MNIST Classification with ANN

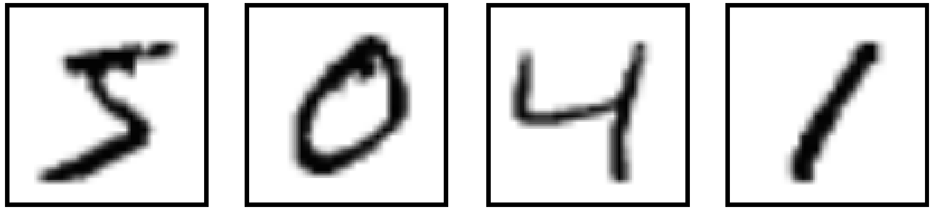
Industrial AI Lab.

Prof. Seungchul Lee

Yunseob Hwang, Illjeok Kim

MNIST database

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



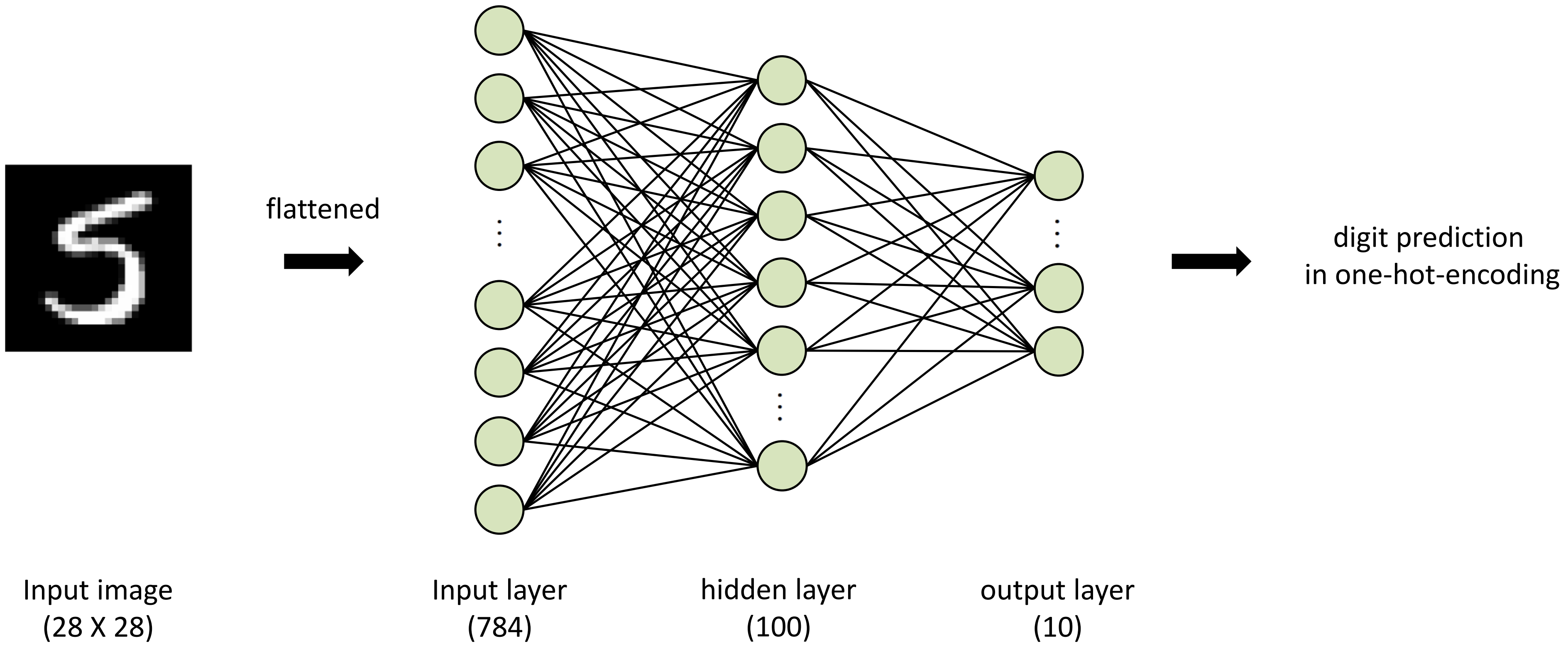
28 x 28
= 784 pixels

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

pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19

pixel 784

Our Network Model

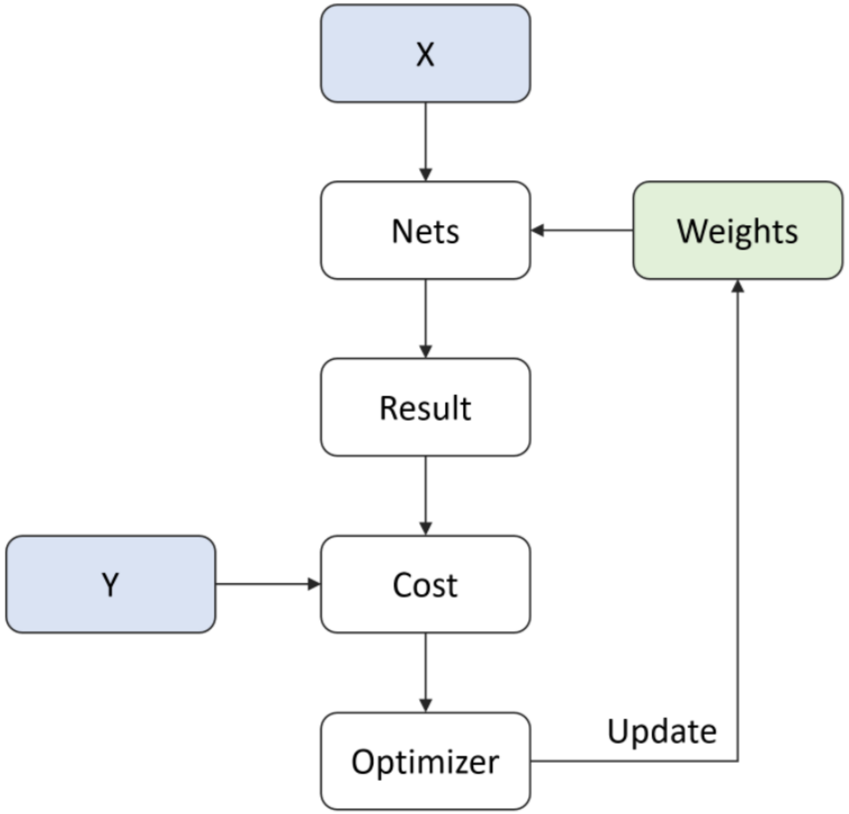


Iterative Optimization

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

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

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



POSTECH

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

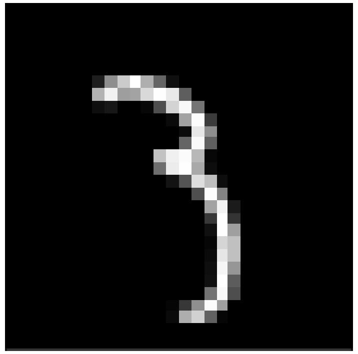
```
from tensorflow.examples.tutorials.mnist import input_data

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()
```

Label : 3



- One hot encoding

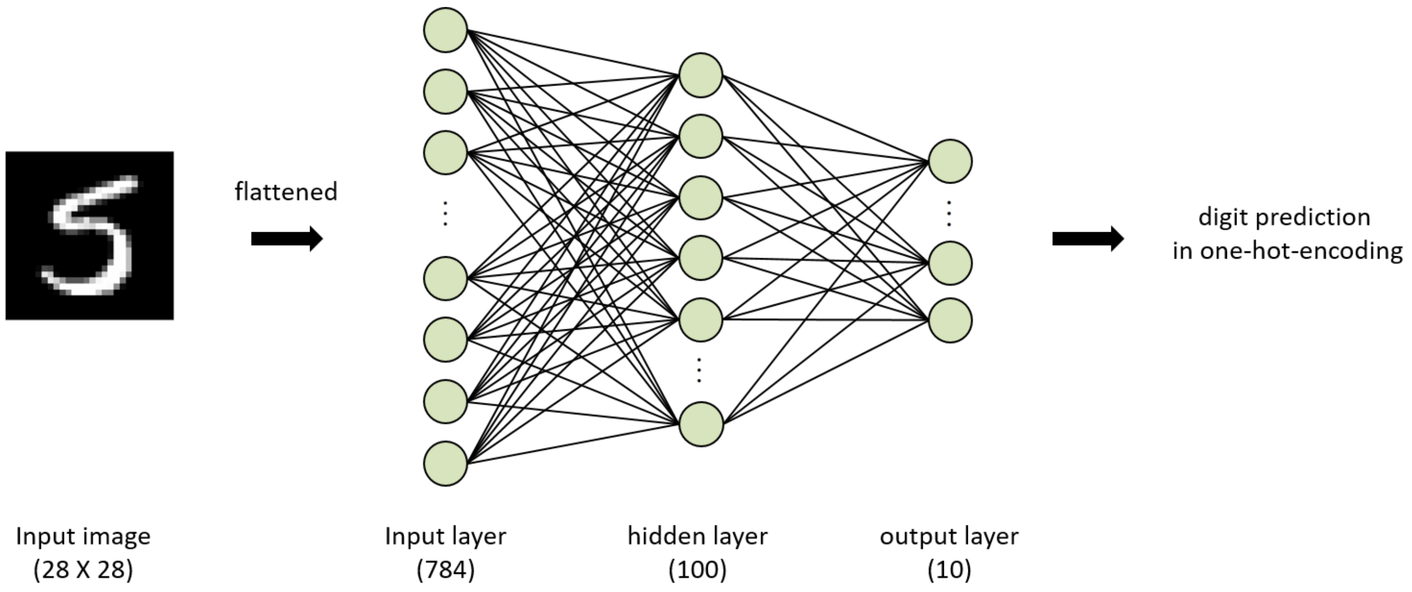
```
print ('Train labels : {}'.format(train_y[0, :]))
```

```
Train labels : [0. 0. 0. 1. 0. 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 $\sigma = 0.1$

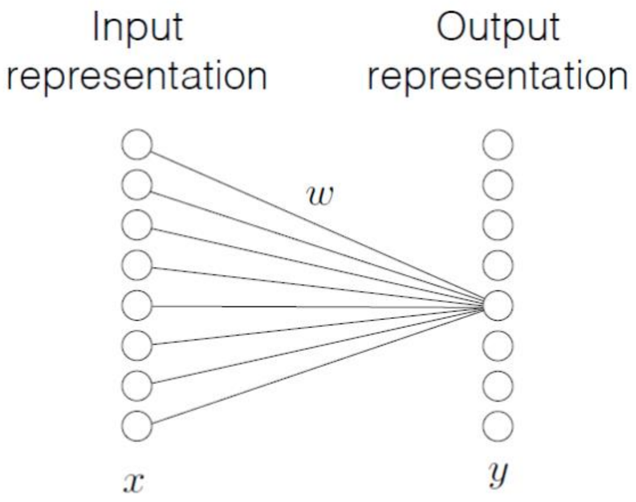
```
weights = {
    'hidden' : tf.Variable(tf.random_normal([n_input, n_hidden], stddev = 0.1)),
    'output' : tf.Variable(tf.random_normal([n_hidden, n_output], stddev = 0.1))
}

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



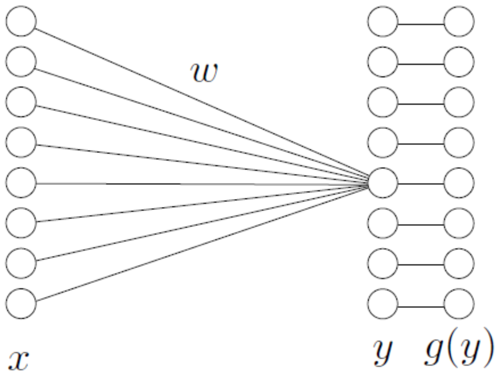
$$y_j = \left(\sum_i \omega_{ij} x_i \right) + b_j$$

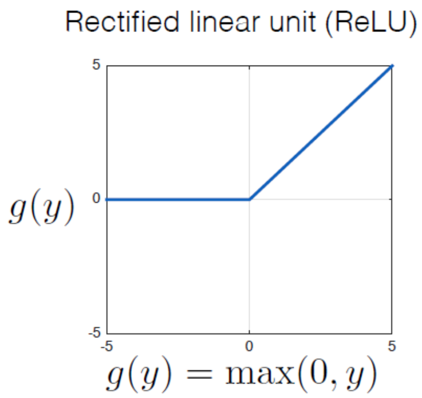
$$y = \omega^T x + b$$

```
# 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

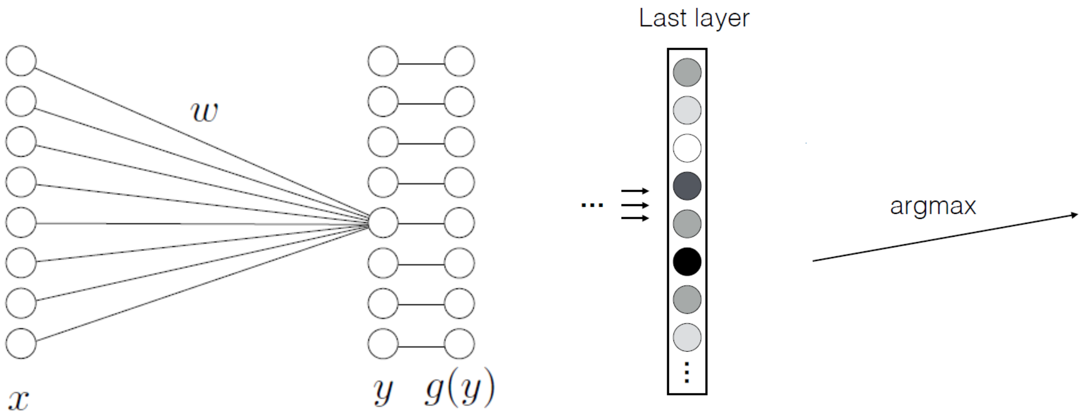





```python
# Define Network
def build_model(x, weights, biases):

    # 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




```python
# 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

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

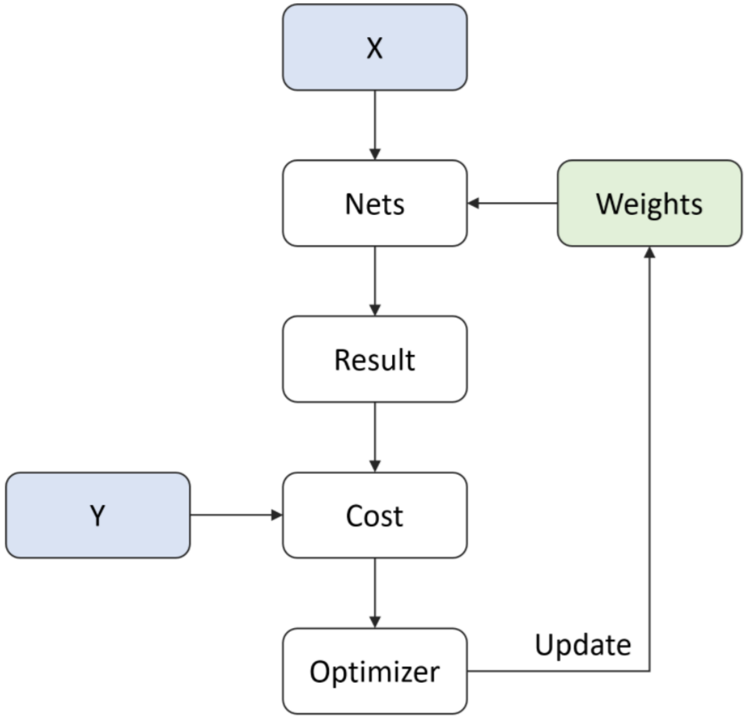
- Optimizer
  - AdamOptimizer: the most popular optimizer

```
# 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)
```

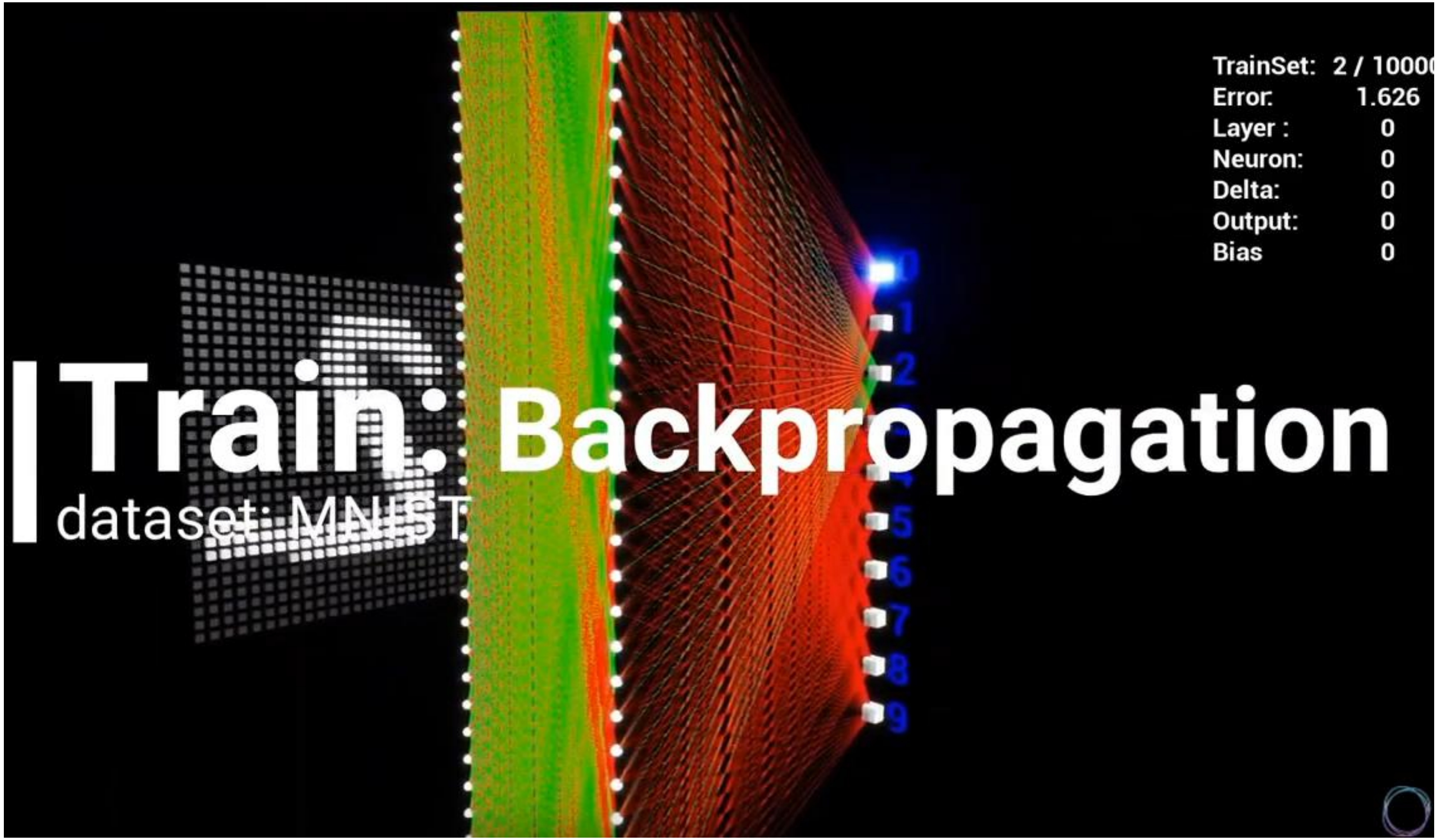
Iteration Configuration

- 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
```



ANN Training Visualization



Source: "3D Neural Network Simulator" - Youtube

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]
```