AI VIET NAM
@aivietnam.edu.vn

Module 05 – Extra Class

Multi-layer Perceptron

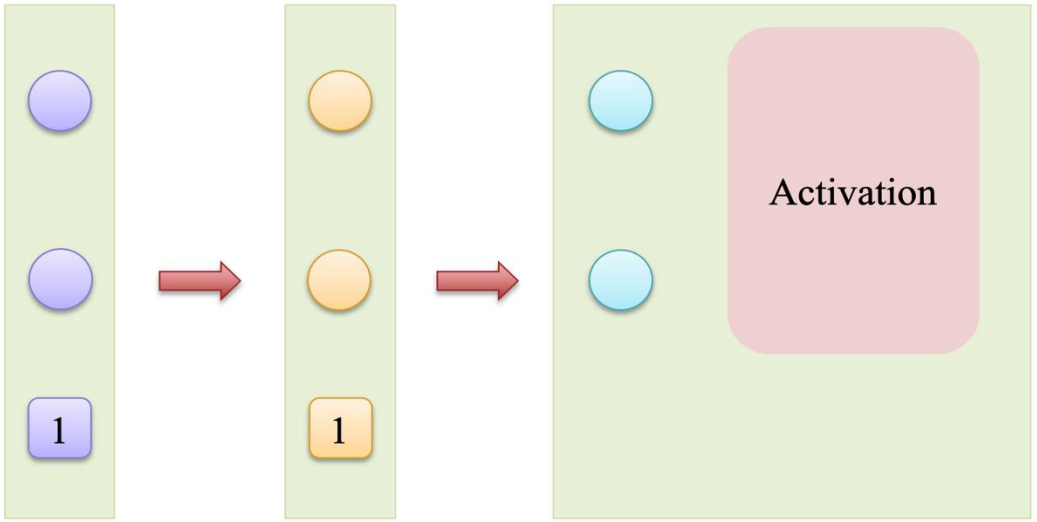
Nguyen Quoc Thai

Code - Data

Objectives

Softmax Regression (Review)

- Logistic Regression
- Softmax Function
- Softmax Regression using SGD



Multi-layer Perceptron

- Hidden Layers
- Activation
- Loss Function
- Optimization

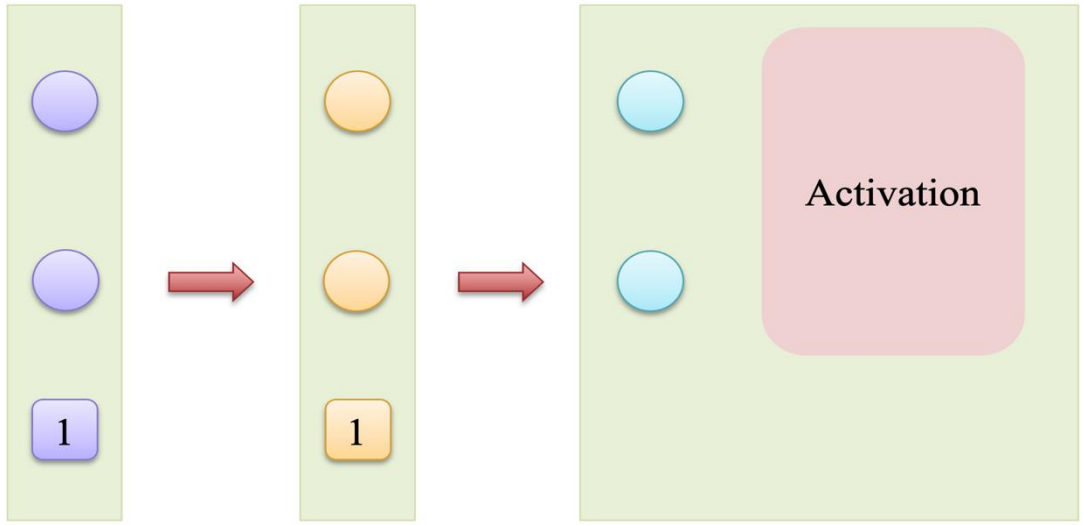
Outline

SECTION 1

Softmax Regression

SECTION 2

Multi-layer Perceptron



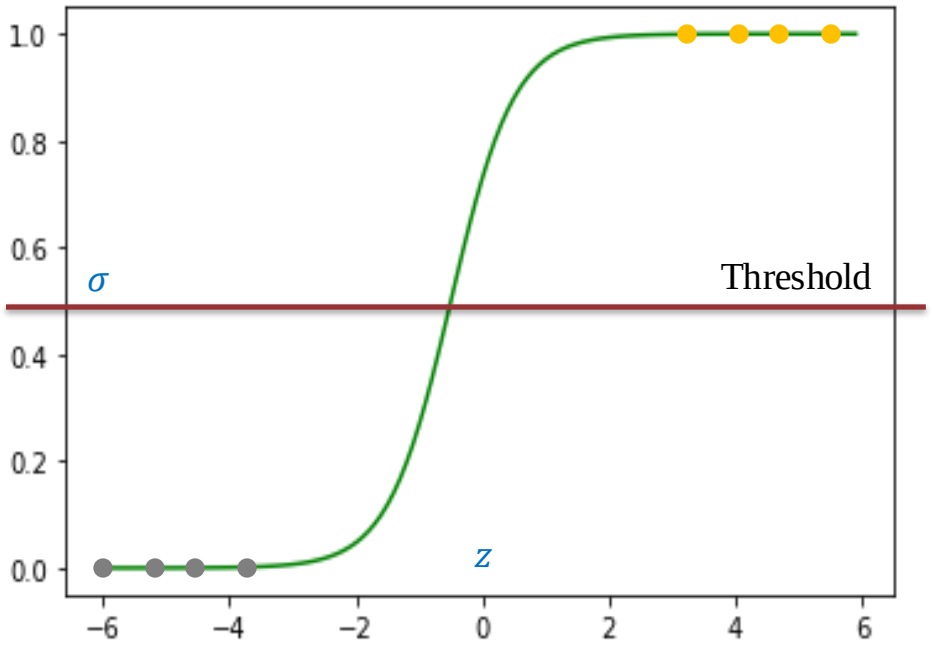
Softmax Regression

Problem

Sigmoid function

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

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



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

Softmax Regression

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

$z_1 = 1.0$

$z_2 = 3.0$

→ **Softmax** →

Probability

$f(z_1) = 0.12$

$f(z_2) = 0.88$

Input

$z_1 = 1.0$

$z_2 = 2.0$

$z_3 = 3.0$

→ **Softmax** →

Probability

$f(z_1) = 0.09$

$f(z_2) = 0.24$

$f(z_3) = 0.67$

Softmax Regression

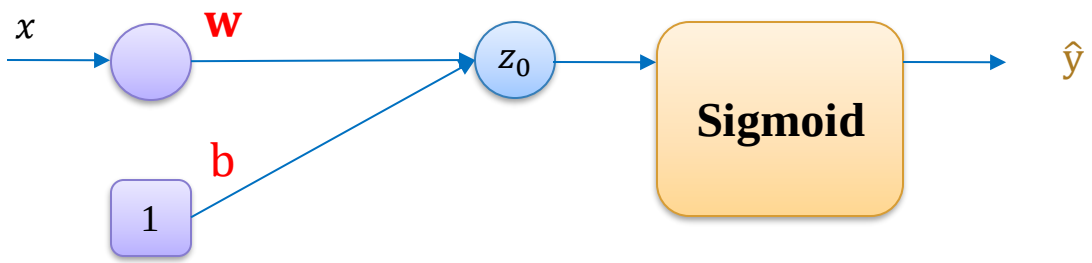
Parameters

Classes: {0, 1}
Binary Classification

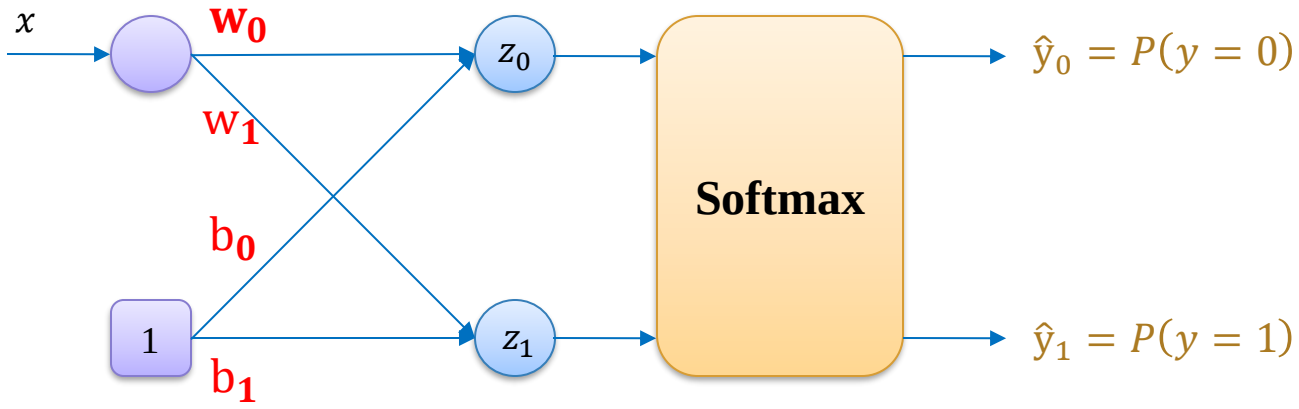
Hours	Pass
0.5	0
2.0	1

#feature: 1
#class: 2

Logistic Regression



Softmax Regression



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

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

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

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

Softmax Regression

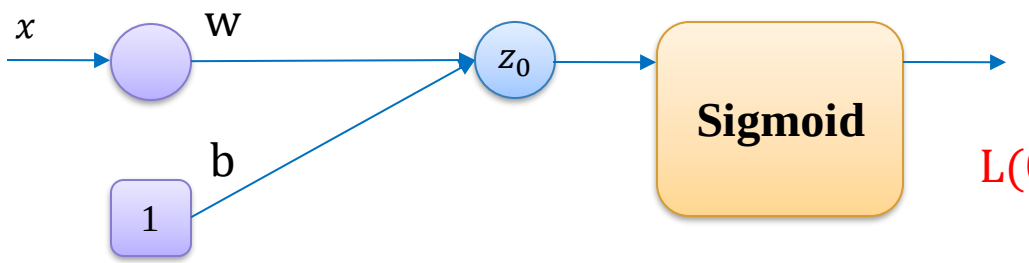
Loss Function

Classes: {0, 1}
Binary Classification

Hours	Pass
0.5	0
2.0	1

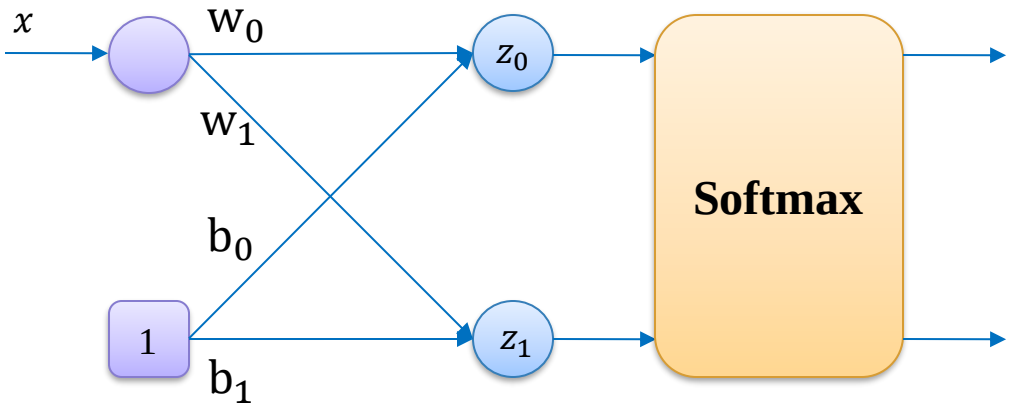
#feature: 1
#class: 2

Logistic Regression



$$L(\boldsymbol{\theta}) = (-y\log\hat{y} - (1-y)\log(1-\hat{y}))$$

Softmax Regression



$$L(\boldsymbol{\theta}) = -y_1\log(\hat{y}_1) - y_0\log(\hat{y}_0)$$

$$= -\sum_i y_i\log(\hat{y}_i)$$

$y_0 = 1$
$y_1 = 0$
One-Hot Encoding
y
y = 0

Softmax Regression

Softmax Regression using Gradient Descent

1) Pick a sample from training data

2) Compute output $\hat{y}$

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

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

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

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

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

4) Compute derivative

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

5) Update parameters

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

η is learning rate

$$\boldsymbol{x} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix}$$

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

One-hot encoding for label

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

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

$$\boldsymbol{\theta} = \begin{bmatrix} \mathrm{b}_0 & \mathrm{b}_1 \\ \mathrm{w}_0 & \mathrm{w}_1 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$

$$\boldsymbol{\eta} = 0.01$$

Hours	Pass
0.5	0
1.0	0
1.5	1
2.0	1

Softmax Regression

Softmax Regression using Gradient Descent

1) Pick a sample from training data

2) Compute output $\hat{y}$

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

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

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

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

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

4) Compute derivative

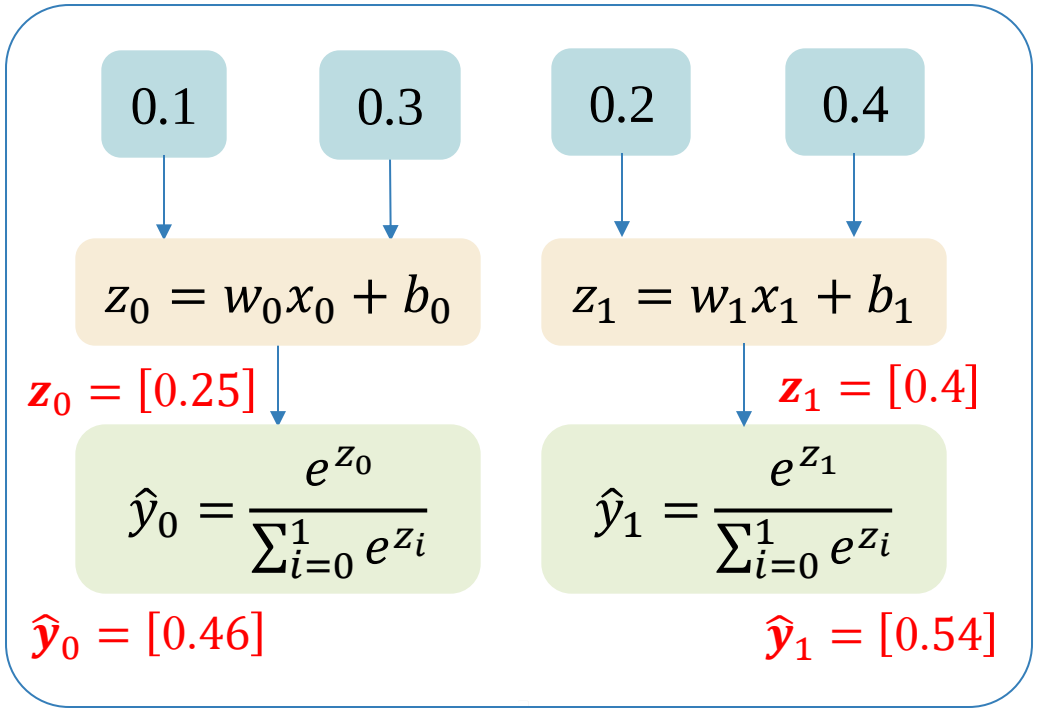
$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

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

η is learning rate

$\boldsymbol{\eta} = 0.01$ $\quad \boldsymbol{x} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix}$ $\quad \boldsymbol{y} = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\quad \boldsymbol{\theta} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$



Softmax Regression

Softmax Regression using Gradient Descent

1) Pick a sample from training data

2) Compute output $\hat{y}$

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

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

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

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

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

4) Compute derivative

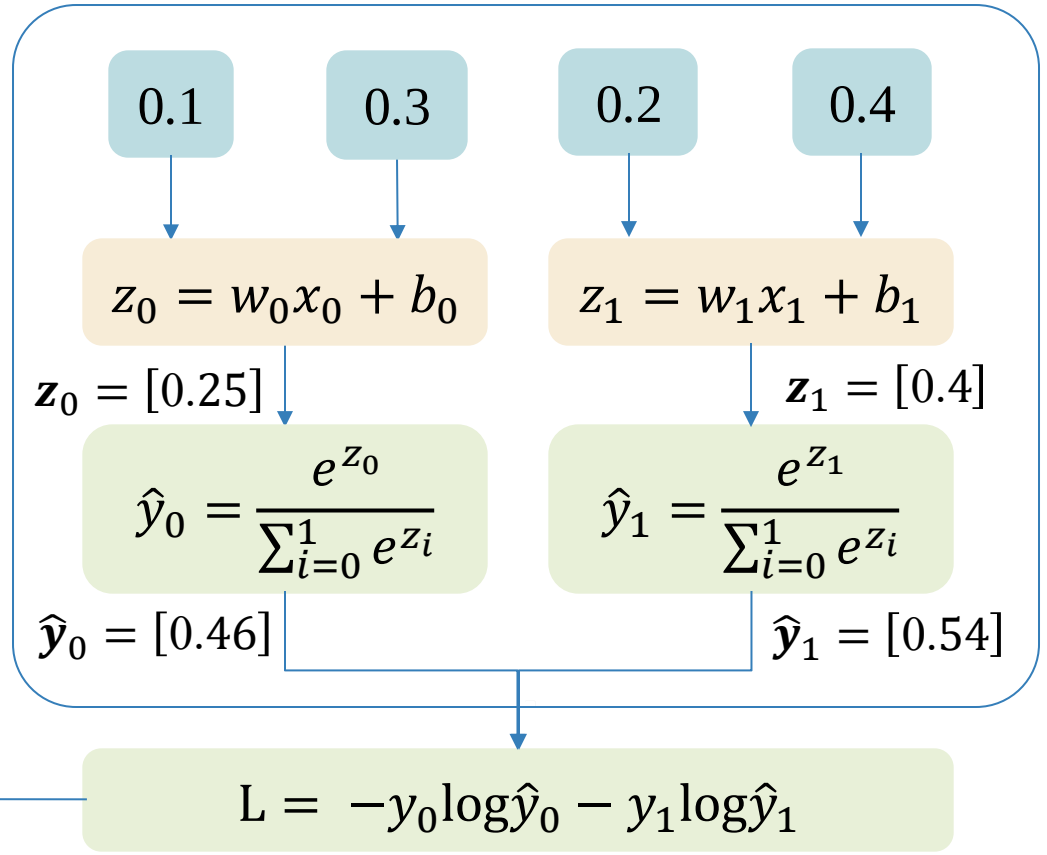
$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

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

η is learning rate

$\boldsymbol{\eta} = 0.01$ $\quad \boldsymbol{x} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix}$ $\quad \boldsymbol{y} = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\quad \boldsymbol{\theta} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$



Softmax Regression

Softmax Regression using Gradient Descent

1) Pick a sample from training data

2) Compute output $\hat{y}$

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

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

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

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

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

4) Compute derivative

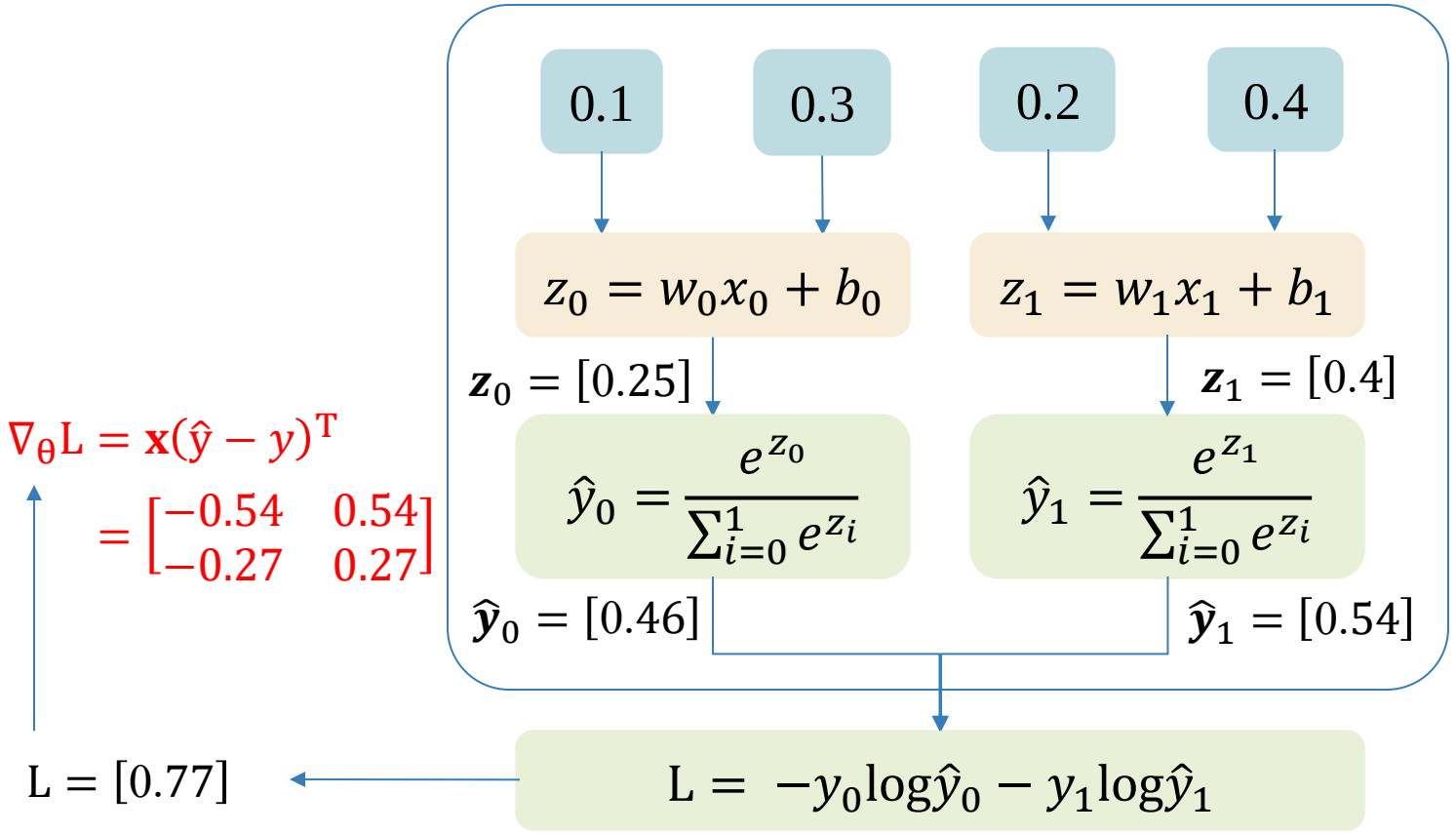
$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

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

η is learning rate

$\boldsymbol{\eta} = 0.01$ $\quad \boldsymbol{x} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix}$ $\quad \boldsymbol{y} = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\quad \boldsymbol{\theta} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$



Softmax Regression

Softmax Regression using Gradient Descent

1) Pick a sample from training data

2) Compute output $\hat{y}$

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

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

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

$\emptyset$ is Hadamard division

3) Compute loss (cross-entropy)

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

4) Compute derivative

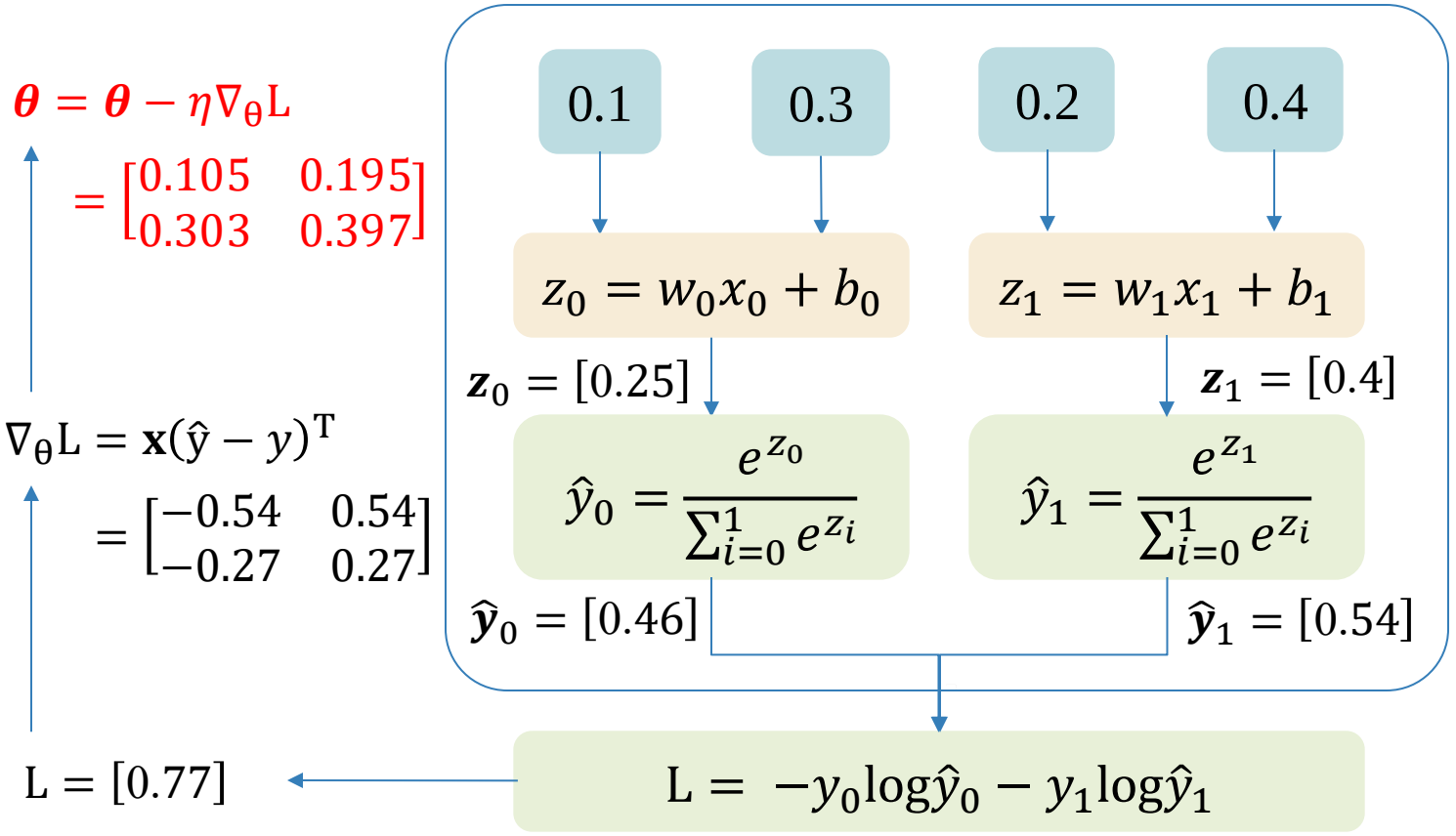
$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

5) Update parameters

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

η is learning rate

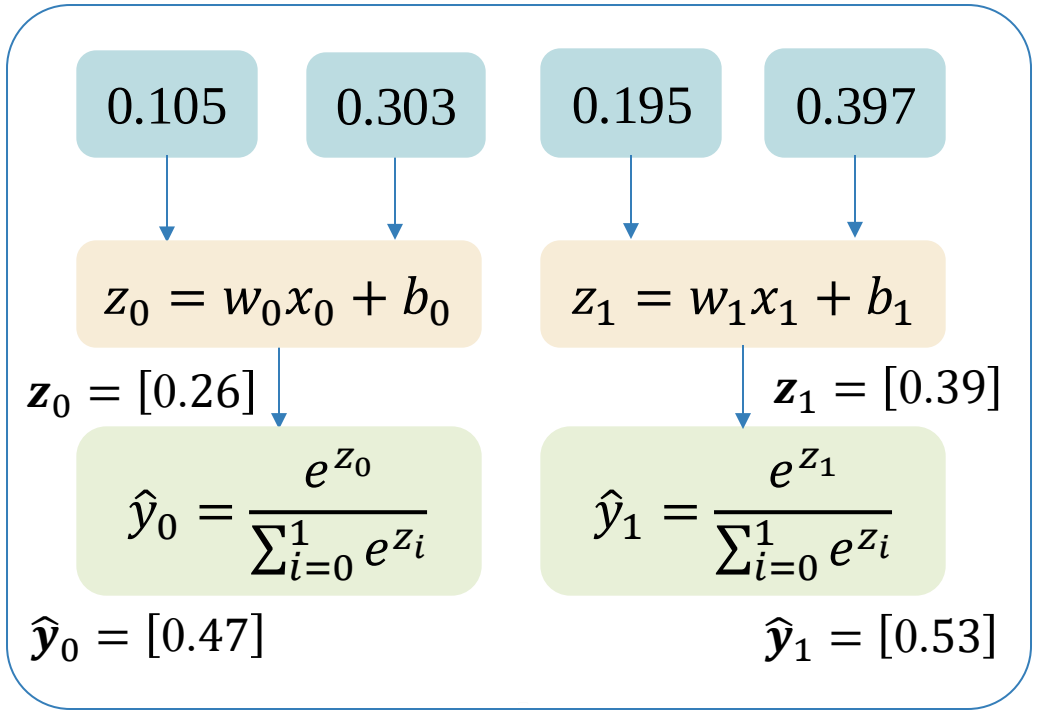
$$\boldsymbol{\eta} = 0.01 \qquad \boldsymbol{x} = \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix} \qquad \boldsymbol{y} = 0 \rightarrow \boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$



Softmax Regression

Softmax Regression using Gradient Descent

Hours	Pass
0.5	?



$y_{pred} = 1$

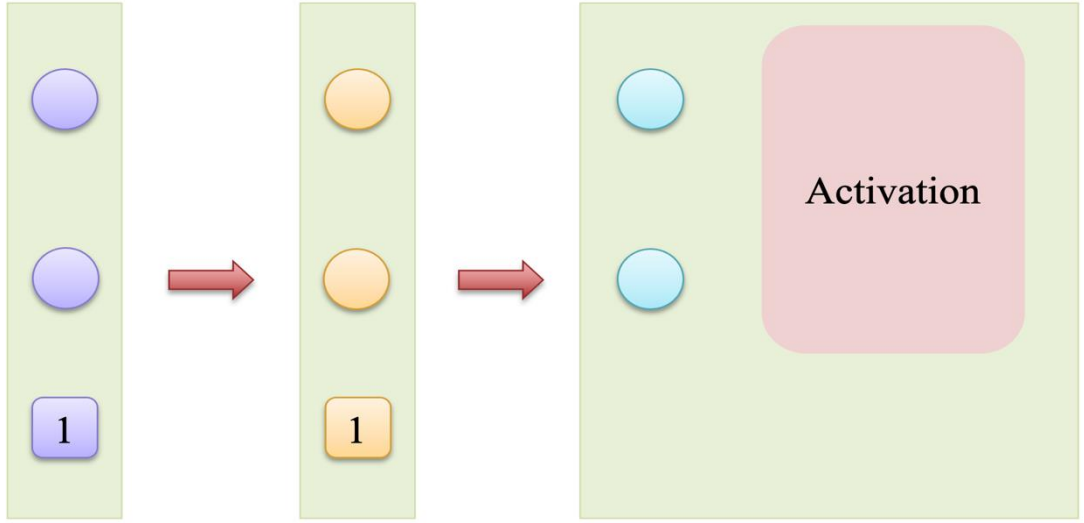
Outline

SECTION 1

Softmax Regression

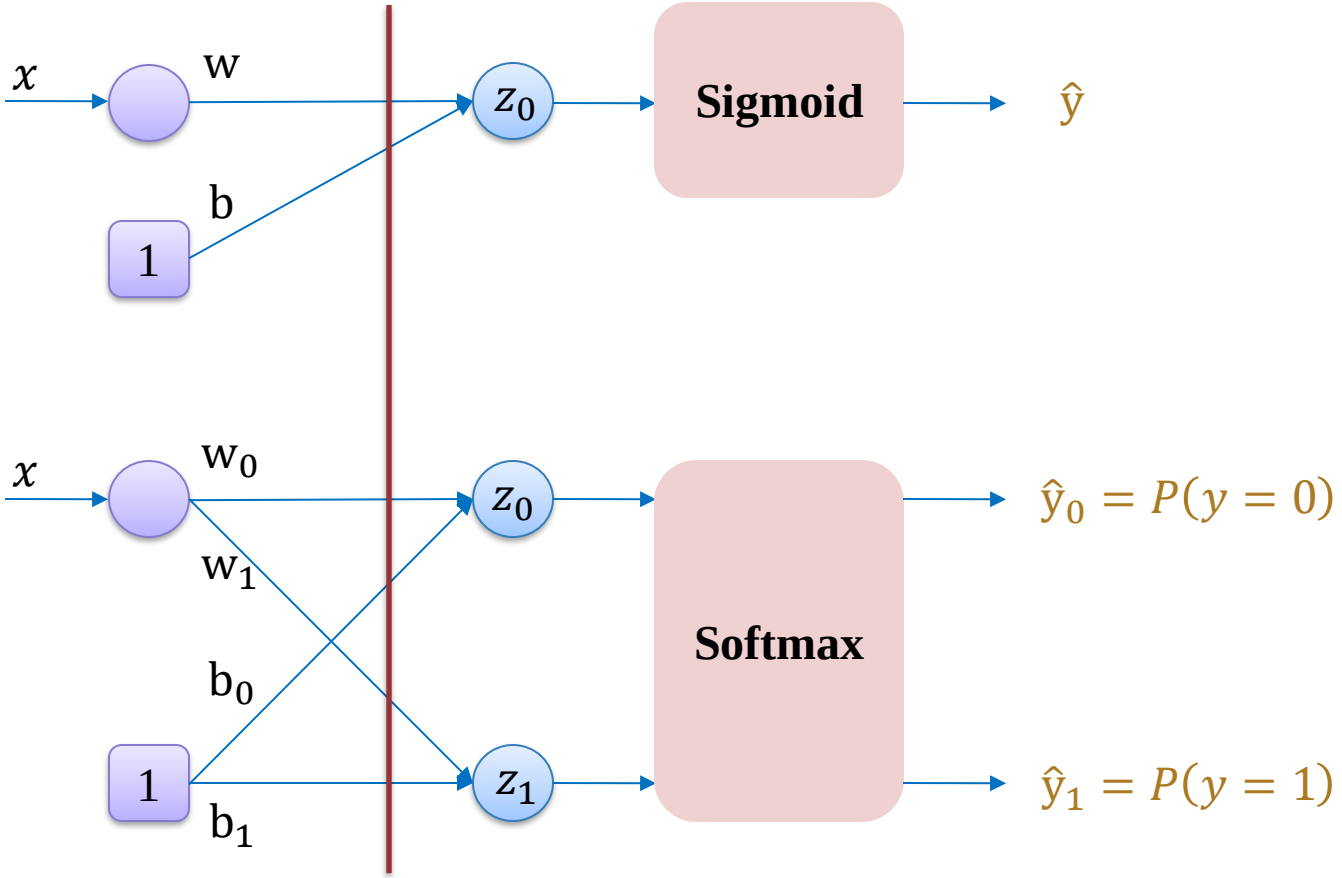
SECTION 2

Multi-layer Perceptron



Multi-layer Perceptron

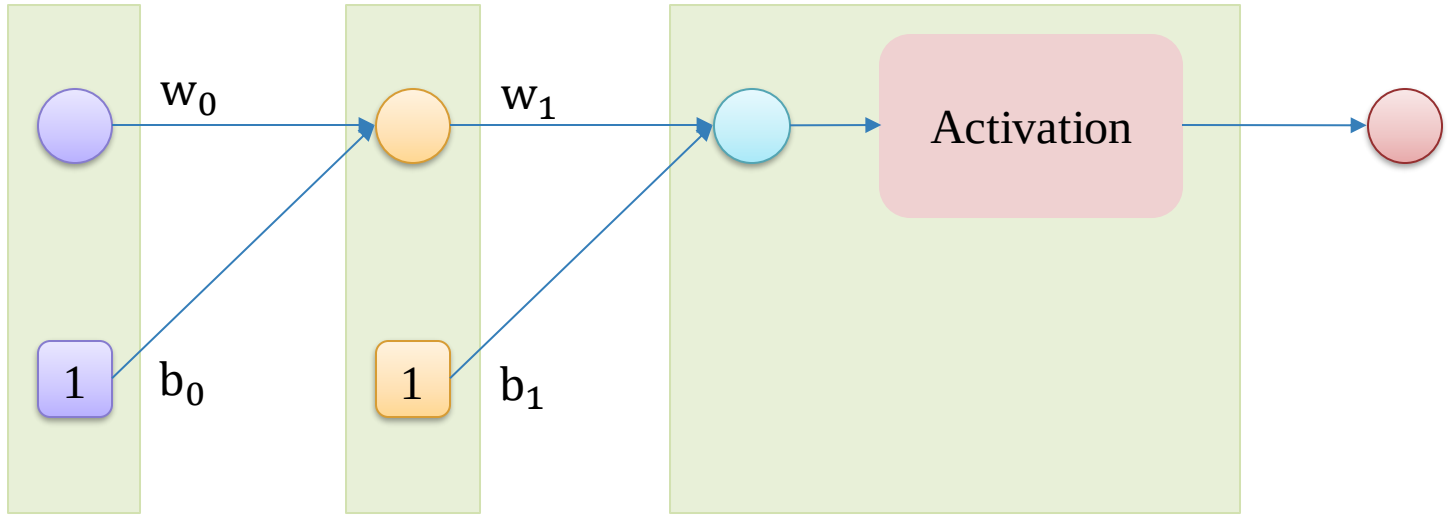
Motivation



Multi-layer Perceptron

Hidden Layer

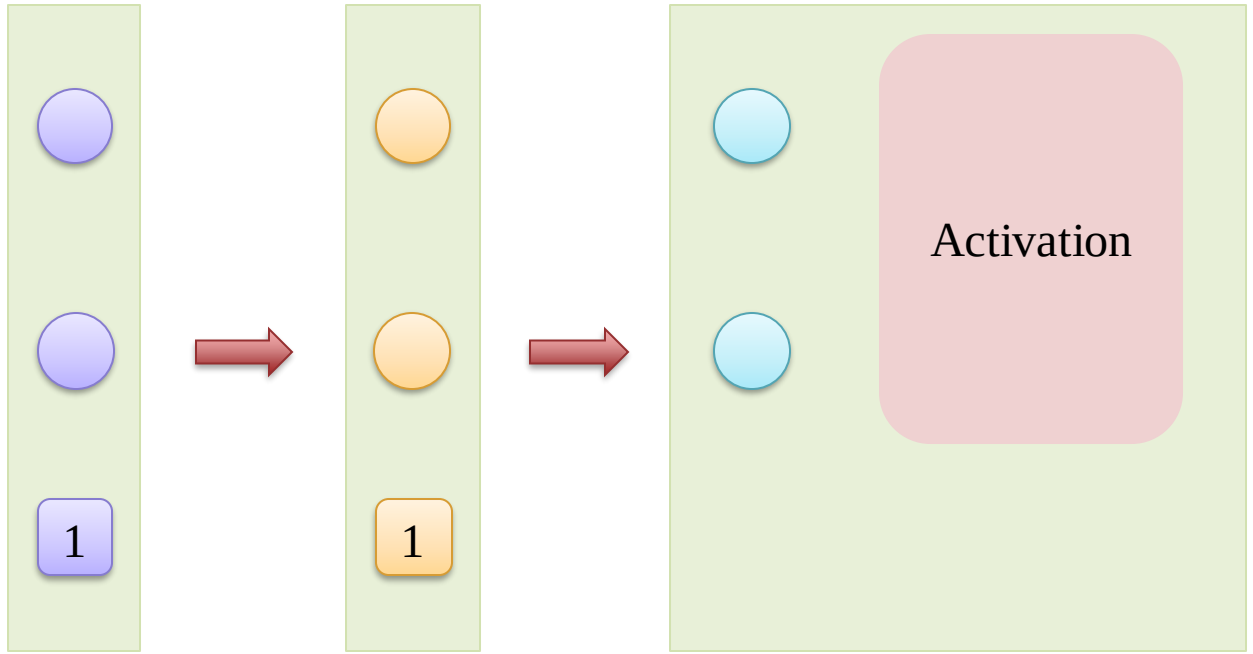
#parameters: 4



Multi-layer Perceptron

Multi-layer Perceptron

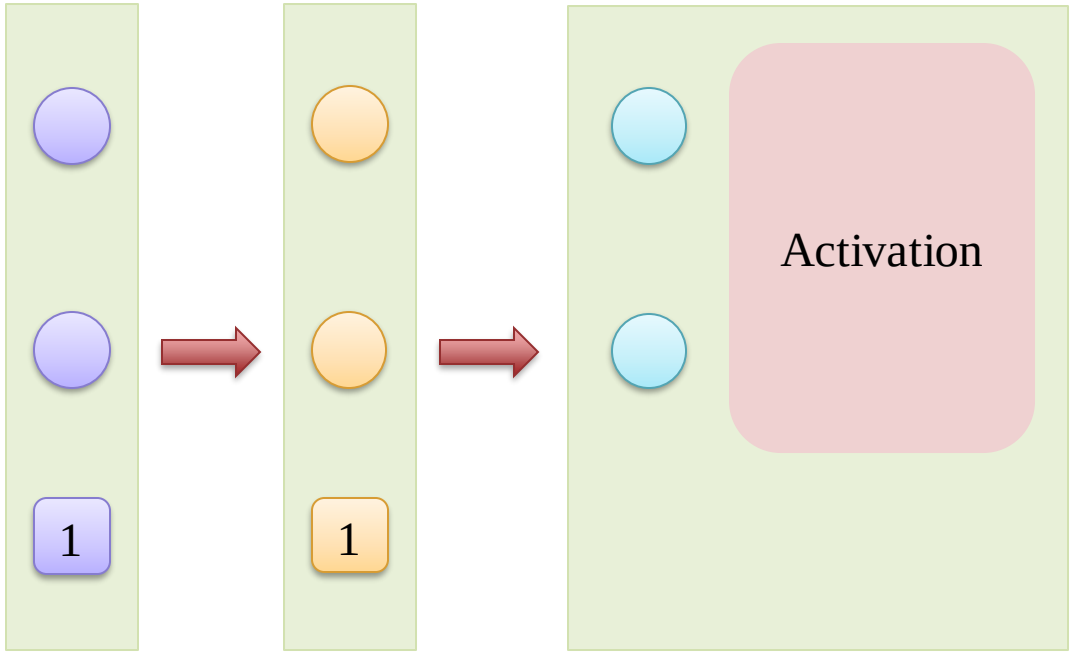
#parameters: 12



Multi-layer Perceptron

Multi-layer Perceptron using Pytorch

#parameters: 12




```
1 model = nn.Sequential(
2     nn.Linear(2, 2),
3     nn.Linear(2, 2),
4     nn.Sigmoid()
5 )
```

```
1 summary(model, (10000000, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1       [-1, 10000000, 2]               6
            Linear-2       [-1, 10000000, 2]               6
           Sigmoid-3       [-1, 10000000, 2]               0
================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 76.29
Forward/backward pass size (MB): 457.76
Params size (MB): 0.00
Estimated Total Size (MB): 534.06
----------------------------------------------------------------
```

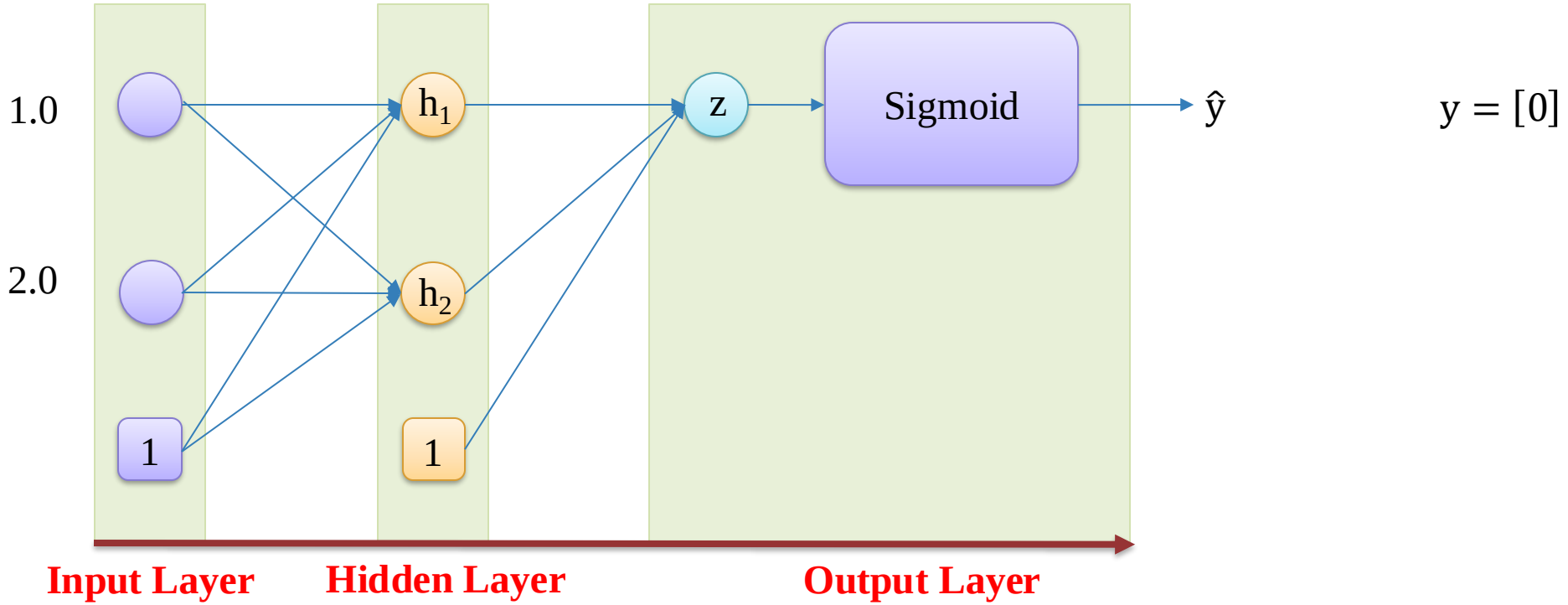
Multi-layer Perceptron

Forward

$$x = [1.0 \quad 2.0]$$

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

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



$$y = [0]$$

Multi-layer Perceptron

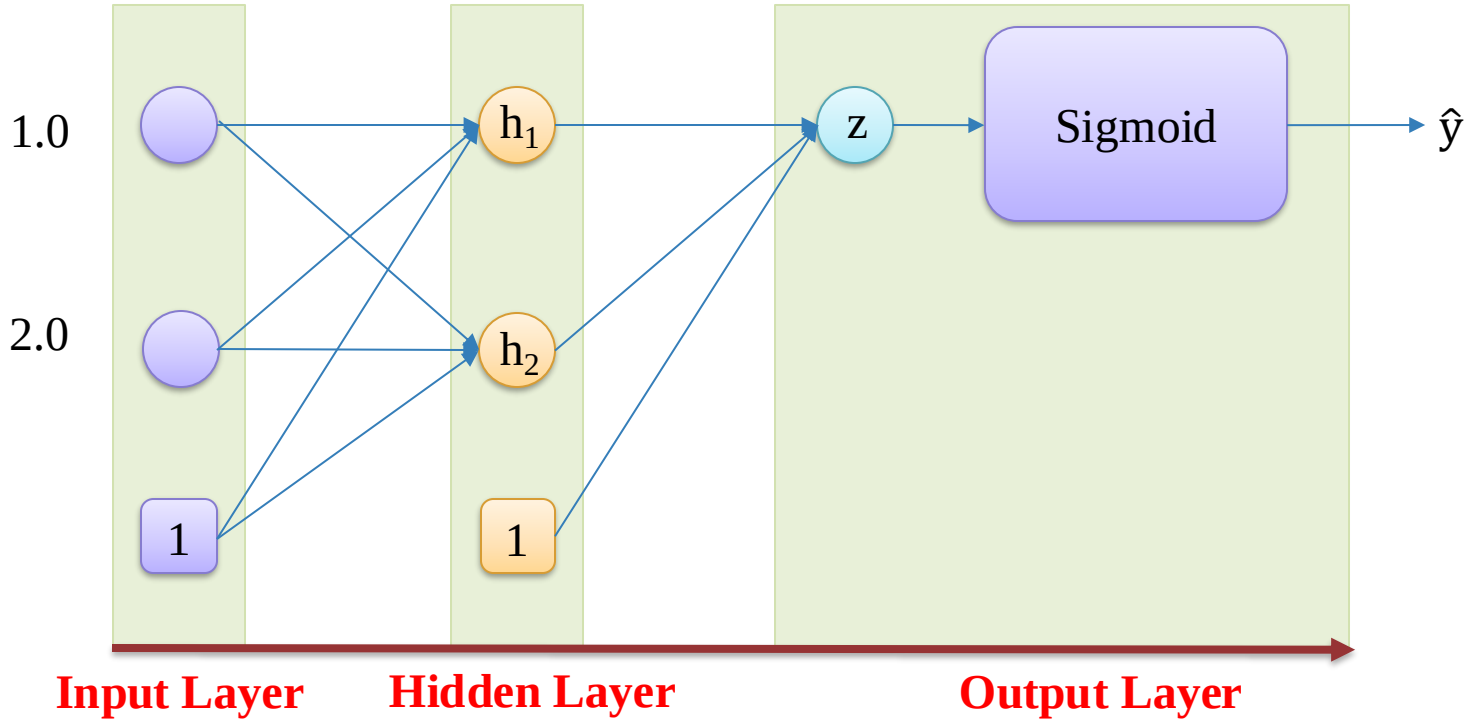
Forward

$x = [1.0 \quad 2.0]$

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

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

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



$y = [0]$

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

Multi-layer Perceptron

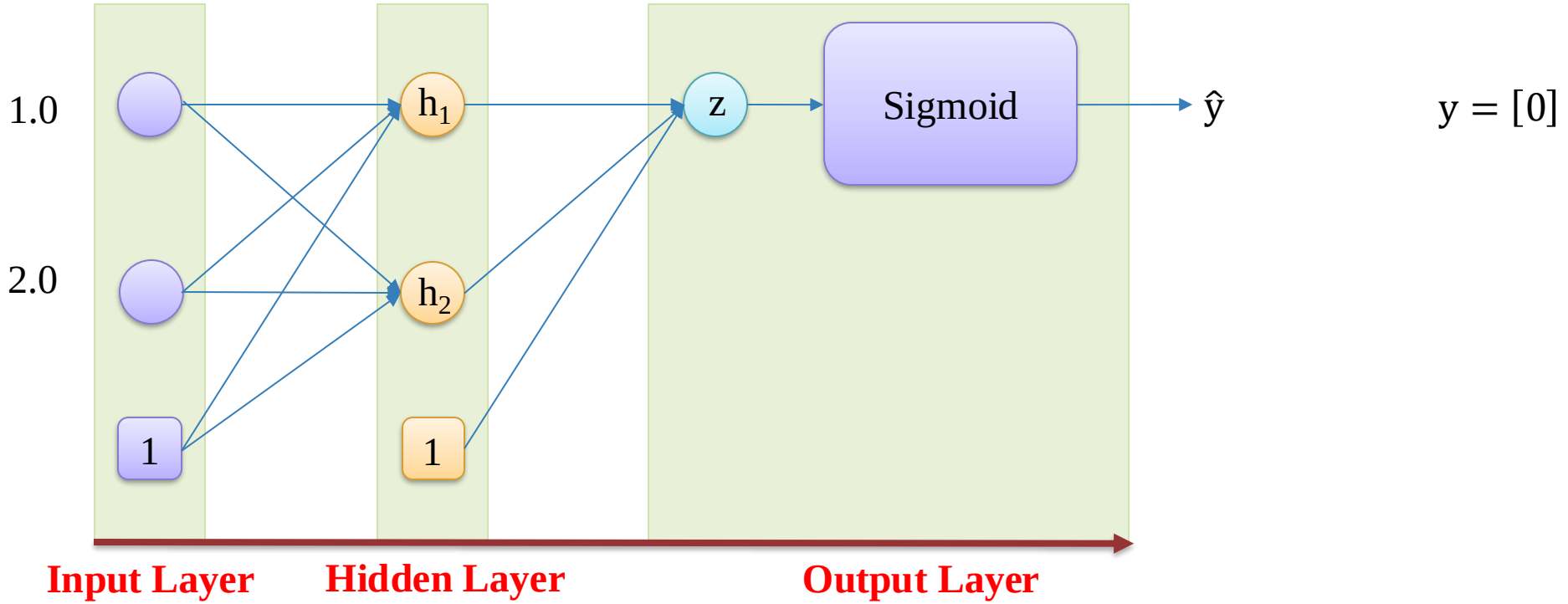
Forward

$x = [1.0 \quad 2.0]$

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

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

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



$y = [0]$

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

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

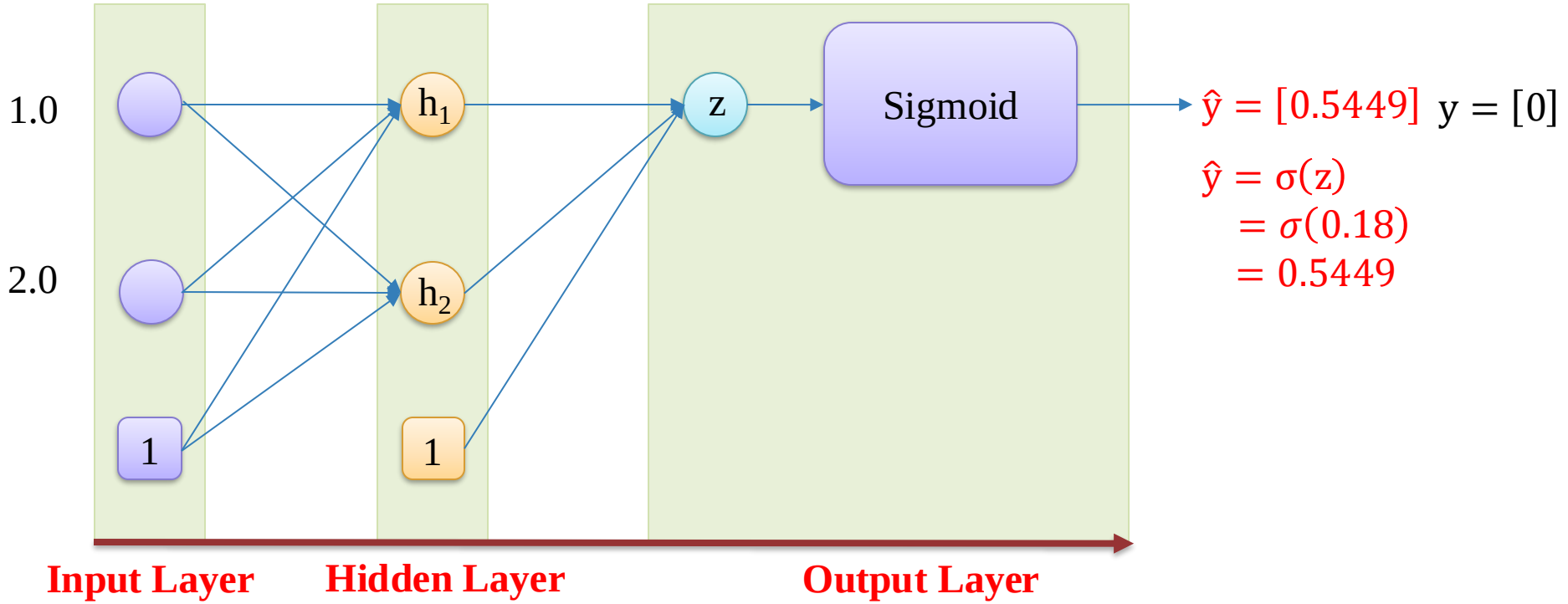
Multi-layer Perceptron

Forward

$x = [1.0 \quad 2.0]$

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

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



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

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

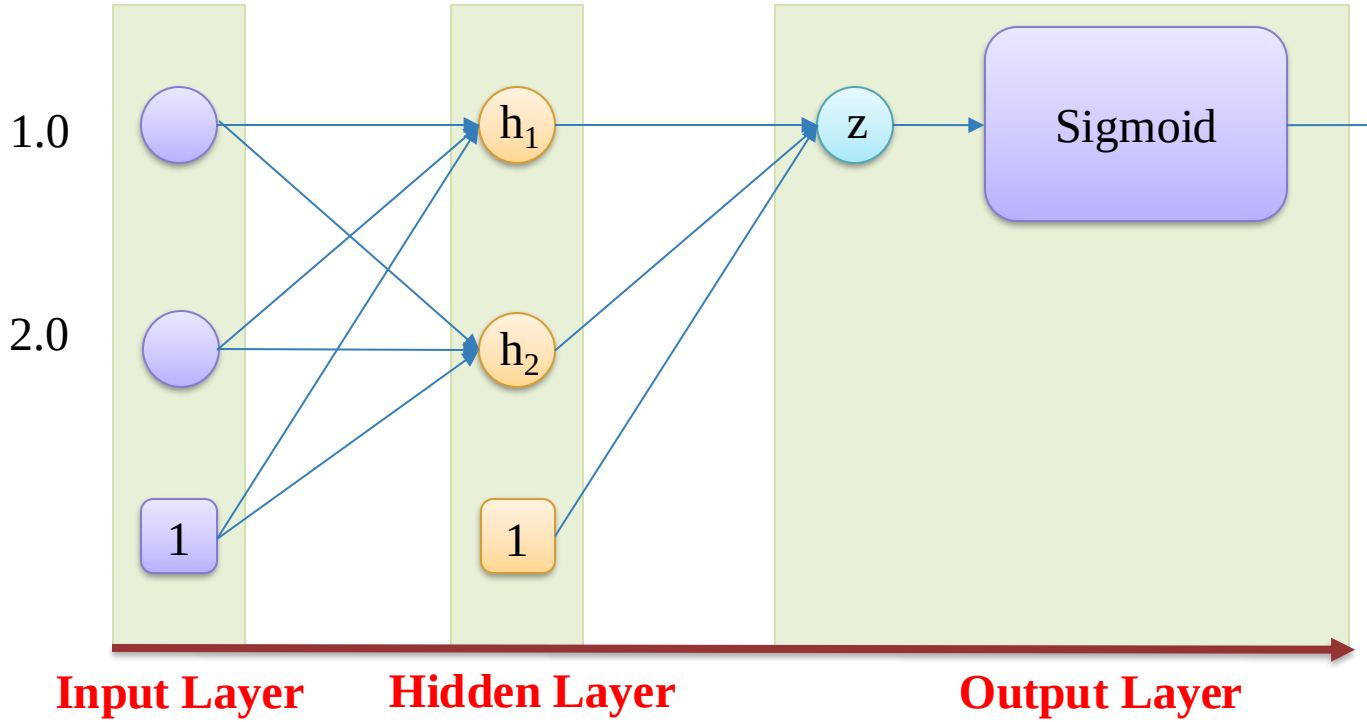
Multi-layer Perceptron

Forward

$x = [1.0 \quad 2.0]$

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

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



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

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

$L = 0.7872$

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

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

Multi-layer Perceptron

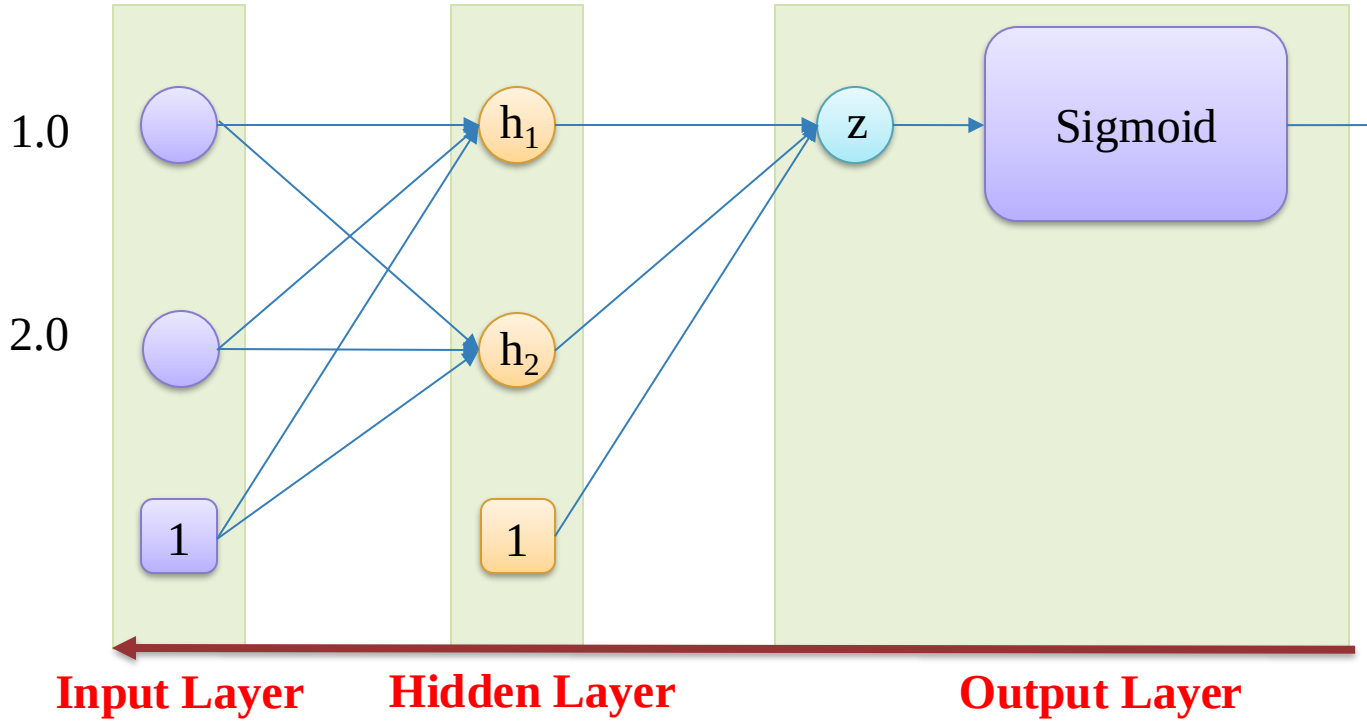
Backward

$x = [1.0 \quad 2.0]$

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

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

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



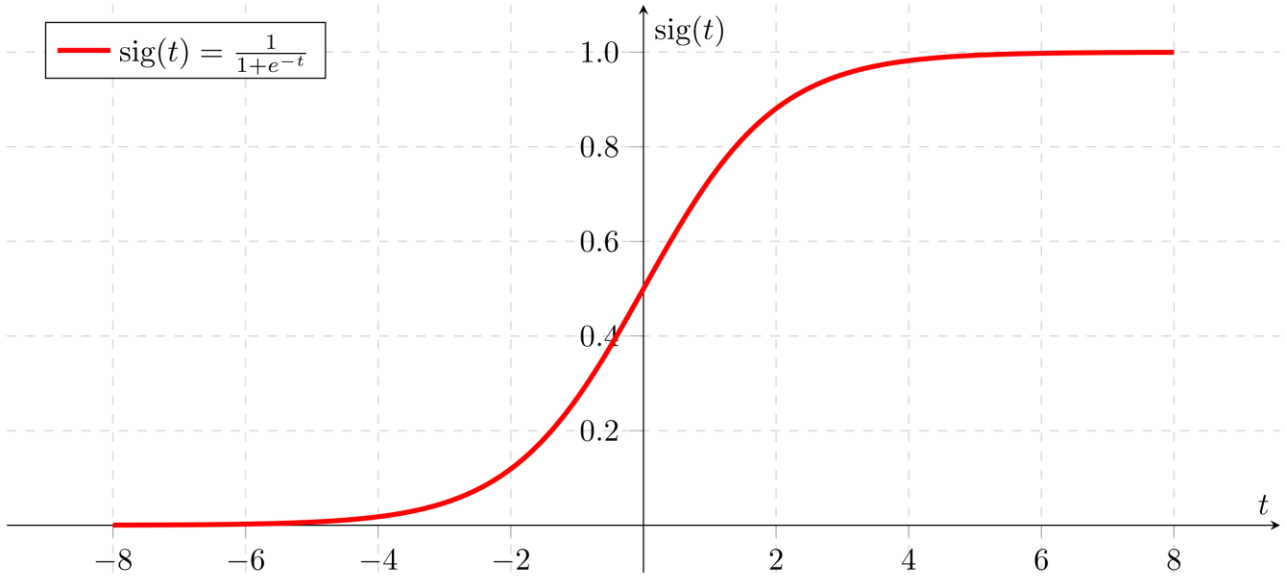
$\hat{y} = [0.5449]$ $y = [0]$

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

$L = 0.7872$

Activation

Sigmoid Function



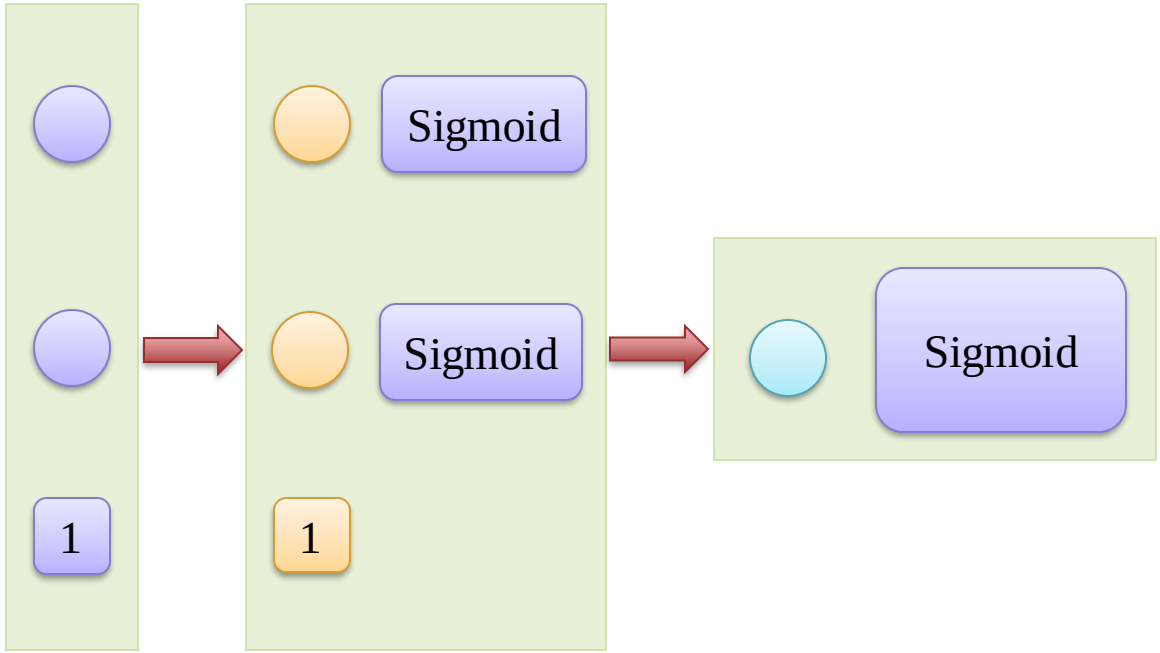

```python
import torch.nn as nn

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

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

Activation

Sigmoid Function




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

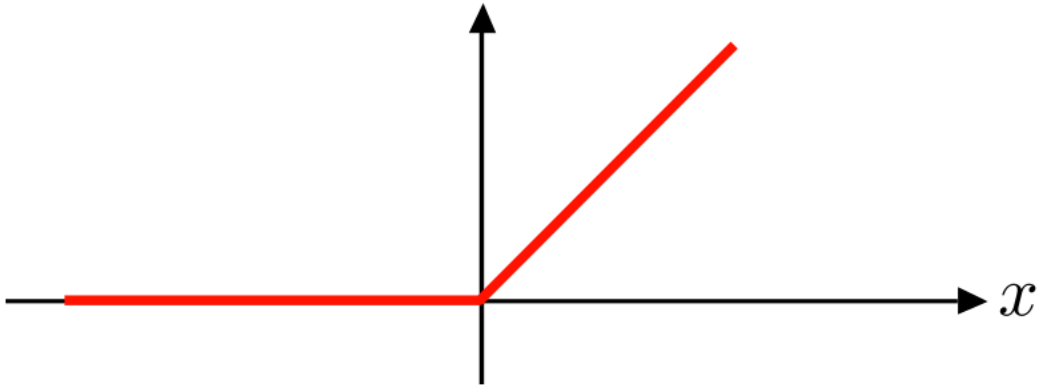
```python
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        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
----------------------------------------------------------------
```

Activation

ReLU Function

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



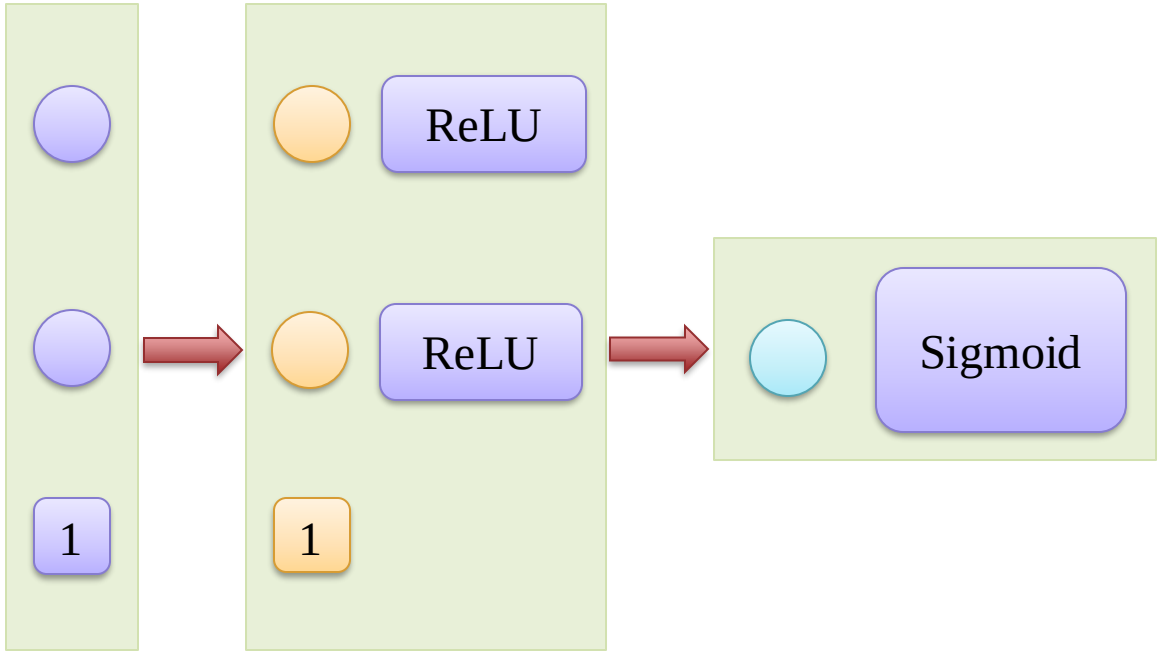

```python
import torch.nn as nn

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

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

Activation

ReLU Function




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

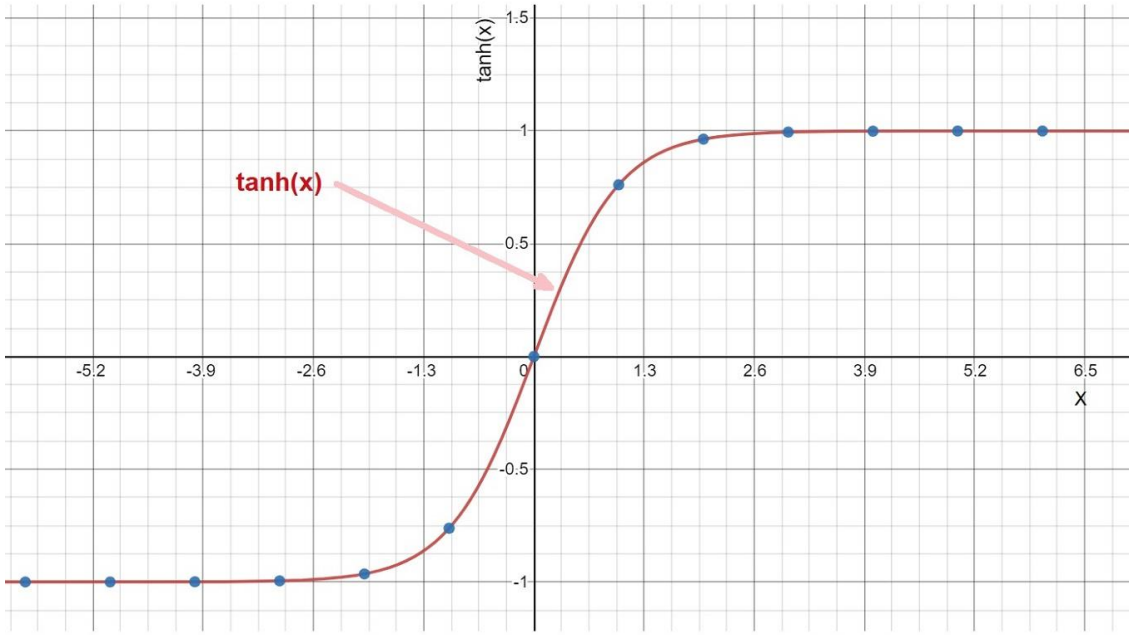
```python
summary(model, (1, 2))
```

```
----------------------------------------------------------------
        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
----------------------------------------------------------------
```

Activation

Tanh Function

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



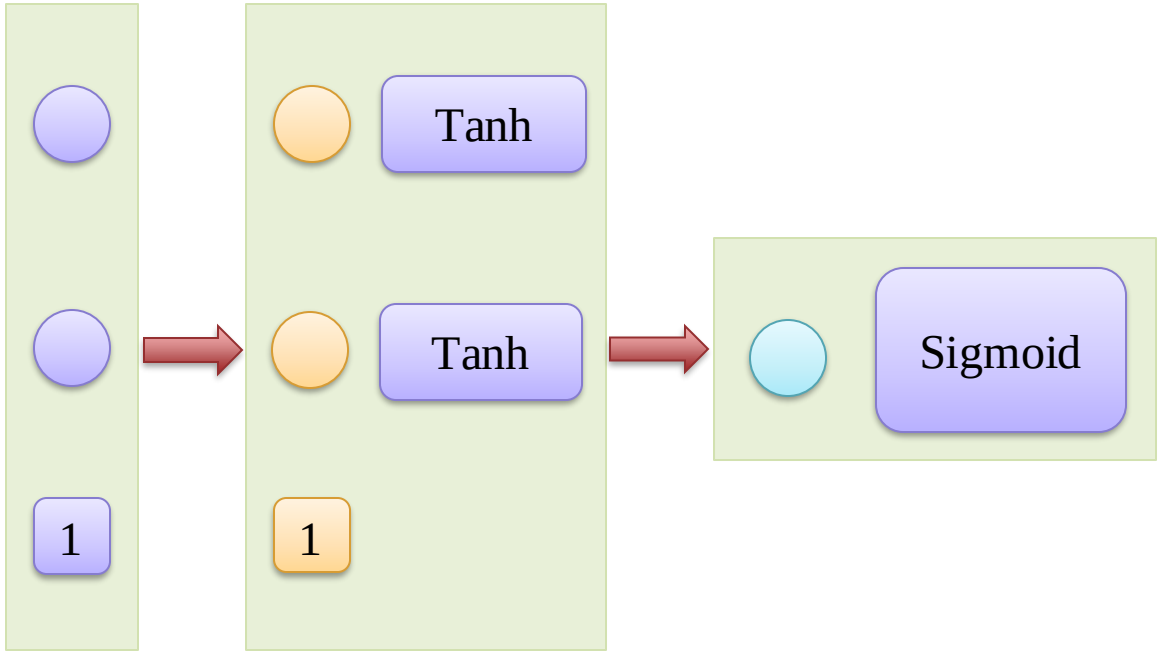

```python
import torch.nn as nn

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

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

Activation

ReLU Function




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

```python
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
----------------------------------------------------------------
```

Loss Function

BCELoss

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

```
1 y_pred = model(x)
2 y_pred
```

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

```
1 y
```

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

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

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

```
1 model = nn.Sequential(
2     nn.Linear(2, 2),
3     nn.Linear(2, 2),
4     nn.Sigmoid()
5 )
```

```
1 summary(model, (10000000, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1          [-1, 10000000, 2]               6
            Linear-2          [-1, 10000000, 2]               6
           Sigmoid-3          [-1, 10000000, 2]               0
================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 76.29
Forward/backward pass size (MB): 457.76
Params size (MB): 0.00
Estimated Total Size (MB): 534.06
----------------------------------------------------------------
```

Loss Function

CrossEntropyLoss

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

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

```
tensor(0)
```

```
1 y_pred
```

```
tensor([ 0.9005, -0.1670], grad_fn=<ViewBackward0>)
```

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

```
tensor(0.2955, grad_fn=<NllLossBackward0>)
```

```
1 model = nn.Sequential(
2     nn.Linear(2, 2),
3     nn.ReLU(),
4     nn.Linear(2, 2),
5 )
```

```
1 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
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

Optimizer

SGD

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

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

```
loss.backward()
```

```
optimizer.step()
```

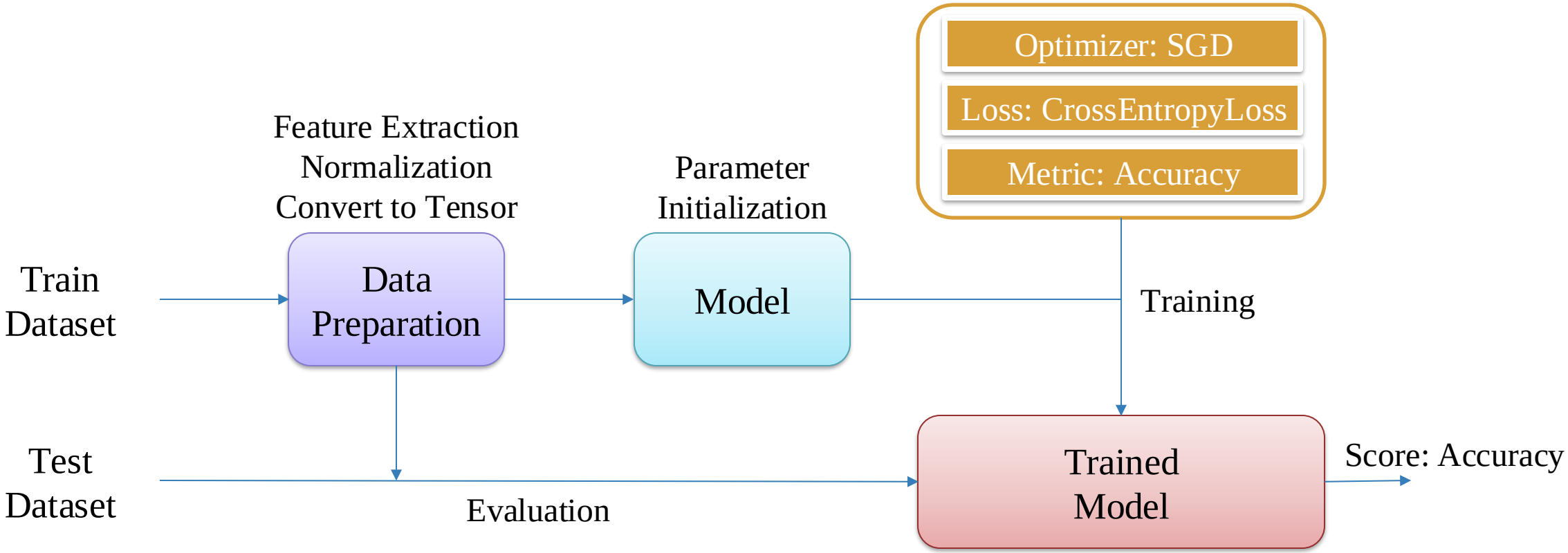
```
for layer in model.children():
    print(layer.state_dict())
```

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


QUIZ TIME

Classification using MLP

IRIS Dataset



Classification using MLP

Load the IRIS dataset

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

```
1 data = load_iris()
```

```
1 data.data.shape
```

```
(150, 4)
```

```
1 data.target.shape
```

```
(150,)
```

Classification using MLP

Train Test Split

```
X_train, X_test, Y_train, Y_test = train_test_split(
    data.data,
    data.target,
    test_size=0.4,
    random_state=7
)
```

```
X_valid, X_test, Y_valid, Y_test = train_test_split(
    X_test,
    Y_test,
    test_size=0.5,
    random_state=7
)
```

```
X_train.shape, X_valid.shape, X_test.shape
```

```
((90, 4), (30, 4), (30, 4))
```

Classification using MLP

Preprocessing

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

```
1 scaler = StandardScaler()
```

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

```
StandardScaler()
```

```
1 X_train = scaler.transform(X_train)
2 X_valid = scaler.transform(X_valid)
3 X_test = scaler.transform(X_test)
```

Classification using MLP

Model

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

```
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
----------------------------------------------------------------
```

Classification using MLP

Loss Function

```
1 X_train[0]
```

```
tensor([-1.0341,  0.9957, -1.2020, -0.7777])
```

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

```
tensor([ 0.1499,  0.0141, -0.4355], grad_fn=<ViewBackward0>)
```

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

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

```
tensor(0.8878, grad_fn=<NllLossBackward0>)
```

```
1 loss.item()
```

```
0.8878384828567505
```

Classification using MLP

Optimizer

```
learning_rate = 0.01

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

```
def evaluate(model_classifier, X_valid, Y_valid):
    with torch.no_grad():
        Y_pred = model_classifier(X_valid)

    Y_pred = torch.argmax(Y_pred, dim=1)
    return sum(Y_pred == Y_valid)/len(Y_valid)
```

```
evaluate(model_classifier, X_valid, Y_valid)
```

```
tensor(0.5000)
```

Classification using MLP

Training

```python
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()
    avg_loss = sum(epoch_loss)/len(epoch_loss)
    losses.append(avg_loss)
    acc = evaluate(model_classifier, X_valid, Y_valid)
    print(f"{avg_loss} -- {acc}")
```

Classification using MLP

Evaluation

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

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

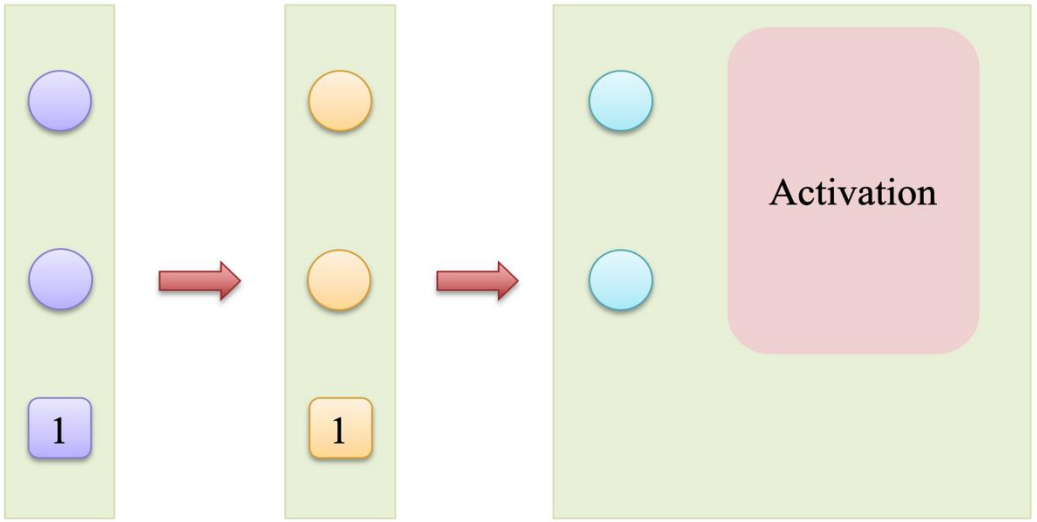
```
1 sum(Y_pred == Y_test)/len(Y_test)
```

```
tensor(0.9000)
```

Objectives

Softmax Regression (Review)

- Logistic Regression
- Softmax Function
- Softmax Regression using SGD



Multi-layer Perceptron

- Hidden Layers
- Activation
- Loss Function
- Optimization

Thanks!

Any questions?