AI VIETNAM
All-in-One Course

Multilayer Perceptron

Nguyen Quoc Thai

Year 2023

CONTENT

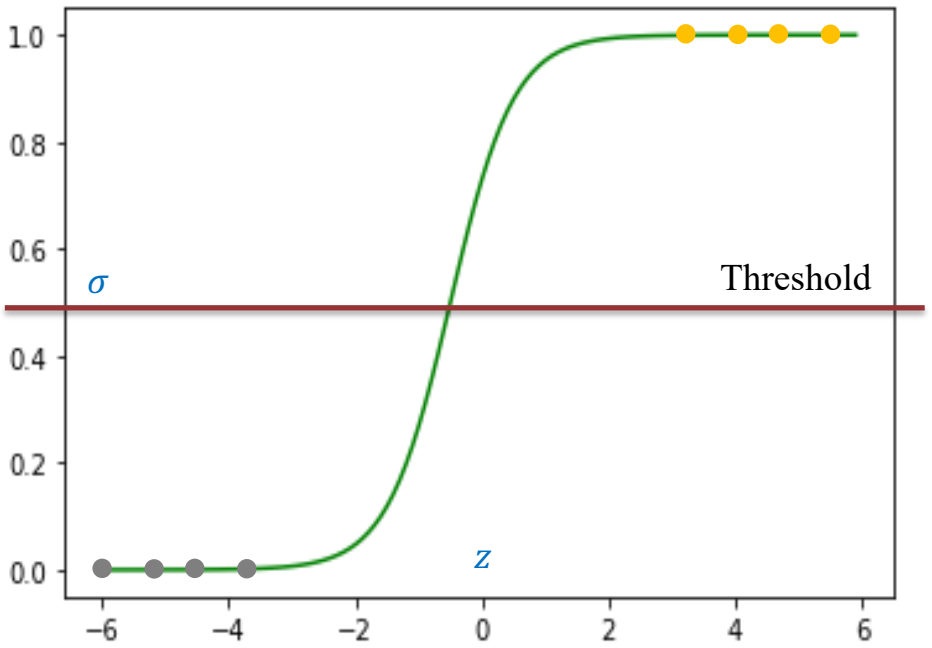
1 - Background

Sigmoid Function

Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$z \in (-\infty \quad +\infty)$ $\sigma(z) \in (0 \quad 1)$



Hours	Pass
0.5	0
1.0	0
1.5	1
2.0	1

Classes: {0, 1}
Binary Classification

Hours	Score
0.5	0
1.0	0
1.5	1
2.0	1
2.5	2
3.0	2
3.5	3
4.0	3

Classes: {0, 1, 2, 3}
Multi-class Classification

1 - Background

Softmax Function

$$P_i = f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$0 \leq f(z_i) \leq 1$$

$$\sum_i f(z_i) = 1$$

Input		Probability
$z_1 = 1.0$	→ Softmax →	$f(z_1) = 0.12$
$z_2 = 3.0$		$f(z_2) = 0.88$

Input		Probability
$z_1 = 1.0$	→ Softmax →	$f(z_1) = 0.09$
$z_2 = 2.0$		$f(z_2) = 0.24$
$z_3 = 3.0$		$f(z_3) = 0.67$

1 - Background

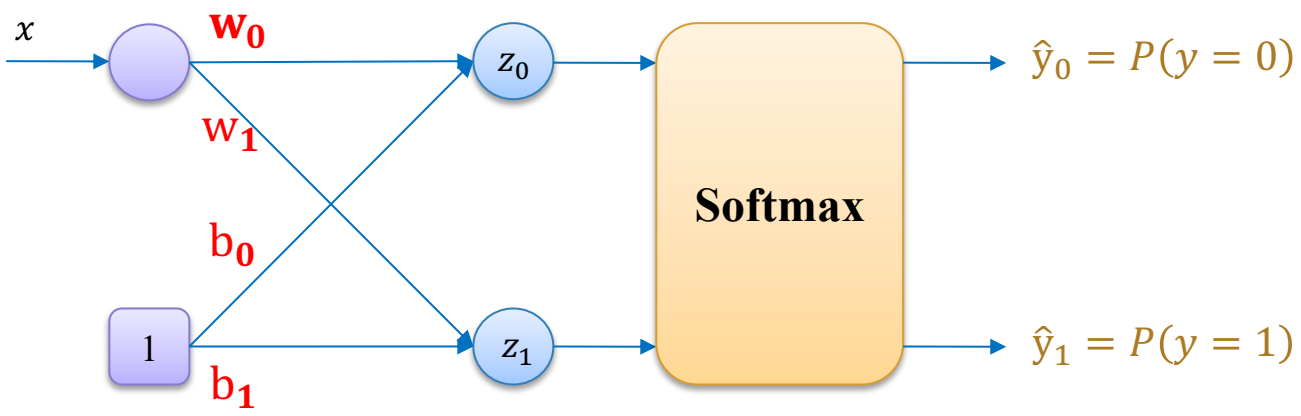
Softmax Regression

Classes: {0, 1}
Binary Classification

Hours	Pass
0.5	0
2.0	1

#feature: 1
#class: 2

Softmax Regression



1 - Background

Softmax Regression

One-hot Encoding

$$\boldsymbol{y} = \begin{bmatrix} y_0 \\ \cdots \\ y_C \end{bmatrix} \qquad y_i \in \{0,1\} \qquad \sum_i y_i = 1 \qquad \text{C = \#classes}$$

Classes: {0, 1}
Binary Classification

Hours	Pass
0.5	0
2.0	1

#feature: 1
#class: 2

$$y = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y = 1 \rightarrow \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Classes: {0, 1, 2}
Multi-class Classification

Hours	Score
0.5	0
1.5	1
3.0	2

#feature: 1
#class: 3

$$y = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$y = 1 \rightarrow \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$y = 2 \rightarrow \boldsymbol{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

1 - Background

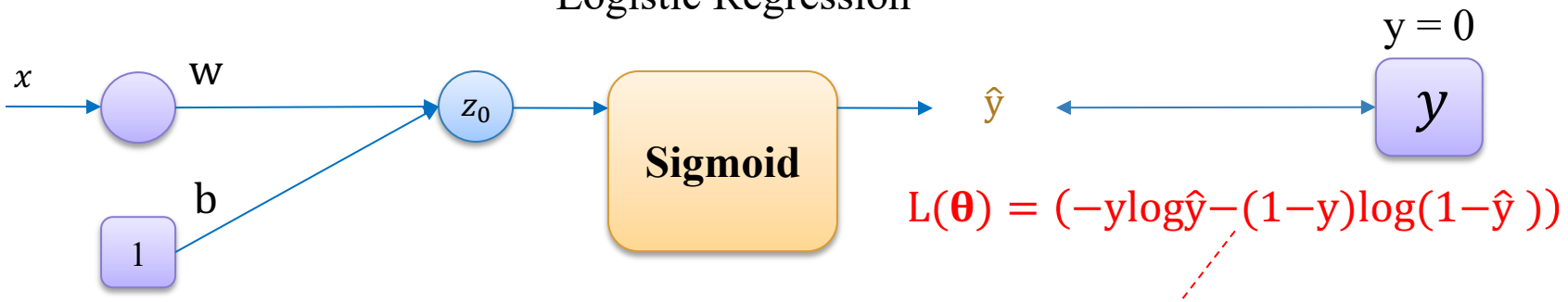
Softmax Regression

Classes: {0, 1}
Binary Classification

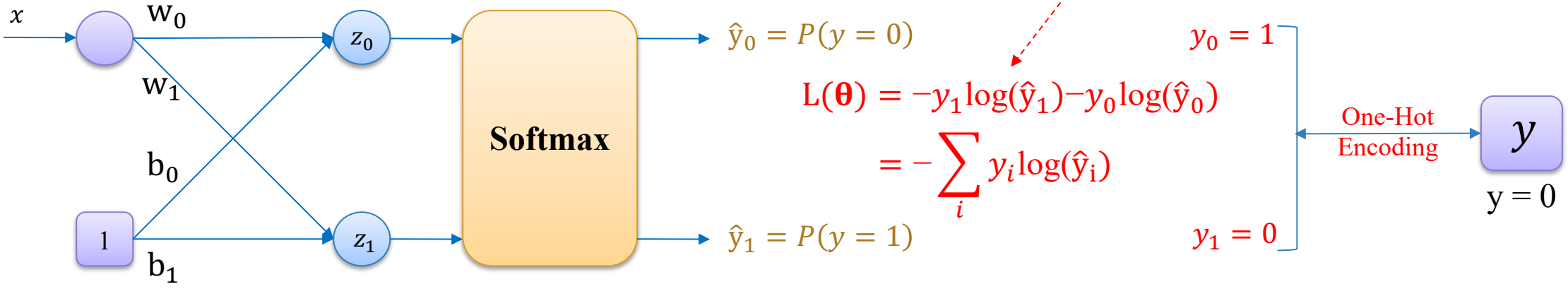
Hours	Pass
0.5	0
2.0	1

#feature: 1
#class: 2

Logistic Regression



Softmax Regression



1 - Background

Softmax Regression

1) Pick a sample from training data

2) Compute output $\hat{y}$

$$\boldsymbol{z} = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\boldsymbol{d} = [1 \ \ldots 1] e^{\boldsymbol{z}}$$

$$\hat{\boldsymbol{y}} = e^{\boldsymbol{z}} \emptyset \boldsymbol{d}$$

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

$$L(\boldsymbol{\theta}) = -\boldsymbol{y}^T log \hat{\boldsymbol{y}}$$

4) Compute derivative

$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta \mathrm{L}$$

η is learning rate

Data #1

Hours	Pass
0.5	0
1.0	0
1.5	1
2.0	1

$$\boldsymbol{x}^T = [1 \quad 0.5]$$

$$\boldsymbol{y} = [0]$$

One-hot encoding for label

$$y = 0 \rightarrow \boldsymbol{y}^T = [1 \quad 0]$$

$$y = 1 \rightarrow \boldsymbol{y}^T = [0 \quad 1]$$

$$\theta = \begin{bmatrix} b_0 & b_1 \\ w_0 & w_1 \end{bmatrix} \qquad \theta = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$

$$\eta = 0.1$$

1 - Background

Softmax Regression

1) Pick a sample from training data

2) Compute output $\hat{y}$

$$\boldsymbol{z} = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\boldsymbol{d} = [1 \ \ldots 1] e^{\boldsymbol{z}}$$

$$\hat{\boldsymbol{y}} = e^{\boldsymbol{z}} \emptyset \boldsymbol{d}$$

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

$$L(\boldsymbol{\theta}) = -\boldsymbol{y}^T log \hat{\boldsymbol{y}}$$

4) Compute derivative

$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

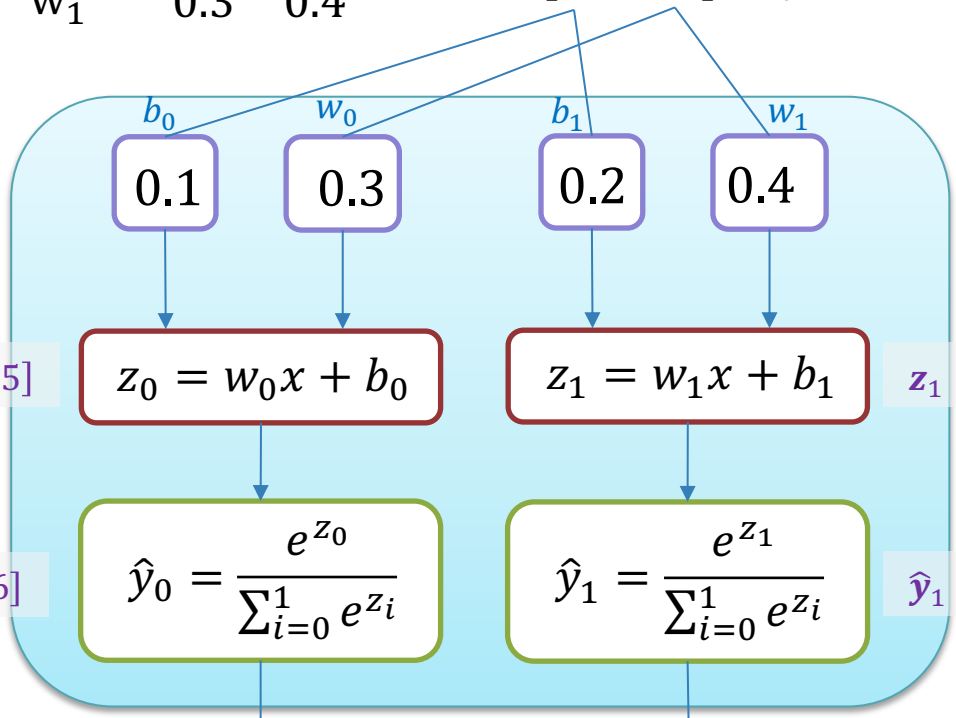
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta \mathrm{L}$$

η is learning rate

$$\eta = 0.1 \quad \theta = \begin{bmatrix} b_0 & b_1 \\ w_0 & w_1 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} \quad \boldsymbol{x}^T = [1 \quad 0.5] \quad y = 0 \rightarrow \boldsymbol{y}^T = [1 \quad 0]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

$$= \begin{bmatrix} 0.105 & 0.194 \\ 0.302 & 0.397 \end{bmatrix}$$



$$\nabla_\theta \mathrm{L} = \mathbf{x}(\hat{y} - y)^{\mathrm{T}}$$

$$= \begin{bmatrix} -0.54 & 0.54 \\ -0.27 & 0.27 \end{bmatrix}$$

1 - Background

Softmax Regression

1) Pick a sample from training data

2) Compute output $\hat{y}$

$$\boldsymbol{z} = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\boldsymbol{d} = [1 \ \ldots 1] e^{\boldsymbol{z}}$$

$$\hat{\boldsymbol{y}} = e^{\boldsymbol{z}} \emptyset \boldsymbol{d}$$

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

$$L(\boldsymbol{\theta}) = -\boldsymbol{y}^T log \hat{\boldsymbol{y}}$$

4) Compute derivative

$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

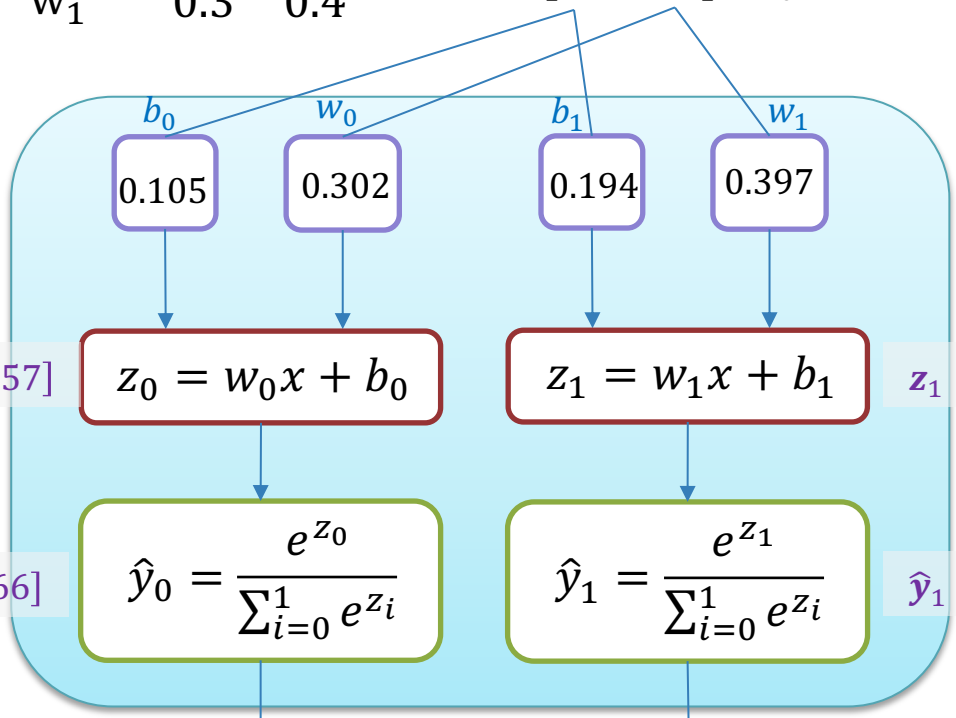
$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta \mathrm{L}$$

η is learning rate

$$\eta = 0.1 \quad \theta = \begin{bmatrix} b_0 & b_1 \\ w_0 & w_1 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} \quad \boldsymbol{x}^T = [1 \quad 0.5] \quad y = 0 \rightarrow \boldsymbol{y}^T = [1 \quad 0]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta L'_{\boldsymbol{\theta}}$$

$$= \begin{bmatrix} 0.105 & 0.194 \\ 0.302 & 0.397 \end{bmatrix}$$

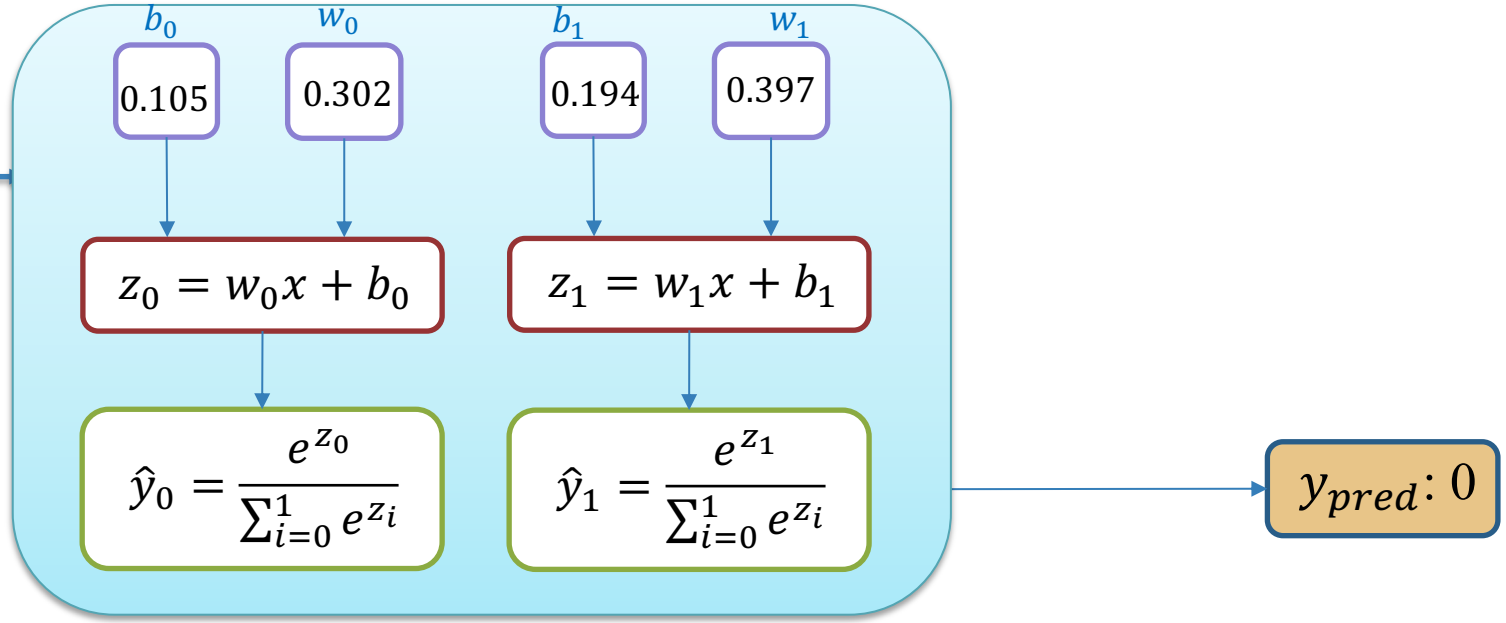


1 - Background

Softmax Regression

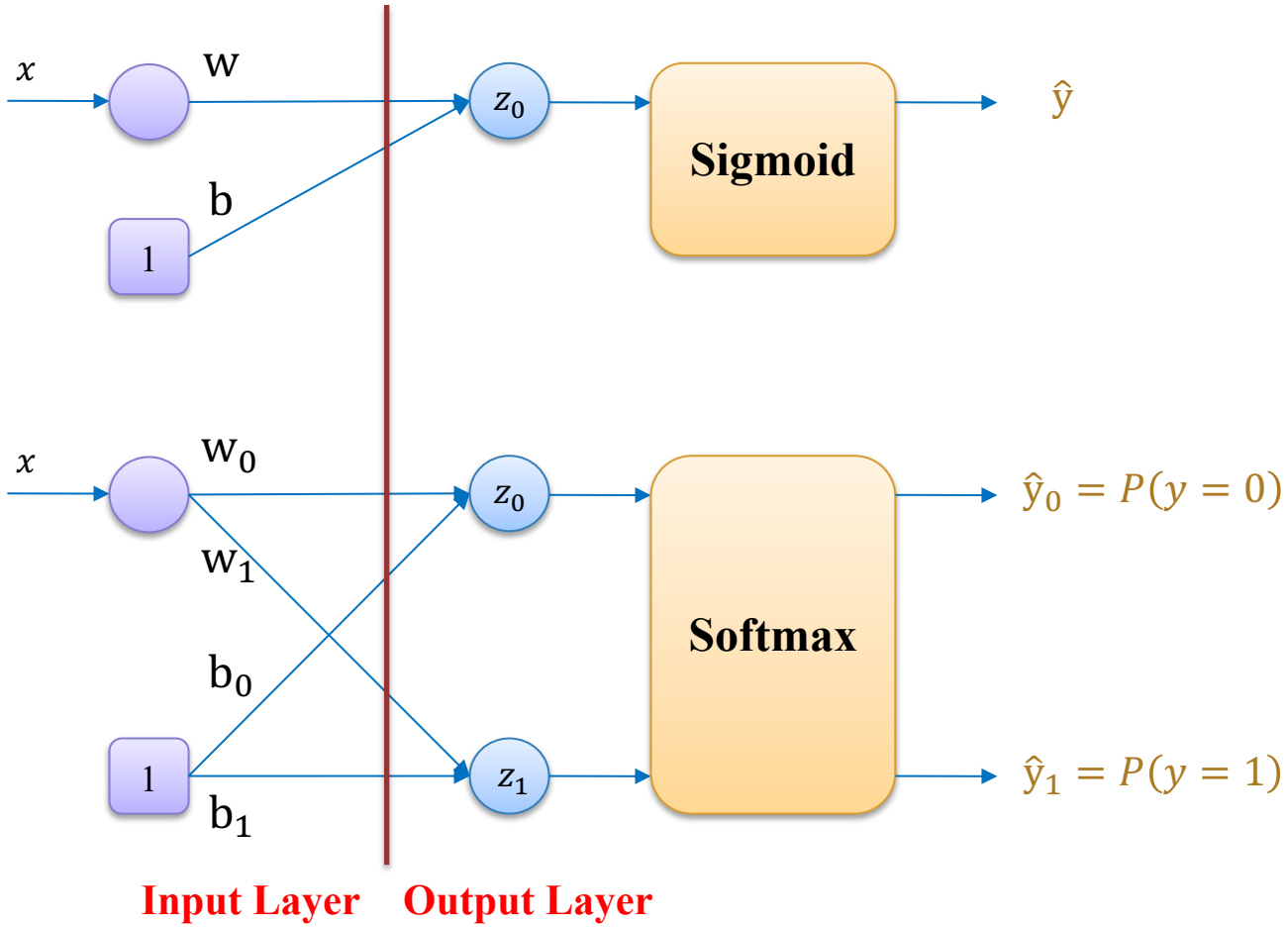
Hours	Pass
0.25	???
4.5	???

Prediction



2 – Multilayer Perceptron

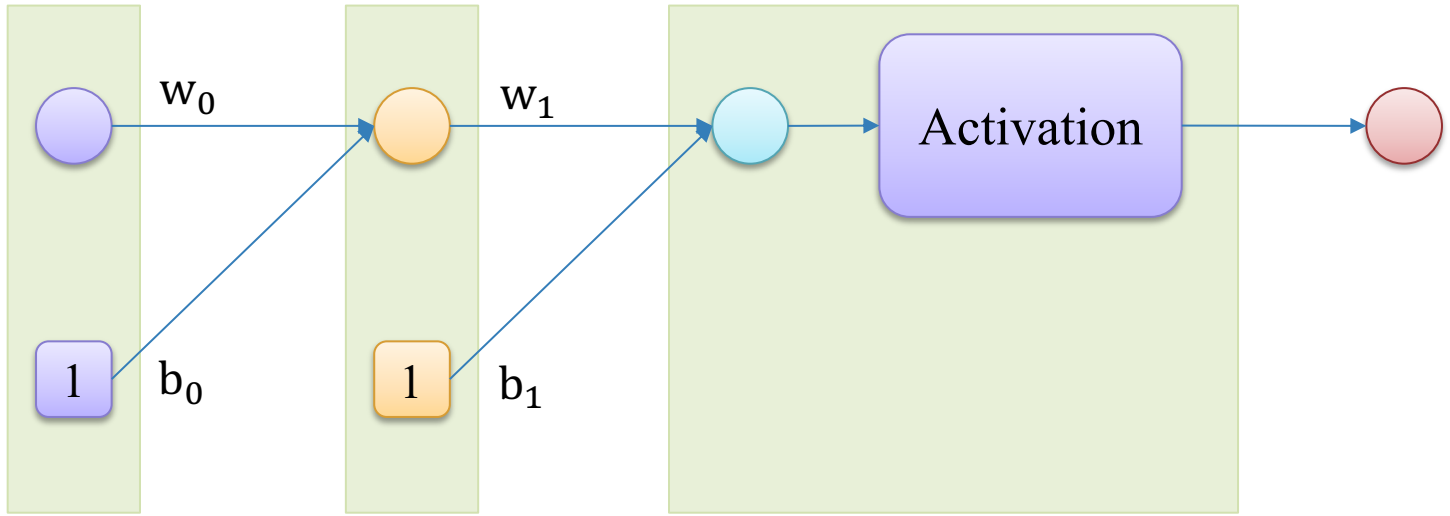
Motivation



2 – Multilayer Perceptron

Multilayer Perceptron

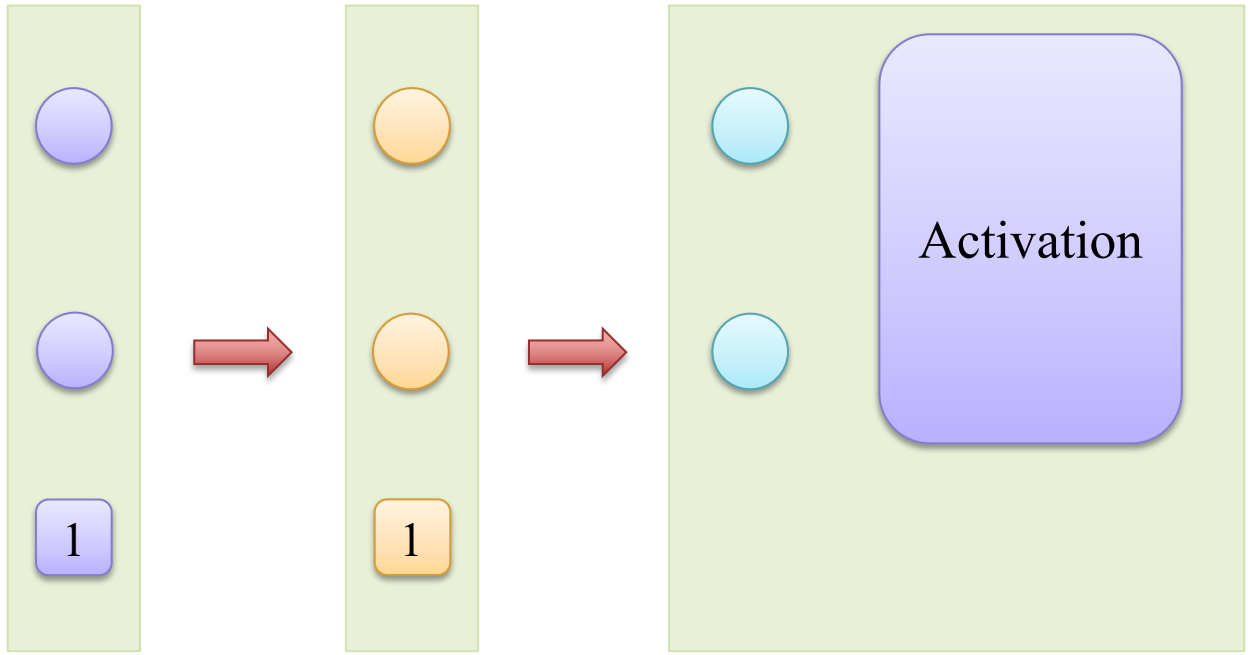
#parameters: 4



2 – Multilayer Perceptron

Multilayer Perceptron

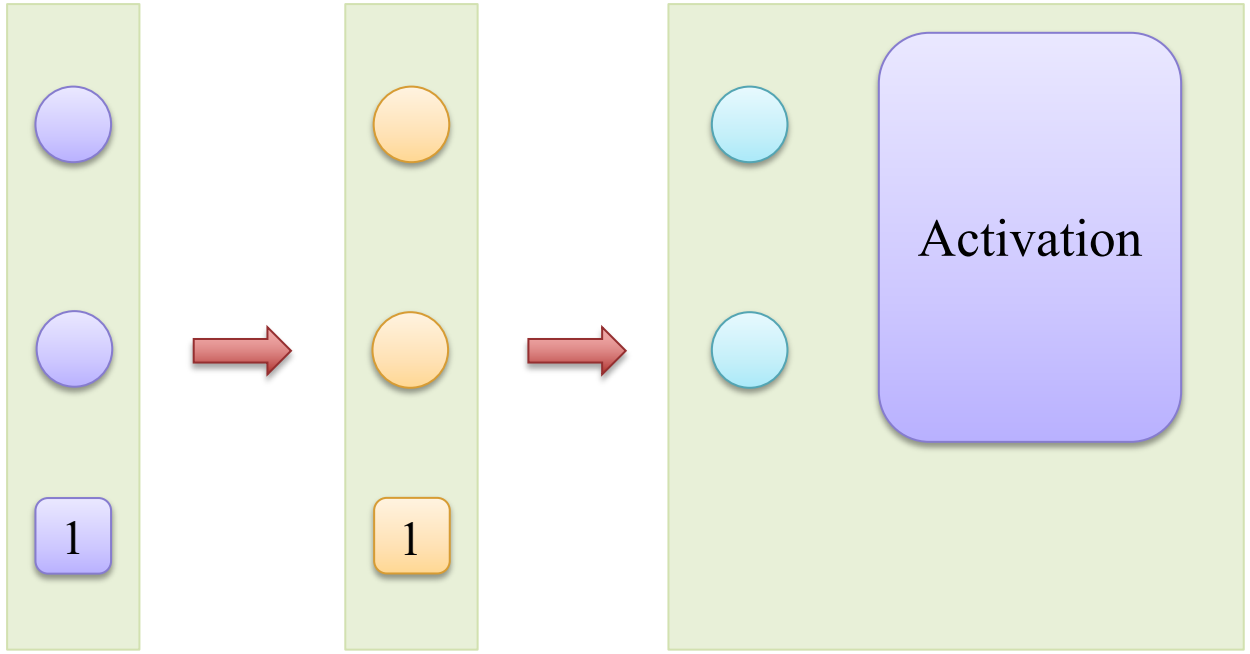
#parameters: 12



2 – Multilayer Perceptron

Multilayer Perceptron

#parameters: 12



Input Layer **Hidden Layer** **Output Layer**

```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.Linear(2, 2),
    nn.Sigmoid()
)
```

```
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 2]               6
            Linear-2                [-1, 1, 2]               6
           Sigmoid-3                [-1, 1, 2]               0
================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

2 – Multilayer Perceptron

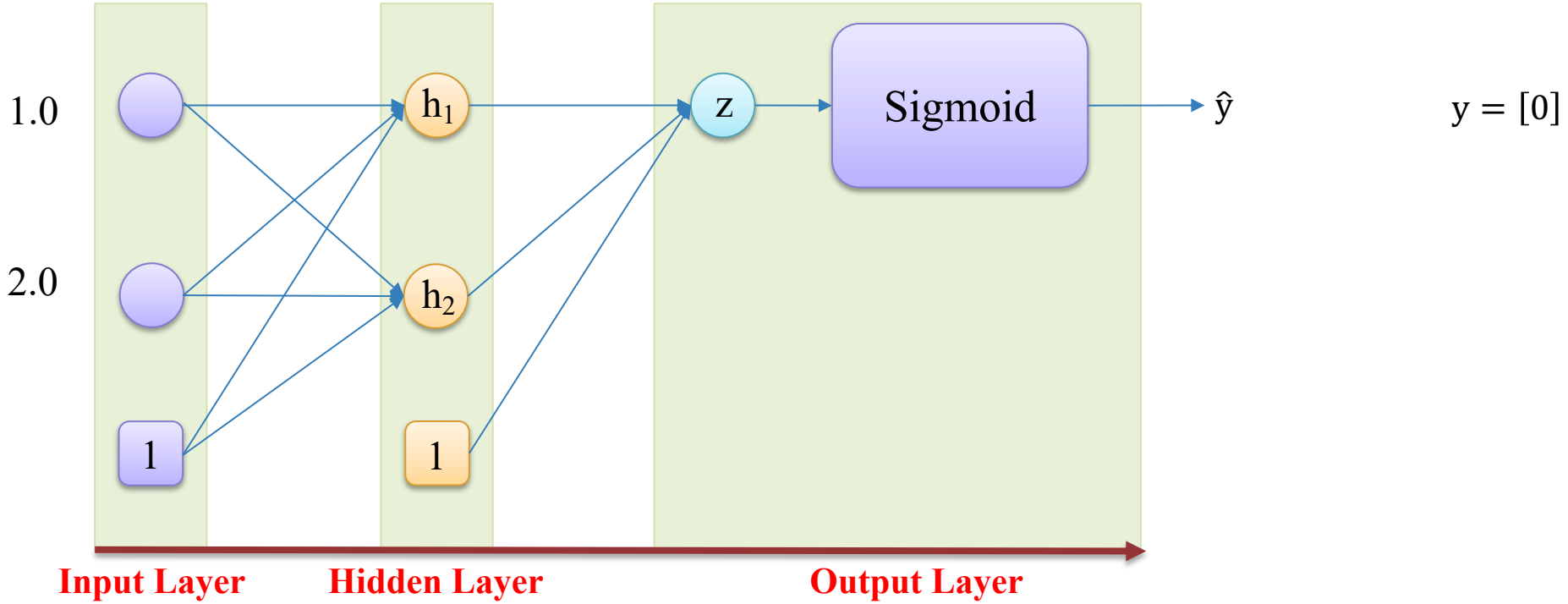
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$y = [0]$

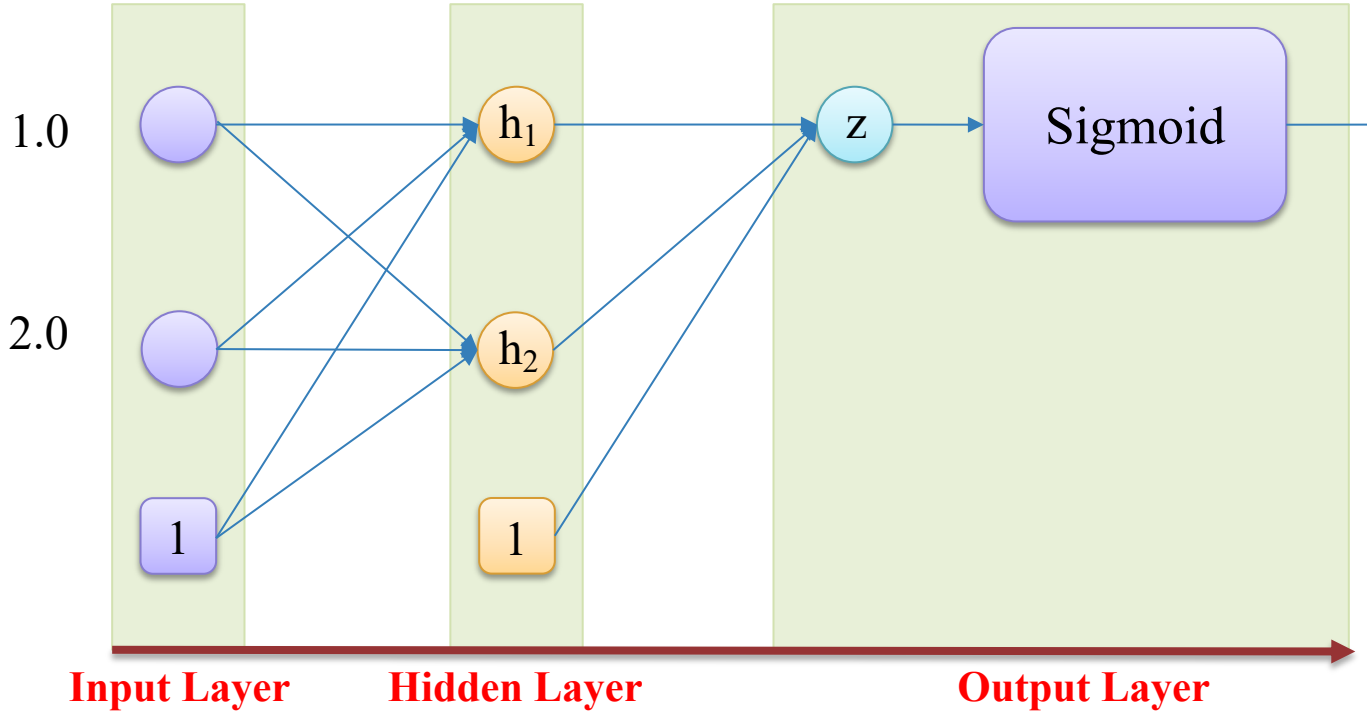
2 – Multilayer Perceptron

Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}] = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$y = [0]$

$h = [1.0 \; x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$

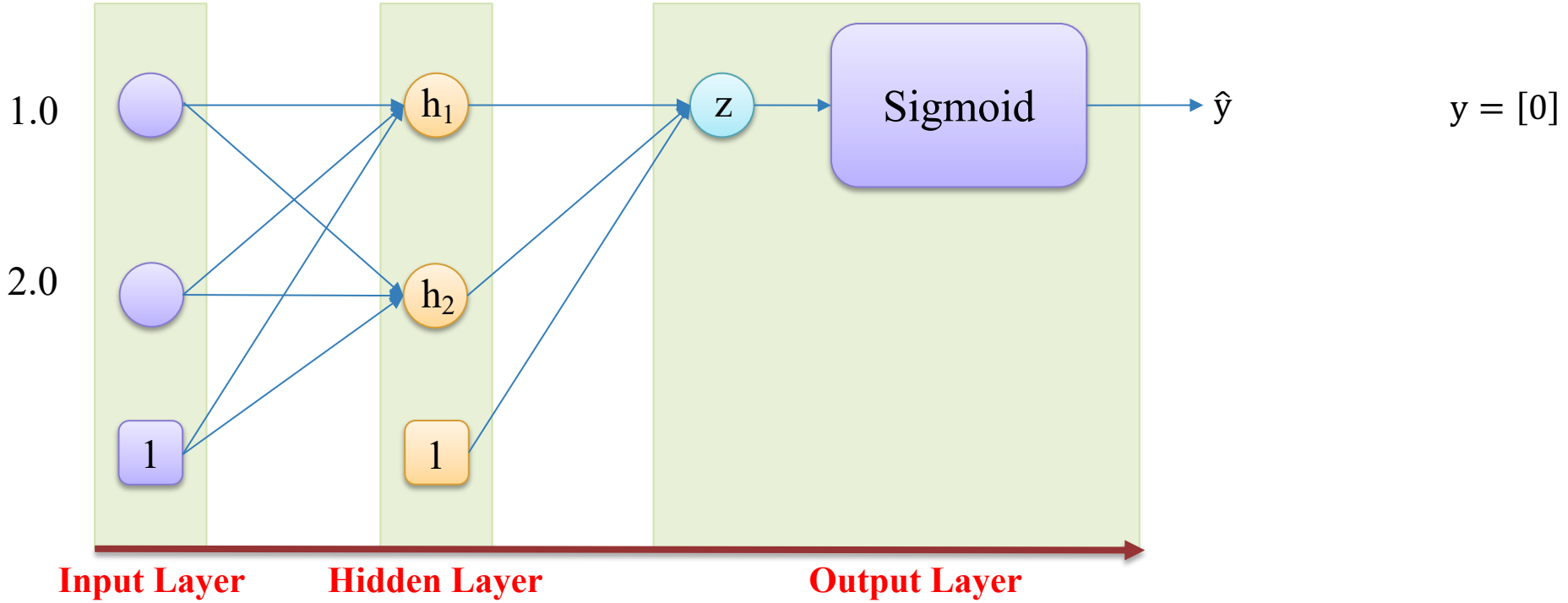
2 – Multilayer Perceptron

Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}] = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$y = [0]$

$h = [1.0 \; x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$

$z = [1.0 \; h]W_z = [1.0 \quad 0.4 \quad 0.4] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$

2 – Multilayer Perceptron

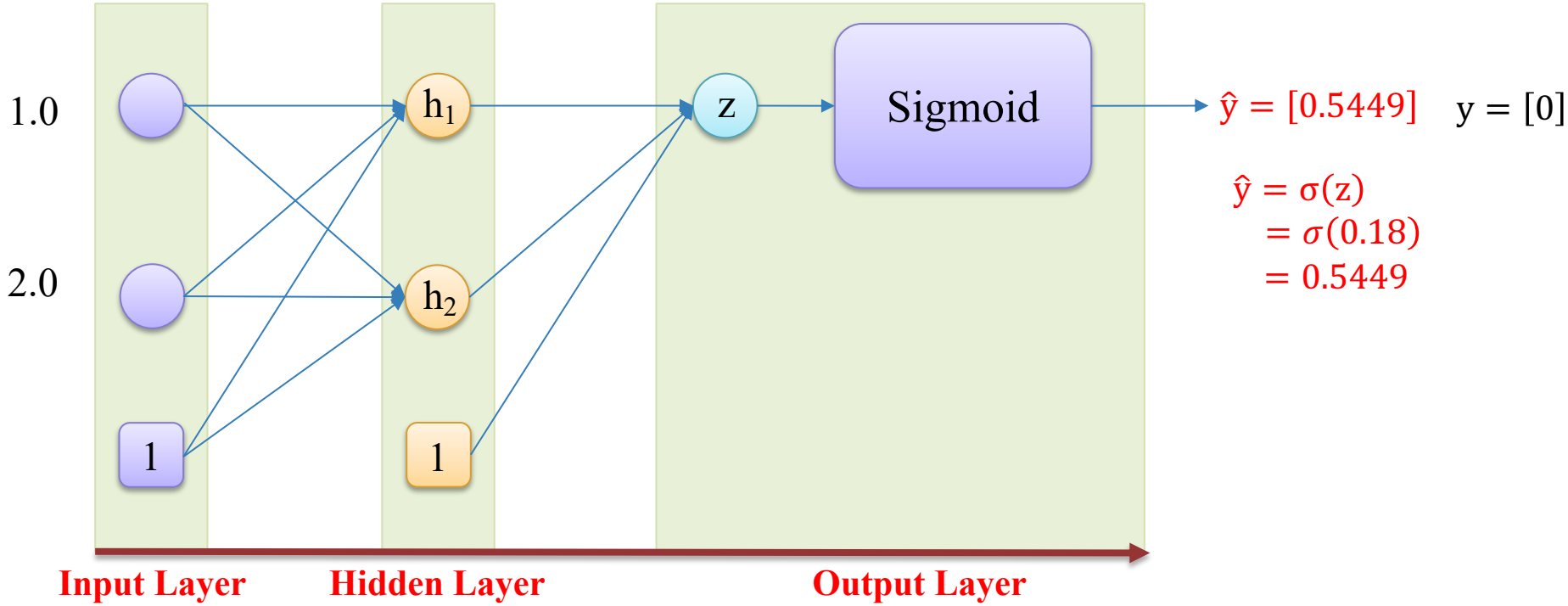
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$\hat{y} = [0.5449]$ $y = [0]$

$\hat{y} = \sigma(z)$
$= \sigma(0.18)$
$= 0.5449$

$h = [1.0 \; x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$

$z = [1.0 \; h]W_z = [1.0 \quad 0.4 \quad 0.4] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$

2 – Multilayer Perceptron

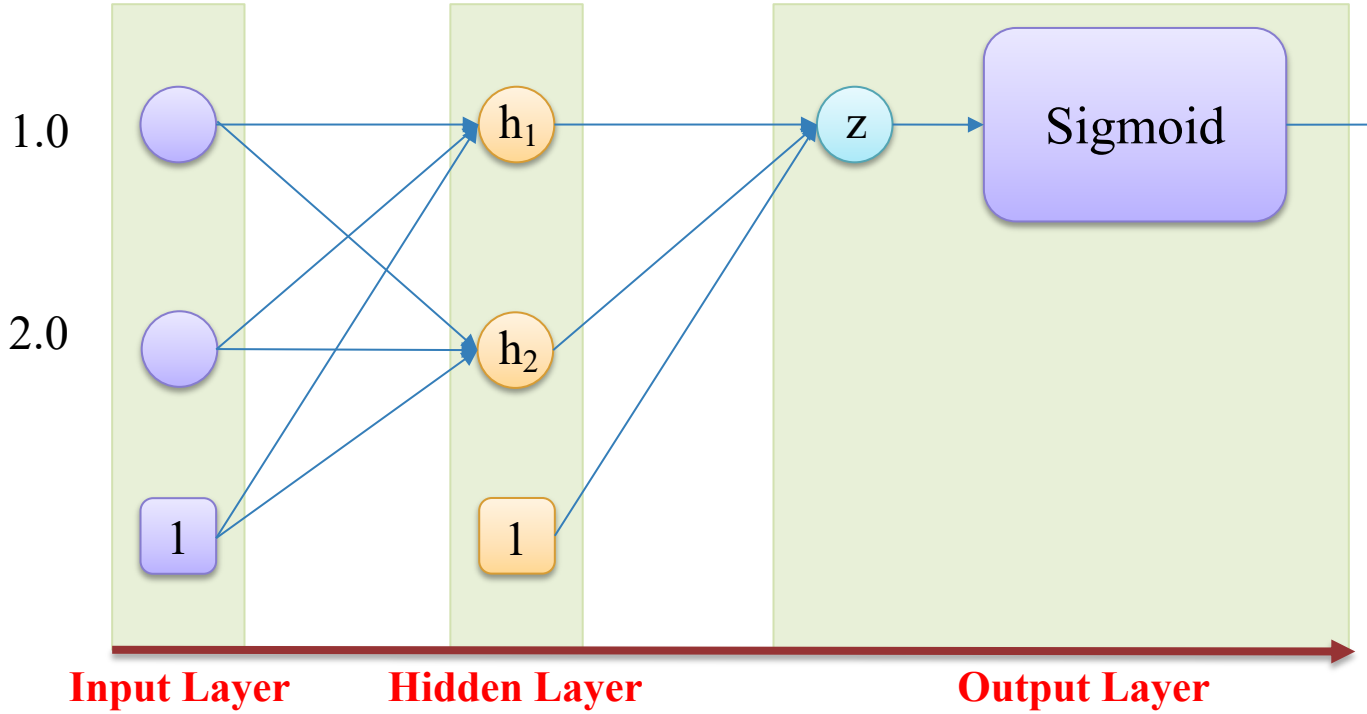
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$\hat{y} = [0.5449] \quad y = [0]$

$\hat{y} = \sigma(z)$
$= \sigma(0.18)$
$= 0.5449$

$L = 0.7872$

$h = [1.0 \; x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$

$z = [1.0 \; h]W_z = [1.0 \quad 0.4 \quad 0.4] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$

2 – Multilayer Perceptron

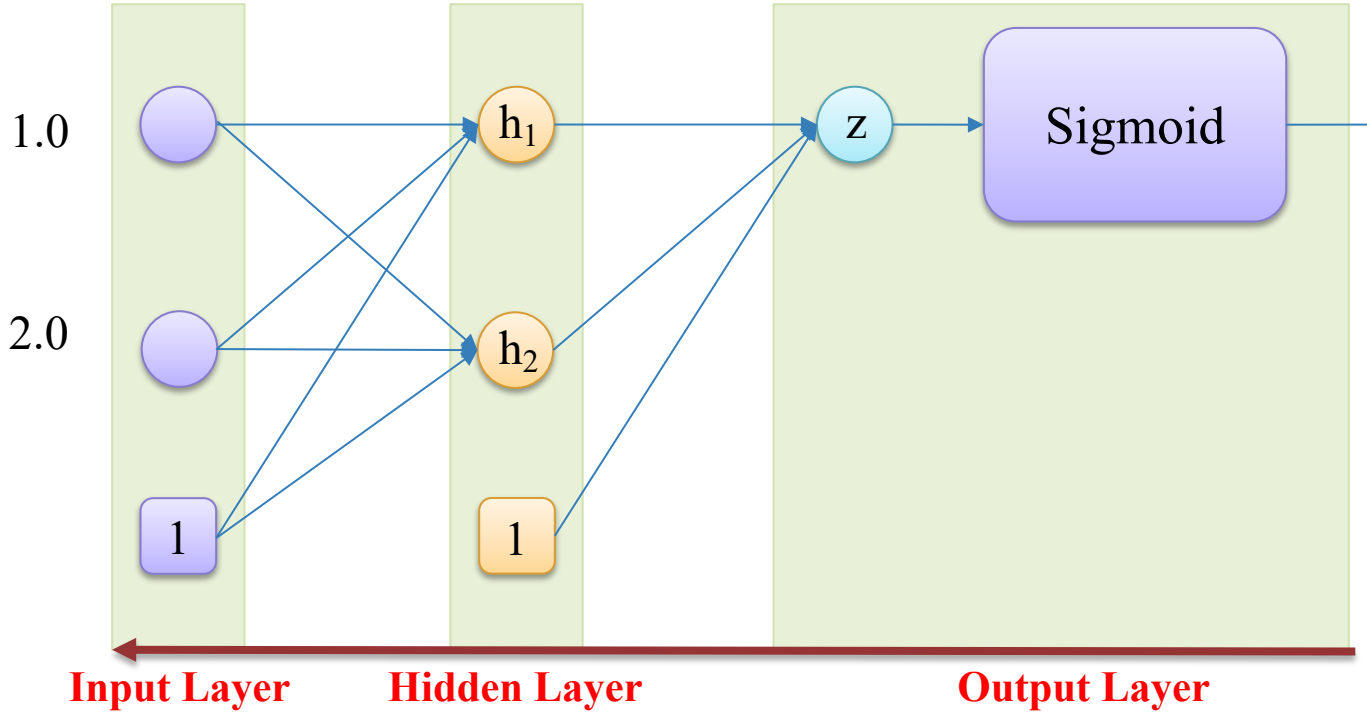
Backward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$= \begin{bmatrix} 0.0946 & 0.0946 \\ 0.0946 & 0.0891 \\ 0.0946 & 0.0891 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.0455 \\ 0.0782 \\ 0.0782 \end{bmatrix}$



$\hat{y} = [0.5449]$ $y = [0]$

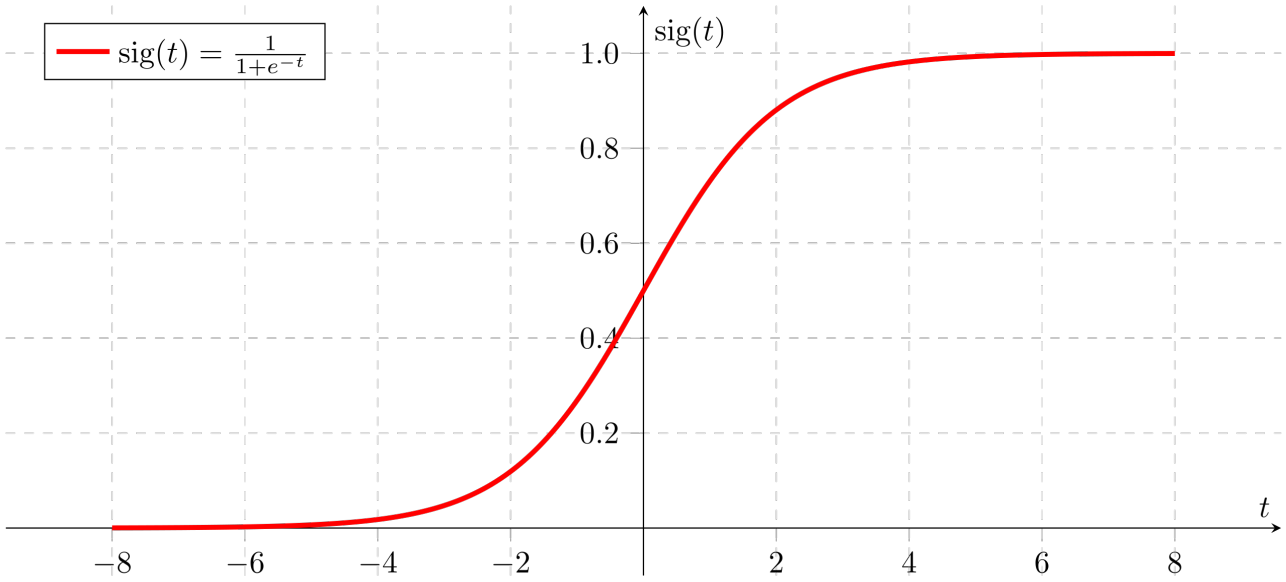
$\hat{y} = \sigma(z)$
$= \sigma(0.18)$
$= 0.5449$

$L = 0.7872$

2 – Multilayer Perceptron

Activation

❖ Sigmoid Function



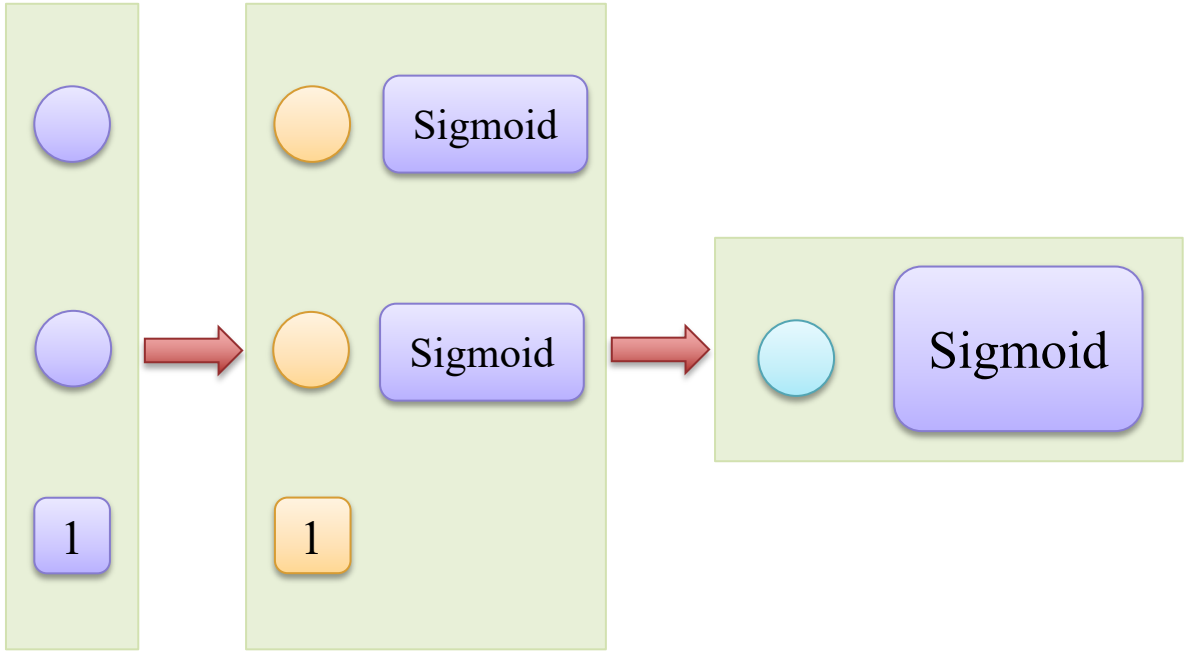

```
import torch.nn as nn

act = nn.Sigmoid()
input = torch.tensor([0.18, -0.18])
act(input)
```

```
tensor([0.5449, 0.4551])
```

2 – Multilayer Perceptron

Activation

❖ Sigmoid Function



Input Layer Hidden Layer Output Layer

```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.Sigmoid(),
    nn.Linear(2, 1),
    nn.Sigmoid()
)
```

```
summary(model, (1, 2))
```

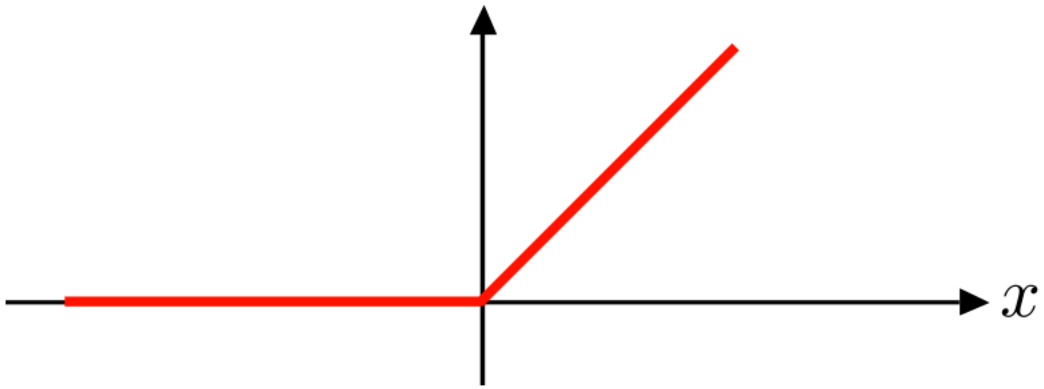
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 2]               6
           Sigmoid-2               [-1, 1, 2]               0
            Linear-3               [-1, 1, 1]               3
           Sigmoid-4               [-1, 1, 1]               0
================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

2 – Multilayer Perceptron

Activation

❖ ReLU Function

$$\text{ReLU}(x) \triangleq \max(0, x)$$



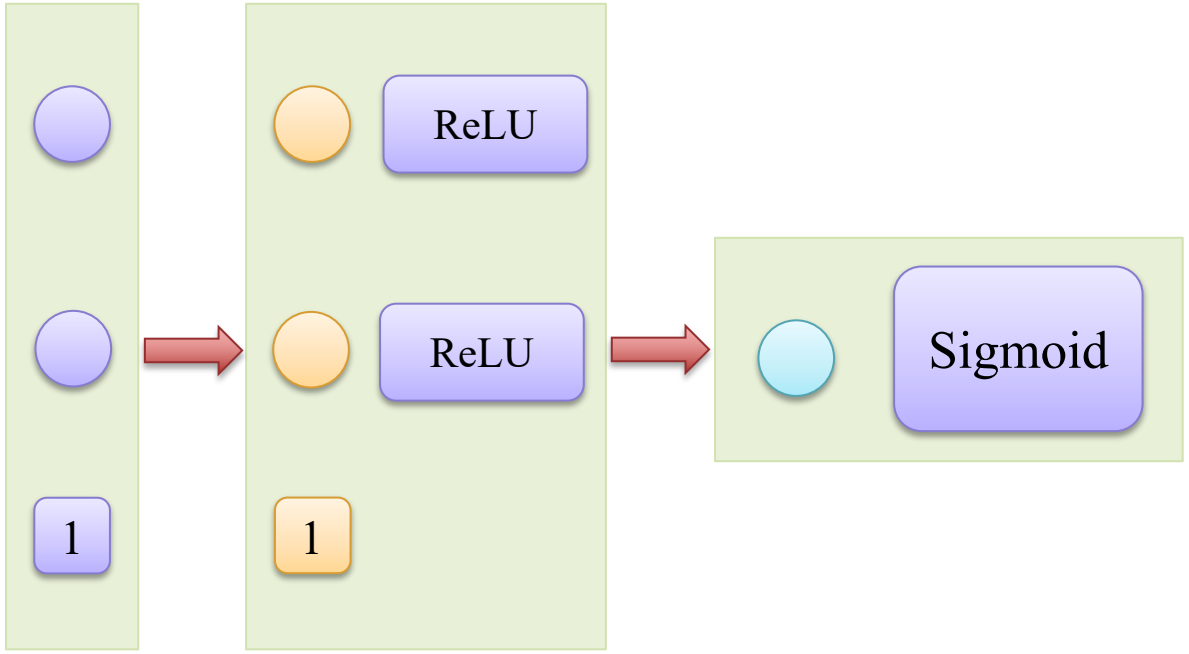

```
import torch.nn as nn

act = nn.ReLU()
input = torch.tensor([0.18, -0.18])
act(input)
```

```
tensor([0.1800, 0.0000])
```

2 – Multilayer Perceptron

Activation

❖ ReLU Function



Input Layer Hidden Layer Output Layer

```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.ReLU(),
    nn.Linear(2, 1),
    nn.Sigmoid()
)
```

```
summary(model, (1, 2))
```

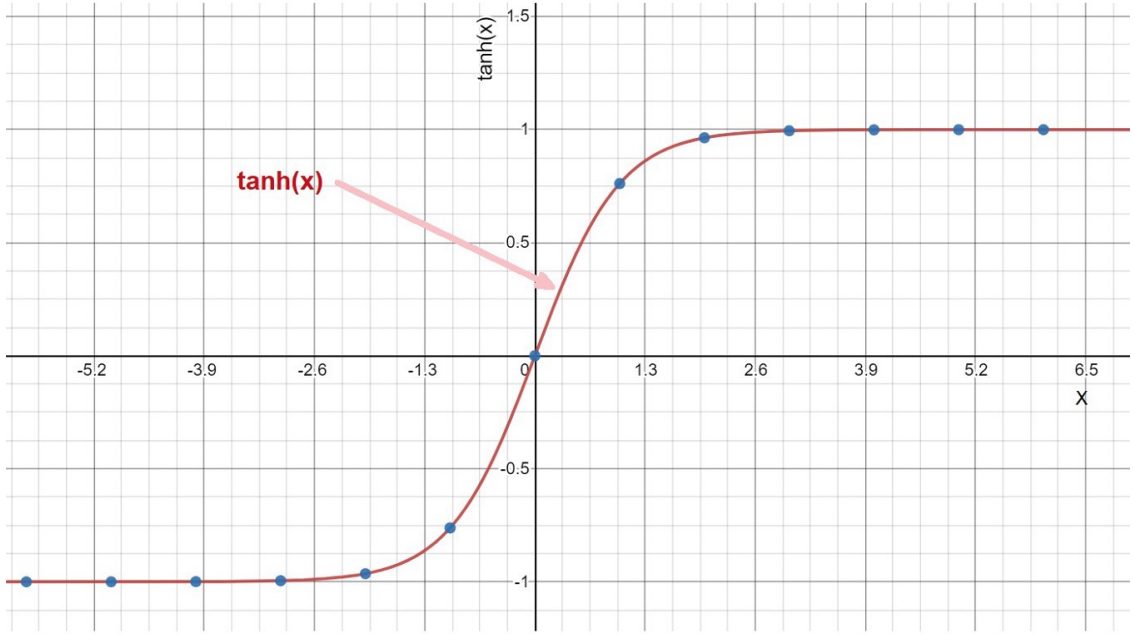
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 2]               6
              ReLU-2                [-1, 1, 2]               0
            Linear-3                [-1, 1, 1]               3
           Sigmoid-4                [-1, 1, 1]               0
================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

2 – Multilayer Perceptron

Activation

❖ Tanh Function

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



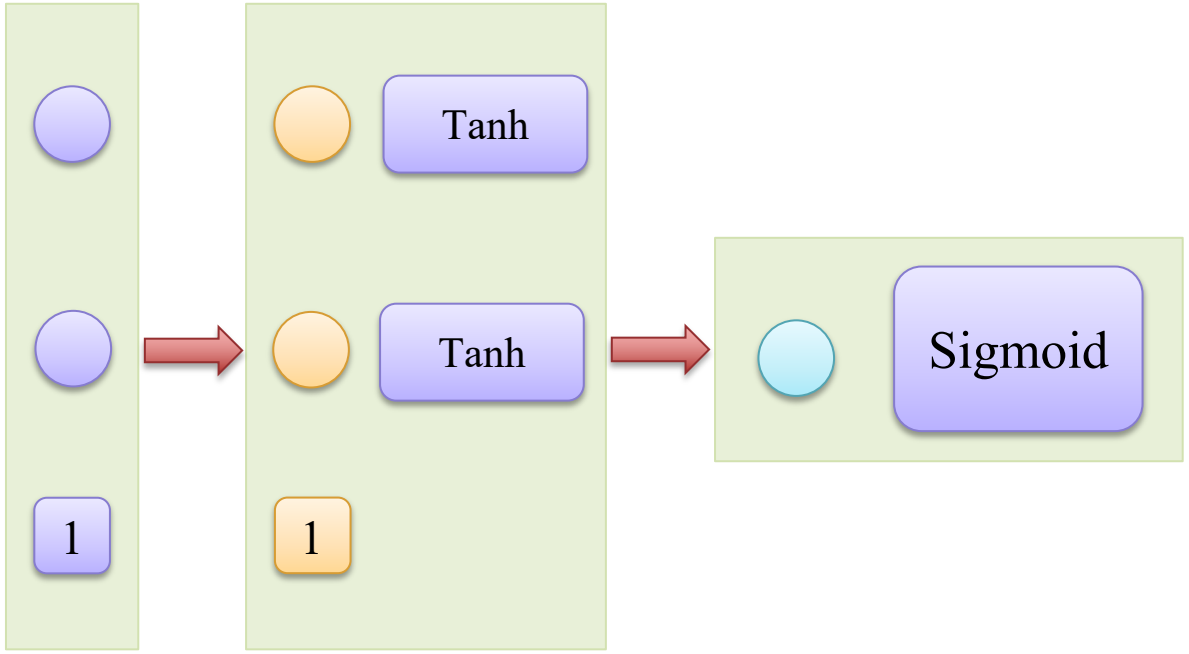

```
import torch.nn as nn

act = nn.Tanh()
input = torch.tensor([0.18, -0.18])
act(input)
```

```
tensor([ 0.1781, -0.1781])
```

2 – Multilayer Perceptron

Activation

❖ Tanh Function




```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.Tanh(),
    nn.Linear(2, 1),
    nn.Sigmoid()
)
```

```
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 2]               6
              Tanh-2                [-1, 1, 2]               0
            Linear-3                [-1, 1, 1]               3
           Sigmoid-4                [-1, 1, 1]               0
================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

2 – Multilayer Perceptron

Loss

❖ BCELoss()

```
import torch.nn as nn
loss_fn = nn.BCELoss()
```

```
y_pred
```

```
tensor([0.5449], grad_fn=<SigmoidBackward0>)
```

```
y
```

```
tensor([0.])
```

```
loss = loss_fn(y_pred, y)
loss
```

```
tensor(0.7872, grad_fn=<BinaryCrossEntropyBackward0>)
```

```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.Linear(2, 1),
    nn.Sigmoid()
)
```

```
print(model)
```

```
Sequential(
  (0): Linear(in_features=2, out_features=2, bias=True)
  (1): Linear(in_features=2, out_features=1, bias=True)
  (2): Sigmoid()
)
```

```
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 2]               6
            Linear-2               [-1, 1, 1]               3
           Sigmoid-3               [-1, 1, 1]               0
================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
----------------------------------------------------------------
```

2 – Multilayer Perceptron

Loss

❖CrossEntropyLoss()

```
import torch.nn as nn
loss_fn = nn.CrossEntropyLoss()
```

```
y = torch.tensor(0)
y
```

```
tensor(0)
```

```
y_pred
```

```
tensor([0.0580, 0.4275], grad_fn=<AddBackward0>)
```

```
loss_fn(y_pred, y)
```

With Softmax Funtion

```
tensor(0.8949, grad_fn=<NllLossBackward0>)
```

```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.ReLU(),
    nn.Linear(2, 2),
)
```

```
print(model)
```

```
Sequential(
  (0): Linear(in_features=2, out_features=2, bias=True)
  (1): ReLU()
  (2): Linear(in_features=2, out_features=2, bias=True)
)
```

```
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 2]               6
              ReLU-2                [-1, 1, 2]               0
            Linear-3                [-1, 1, 2]               6
================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
```

2 – Multilayer Perceptron

Optimizer

❖ **SGD()**

```
for layer in model.children():
    print(layer.state_dict())
```

```
OrderedDict([('weight', tensor([[0.1000, 0.1000],
        [0.1000, 0.1000]])), ('bias', tensor([0.1000, 0.1000]))])
OrderedDict()
OrderedDict([('weight', tensor([[0.1000, 0.1000],
        [0.1000, 0.1000]])), ('bias', tensor([0.1000, 0.1000]))])
```

```
learning_rate = 0.1
optimizer = optim.SGD(model.parameters(), learning_rate)
```

```
loss.backward()
```

```
optimizer.step()
```

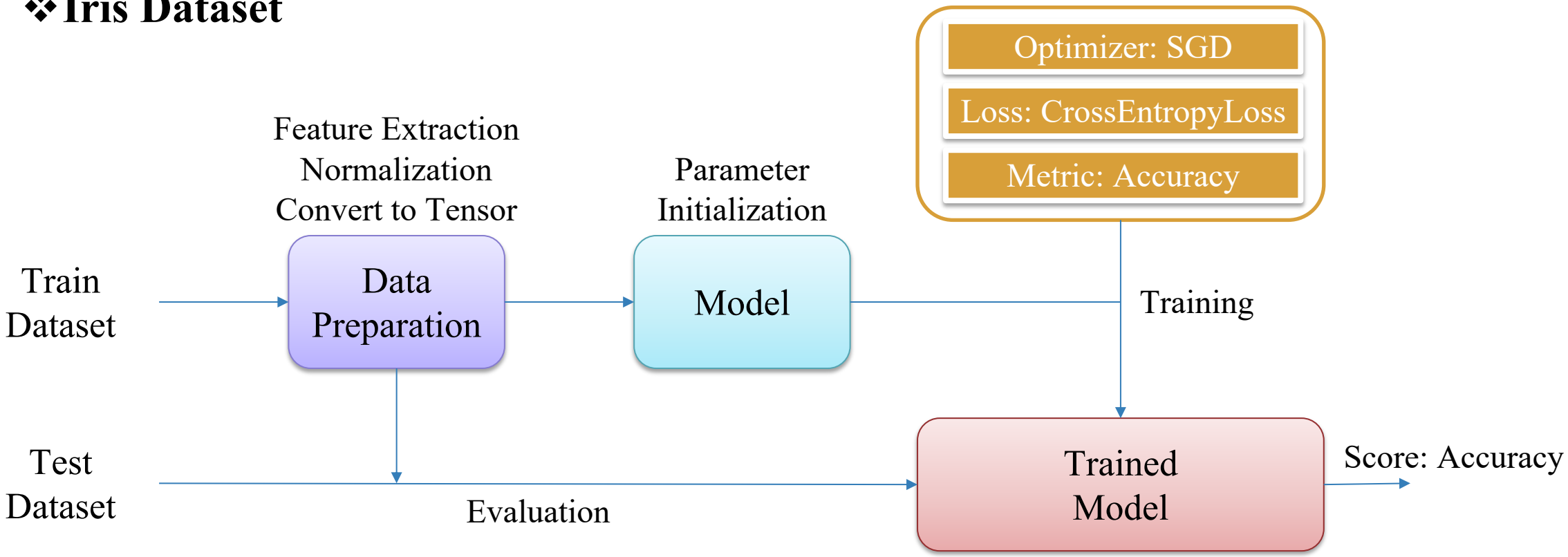
```
for layer in model.children():
    print(layer.state_dict())
```

```
OrderedDict([('weight', tensor([[0.0946, 0.0891],
        [0.0946, 0.0891]])), ('bias', tensor([0.0946, 0.0946]))])
OrderedDict([('weight', tensor([[0.0782, 0.0782]])), ('bias', tensor([0.0455]))])
OrderedDict()
```

3 – Classification using MLP

Pipeline

❖ **Iris Dataset**



3 – Classification using NN

Load Iris Dataset

- Load Iris Dataset from sklearn
- Train: Test = 0.7 : 0.3

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
data = load_iris()
```

```
data.data.shape
```

```
(150, 4)
```

```
data.target.shape
```

```
(150,)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(
    data.data,
    data.target,
    test_size=0.3
)
```

3 – Classification using NN

Data Preparation

- Normalization: StandardScaler()
- Convert to tensor

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(X_train)
```

```
▾ StandardScaler
StandardScaler()
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = torch.tensor(X_train, dtype=torch.float32)
Y_train = torch.tensor(Y_train)
X_test = torch.tensor(X_test, dtype=torch.float32)
Y_test = torch.tensor(Y_test)
```

3 – Classification using NN

Model

```
model_classifier = nn.Sequential(
    nn.Linear(4, 8),
    nn.ReLU(),
    nn.Linear(8, 3)
)
```

```
model_classifier
```

```
Sequential(
  (0): Linear(in_features=4, out_features=8, bias=True)
  (1): ReLU()
  (2): Linear(in_features=8, out_features=3, bias=True)
)
```

```
summary(model_classifier, (1, 4))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 8]              40
              ReLU-2                [-1, 1, 8]               0
            Linear-3                [-1, 1, 3]              27
================================================================
Total params: 67
Trainable params: 67
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

3 – Classification using NN

Loss Function, Optimizer

```
X_train[0]
```

```
tensor([0.7150, 0.0113, 1.0488, 0.8702])
```

```
y_pred = model_classifier(X_train[0])
y_pred
```

```
tensor([ 0.0074, -0.2822,  0.1523], grad_fn=<AddBackward0>)
```

```
loss_fn = nn.CrossEntropyLoss()
```

```
loss = loss_fn(y_pred, Y_train[0])
loss
```

```
tensor(0.9214, grad_fn=<NllLossBackward0>)
```

```
loss.item()
```

```
0.9213845729827881
```

```
learning_rate = 0.01

optimizer = optim.SGD(
    model_classifier.parameters(),
    learning_rate
)
```

3 – Classification using NN

Training

```
num_epochs = 20
losses = []
for epoch in range(num_epochs):
    epoch_loss = []
    for x_train, y_train in zip(X_train, Y_train):
        y_pred = model_classifier(x_train)
        loss = loss_fn(y_pred, y_train)
        epoch_loss.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    losses.append(sum(epoch_loss)/len(epoch_loss))
```

```
losses
```

```
[1.0220808364096141,
 0.7892919631231399,
 0.5656777904147193,
 0.43206991709413983,
 0.3591597341001034,
 0.31344637402466363,
 0.2815234812508736,
 0.25728054582363086,
 0.23809761630726003,
 0.22223558177107147,
 0.2087183050567373,
 0.19690465535746798,
 0.1863620154160474,
 0.17690506201636577,
 0.1683366762330046,
 0.16056279294702802,
 0.15347826486963423,
 0.14700075439288326,
 0.14108285806917895,
 0.13565213933332068]
```

3 – Classification using NN

Evaluation

```
with torch.no_grad():
    Y_pred = model_classifier(X_test)
```

```
Y_pred = torch.argmax(Y_pred, dim=1)
```

```
Y_pred
```

```
tensor([1, 2, 1, 1, 1, 1, 1, 2, 2, 0, 1, 0, 2, 0, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2,
        2, 0, 1, 1, 2, 0, 1, 2, 0, 0, 1, 0, 2, 1, 2, 1, 0, 1, 1, 2, 2])
```

```
sum(Y_pred == Y_test)/len(Y_test)
```

```
tensor(0.9778)
```

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?