```
# Define decision variables in tf

weights = {
    'w' : tf.Variable(tf.random_normal([1], stddev=0.1))
}
biases = {
    'b' : tf.Variable(tf.random_normal([1], stddev=0.1))
}
```

In [14]:
```
x = tf.placeholder(tf.float32, [10])
y = tf.placeholder(tf.float32, [10])
```

$\hat{y}_i = \omega x_i + b$

In [15]:
```
# define model

def model(x, weights, biases):
    output = tf.add(tf.multiply(x, weights['w']), biases['b'])
    return output
```

$$\min_{\omega,b} \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

In [16]:
```
# define loss

pred = model(x, weights, biases)
loss = tf.square(tf.subtract(y, pred))
loss = tf.reduce_mean(loss)
```

In [17]:
```
# define optimizer

LR = 0.04
# optm = tf.train.AdamOptimizer(LR).minimize(loss)
optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)
```

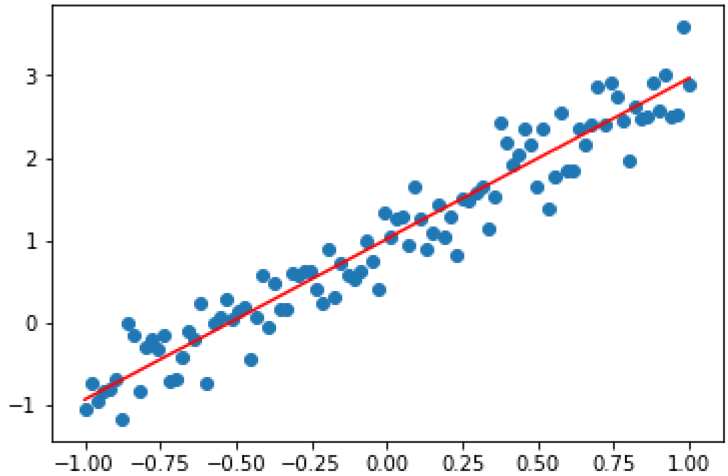
In [18]:
```
# tf.Variable initializer

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

In [19]:
```
# optimizing

n_iter = 200
n_prt = 20

for epoch in range(n_iter):
    idx = np.random.randint(0, 100, 10)
    train_x, train_y = data_x[idx], data_y[idx]
    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))


w_hat = sess.run(weights['w'])
b_hat = sess.run(biases['b'])

sess.close()
```

```
Iter : 0
Cost : 1.932685136795044
Iter : 20
Cost : 0.4764551520347595
Iter : 40
Cost : 0.1376667022705078
Iter : 60
Cost : 0.08908583968877792
Iter : 80
Cost : 0.12191130220890045
Iter : 100
Cost : 0.09848610311746597
Iter : 120
Cost : 0.13959051668643951
Iter : 140
Cost : 0.08917944133281708
Iter : 160
Cost : 0.1182650700211525
Iter : 180
Cost : 0.09228118509054184
```

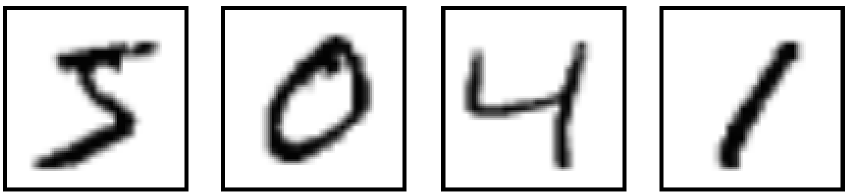
```
In [20]: print ("w_hat : {}".format(w_hat))
         print ("b_hat : {}".format(b_hat))

         plt.scatter(data_x, data_y)
         learned_y = data_x*w_hat + b_hat
         plt.plot(data_x, learned_y, 'r')
         plt.show()
```

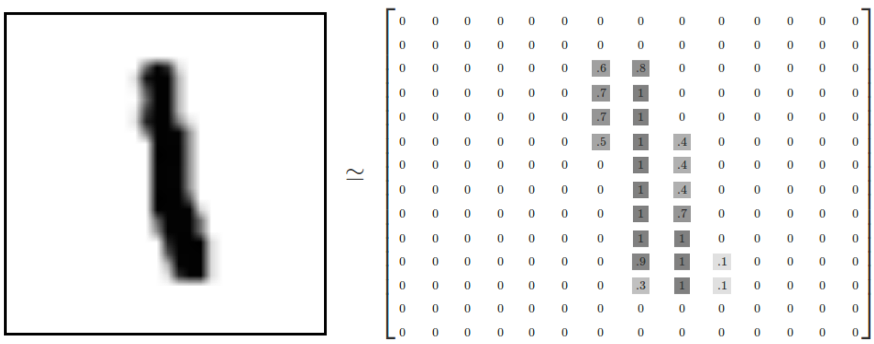
```
w_hat : [ 1.95228994]
b_hat : [ 1.01526022]
```

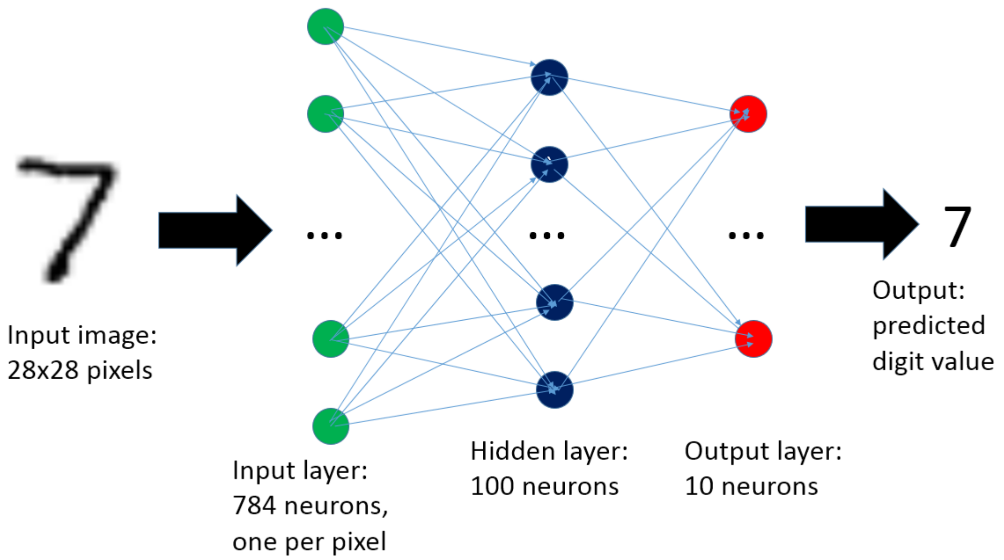


4. ANN with TensorFlow

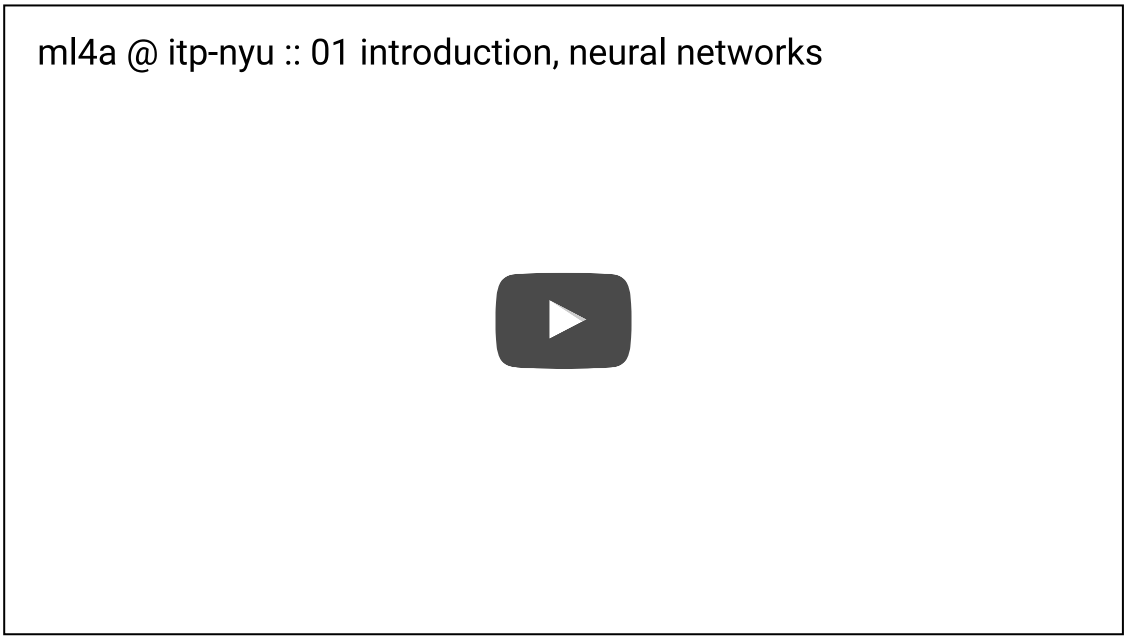
- MNIST (Mixed National Institute of Standards and Technology database) database
    - Handwritten digit database
    - 28×28 gray scaled image
    - Flattened array into a vector of $28 \times 28 = 784$







In [21]:
```
%%html
<center><iframe src="https://www.youtube.com/embed/z0bynQjEpII?start=2088&end=3137"
width="560" height="315" frameborder="0" allowfullscreen></iframe></center>
```



4.1. Import Library

In [22]:
```
# Import Library
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

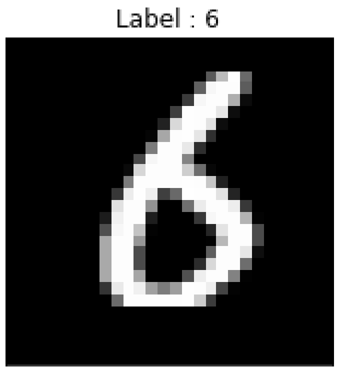
4.2. Load MNIST Data

- Download MNIST data from tensorflow tutorial example

```
In [23]: 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 [24]: 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()
```



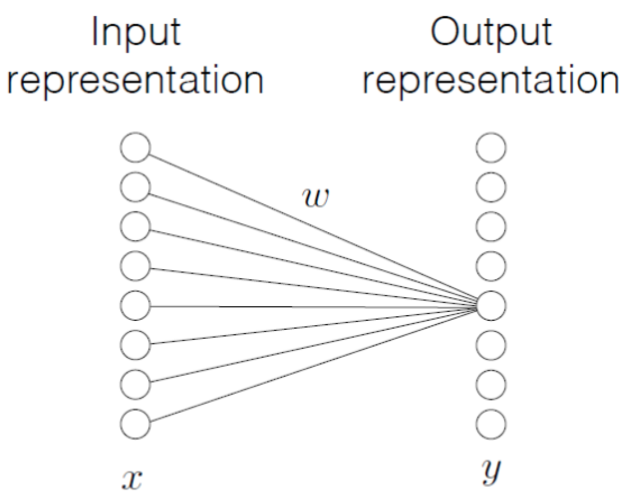
One hot encoding

```
In [25]: print ('Train labels : {}'.format(train_y[3, :]))
```

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

4.3. Build a Model

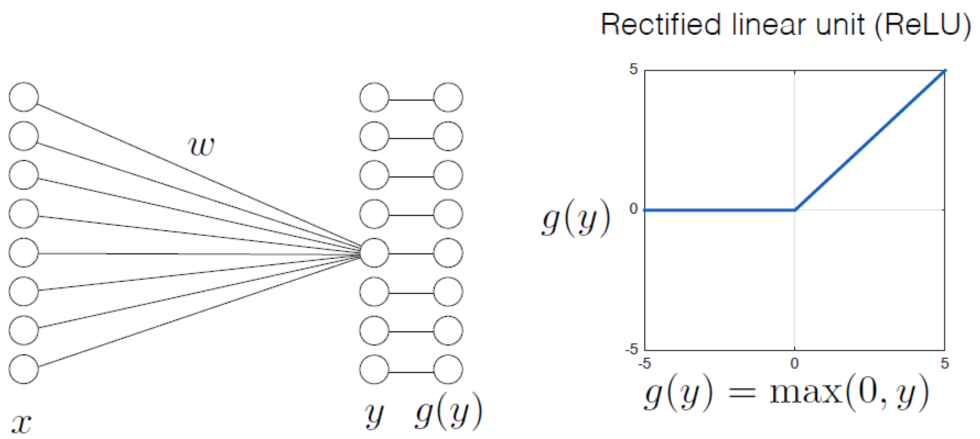
First, the layer performs several matrix multiplication to produce a set of linear activations



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

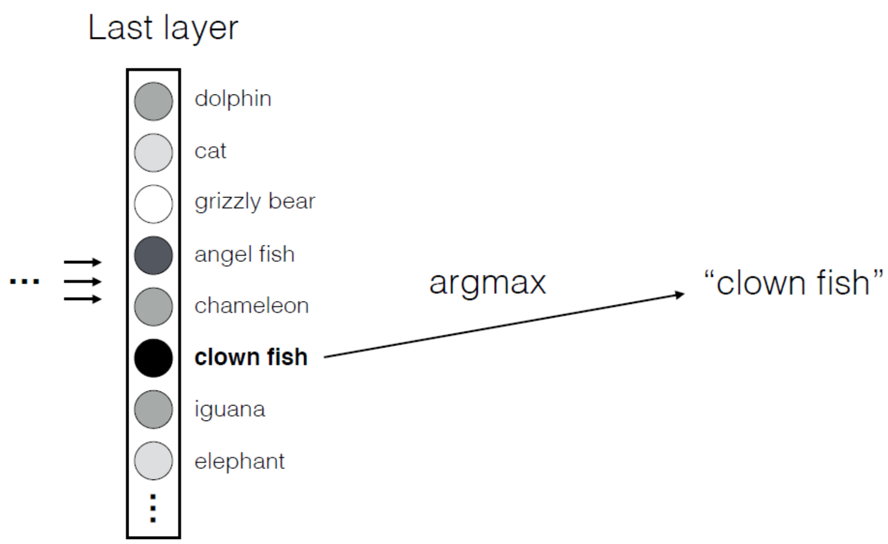
```
# hidden1 = tf.matmul(x, weights['hidden1']) + biases['hidden1']
hidden1 = tf.add(tf.matmul(x, weights['hidden1']), biases['hidden1'])
```

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




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

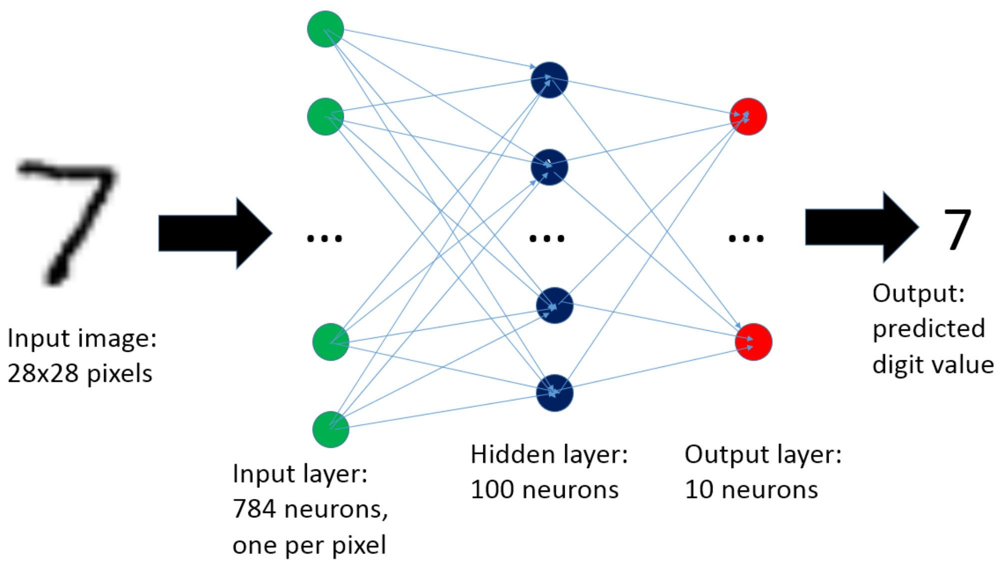
Third, predict values with affine transformation




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

4.4. Define an ANN Shape

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



In [26]:
```
n_input = 28*28
n_hidden1 = 100
n_output = 10
```

4.5. Define Weights, Biases and Network

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

In [27]:
```
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])
```

In [28]:
```
# 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
```

4.6. Define Cost, Initializer and Optimizer

Loss

- Classification: Cross entropy
  - Equivalent to apply logistic regression

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

Initializer

- Initialize all the empty variables

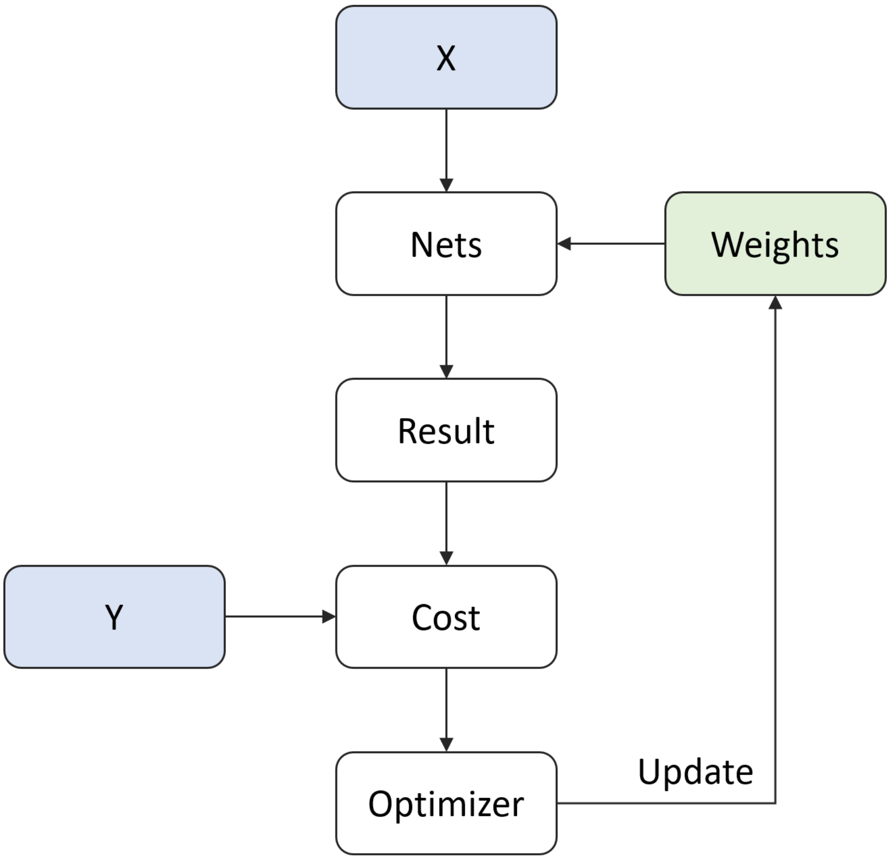
Optimizer

- AdamOptimizer: the most popular optimizer

In [29]:

```
# Define Cost
LR = 0.0001

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)
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 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

In [30]:
```
n_batch = 50      # Batch Size
n_iter = 2500     # Learning Iteration
n_prt = 250       # Print Cycle
```

4.9. Optimization

In [31]:

```
# 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 : 2.5034923553466797
Iter : 250
Cost : 1.3030091524124146
Iter : 500
Cost : 0.8532398343086243
Iter : 750
Cost : 0.7425006031990051
Iter : 1000
Cost : 0.6172652244567871
Iter : 1250
Cost : 0.361349493265152
Iter : 1500
Cost : 0.21700823307037354
Iter : 1750
Cost : 0.34678104519844055
Iter : 2000
Cost : 0.4176177680492401
Iter : 2250
Cost : 0.2827926278114319
```

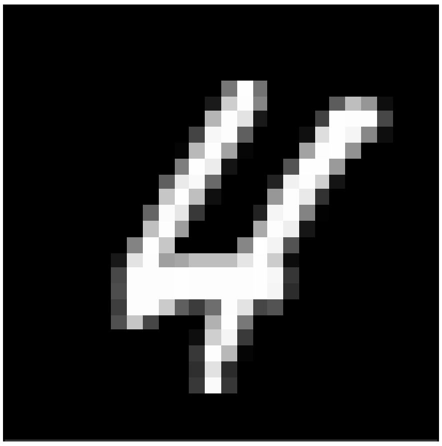
4.10. Test

In [32]:

```
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 : 90.0%
```

In [33]:
```
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 : [ 0.01  0.    0.02  0.    0.58  0.01  0.09  0.    0.19  0.09]
```

In [34]:
```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_to
c.js')
```