Source:

- 신경망 첫걸음, 타리크 라시드, 한빛미디어
- 딥러닝 첫걸음, 김성필, 한빛미디어

20170534

Introduction to machine learning

Lecture #11 : Python programming for ANN – 1

Instructor: Jeon, Seung-Bae

Assistant Professor
Hanbat National University
Department of Electronic Engineering

Contents

1 Lecture #10 Review

2 Simple programming (XOR)

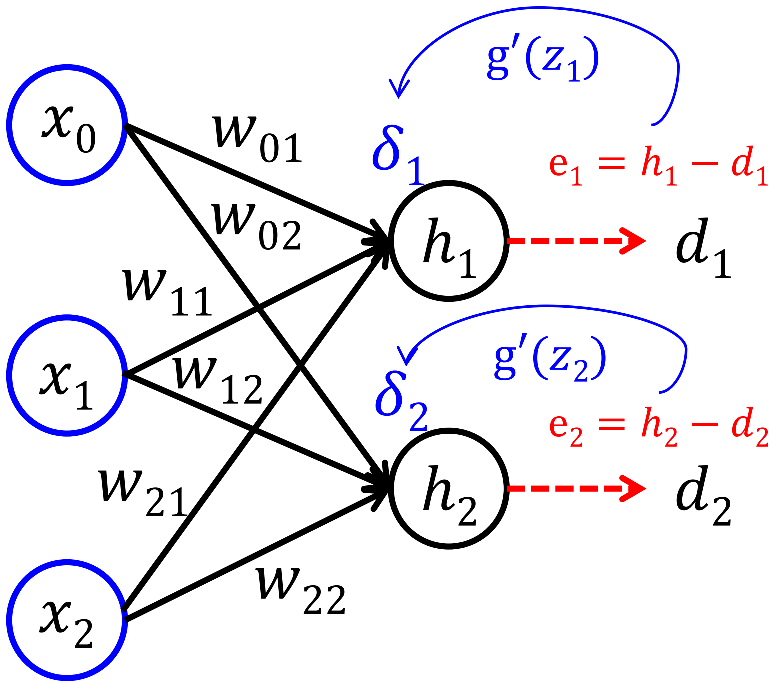
3 MNIST data

Contents

1 Lecture #10 Review

Weight update (training) summary

Neural networks also use gradient descent for parameter (weight) update.



d_1, d_2 : desired values

$$w_{jk} = w_{jk} - \alpha \frac{\partial J(w_{01}, w_{02}, w_{11}, w_{12}, w_{21}, w_{22})}{\partial w_{jk}}$$

$$= w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}}, \quad \alpha = \text{learning rate}$$

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial J}{\partial h_k}\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = (h_k - d_k)\, h_k(1 - h_k)\, x_j$$

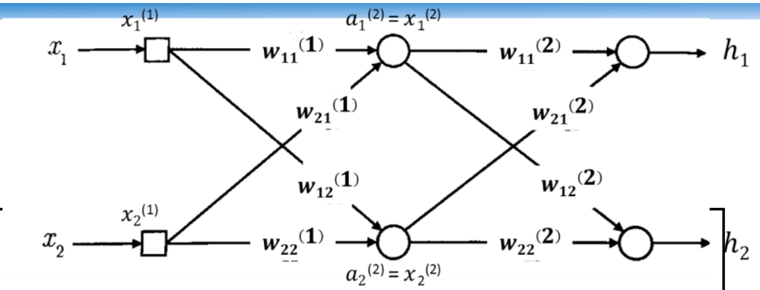
$$= e_k\, g(z_k)(1 - g(z_k))\, x_j$$

$$= e_k\, g'(z_k)\, x_j = \delta_k\, x_j$$

$$\rightarrow w_{jk} = w_{jk} - \alpha \frac{\partial J}{\partial w_{jk}} = w_{jk} - \alpha \delta_k\, x_j$$

(this is called as the delta rule.)

Process of the weight update in multi-layer ANN



Summary

1) **Initialize the weight values properly**

2) **By entering the feature values of the training data into the neural network and feeding forward, obtain the output value h (this process is also called inference)**

$$z^{(2)} = x^{(1)} \cdot w^{(1)} \qquad x^{(2)} = g(z^{(2)})$$
$$z^{(3)} = x^{(2)} \cdot w^{(2)} \qquad x^{(3)} = g(z^{(3)}) = h$$

3) **Calculate the error at the output layer**

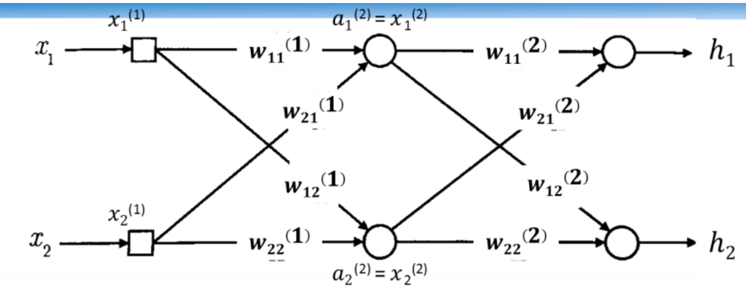
$$e^{(3)} = h - d$$

4) Conduct back propagation to calculate delta at the hidden layer.

$$e^{(3)} = h - d\ ,\ \delta^{(3)} = e^{(3)} * g'(z^{(3)})$$
$$e^{(2)} = \delta^{(3)} \cdot (w^{(2)})^T\ ,\ \delta^{(2)} = e^{(2)} * g'(z^{(2)})$$

다층신경망의 가중치 학습 – process

Summary



5) Calculate the amount of weight update based on the delta values.

$$\Delta w^{(2)} = -\alpha (x^{(2)})^T \cdot \delta^{(3)}$$
$$\Delta w^{(1)} = -\alpha (x^{(1)})^T \cdot \delta^{(2)}$$

6) Update the weights.

$$w^{(2)} = w^{(2)} + \Delta w^{(2)}$$
$$w^{(1)} = w^{(1)} + \Delta w^{(1)}$$

1) 7) Repeat the above 1-6 for the entire training data (1 epoch)

1) 8) Repeat the above epoch until the error becomes sufficiently small.

Even if the number of hidden layers increases, the overall concept is the same.

Different cost function (cross entropy)

- **It doesn't change the process much.**
- **Only the method for obtaining delta at the output layer changes.**

1) Initialize the weight values properly

2) By entering the feature values of the training data into the neural network and feeding forward, obtain the output value h (this process is also called inference)

$$z^{(1)} = x^{(1)} \cdot w^{(1)} \qquad x^{(2)} = g(z^{(1)})$$
$$z^{(2)} = x^{(2)} \cdot w^{(2)} \qquad x^{(3)} = g(z^{(2)}) = h$$

3) Calculate the error at the output layer

$$e^{(3)} = h - d$$

4) Conduct back propagation to calculate delta at the hidden layer.

$$e^{(3)} = h - d\ ,\ \delta^{(3)} = e^{(3)}$$
$$e^{(2)} = \delta^{(3)} \cdot (w^{(2)})^T\ ,\ \delta^{(2)} = e^{(2)} * g'(z^{(2)})$$

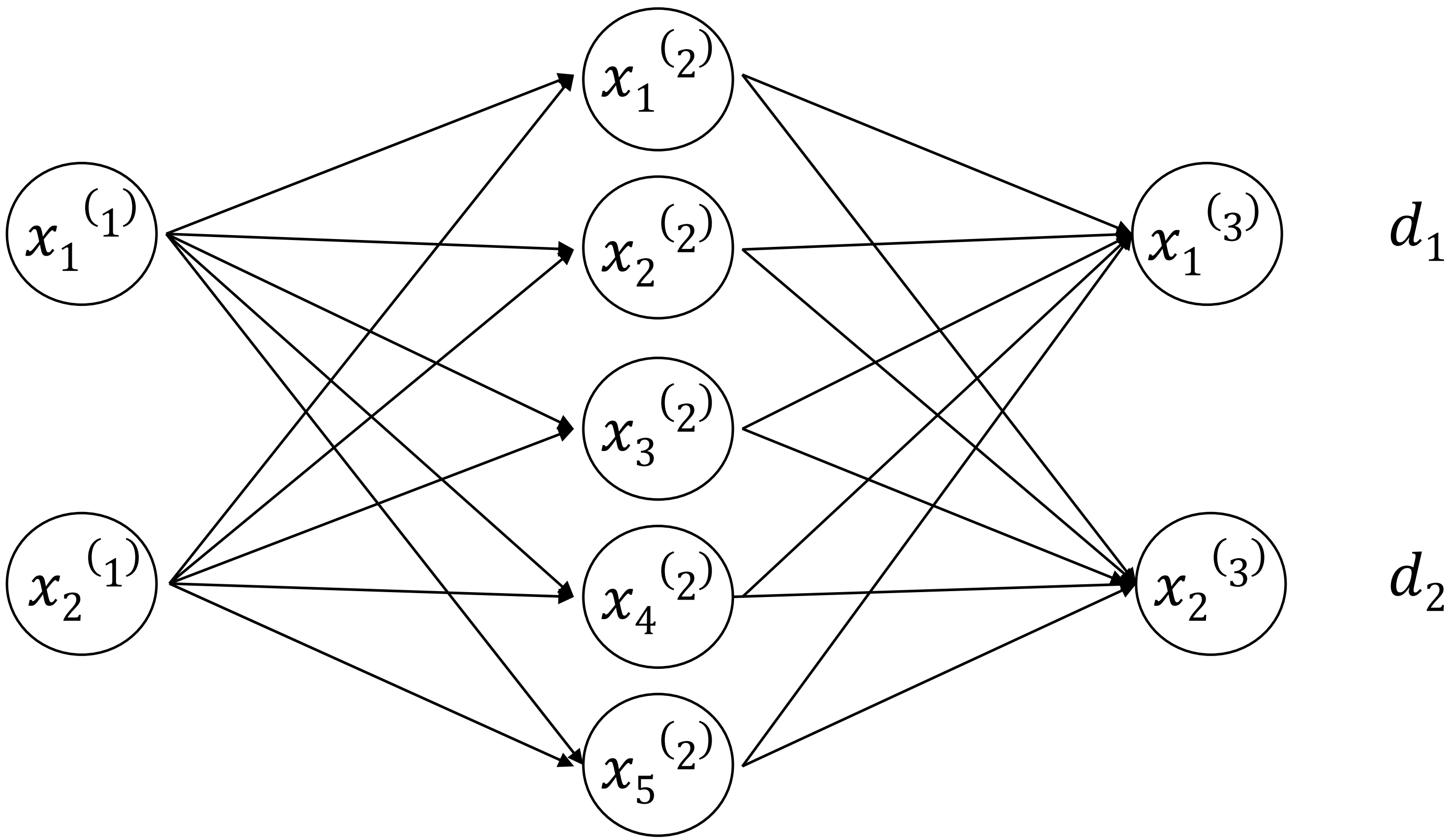
Contents

2 Simple programming (XOR)

ANN programming – XOR

XOR	Input 1	Input 2
0	1	1
1	1	0
1	0	1
0	0	0

- **Let's make the ANN with the 5 neurons in the hidden layer. (there is no bias)**



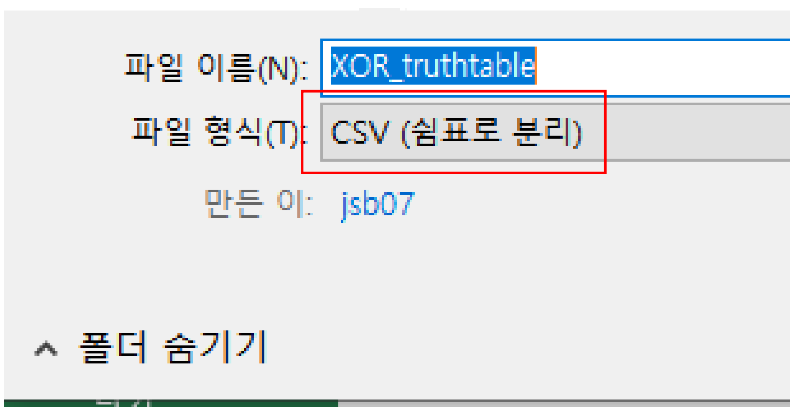
ANN programming – XOR

1. Dataset preparation

XOR	Input 1	Input 2
0	1	1
1	1	0
1	0	1
0	0	0

0	1	1
1	1	0
0	1	1
0	0	0

Excel → save as CSV format



CSV: comma-separated values

→ If we open the file with notepad, it will be displayed as the picture on the rieght.

0,1,1
1,1,0
0,1,1
0,0,0

ANN programming – XOR

0	1	1
1	1	0
0	1	1
0	0	0

2. Data Reading

```
import numpy as np

training_data_file = open('C:/Users/jsb07/.spyder-py3/XOR_truthtable.csv', 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

print (training_data_list)
```

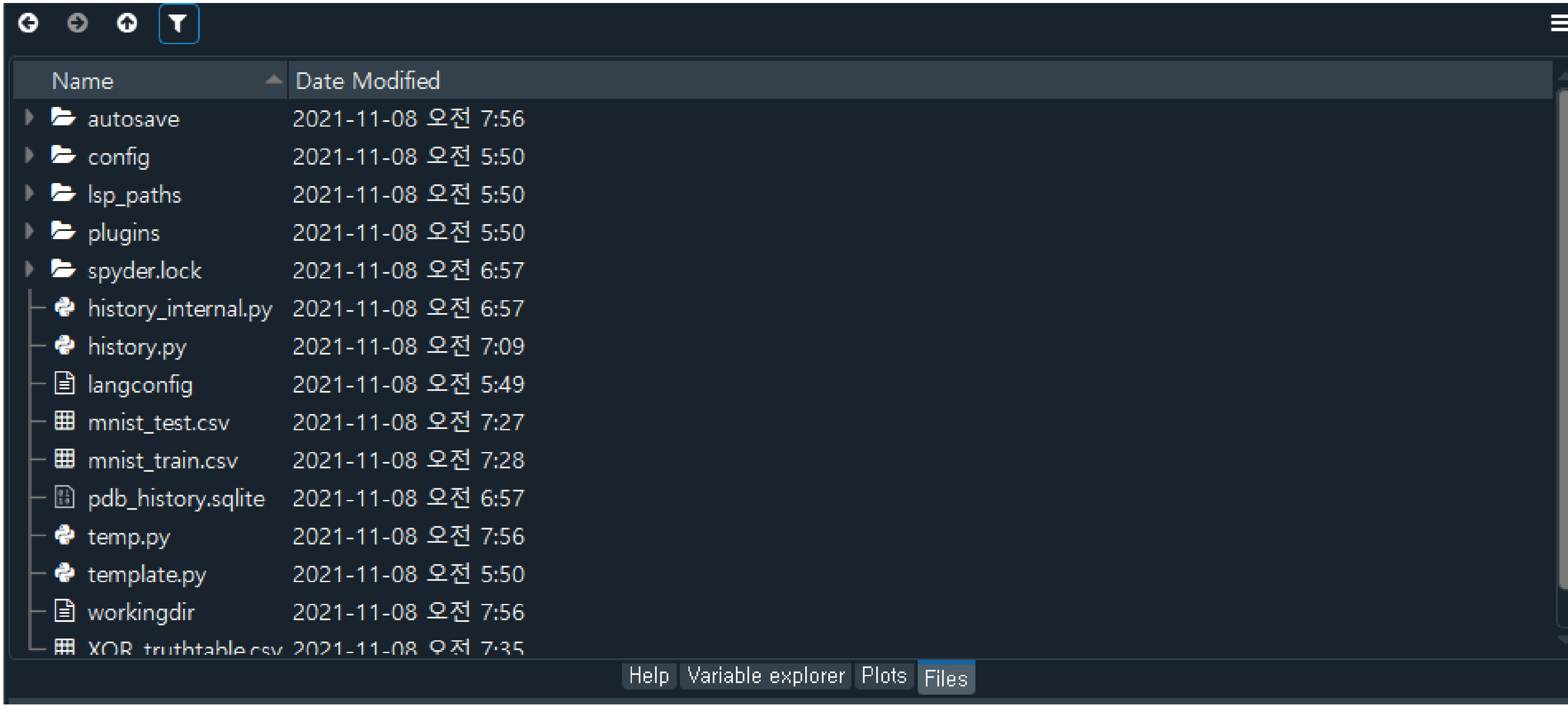
```
['0,1,1\n', '1,1,0\n', '0,1,1\n', '0,0,0\n']
```

Readlines() : Read all the line of the file and save the lines as a list

Line space \n is also included. Then all the elements (lines) are saved in str type.

ANN programming – XOR

2. Data Reading (tip for path error)



files tab (on the right in spyder), you can find your file and path, you can 'copy absolute path' by the right click and paste the copied path on your code.

ANN programming – XOR

3. Appropriate data transforming (reading, converting to the number, dimension scaling)

```
import numpy as np
import matplotlib.pyplot as plt

training_data_file = open('C:/Users/jsb07/Documents/XOR_truthtable.csv', 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

print(training_data_list)
for record in training_data_list:
        all_values = record.split(',')
        print (all_values)
        feature = np.asfarray(all_values[1:])
        print (feature)
        x = np.array(feature,ndmin = 2)
        print (x)

        correct_label = int(all_values[0])
        d = np.zeros(2)
        d [correct_label] = 1
        print (d)
```

```
['0,1,1\n', '1,1,0\n', '1,0,1\n', '0,0,0\n']
['0', '1', '1\n']
[1. 1.]
[[1. 1.]]
[1. 0.]
['1', '1', '0\n']
[1. 0.]
[[1. 0.]]
[0. 1.]
['1', '0', '1\n']
[0. 1.]
[[0. 1.]]
[0. 1.]
['0', '0', '0\n']
[0. 0.]
[[0. 0.]]
[1. 0.]
```

- **record.split : data dividing (by ,)**
- **np.asfarray() : convert the array to float array**
- **x=np.array(feature,ndmin=2) → feature (2,) → x = (1x2)**

ANN programming – XOR

3. Appropriate data transforming (one-hot encoding)

```
import numpy as np
import matplotlib.pyplot as plt

training_data_file = open('C:/Users/jsb07/Documents/XOR_truthtable.csv', 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

print(training_data_list)
for record in training_data_list:
        all_values = record.split(',')
        print (all_values)
        feature = np.asfarray(all_values[1:])
        print (feature)
        x = np.array(feature,ndmin = 2)
        print (x)

        correct_label = int(all_values[0])
        d = np.zeros(2)
        d [correct_label] = 1
        print (d)
```

```
['0,1,1\n', '1,1,0\n', '1,0,1\n', '0,0,0\n']
['0', '1', '1\n']
[1. 1.]
[[1. 1.]]
[1. 0.]
['1', '1', '0\n']
[1. 0.]
[[1. 0.]]
[0. 1.]
['1', '0', '1\n']
[0. 1.]
[[0. 1.]]
[0. 1.]
['0', '0', '0\n']
[0. 0.]
[[0. 0.]]
[1. 0.]
```

- **One-Hot encoding**

→ Save the first element in the line to the correct_label.

→ After defining the zero array, replace the element which index is correct_label to 1.

ANN programming – XOR

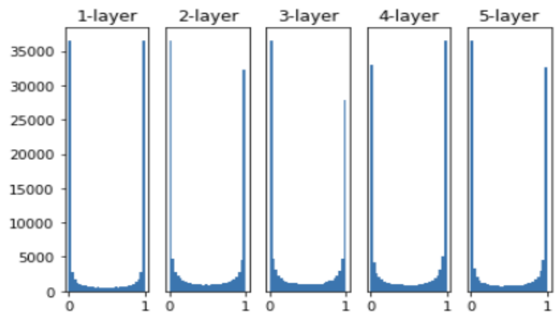
4. Declaring the variables

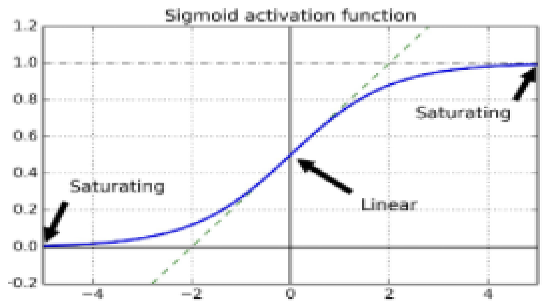
```
epoch = 10000
learning_rate = 0.03
input_layer = 2
hidden_layer = 5
output_layer = 2
w1 = np.random.rand(input_layer,hidden_layer)
w2 = np.random.rand(hidden_layer, output_layer)

classification_accuracy_per_epoch = np.array([])
total_cost_per_epoch = np.array([])
n_epoch = np.array([])
```

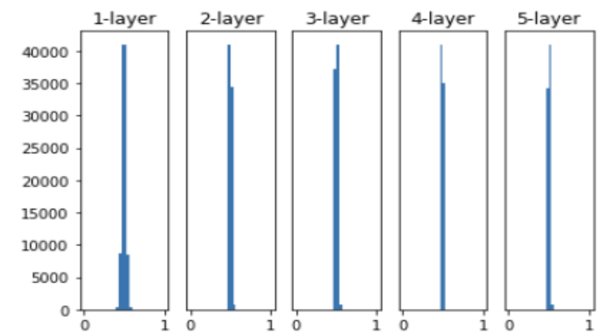
- **np.random.rand(row,col) → fill the array with row and col size by random values from 0~1.**
- **Empty arrays are also needed to store the accuracy, cost and epoch number**

Other weight initialization techniques

It is not good if the variance is too large or too small.







Xavier initialization

- Xavier Normal Initialization

$$W \sim N(0, Var(W))$$

$$sigma = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

- Xavier Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \ +\sqrt{\frac{6}{n_{in} + n_{out}}})$$

```
w1 = np.random.normal(0,np.sqrt(2/(input_layer+hidden_layer)),size=(input_layer,hidden_layer))
w2 = np.random.normal(0,np.sqrt(2/(hidden_layer+output_layer)),size=(hidden_layer,output_layer))
```

He initialization (for ReLU)

- He Normal Initialization

$$W \sim N(0, Var(W))$$

$$sigma = \sqrt{\frac{2}{n_{in}}}$$

(n_{in} : 이전 layer(input)의 노드 수)

- He Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, \ +\sqrt{\frac{6}{n_{in}}})$$

(n_{in} : 이전 layer(input)의 노드 수)

5. Function define

Activation function

```
def activation (z):
    g_z = 1/(1+np.exp(-z))
    return g_z
```

Feedforward

```
def feedforward (feature,w1,w2):
    x = np.array(feature, ndmin=2)
    z2 = np.dot(x,w1)
    x2 = activation(z2)
    z3 = np.dot(x2,w2)
    h = activation(z3)
    return x2, h
```

※ dimension scaling of the feature is conducted in the Feedforward function.

ANN programming – XOR

5. Function define

Backpropagation

```
def backpropagation (h,d,x1,x2,x3):
    x = np.array(x1, ndmin=2)
    dd = np.array(d,ndmin=2)
    e3 = h-dd
    delta3 = e3
    e2 = np.dot(delta3,w2.T)
    delta2 = e2*x2*(1-x2)

    w2_update = -learning_rate*np.dot(x2.T,delta3)
    w1_update = -learning_rate*np.dot(x.T,delta2)

    return w1_update, w2_update
```

※ dimension scaling of the d is conducted in the backpropagation function

Inference

```
def inference (h):
    inferred_label = np.argmax(h)
    return inferred_label
```

ANN programming – XOR

6. Main Body – 1) Evaluation of classification accuracy before training

```
scorecard = np.array([])   Empty arrays for the score and cost for each example
cost = np.array([])
for record in training_data_list:    Open the example one by one
        all_values = record.split(',')
        feature = np.asfarray(all_values[1:])
        correct_label = int(all_values[0])    Make feature and d (correct answer)
        d = np.zeros(2)
        d [correct_label] = 1

        x2, h = feedforward (feature,w1,w2)    Conduct feedforward and print h vector
        print (h)

        if (inference (h)==correct_label):    If inference(h) is equal to the correct_label
            scorecard = np.append(scorecard,1)  → Append 1 to the score card.
        else:                                  → If it is not ,append 0 to the score card
          scorecard = np.append(scorecard,0)
        cost_example = np.sum(-d*np.log(h)-(1-d)*np.log(1-h))  Calculating the cost for the example
        cost = np.append(cost, cost_example)                    and append the cost to the cost array
classification_accuracy = np.sum(scorecard)/4
classification_accuracy_per_epoch = np.append(classification_accuracy_per_epoch, classification_accuracy)
total_cost = np.sum(cost)/4
total_cost_per_epoch = np.append(total_cost_per_epoch, total_cost)
n_epoch = np.append(n_epoch, 0)
```

Check all examples and calculate the classification accuracy and store the results in athe empty array.
Calculate the total cost (average after adding all costs) and store it in the empty array.
Save Epoch number (save as 0 because it is before the learning).

ANN programming – XOR

6. Main Body – 1) Evaluation of classification accuracy before training

```
print ("initial cost: ", total_cost)
print ("initial accuracy: ", classification_accuracy)
```

```
[[0.63800817 0.73023477]]
[[0.62912345 0.71562709]]
[[0.62664513 0.71221607]]
[[0.61747738 0.69675864]]
initial cost =   1.5215071060210743
initial_accuracy =   0.5
```

ANN programming – XOR

6. Main Body – 2) Training

```
for i in range(epoch):
    scorecard = np.array([])
    cost = np.array([])

    for record in training_data_list:
        all_values = record.split(',')

        feature = np.asfarray(all_values[1:])
        correct_label = int(all_values[0])
        d = np.zeros(2)
        d [correct_label]=1

        x2, h = feedforward(feature,w1,w2)
        w1_update, w2_update = backpropagation (h,d,feature,x2,h)

        w1 = w1+w1_update
        w2 = w2+w2_update
```

Open the example one by one

Make feature and d (correct answer)

Conduct feedforward
And calculate the amount of the update by the backpropagation

W1, W2 update

ANN programming – XOR

6. Main Body – 2) Epoch test

```
    w1 = w1+w1_update
    w2 = w2+w2_update
```

```
for record in training_data_list:
    all_values = record.split(',')
    feature = np.asfarray(all_values[1:])
    correct_label = int(all_values[0])
    d = np.zeros(2)
    d [correct_label] = 1

    x2, h = feedforward (feature,w1,w2)


    if (inference (h)==correct_label):
        scorecard = np.append(scorecard,1)
    else:
      scorecard = np.append(scorecard,0)
    cost_example = np.sum(-d*np.log(h)-(1-d)*np.log(1-h))
    cost = np.append(cost, cost_example)
classification_accuracy = np.sum(scorecard)/np.size(scorecard)
classification_accuracy_per_epoch = np.append(classification_accuracy_per_epoch, classification_accuracy)
total_cost = np.sum(cost)/4
total_cost_per_epoch = np.append(total_cost_per_epoch, total_cost)
n_epoch = np.append(n_epoch, i+1)
```

Open the example one by one

Same with the evaluation code before the training

Save accuracy and cost of this epoch

ANN programming – XOR

6. Main Body – 3) Checking the final state

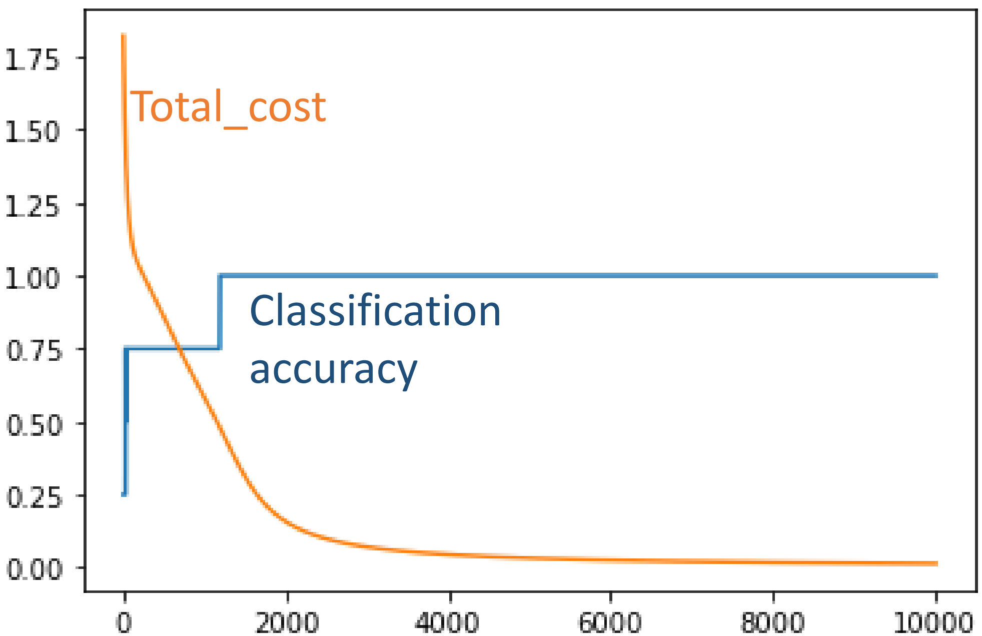
```
scorecard = np.array([])
cost = np.array([])
for record in training_data_list:
    all_values = record.split (',')
    feature = np.asfarray(all_values[1:])
    correct_label = int(all_values[0])
    d = np.zeros(2)
    d [correct_label] = 1
    x2, h = feedforward (feature, w1,w2)
    print (h)
    if (inference(h)==correct_label):
        scorecard = np.append(scorecard,1)
    else:
        scorecard = np.append(scorecard,0)
    cost_example = np.sum(-d*np.log(h)-(1-d)*np.log(1-h))
    cost = np.append(cost, cost_example)

classification_accuracy = np.sum(scorecard)/4
total_cost = np.sum(cost)/4
print ("Final cost: ", total_cost)
print ("Final accuracy: ", classification_accuracy)
plt.plot(n_epoch,classification_accuracy_per_epoch)
plt.plot(n_epoch,total_cost_per_epoch)
```

Same with the evaluation code before the training

Print final cost and accuracy
Plot the graphs

ANN programming – XOR

6. Main Body – 3) Checking the final state




```
[[9.99755932e-01 2.79294619e-04]]
[[0.0058944 0.9943169]]
[[9.99755932e-01 2.79294619e-04]]
[[0.9927184  0.00700539]]
Final cost:  0.00674907411360905
Final accuracy:  1.0
```

Mini batch

If batch = 2, we can modify the code as follows

```
for i in range(epoch):
    scorecard = np.array([])
    cost = np.array([])
    w1_update_temp = np.zeros([input_layer, hidden_layer])
    w2_update_temp = np.zeros([hidden_layer, output_layer])
    batch = 2
    batch_count= 0
    for record in training_data_list:
        all_values = record.split(',')

        feature = np.asfarray(all_values[1:])
        correct_label = int(all_values[0])
        d = np.zeros(2)
        d [correct_label]=1

        x2, h = feedforward(feature,w1,w2)
        w1_update, w2_update = backpropagation (h,d,feature,x2,h)

        w1_update_temp = w1_update_temp+w1_update
        w2_update_temp = w2_update_temp+w2_update
        batch_count=batch_count+1

        if (batch_count==batch):
            w1 = w1+w1_update_temp/batch
            w2 = w2+w2_update_temp/batch
            batch_count = 0
            w1_update_temp = np.zeros([input_layer, hidden_layer])
            w2_update_temp = np.zeros([hidden_layer, output_layer])
```

1) Temp array
2) Batch size & batch count

1) Calculate the amount of the updates and add to the temp arrays
2) batch_count +1

If batch_count == batch
Conduct update and initialize the count and temp arrays

Class Exersize (XNOR)

- **Implent the program for the XNOR (with 2 hidden layers, minibatch of 2)**

XNOR	Input 1	Input 2
1	1	1
0	1	0
0	0	1
1	0	0

Contents

3 MNIST data

MNIST data

- **https://pjreddie.com/projects/mnist-in-csv/**

home darknet coq tactics publications projects résumé

MNIST in CSV

Here's the train set and test set. Click & download

The format is:

```
label, pix-11, pix-12, pix-13, ...
```

where `pix-ij` is the pixel in the `ith` row and `jth` column.

For the curious, this is the script to generate the csv files from the original data.

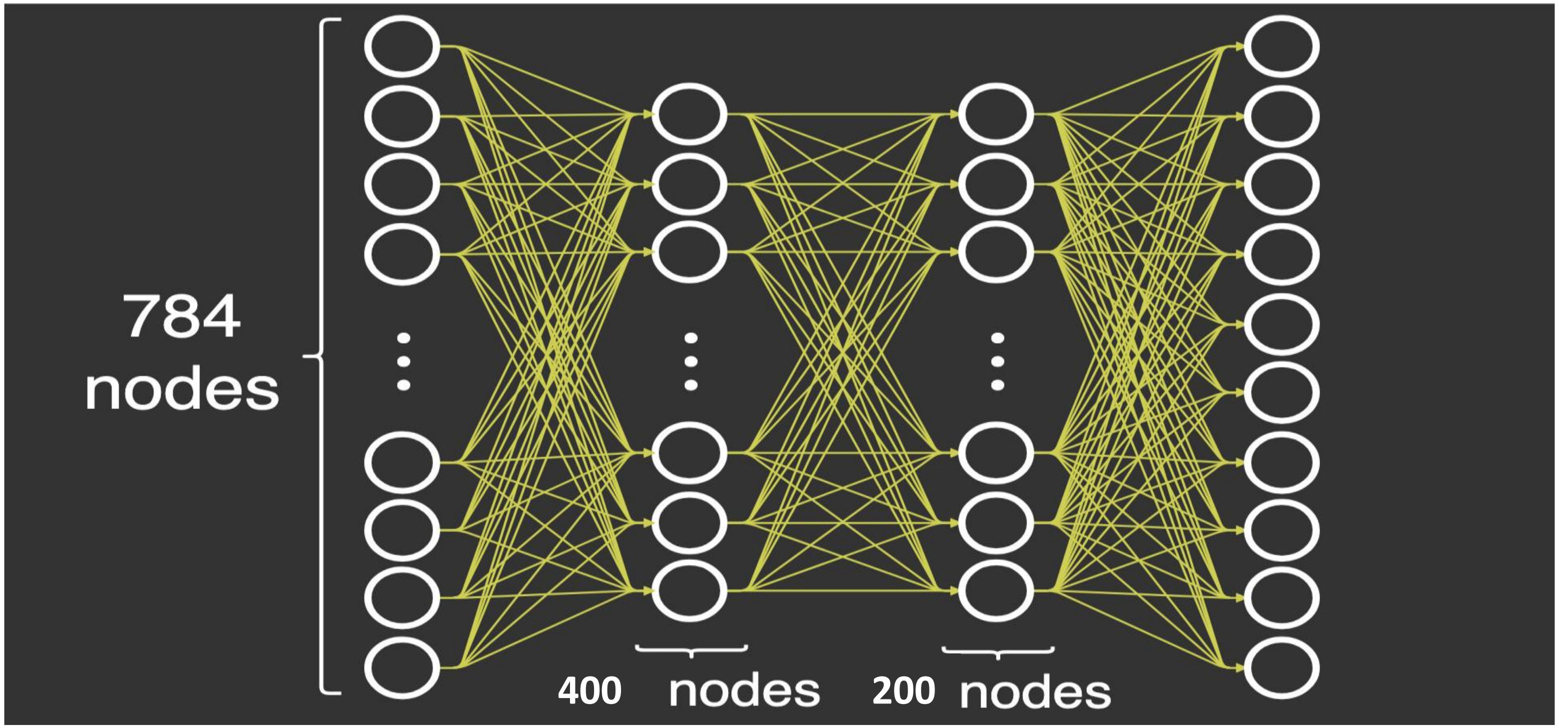
```
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")
```

MNIST data

5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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
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
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Label Pixel value (x_1~x_{784})

MNIST Programming (Project 2)



MNIST Programming (Project 2)

- We should modify the number of neurons
- The test should be carried out with the test set
  → Statements for Test set open, readline & close are needed.
  → Classification accuracy & cost calculation should be conducted with the test set.
- Use the mini-batch (start from the batch size of 5)
- Activation function : tanh, use the softmax only for the output layer
- We will only use the first 10,000 examples in the training set and the first 1,000 examples in the test set (because it will take so long time if we use the entire data set.)
- If you finish to implement the code with the guideline above, change the activation function (expect the softmax for the output layer) and compare the results (tanh vs sigmoid)
- Change the batch size to 100 and 200 then compare the results (5 vs 100 vs 200)