6강. 딥러닝을 통한 데이터분석

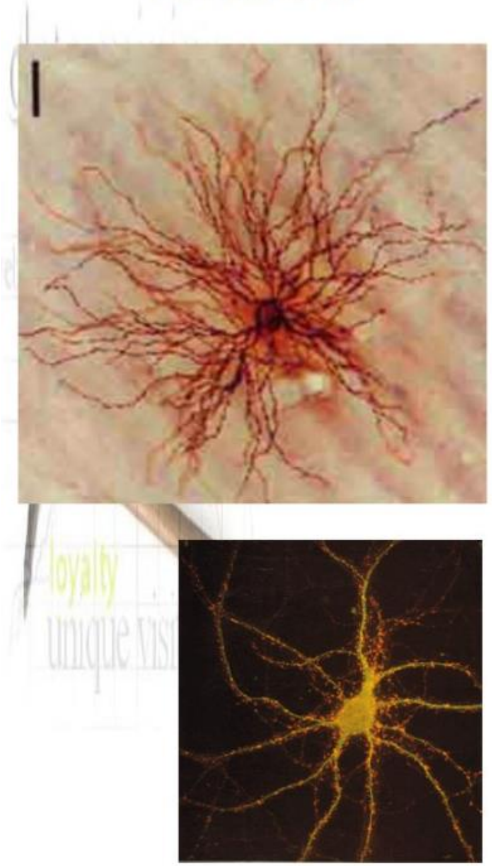
2023.01.06.

양희철
hcyang@cnu.ac.kr

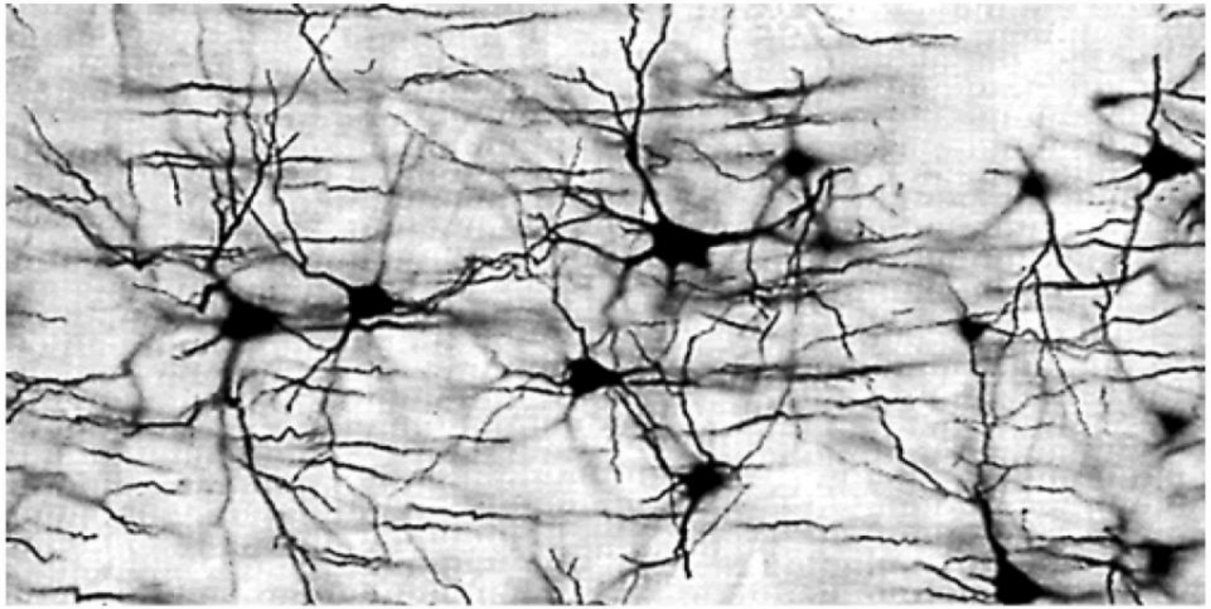
Perceptron

Neural networks

Neuron



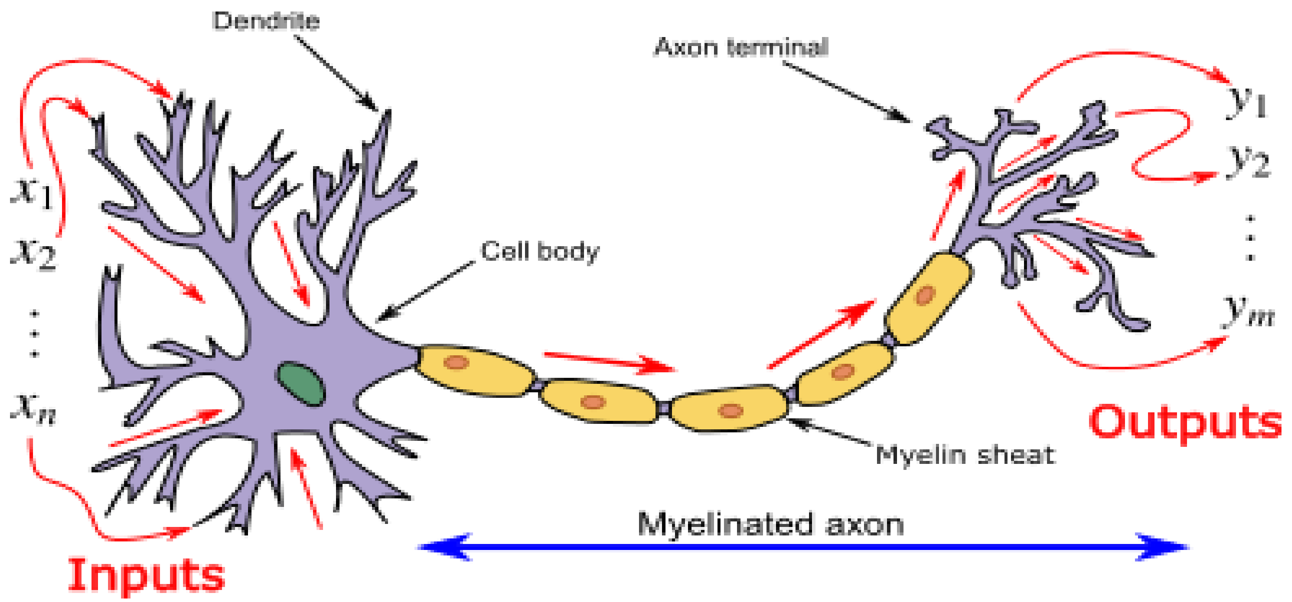
Neural Network



Perceptron

Perceptron modeling

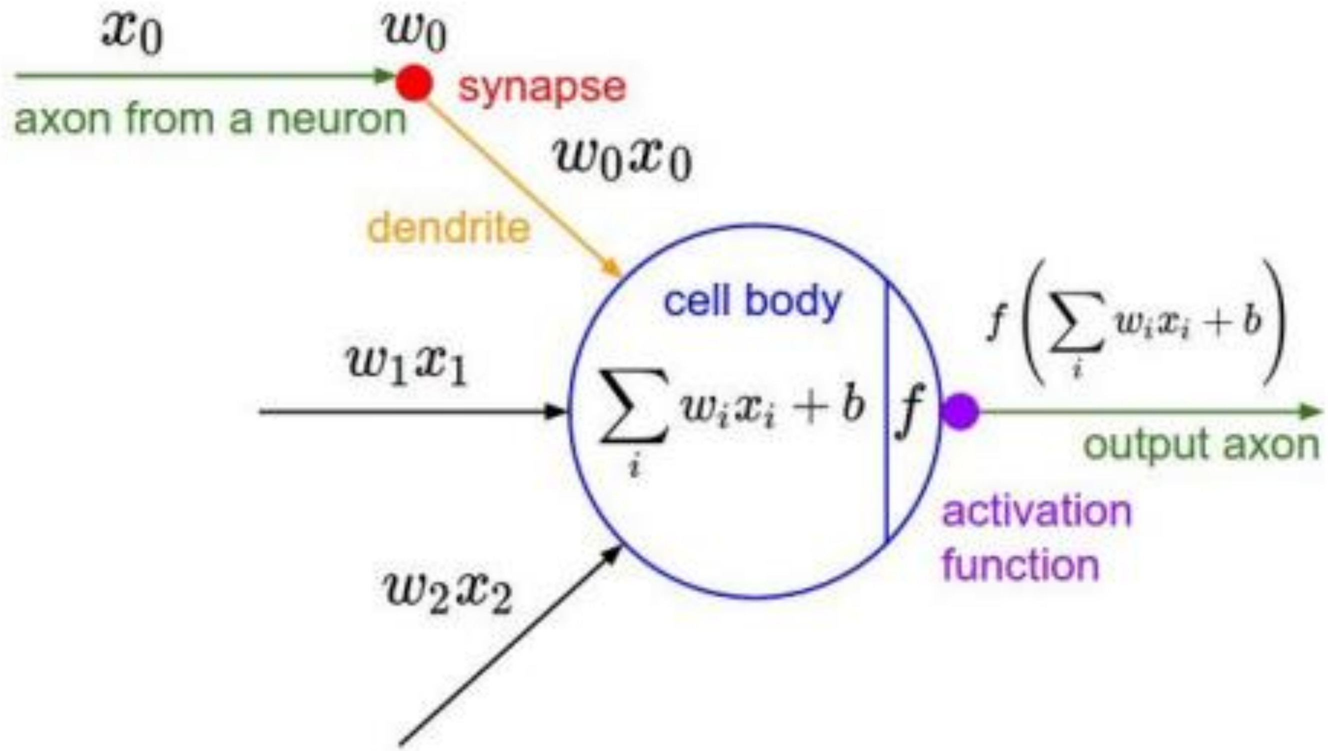
- Proposed by Frank Rosenblatt (1957)
- A mathematical model that mimics human brain



Perceptron

Perceptron modeling

- We can express a linear function as a network

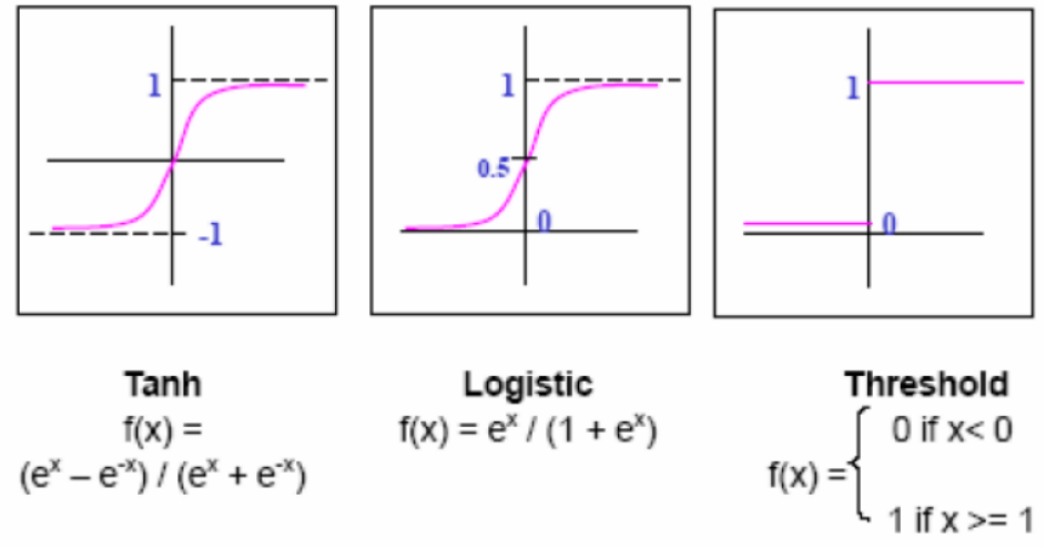


Perceptron

Activation function

- Continuous output → Probability of decision

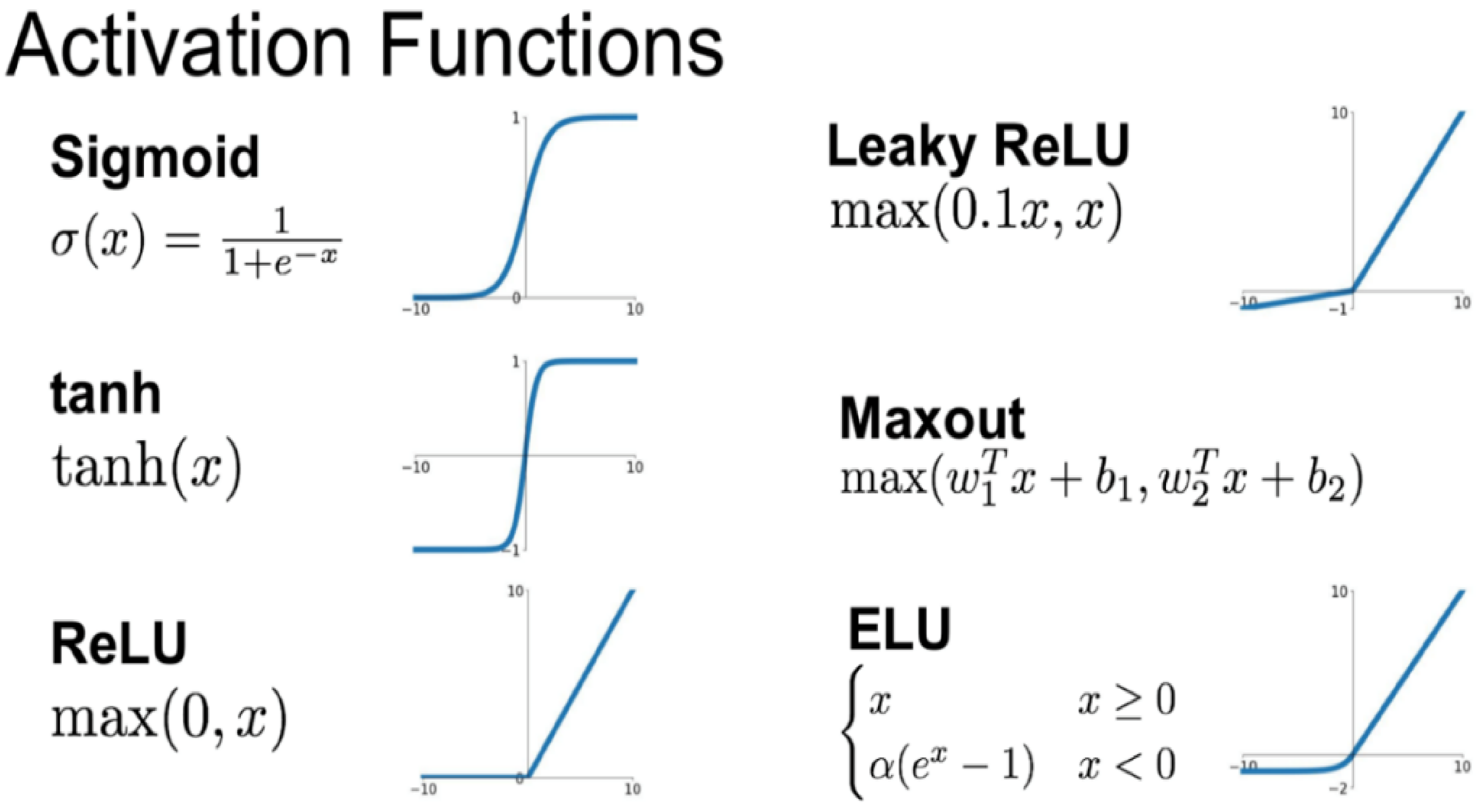
$$out = f(\sum_{i=1}^{d} w_i x_i + w_o) = f(\sum_{i=0}^{d} w_i x_i) = f(\mathrm{w}^T \mathrm{x})$$



Perceptron

Activation function

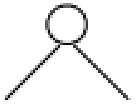
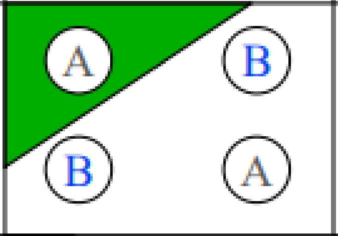
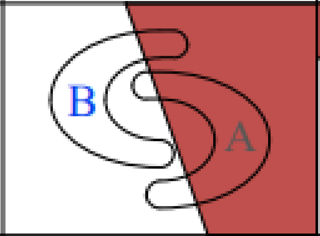
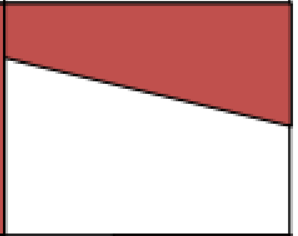
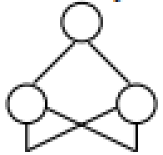
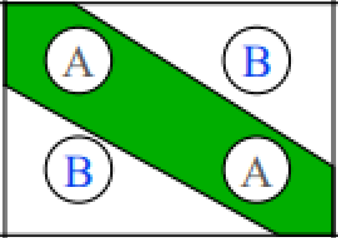
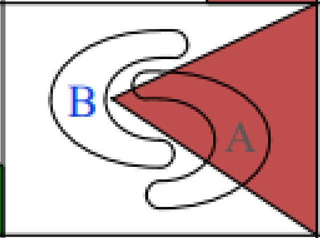
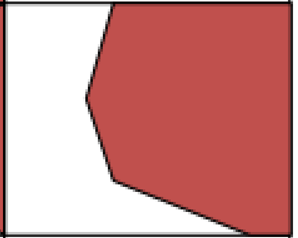
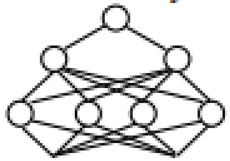
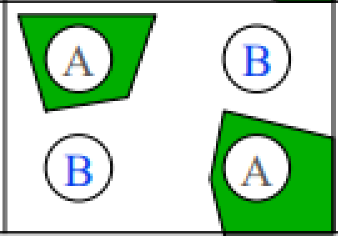
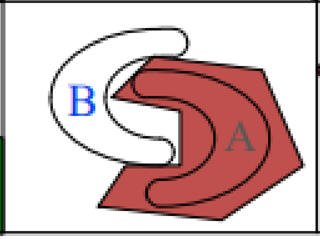
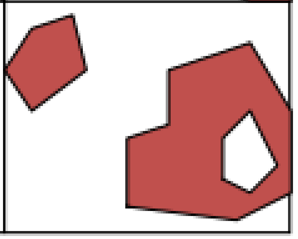
- Examples



Multilayer Perceptron (MLP)

Nonlinear classification

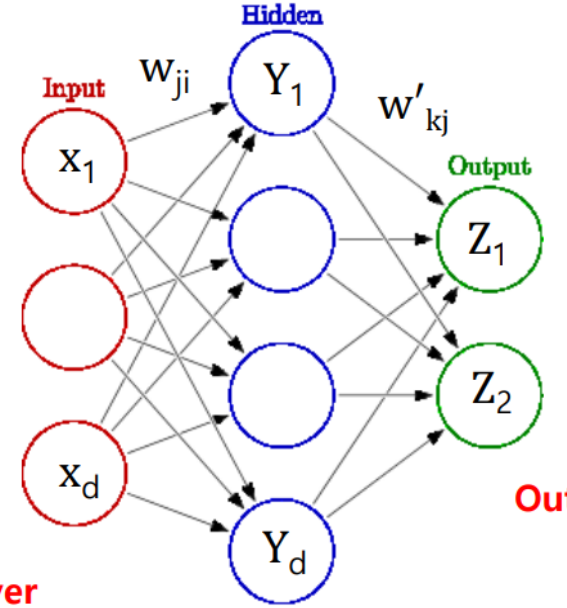
- MLP can handle nonlinear problems (Combination of multiple linear classifier)

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Abitrary (Complexity Limited by No. of Nodes)			

Multilayer Perceptron (MLP)

Multilayer perceptron

- Feedforward process



Output layer

Input layer

Hidden layer

$$y_j = f(\sum_{i=0}^{d} w_{ji} x_i) = f(\mathrm{w}^T \mathrm{x})$$

$$Z_k = f(\sum_{j=0}^{h} w'_{kj} y_j) = f(\mathrm{w}'^T y)$$

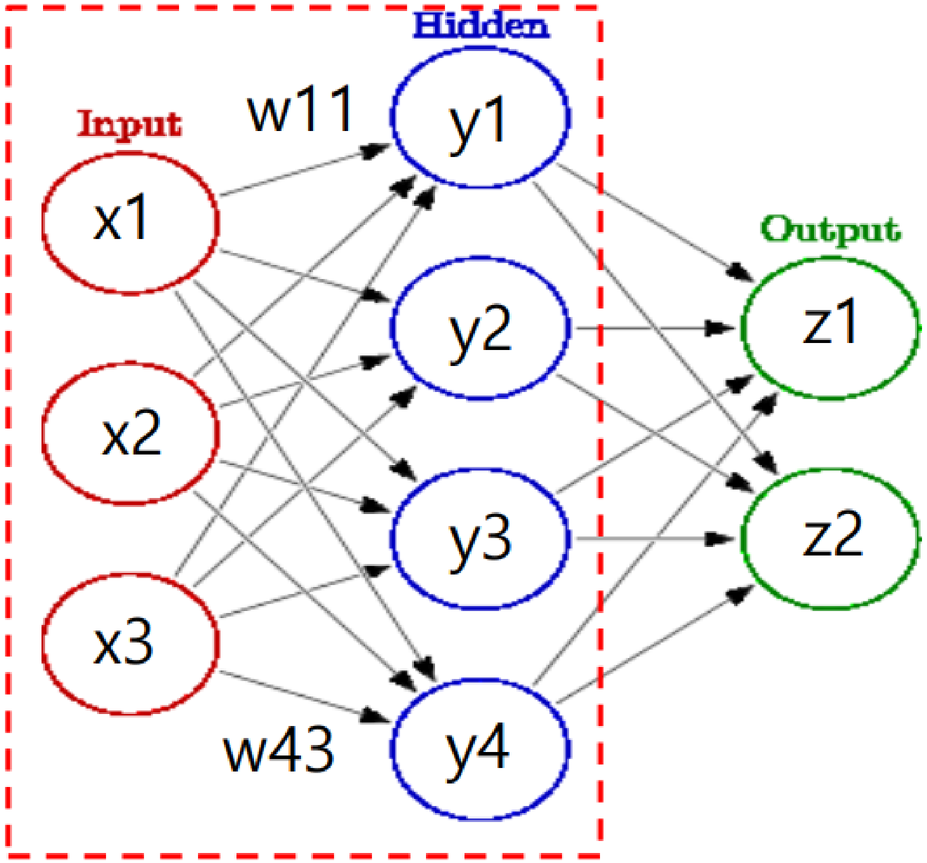
$$z_k = f(\mathrm{w}'^T y) = f(\mathrm{w}'^T f(wTx))$$

Multilayer Perceptron (MLP)

Multilayer perceptron

- Feedforward process (input to hidden)

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv w_j^t . x,$$

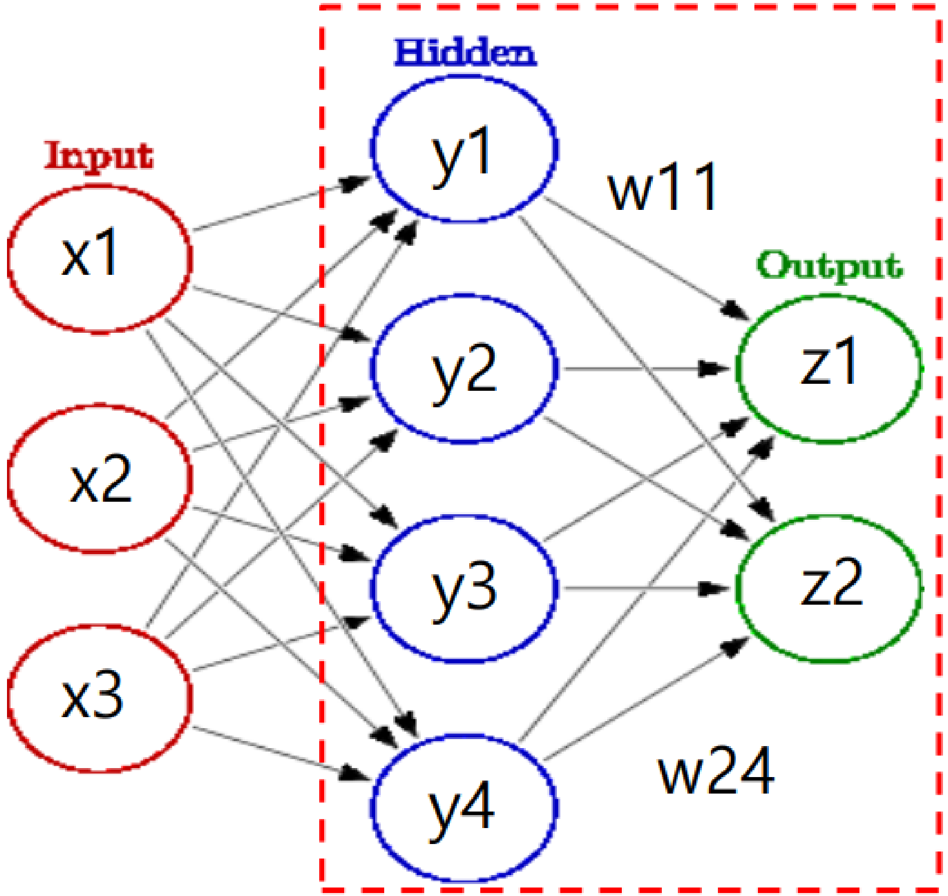


Multilayer Perceptron (MLP)

Multilayer perceptron

- Feedforward process (hidden to output)

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = w_k^t . y,$$



Multilayer Perceptron (MLP)

Multilayer perceptron

- Feedforward process (input to output)

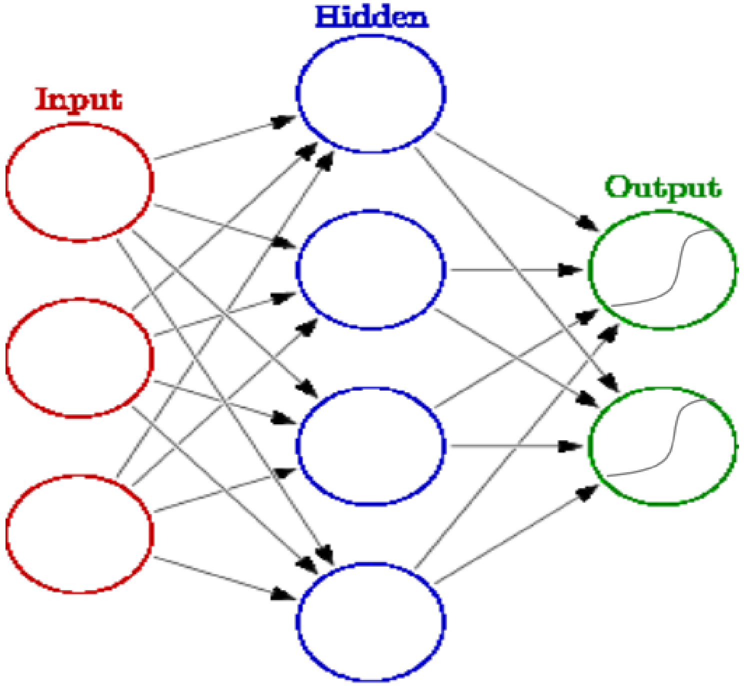
$$g_k(x) \equiv z_k = f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right) \qquad (1)$$

$$(k = 1,\ldots,c)$$

Forward Propagation

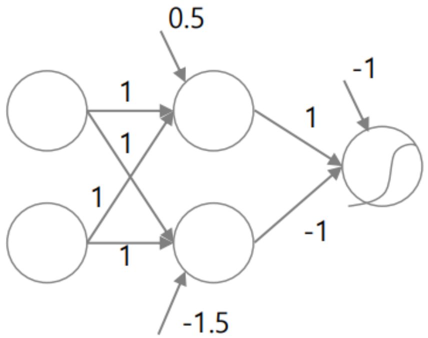
NN structure: activation function

- Activation function for output units: sigmoid units (2-class classification)



$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$

$$where\ \sigma(x) = \frac{1}{1 + \exp(-x)}$$

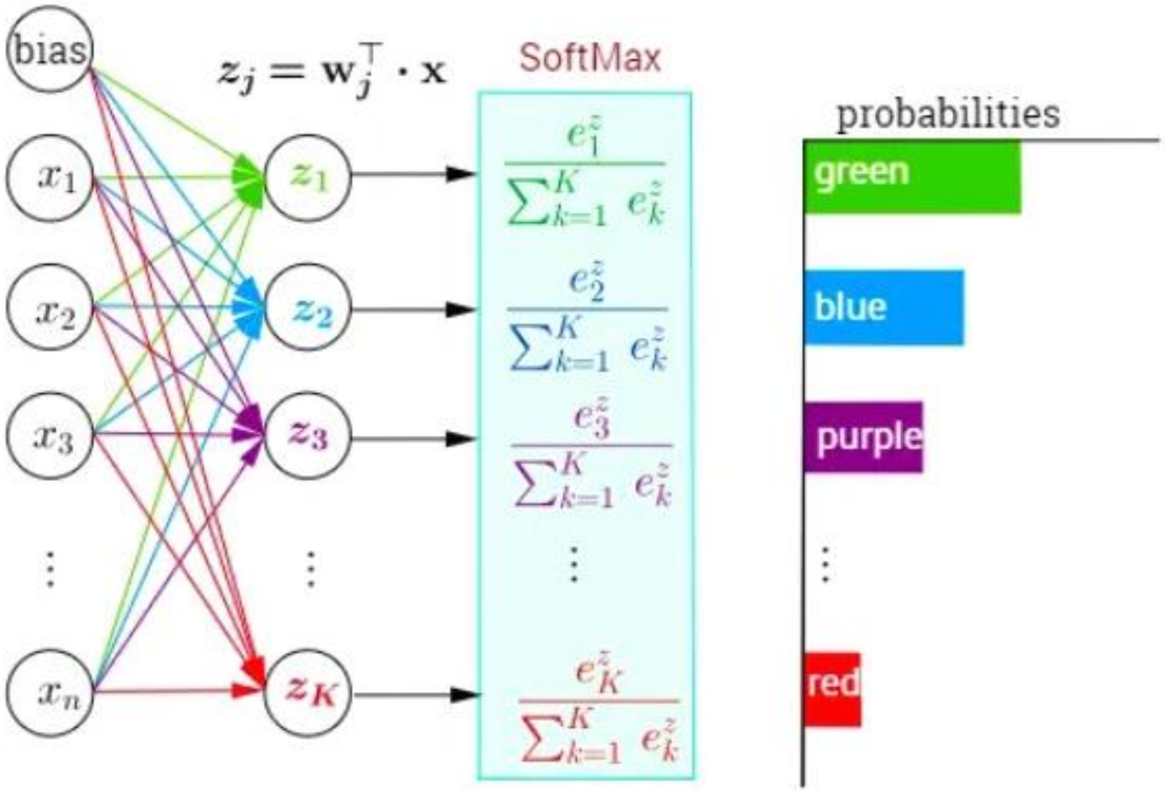


Forward Propagation

NN structure: activation function

- Activation function for output units: SoftMax units (multi-class classification)

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



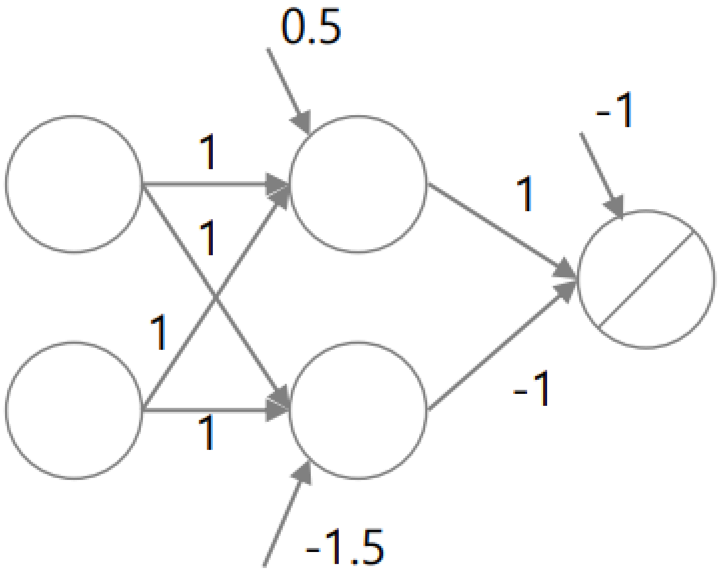
$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Forward Propagation

NN structure: activation function

- Activation function for output units: linear units (regression)

$$\hat{y} = \boldsymbol{W}^{\top}\boldsymbol{h} + \boldsymbol{b}$$

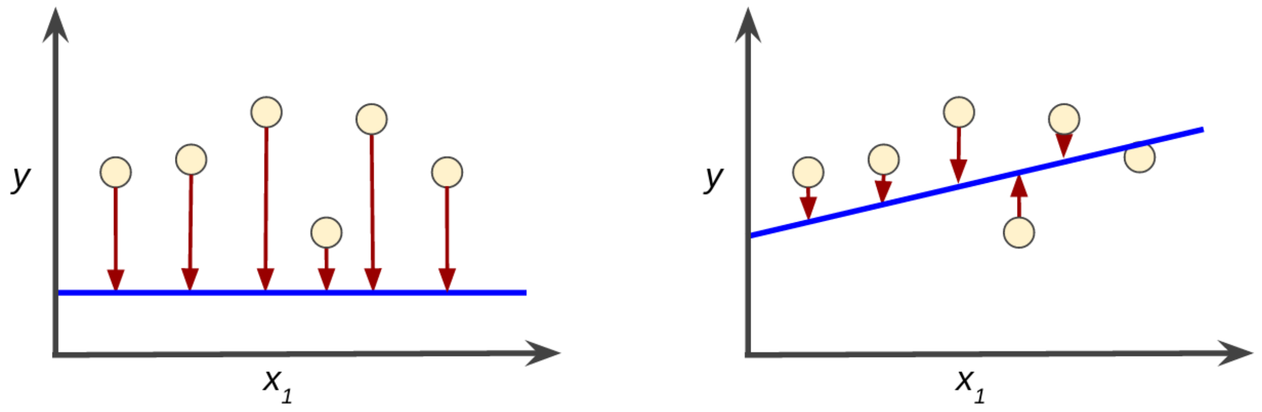


Forward Propagation

NN structure: loss function

- Regression: Minimize MSE (Mean Square Error)

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$



Forward Propagation

NN structure: loss function

- Classification: Minimize the cross-entropy

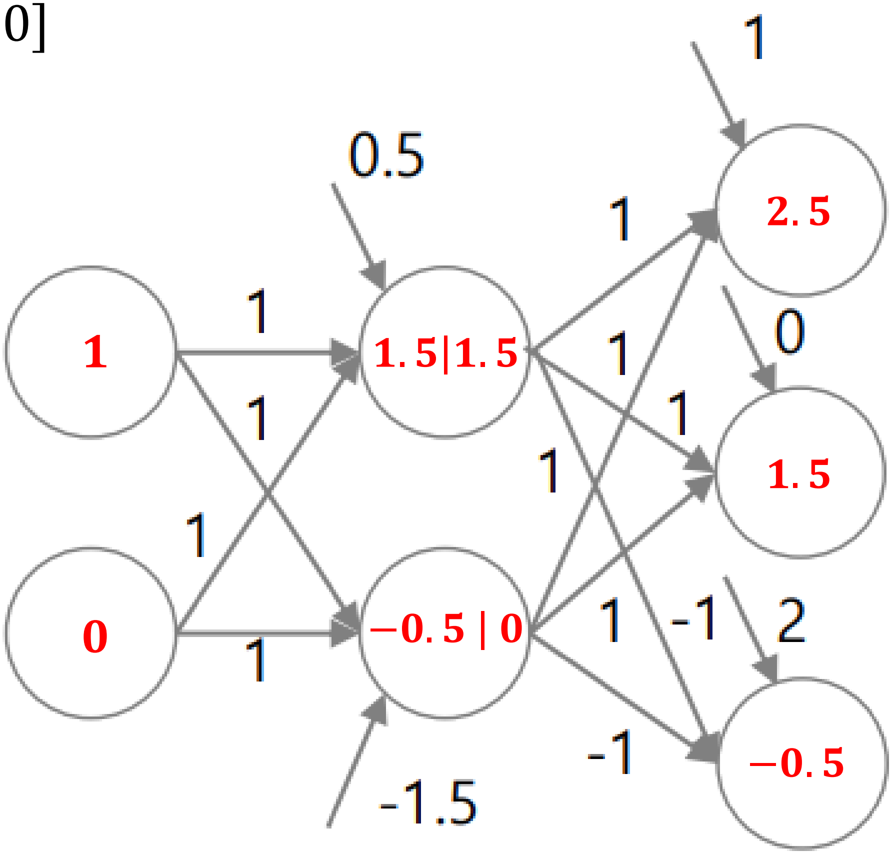
$$H(P,Q) = -\sum_i P(x_i) \log Q(x_i)$$

Forward Propagation

Feedforward process

- Example: $\boldsymbol{x} = [1, 0]$, $y = [1\ 0\ 0]$

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



$$\frac{e^{2.5}}{e^{2.5} + e^{1.5} + e^{-0.5}} = 0.705$$

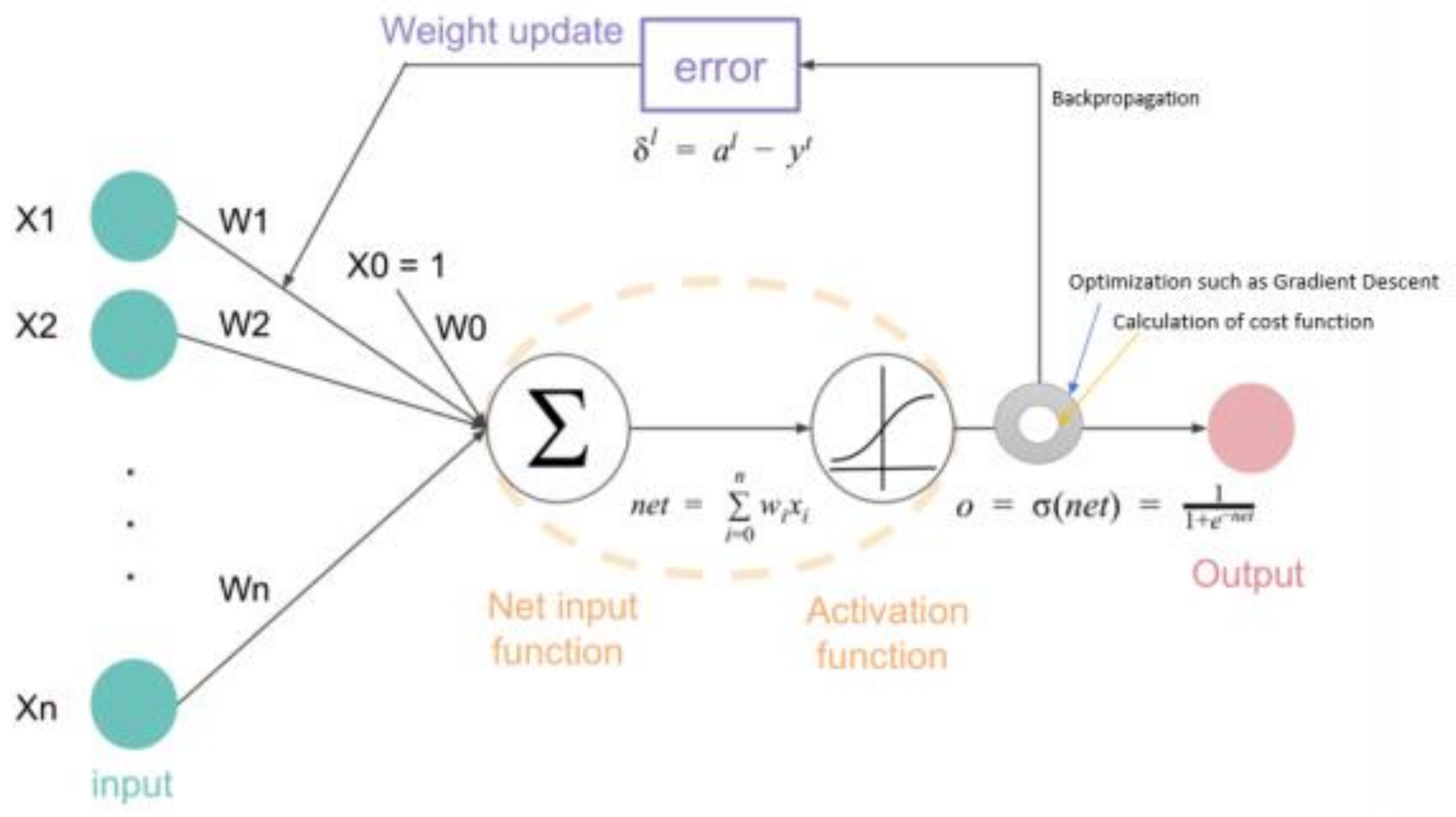
$$\frac{e^{2.5}}{e^{2.5} + e^{1.5} + e^{-0.5}} = 0.259$$

$$\frac{e^{2.5}}{e^{2.5} + e^{1.5} + e^{-0.5}} = 0.035$$

Backpropagation

Error backpropagation

- Weight optimization



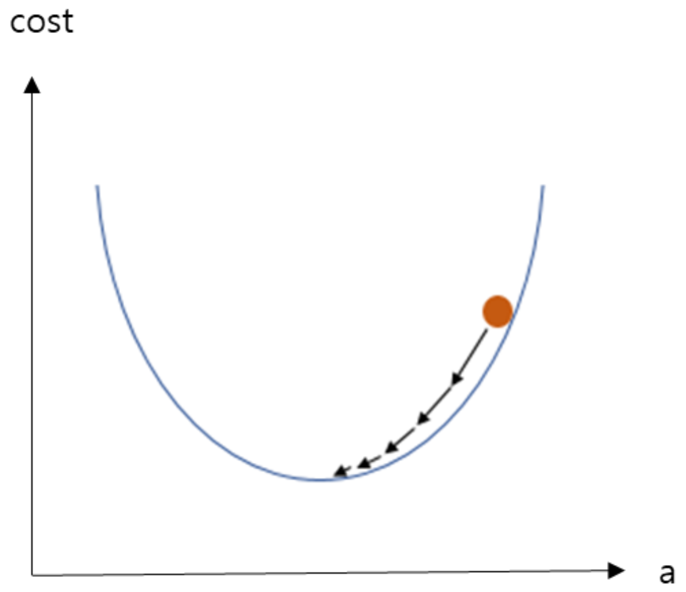
Backpropagation

Error backpropagation

- Weight optimization
    - Recap) Gradient descent

$$w_{k+1} = w_k + \eta\left(-\frac{\partial J(w)}{\partial w}\right)$$

$$= w_k - \eta\frac{\partial J(w)}{\partial w}$$



Backpropagation

Error backpropagation

- Example

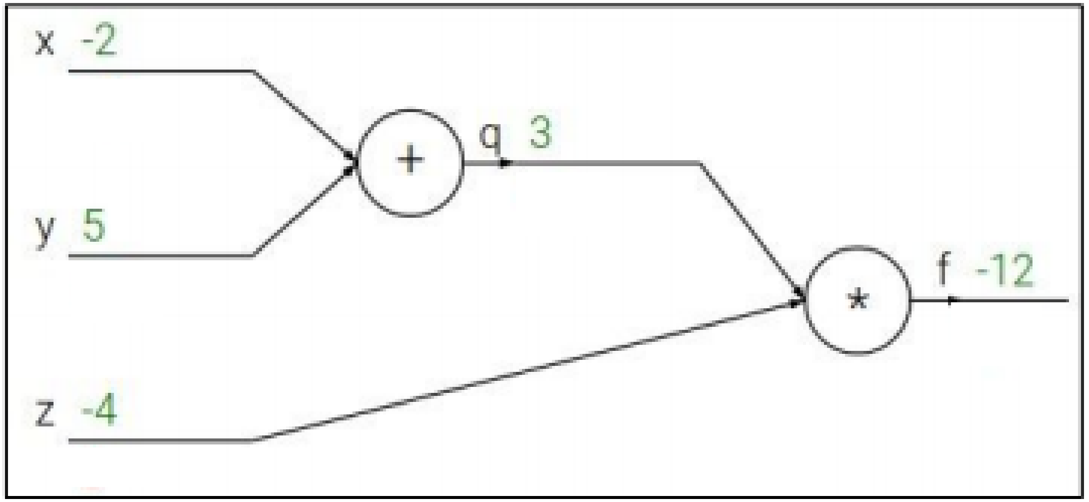
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation

Error backpropagation

- Example

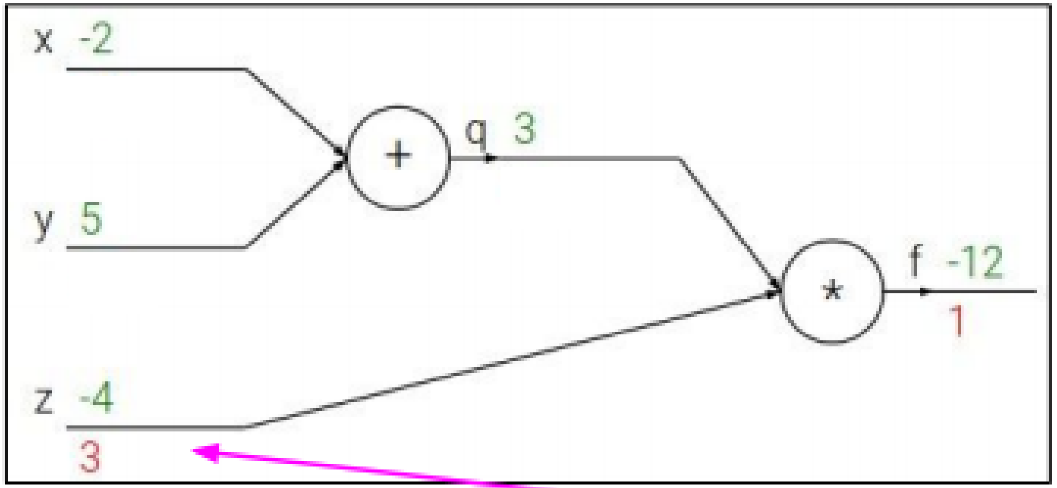
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

Backpropagation

Error backpropagation

- Example

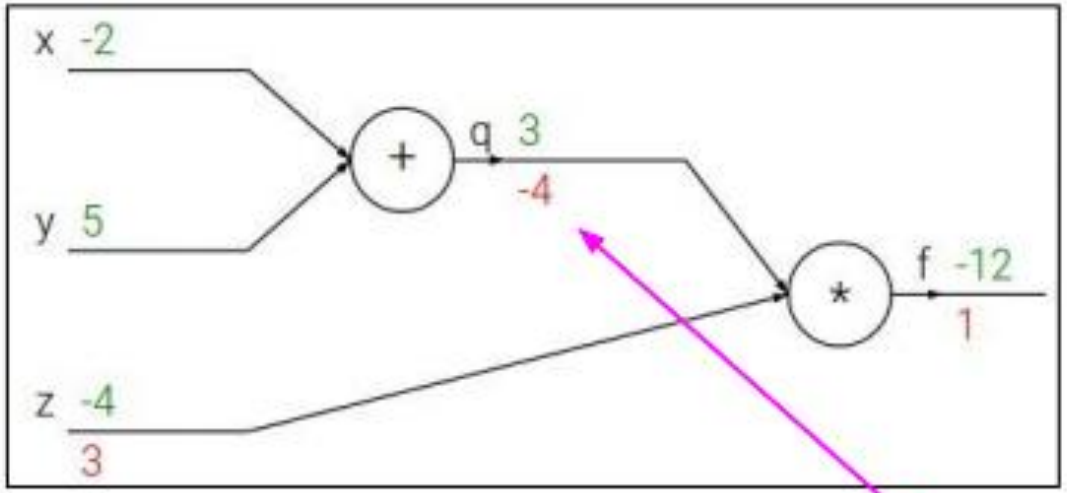
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial q}}$$

Backpropagation

Error backpropagation

- Example

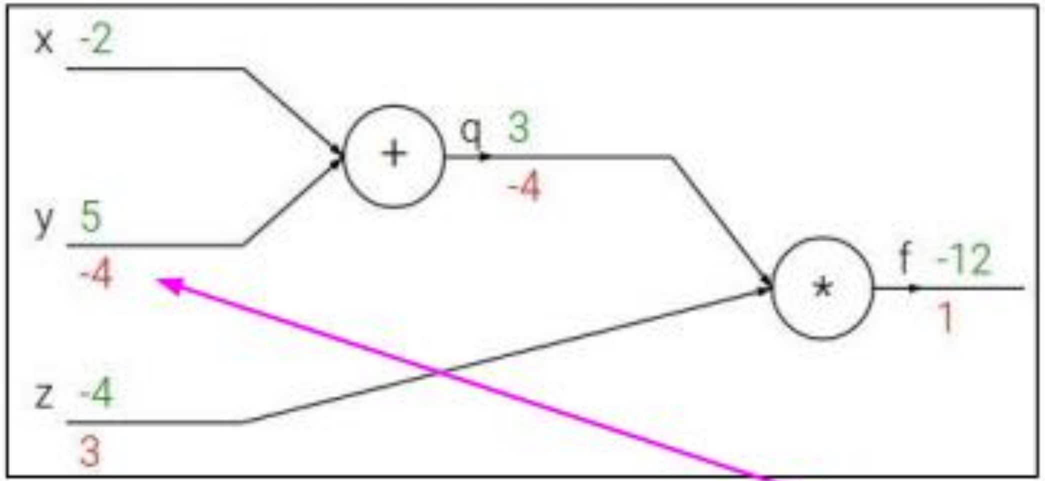
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

Upstream gradient

Local gradient

Backpropagation

Error backpropagation

- Example

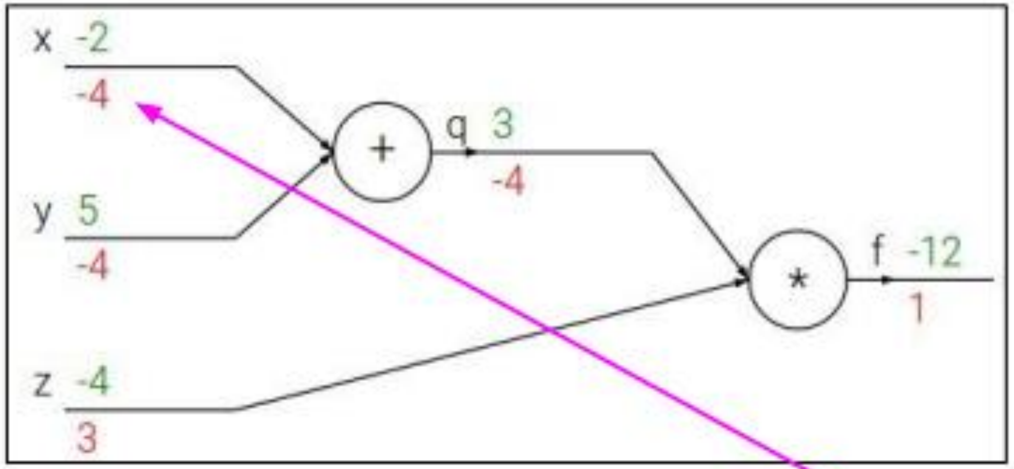
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\boxed{\frac{\partial f}{\partial x}}$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Upstream gradient

Local gradient

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
  - 1) 건강 데이터를 판다스를 통해 불러오자.

```python
import pandas as pd

life = pd.read_csv('https://raw.githubusercontent.com/dongupak/DataSciPy₩
/master/data/csv/life_expectancy.csv')
print(life.head())
```

```python
# 기대수명 데이터를 살펴보는 메소드 describe()
life.describe()
```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
  - 2) 건강 데이터 중 기대수명과 큰 상관관계를 가지는 열만 추출하자. 또한 결손값이 있는 데이터는 삭제하자.

```
life = life[['Life expectancy', 'Alcohol', 'Percentage expenditure', 'Measles', 'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']]
life.dropna(inplace = True)
print(life)
```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
  - 3) 데이터를 훈련용 80%과 테스트용 20%으로 나누자.

```python
from sklearn.model_selection import train_test_split

X = life[['Alcohol', 'Percentage expenditure', 'Measles', 'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']]
y = life[['Life expectancy']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
    - 4) 모든 데이터를 평균 0, 표준편차 1이 되도록 변경하고자 한다. sklearn의 StandardScaler를 활용하자.

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train, X_test
```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
  - 5) Regression을 위한 인공신경망 모델을 만들자. 하나의 값을 예측하는 회귀 문제이므로 마지막 네트워크는 출력이 1이 되도록 한다.

```
import tensorflow as tf
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(1, activation='relu')
])

```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
    - 6) 평균제곱오차(MSE)를 손실 함수로 사용하고 adam 최적화 함수를 사용해 모델을 학습하자. (epochs=100)

```
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])
```

```
model.fit(X_train, y_train, epochs=100)
```

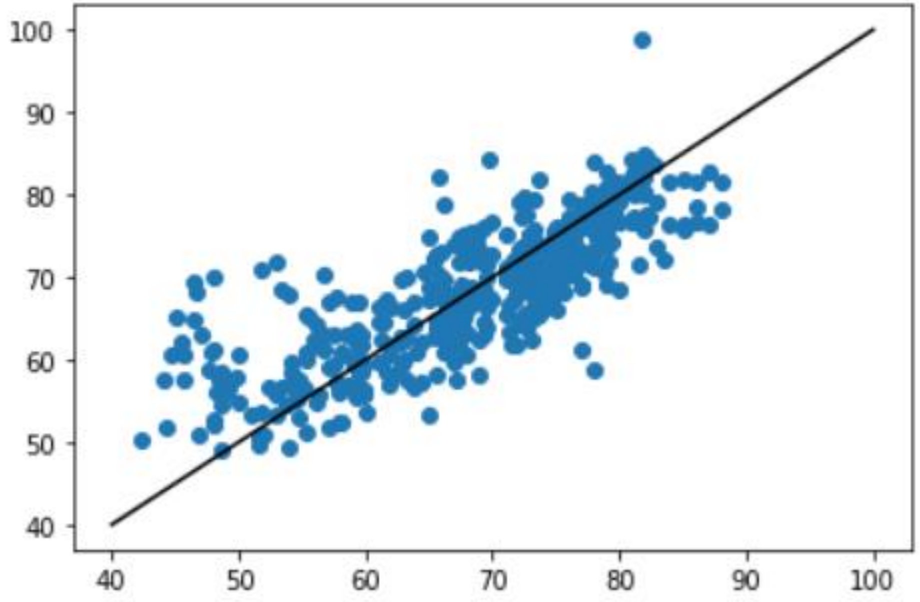
```
Epoch 99/100
58/58 [==============================] - 0s 2ms/step - loss: 28.0779 - mse: 28.0779
Epoch 100/100
58/58 [==============================] - 0s 2ms/step - loss: 28.0815 - mse: 28.0815
<keras.callbacks.History at 0x7fb728b56850>
```

Deep Learning Practice 1

Practice: Regression

- 건강 데이터를 통해 기대수명을 예측하는 인공신경망을 구현하고자 한다.
  - 7) X_test 데이터를 입력으로 사용하여 기대수명을 예측하자. 예측값을 목표값 y_test와 비교하여 일치하는지를 산포도 그래프로 확인하자.

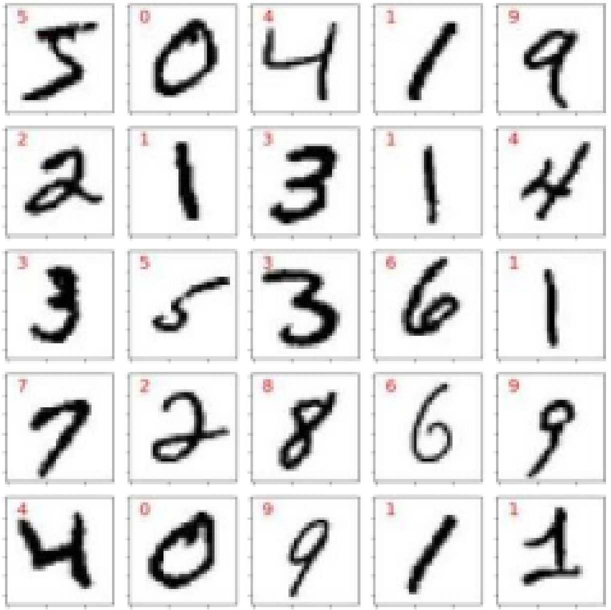
```python
import matplotlib.pyplot as plt
y_test_predict = model.predict(X_test)
plt.scatter(y_test, y_test_predict)
plt.plot([40, 100], [40, 100], c='k')
plt.show()
```



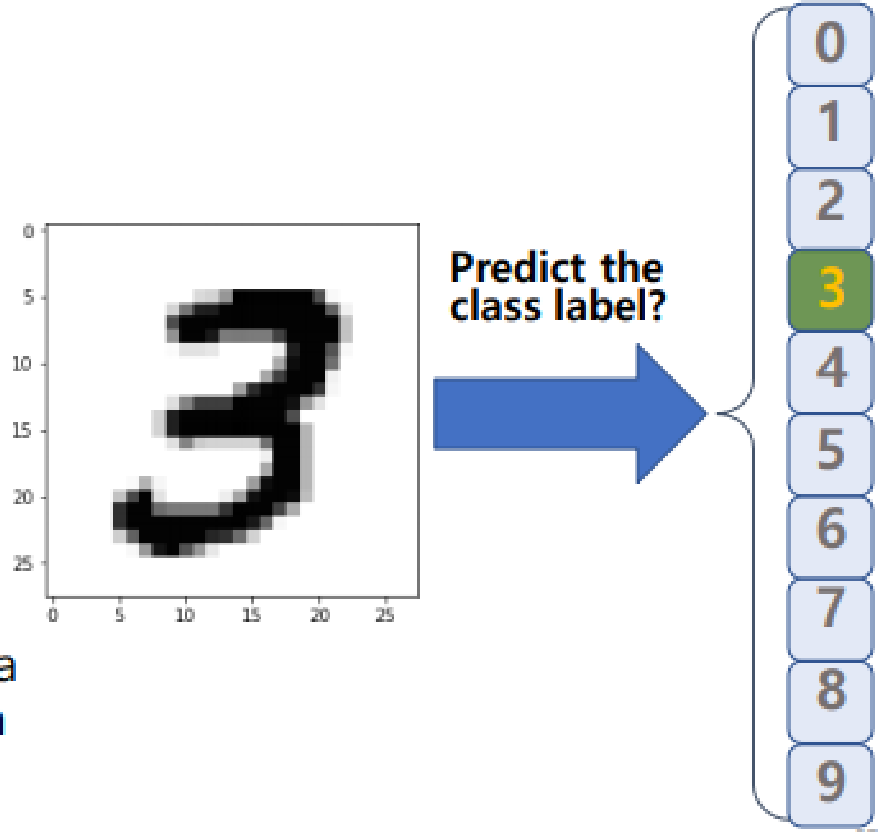
Deep Learning Practice 2

Practice: Classification

- Handwritten DIGIT classification



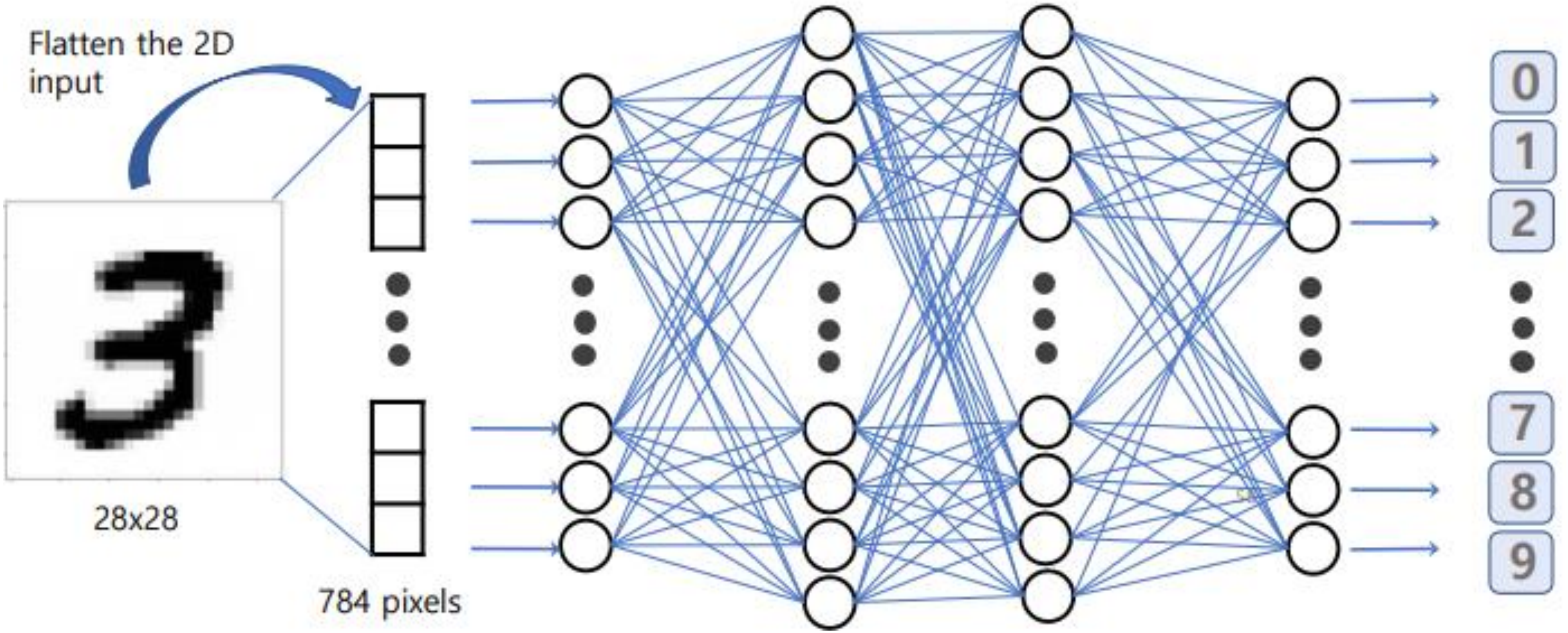
Each sample is a 28x28 gray-scale image of hand-written digits 0—9. Ground-truth labels are shown in red.



Deep Learning Practice 2

Practice: Classification

- Handwritten DIGIT classification



Deep Learning Practice 2

Import package & dataset

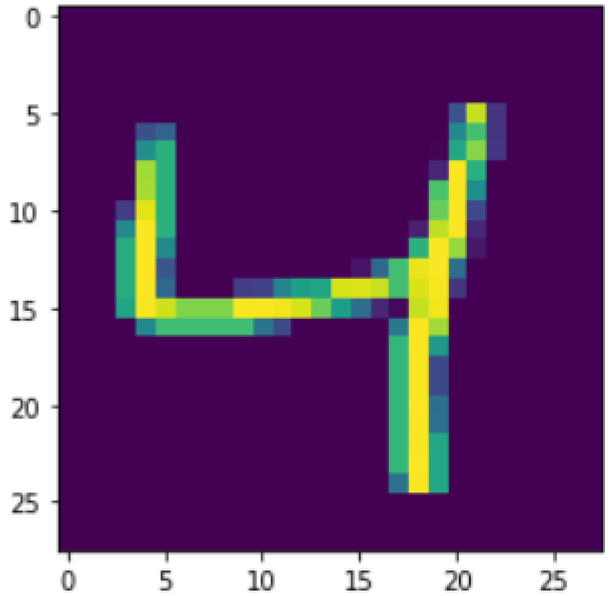
```
[1]
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras

data_mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = data_mnist.load_data()

print(train_images.shape)
print(train_labels.shape)
print(test_images.shape)
print(test_labels.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

Deep Learning Practice 2

Import package & dataset

```
[2]   1 img = plt.imshow(train_images[2])
```



Deep Learning Practice 2

Data reshaping & normalization

```
[3]  1 train_images_reshape = train_images.reshape([train_images.shape[0], train_images.shape[1]*train_images.shape[2]])
     2 test_images_reshape = test_images.reshape([test_images.shape[0], test_images.shape[1]*test_images.shape[2]])
     3 print(train_images_reshape.shape)
     4 print(train_images_reshape[0,0:300])
```

```
(60000, 784)
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   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  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0]
```

Deep Learning Practice 2

Data reshaping & normalization

[4]

```
train_images_reshape_norm = train_images_reshape/255
test_images_reshape_norm = test_images_reshape/255
print('최대: {}, 최소: {}'.format(np.max(train_images_reshape_norm),np.min(train_images_reshape_norm)))
```

```
최대: 1.0, 최소: 0.0
```

Deep Learning Practice 2

Two-class classifier

[5]

```
idx1 = np.where(train_labels == 6)
idx2 = np.where(train_labels == 8)
idx12 = np.union1d(idx1, idx2)

idx3 = np.where(test_labels == 6)
idx4 = np.where(test_labels == 8)
idx34 = np.union1d(idx3, idx4)

train_x = train_images_reshape_norm[idx12,:]
train_y = train_labels[idx12]
test_x = test_images_reshape_norm[idx34,:]
test_y = test_labels[idx34]

print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)

```

```
(11769, 784)
(11769,)
(1932, 784)
(1932,)
```

Deep Learning Practice 2

Two-class classifier

[6]

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=(10,))
clf.fit(train_x, train_y)
test_y_hat = clf.predict(test_x)

print(clf)
print(clf.loss_curve_)
print(test_y[0:100])
print(test_y_hat[0:100])
```

```
MLPClassifier(hidden_layer_sizes=(10,))
[0.3273636147565900４, 0.0873700725640207, 0.04807722614932833, 0.0374507214836376３6, 0.03273010166916536, 0.029507229833757413,
[6 6 6 6 6 8 6 6 8 6 6 6 6 8 6 8 6 6 8 6 6 8 6 6 6 8 8 8 8 6 6 6 8 8 8 8 8
 6 8 8 8 8 6 8 8 6 8 6 6 8 6 8 6 6 6 8 8 6 6 8 6 8 8 8 8 8 8 6 6 6 6 8 6 6
 6 8 8 8 6 8 8 8 6 6 6 6 8 8 6 8 8 8 8 6 8 6 6 8 8 8]
[6 6 6 6 6 8 6 6 8 6 6 6 6 8 6 8 6 6 8 6 6 8 6 6 6 8 8 8 8 6 6 6 8 8 8 8 8
 6 8 8 8 8 6 8 8 6 8 6 6 8 6 8 6 6 6 8 8 6 6 8 6 8 8 8 8 8 8 6 6 6 6 8 6 6
 6 8 8 8 6 8 8 8 6 6 6 6 8 8 6 8 8 8 8 6 8 6 6 8 8 8]
```

Deep Learning Practice 2

Two-class classifier

[7]

```
from sklearn.metrics import accuracy_score

accuracy_score(test_y, test_y_hat)
```

```
0.9953416149068323
```

Deep Learning Practice 2

Multi-class classifier

[8]

```
from sklearn.preprocessing import LabelBinarizer

encoder = LabelBinarizer()
train_y_onehot = encoder.fit_transform(train_labels)
test_y_onehot = encoder.fit_transform(test_labels)

print(train_labels[0:10])
print(train_y_onehot[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
[[0 0 0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]]
```

Deep Learning Practice 2

Multi-class classifier

[9]

```
clf2 = MLPClassifier(hidden_layer_sizes=(50,))

train_x = train_images_reshape_norm
train_y = train_y_onehot
test_x = test_images_reshape_norm
test_y = test_y_onehot
clf2.fit(train_x, train_y)
test_y_hat = clf2.predict(test_x)
```

[10]

```
test_y_argmax = np.argmax(test_y, axis=1)
test_y_hat_argmax = np.argmax(test_y_hat, axis=1)
print(test_y_argmax[0:10])
print(test_y_hat_argmax[0:10])

accuracy_score(test_y_argmax, test_y_hat_argmax)
```

```
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
0.9528
```

Deep Learning Practice 2

Multi-class classifier

[11]

```
clf3 = MLPClassifier(hidden_layer_sizes=(50, 100, 50))

train_x = train_images_reshape_norm
train_y = train_y_onehot
test_x = test_images_reshape_norm
test_y = test_y_onehot
clf3.fit(train_x, train_y)
test_y_hat = clf3.predict(test_x)

test_y_argmax = np.argmax(test_y, axis=1)
test_y_hat_argmax = np.argmax(test_y_hat, axis=1)

accuracy_score(test_y_argmax, test_y_hat_argmax)
```

```
0.9735
```

Deep Learning Practice 3

Import package & dataset, data reshaping & normalization, one-hot encoding

[1]

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.utils import to_categorical


data_mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = data_mnist.load_data()

train_images_reshape = train_images.reshape([train_images.shape[0], train_images.shape[1]*train_images.shape[2]])
test_images_reshape = test_images.reshape([test_images.shape[0], test_images.shape[1]*test_images.shape[2]])

train_images_reshape_norm = train_images_reshape/255
test_images_reshape_norm = test_images_reshape/255

train_y_onehot = to_categorical(train_labels)
test_y_onehot = to_categorical(test_labels)

print(train_labels[0:10])
print(train_y_onehot[0:10])
```

Deep Learning Practice 3

Import package & dataset, data reshaping & normalization, one-hot encoding

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
[5 0 4 1 9 2 1 3 1 4]
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

Deep Learning Practice 3

Multi-class classifier

[2]

```
train_x = train_images_reshape_norm
train_y = train_y_onehot
test_x = test_images_reshape_norm
test_y = test_y_onehot

input_shape = (train_x.shape[1], )

mlp_model = models.Sequential()
mlp_model.add(layers.Dense(units = 50, activation = 'relu', input_shape=input_shape))
mlp_model.add(layers.Dense(units = 100, activation = 'relu'))
mlp_model.add(layers.Dense(units = 50, activation = 'relu'))
mlp_model.add(layers.Dense(units = 10, activation = 'softmax'))

mlp_model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
mlp_model.summary()
```

Deep Learning Practice 3

Multi-class classifier

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                39250

 dense_1 (Dense)             (None, 100)               5100

 dense_2 (Dense)             (None, 50)                5050

 dense_3 (Dense)             (None, 10)                510

=================================================================
Total params: 49,910
Trainable params: 49,910
Non-trainable params: 0
_________________________________________________________________
```

Deep Learning Practice 3

Training (batch size = 50, epochs = 100)

```
[3]  history = mlp_model.fit(train_x, train_y, validation_data = [test_x, test_y], batch_size=50, epochs=100)
```

```
Epoch 1/100
1200/1200 [==============================] - 6s 3ms/step - loss: 0.3027 - accuracy: 0.9134 - val_loss: 0.1627 - val_accuracy: 0.9513
Epoch 2/100
1200/1200 [==============================] - 4s 3ms/step - loss: 0.1372 - accuracy: 0.9588 - val_loss: 0.1138 - val_accuracy: 0.9667
Epoch 3/100
1200/1200 [==============================] - 7s 6ms/step - loss: 0.1003 - accuracy: 0.9699 - val_loss: 0.1019 - val_accuracy: 0.9695
Epoch 4/100
1200/1200 [==============================] - 7s 6ms/step - loss: 0.0806 - accuracy: 0.9756 - val_loss: 0.0958 - val_accuracy: 0.9710
Epoch 5/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0690 - accuracy: 0.9778 - val_loss: 0.0947 - val_accuracy: 0.9696
Epoch 6/100
1200/1200 [==============================] - 4s 3ms/step - loss: 0.0569 - accuracy: 0.9821 - val_loss: 0.0959 - val_accuracy: 0.9714
Epoch 7/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0508 - accuracy: 0.9838 - val_loss: 0.0938 - val_accuracy: 0.9715
Epoch 8/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0445 - accuracy: 0.9854 - val_loss: 0.0998 - val_accuracy: 0.9699
Epoch 9/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0408 - accuracy: 0.9866 - val_loss: 0.1208 - val_accuracy: 0.9674
Epoch 10/100
1200/1200 [==============================] - 4s 3ms/step - loss: 0.0343 - accuracy: 0.9885 - val_loss: 0.0934 - val_accuracy: 0.9748
Epoch 11/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0300 - accuracy: 0.9903 - val_loss: 0.0957 - val_accuracy: 0.9749
Epoch 12/100
1200/1200 [==============================] - 3s 3ms/step - loss: 0.0301 - accuracy: 0.9903 - val_loss: 0.1222 - val_accuracy: 0.9708
```

Deep Learning Practice 3

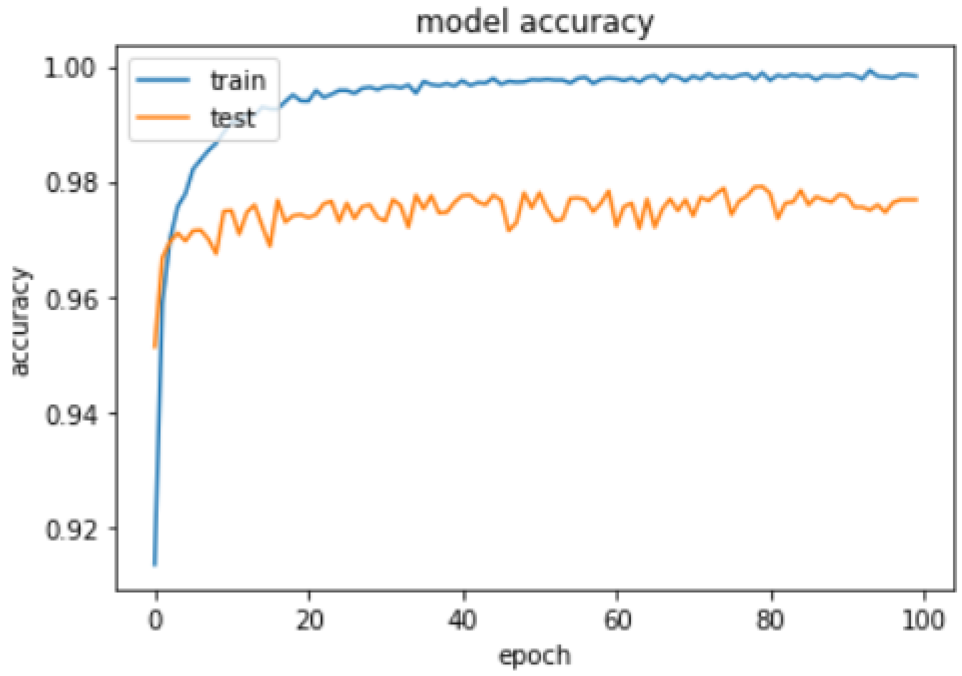
Training (batch size = 50, epochs = 100)

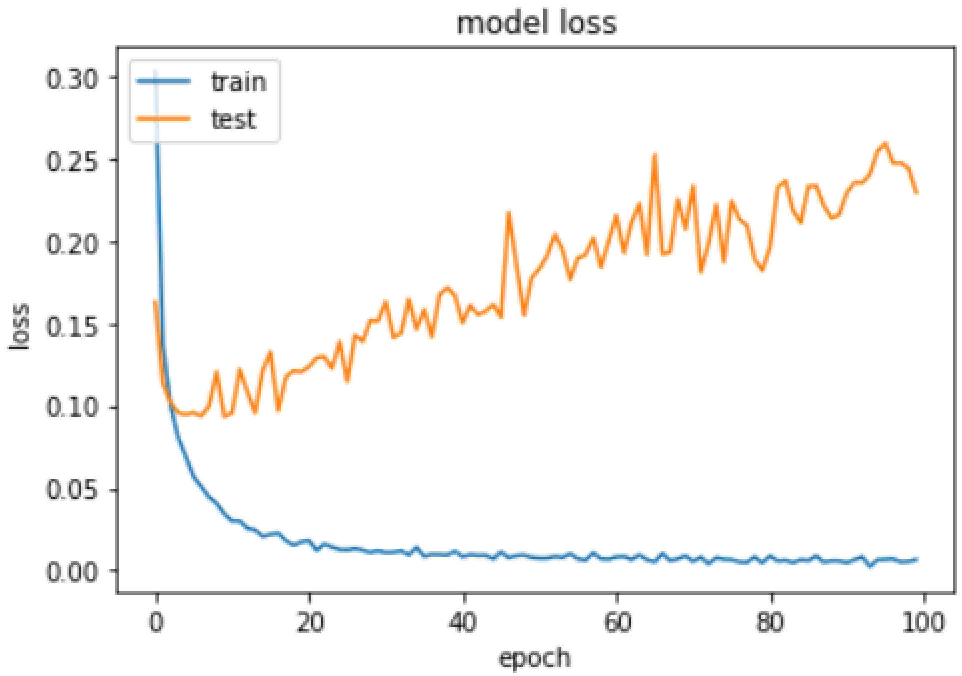
[4]

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

print(history.history['val_accuracy'])
print(np.max(history.history['val_accuracy']))
```

Deep Learning Practice 3

Training (batch size = 50, epochs = 100)





[0.9513000249862671, 0.96670001745224, 0.9695000052452087, 0.9710000157356262, 0.9696000218391418, 0.9714000225067139,
0.9790999889373779