AI VIETNAM
All-in-One Course

Pytorch

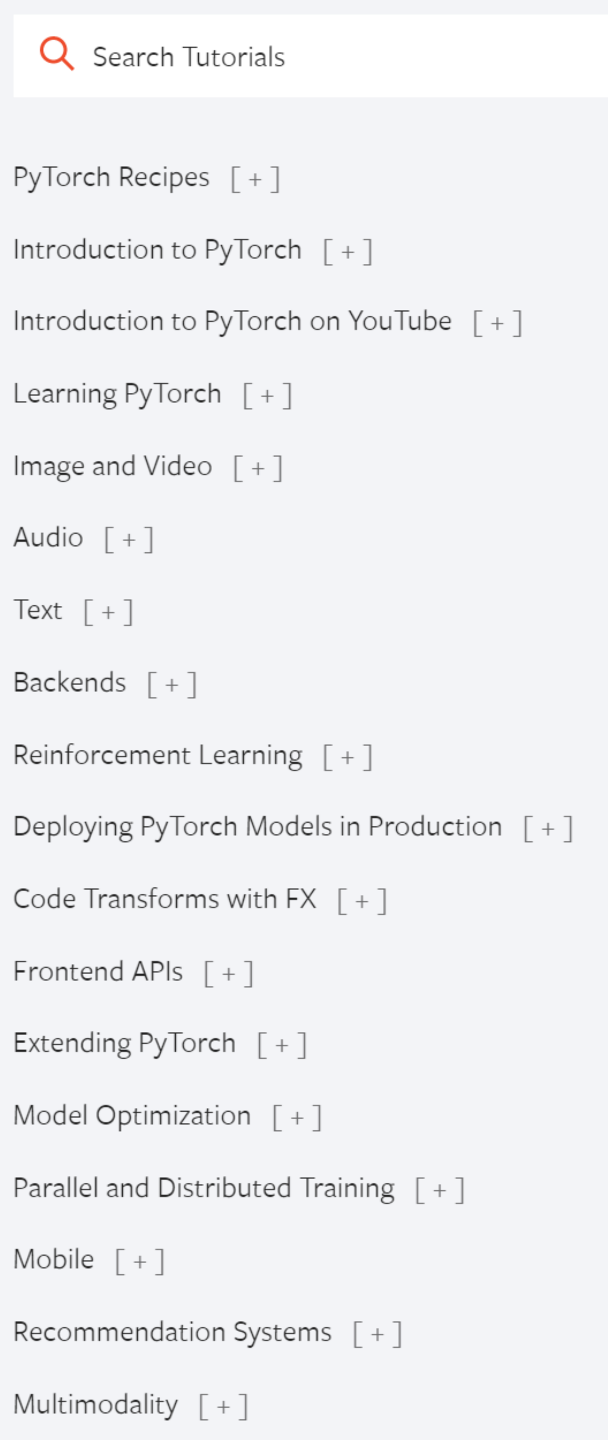
Deep Learning Framework

Quang-Vinh Dinh
Ph.D. in Computer Science

Outline

- Introduction to PyTorch
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Introduction



Create a tensor in PyTorch

```python
# From a list
import torch

data = [1, 2, 3]
data = torch.tensor(data)
print(data)
```

```
tensor([1, 2, 3])
```

```python
# From a NumPy Array
import numpy as np

data = np.array([1, 2, 3])
data = torch.from_numpy(data)
print(data)
```

```
tensor([1, 2, 3], dtype=torch.int32)
```

```python
# random uniform
import torch

# Creates a 3x3 tensor with
# random values between 0 and 1
data = torch.rand(3, 3)
print(data)
```

```
tensor([[0.0615, 0.2716, 0.2160],
        [0.1595, 0.9780, 0.6053],
        [0.7718, 0.1605, 0.4205]])
```

```python
# random normal
import torch

# Creates a 3x3 tensor with values
# sampled from a standard normal distribution
data = torch.randn(3, 3)
print(data)
```

```
tensor([[-1.0891, -0.2750,  0.8105],
        [ 1.0836, -1.1107,  0.3683],
        [ 2.0638, -1.2075, -0.5610]])
```

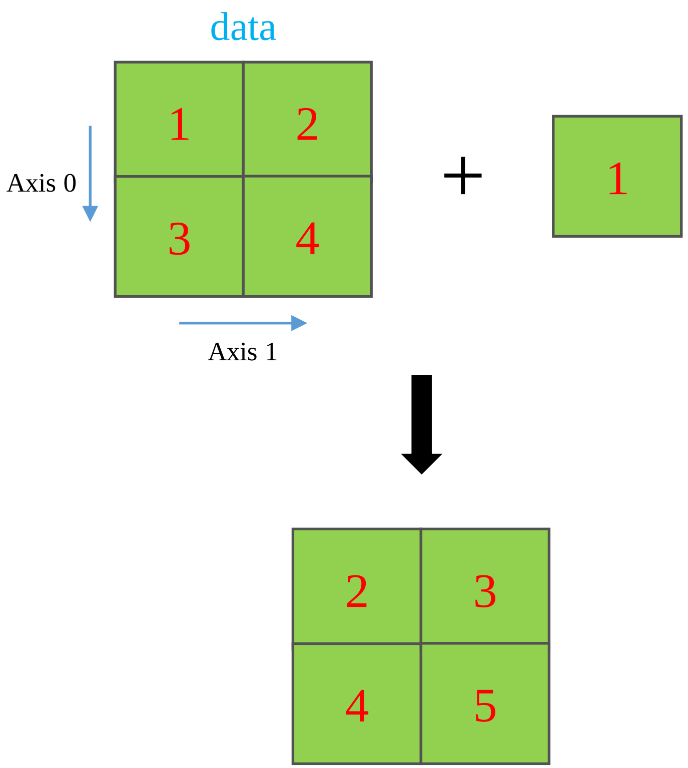
```python
# Using the arange function
import torch

# Creates a tensor [0, 1, 2, 3, 4]
data = torch.arange(start=0, end=5, step=1)
print(data)
```

```
tensor([0, 1, 2, 3, 4])
```

Introduction

❖ Tensor

❖ Broadcasting



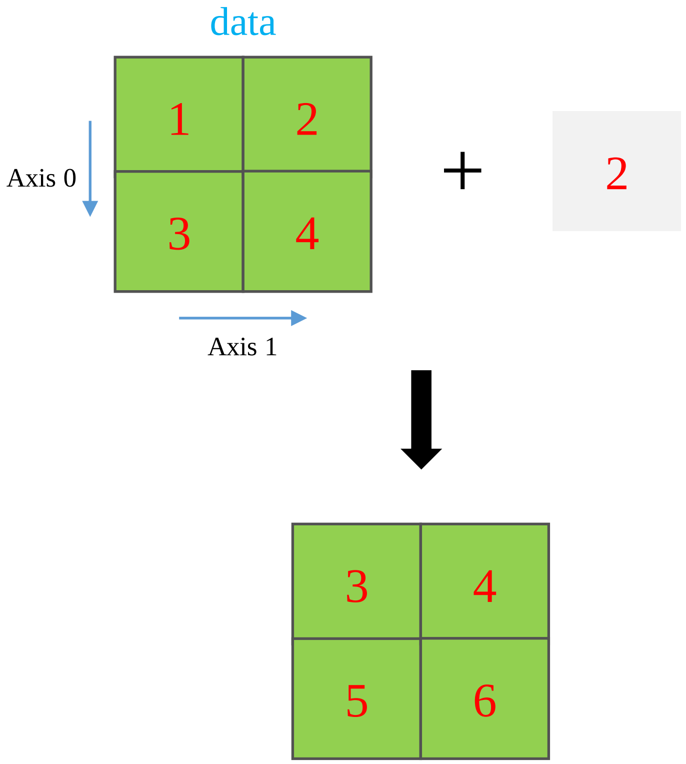

```python
# broadcasting
import numpy as np

# create a tensor
tensor1 = torch.tensor([[1, 2],
                        [3, 4]])
tensor2 = torch.tensor([1])
print(f'Tensor1:\n {tensor1}')
print(f'Tensor2:\n {tensor2}')

# Addition between two tensors
tensor3 = tensor1 + tensor2
print(f'Tensor3:\n {tensor3}')
```

```
Tensor1:
 tensor([[1, 2],
        [3, 4]])
Tensor2:
 tensor([1])
Tensor3:
 tensor([[2, 3],
        [4, 5]])
```

Introduction

❖ Tensor

❖ Broadcasting




```python
# broadcasting
import numpy as np

# create a tensor
tensor1 = torch.tensor([[1, 2],
                        [3, 4]])
print(f'Tensor1:\n {tensor1}')

# Between a tensor and a number
tensor2 = tensor1 + 2
print(f'Tensor2:\n {tensor2}')
```

```
Tensor1:
 tensor([[1, 2],
        [3, 4]])
Tensor2:
 tensor([[3, 4],
        [5, 6]])
```

Introduction

❖ Tensor

❖ Important functions

Squared Difference

$$sd = (x - y)^2$$

x		sd
1		16
2		9
3	$(x - y)^2 =$	4
4		1

```python
# Compute squared difference
import torch

# Create two tensors
x = torch.tensor([1.0, 2.0, 3.0, 4.0])
y = 5

# Compute squared difference
squared_diff = (x - y) ** 2

print(f'x:\n {x}')
print(f'y = {y}')
print(f'squared_diff:\n {squared_diff}')
```

```
x:
 tensor([1., 2., 3., 4.])
y = 5
squared_diff:
 tensor([16.,  9.,  4.,  1.])
```

Introduction

❖ Tensor

❖ Important functions

Mean Squared Error

$$sd = \sum_i (\boldsymbol{x}_i - \boldsymbol{y}_i)^2$$

x	y	sd
1	5	16
2	5	9
3	5	4
4	5	1

$$\Sigma_i(\boldsymbol{x}_i - \boldsymbol{y}_i)^2 = \Sigma_i(sd) = 7.5$$

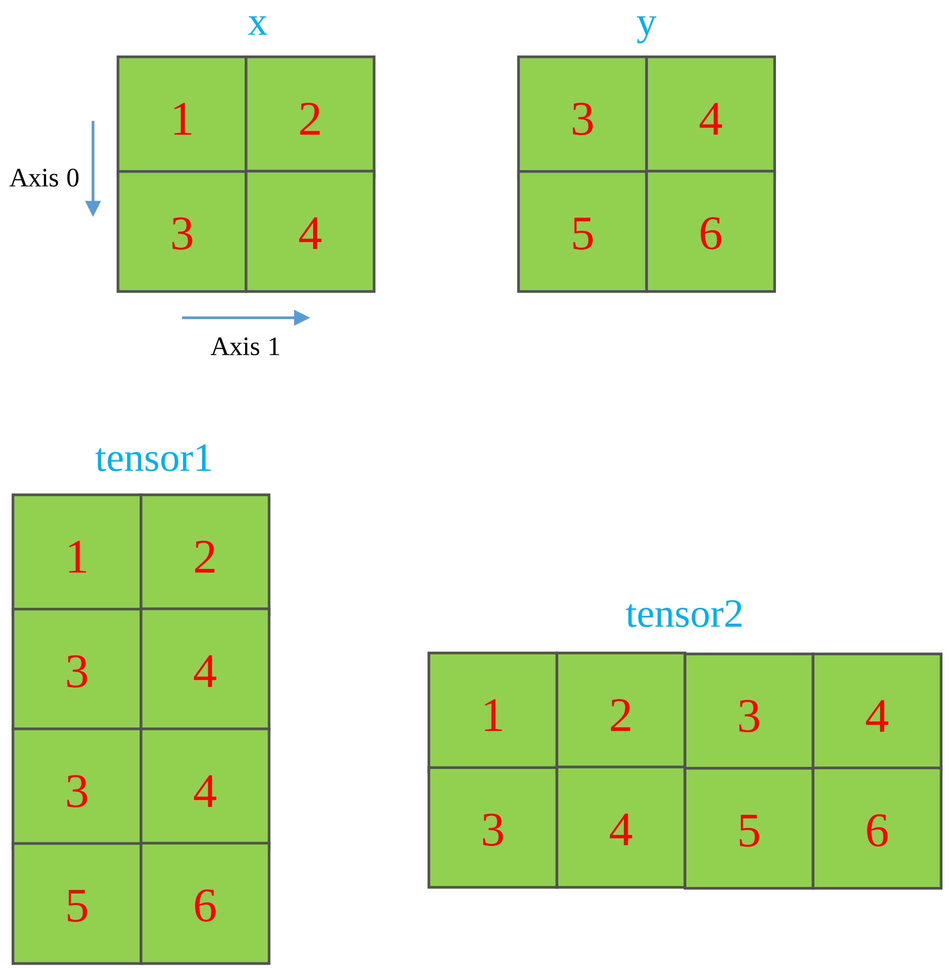
```python
# Compute squared difference
import torch

# Create two tensors
x = torch.tensor([1.0, 2.0, 3.0, 4.0])
y = torch.tensor([5.0, 5.0, 5.0, 5.0])

# Compute squared difference
loss_fn = torch.nn.MSELoss()
mse = loss_fn(x,y)

print(f'x:\n {x}')
print(f'y = {y}')
print(f'mse:\n {mse}')
```

```
x:
 tensor([1., 2., 3., 4.])
y = tensor([5., 5., 5., 5.])
mse:
 7.5
```

Introduction

❖ Tensor

❖ Important functions




```python
# Concatenate tensors
import torch

# Create two tensors
x = torch.tensor([[1, 2],
                  [3, 4]])
y = torch.tensor([[3, 4],
                  [5, 6]])

# Concat tensors along the first dim
tensor1 = torch.cat((x, y), dim=0)
print(f'Tensor1:\n {tensor1}')

# Concat tensors along the second dim
tensor2 = torch.cat((x, y), dim=1)
print(f'Tensor2:\n {tensor2}')
```

```
Tensor1:
 tensor([[1, 2],
        [3, 4],
        [3, 4],
        [5, 6]])
Tensor2:
 tensor([[1, 2, 3, 4],
        [3, 4, 5, 6]])
```

Introduction

❖ Tensor

❖ Important functions

$$\begin{bmatrix} 3 & 4 \\ 4 & 2 \\ 1 & 1 \end{bmatrix}.\text{argmax}(\text{axis}=0) = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 \\ 4 & 2 \\ 1 & 1 \end{bmatrix}.\text{argmax}(\text{axis}=1) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

```python
# argmax
import torch

# Creates a 3x2 tensor
data = torch.randint(low=0, high=9, size=(3, 2))
print(f'data:\n {data}')

# Compute argmax across the rows (dimension 0)
argmax_dim0 = torch.argmax(data, dim=0)
print(f'argmax1:\n {argmax_dim0}')

# Compute argmax across the columns (dimension 1)
argmax_dim1 = torch.argmax(data, dim=1)
print(f'argmax1:\n {argmax_dim1}')
```

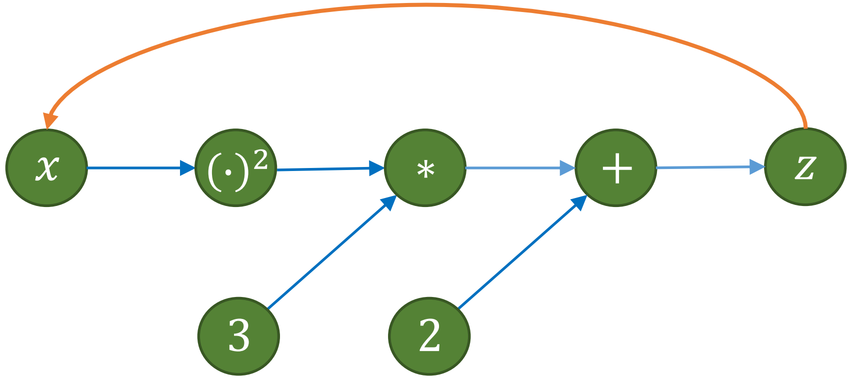
```
data:
 tensor([[3, 4],
        [4, 2],
        [1, 1]])
argmax1:
 tensor([1, 0])
argmax1:
 tensor([1, 0, 0])
```

Introduction

❖ Gradient computation

$$y = x^2$$

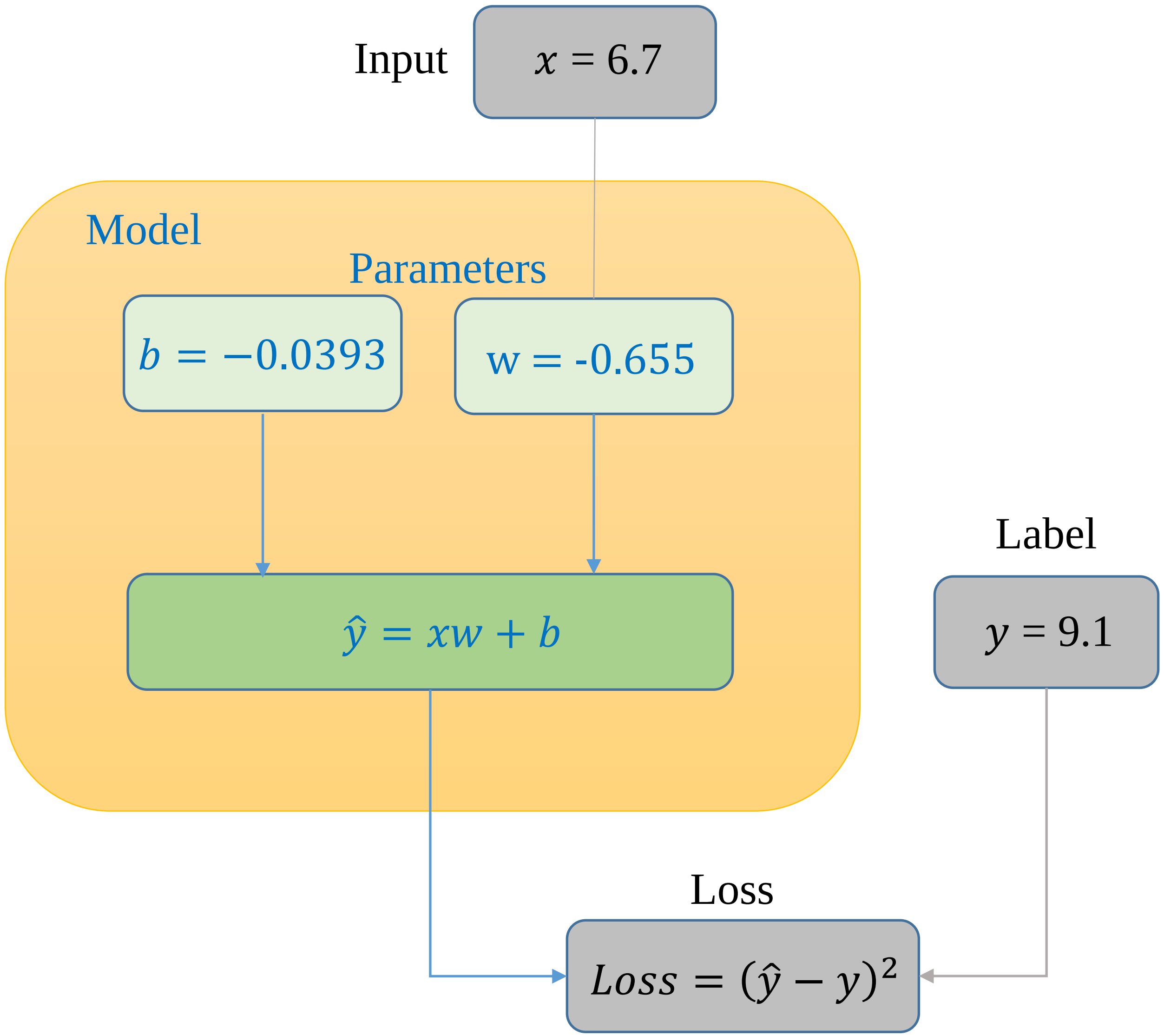
$$z = 3y + 2$$

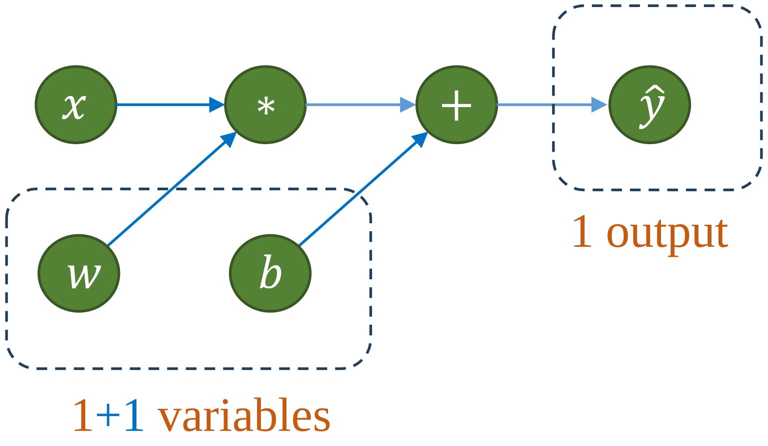



```python
# autograd
import torch

# Create a tensor to compute gradients
x = torch.tensor(2.0, requires_grad=True)

# operation
y = x ** 2
z = 3*y + 2
print(f'z: {z}')

# Backpropagate to compute gradients
z.backward()

# Print the gradient. dz/dx at x=2.0
print(f"Gradient of z w.r.t. x: {x.grad}")
```

```
z: 14.0
Gradient of z with respect to x: 12.0
```

Example 1

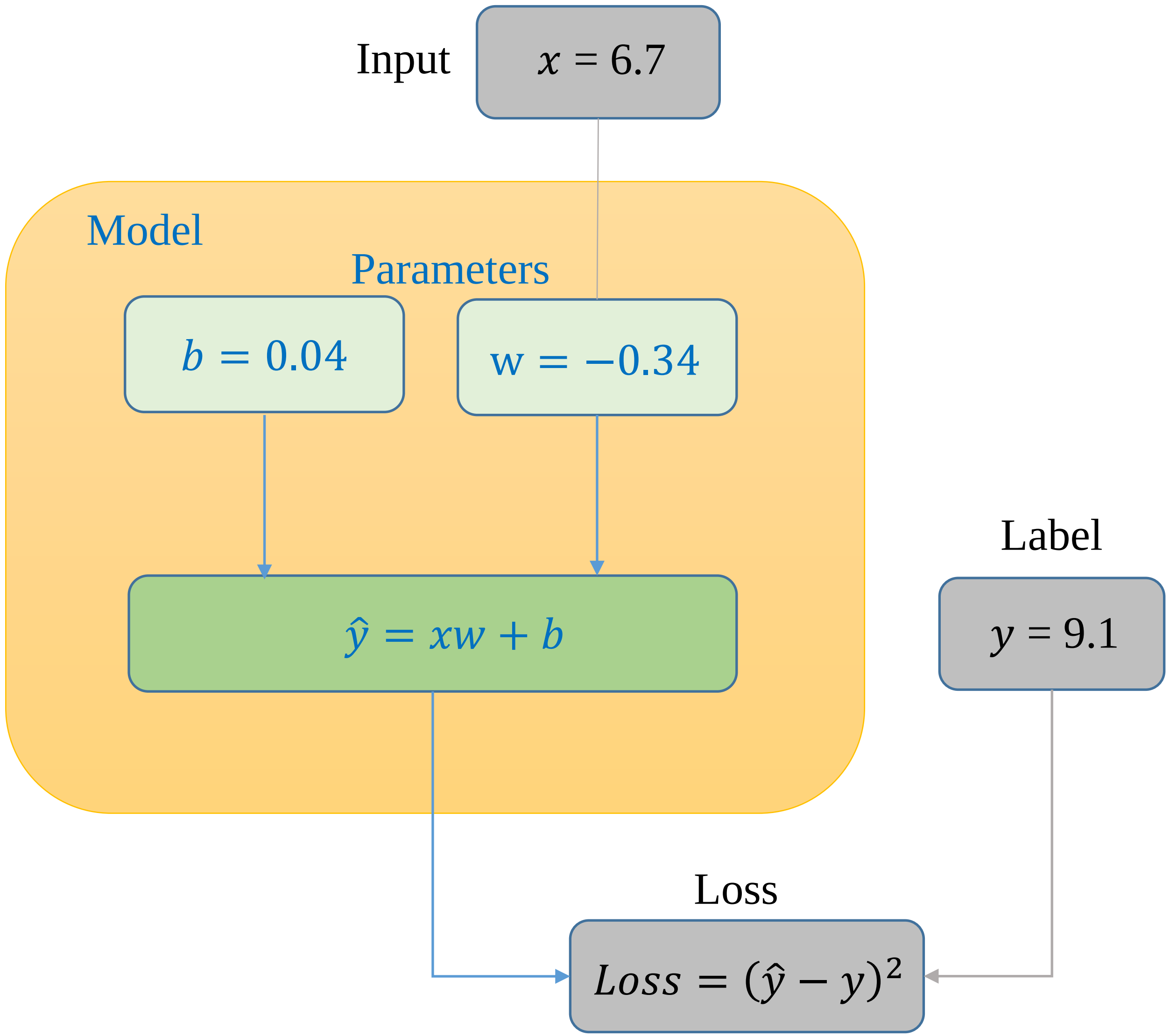


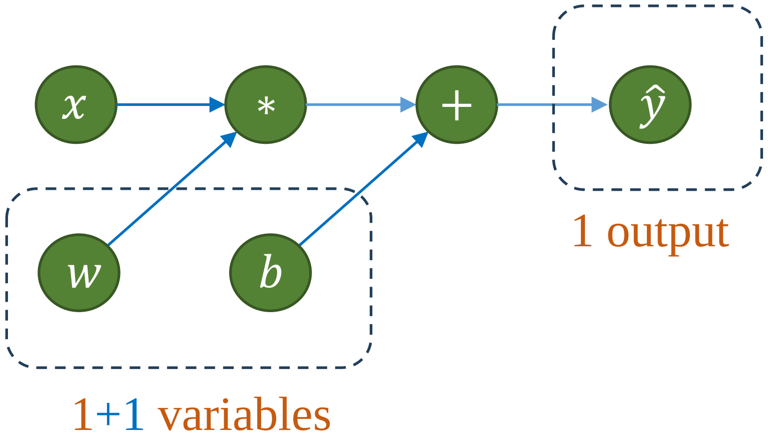



```python
import torch.nn as nn

# Create a linear layer
linear = nn.Linear(1, 1)

print(f'b - {linear.bias}')
print(f'w - {linear.weight}')
```

```
b - Parameter containing:
tensor([-0.0393], requires_grad=True)
w - Parameter containing:
tensor([[-0.6550]], requires_grad=True)
```

Example 1



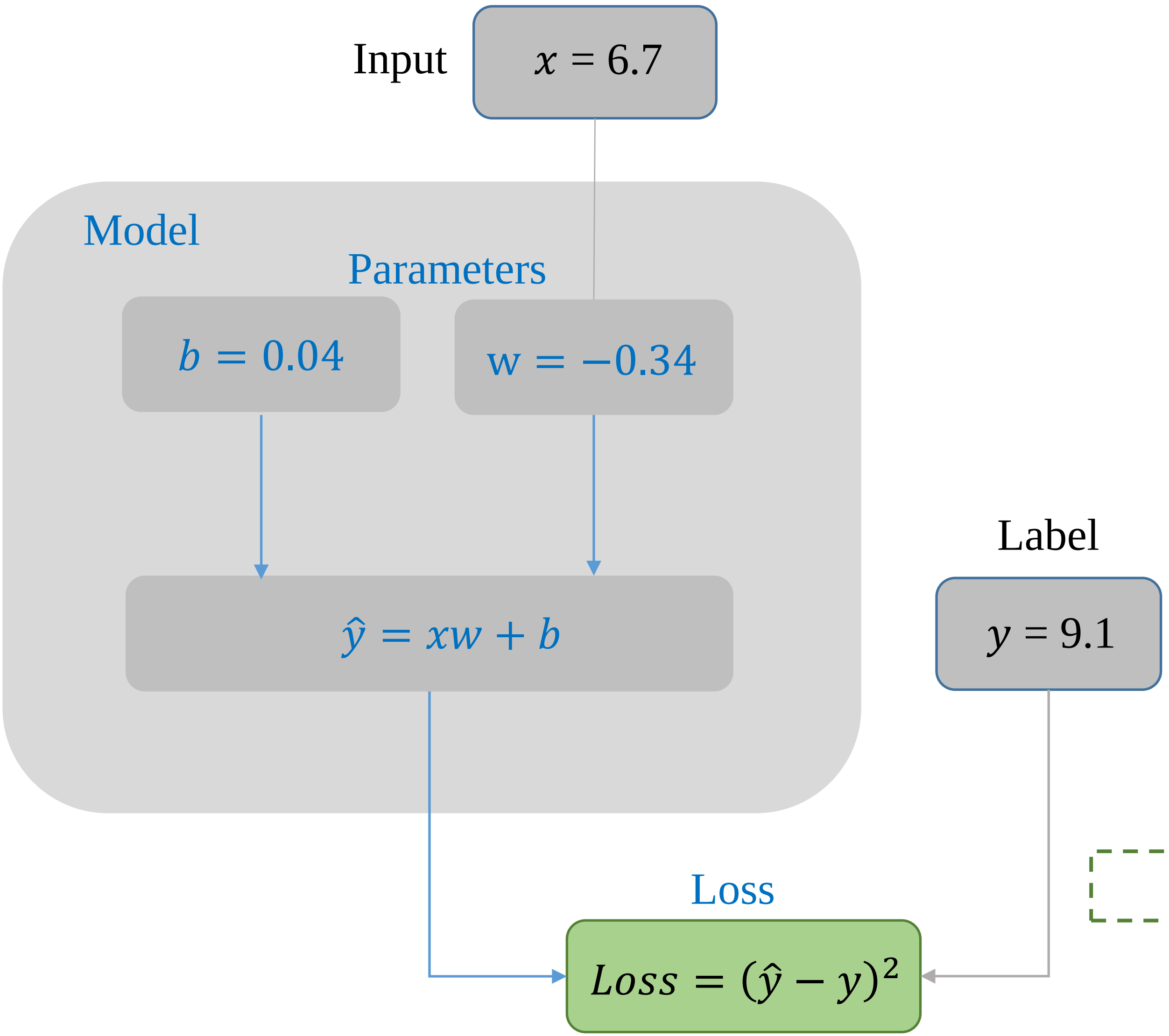


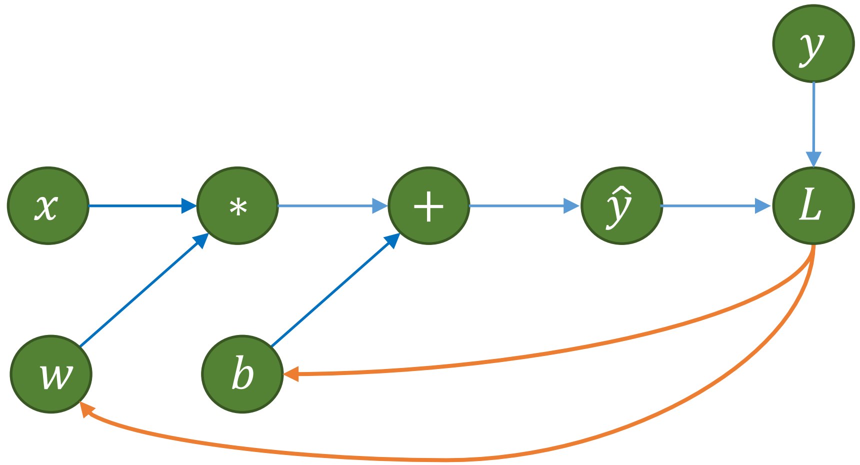

```python
import torch.nn as nn
import torch

# Create a linear layer
linear = nn.Linear(1, 1)

# set values
linear.bias.data = torch.Tensor([0.04])
linear.weight.data = torch.Tensor([[-0.34]])

print(f'b - {linear.bias}')
print(f'w - {linear.weight}')
```

```
b - Parameter containing:
tensor([0.0400], requires_grad=True)
w - Parameter containing:
tensor([[-0.3400]], requires_grad=True)
```

Example 1

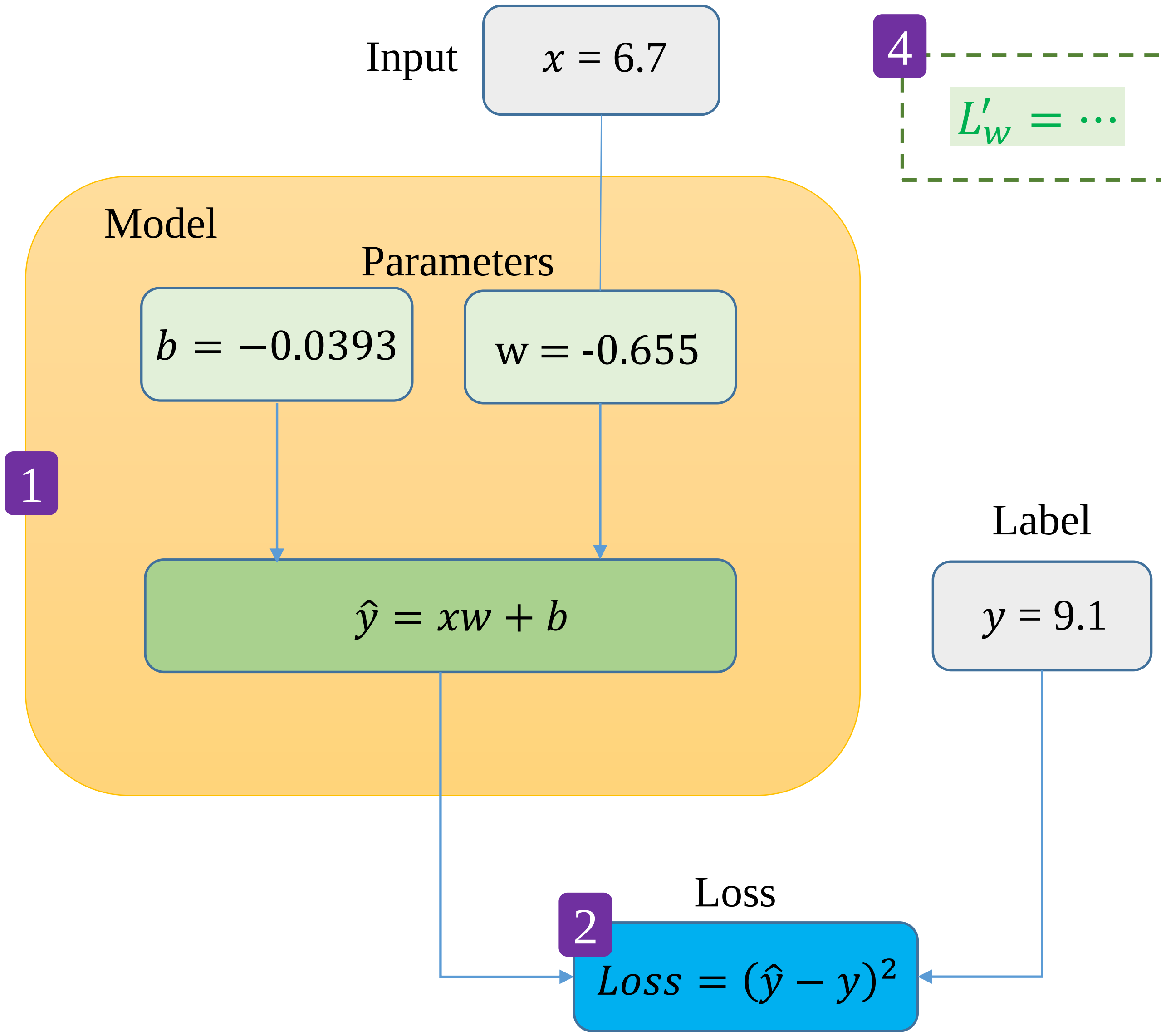
L'_w L'_b






```python
import torch.nn as nn
import torch

# Create a linear layer
linear = nn.Linear(1, 1)

# set values
linear.bias.data = torch.Tensor([0.04])
linear.weight.data = torch.Tensor([[-0.34]])

# loss function and optimizer
loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.SGD(linear.parameters(),
                            lr=0.01)
```

Example 1

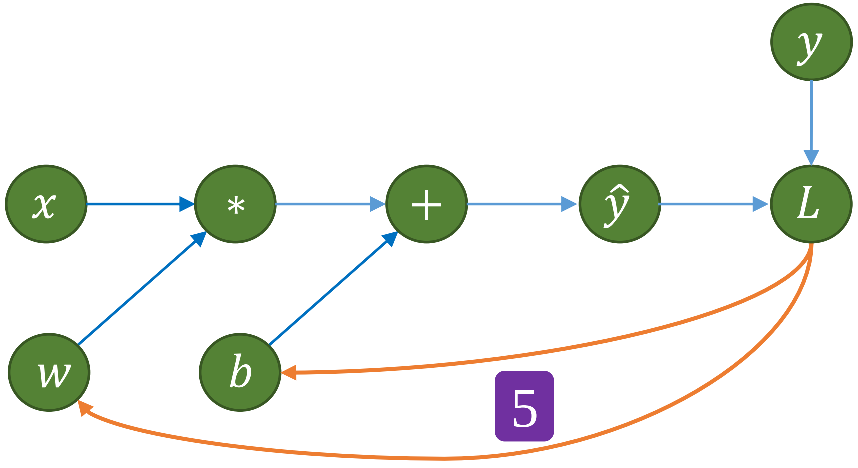


3

$L'_w = 0$ $L'_b = 0$

4

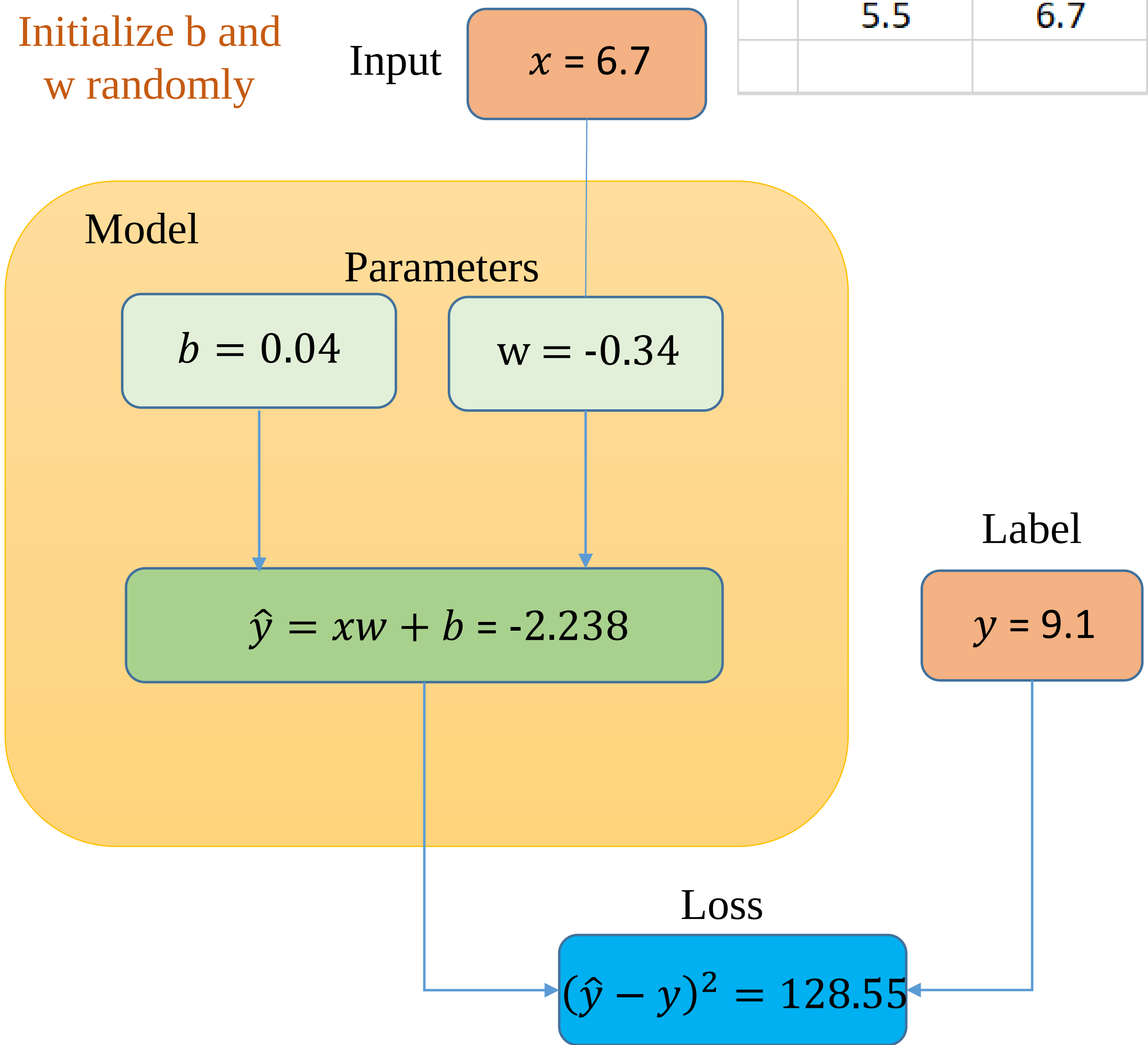
$L'_w = \cdots$ $L'_b = \cdots$




```
# y_hat
y_hat = linear(x)          1

# loss
loss = loss_fn(y_hat, y)   2

# compute gradient
optimizer.zero_grad()      3
loss.backward()            4

# update
optimizer.step()           5
```

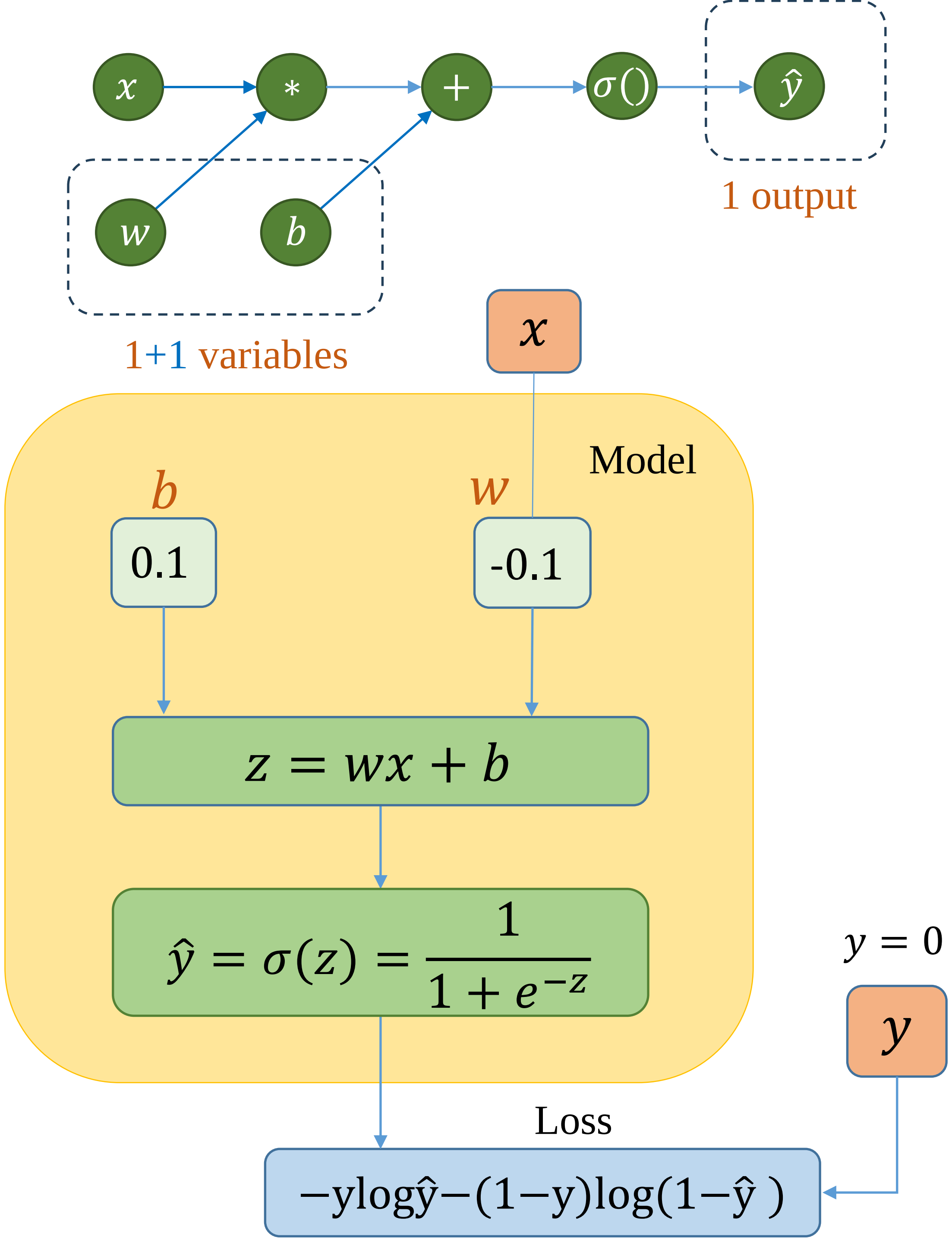
Example 1

area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7




```python
import numpy as np
import torch
import torch.nn as nn

### Data preparation
data = np.genfromtxt('data.csv', delimiter=',')
x_data = torch.from_numpy(data[:, 0:1]).float()
y_data = torch.from_numpy(data[:, 1:]).float()

# Create model, loss and optimizer
linear = nn.Linear(1, 1)
loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.SGD(linear.parameters(),
                            lr=0.01)

# training
epochs = 100
for epoch in range(epochs):
    y_hat = linear(x_data)
    loss = loss_fn(y_hat, y_data)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Logistic Regression

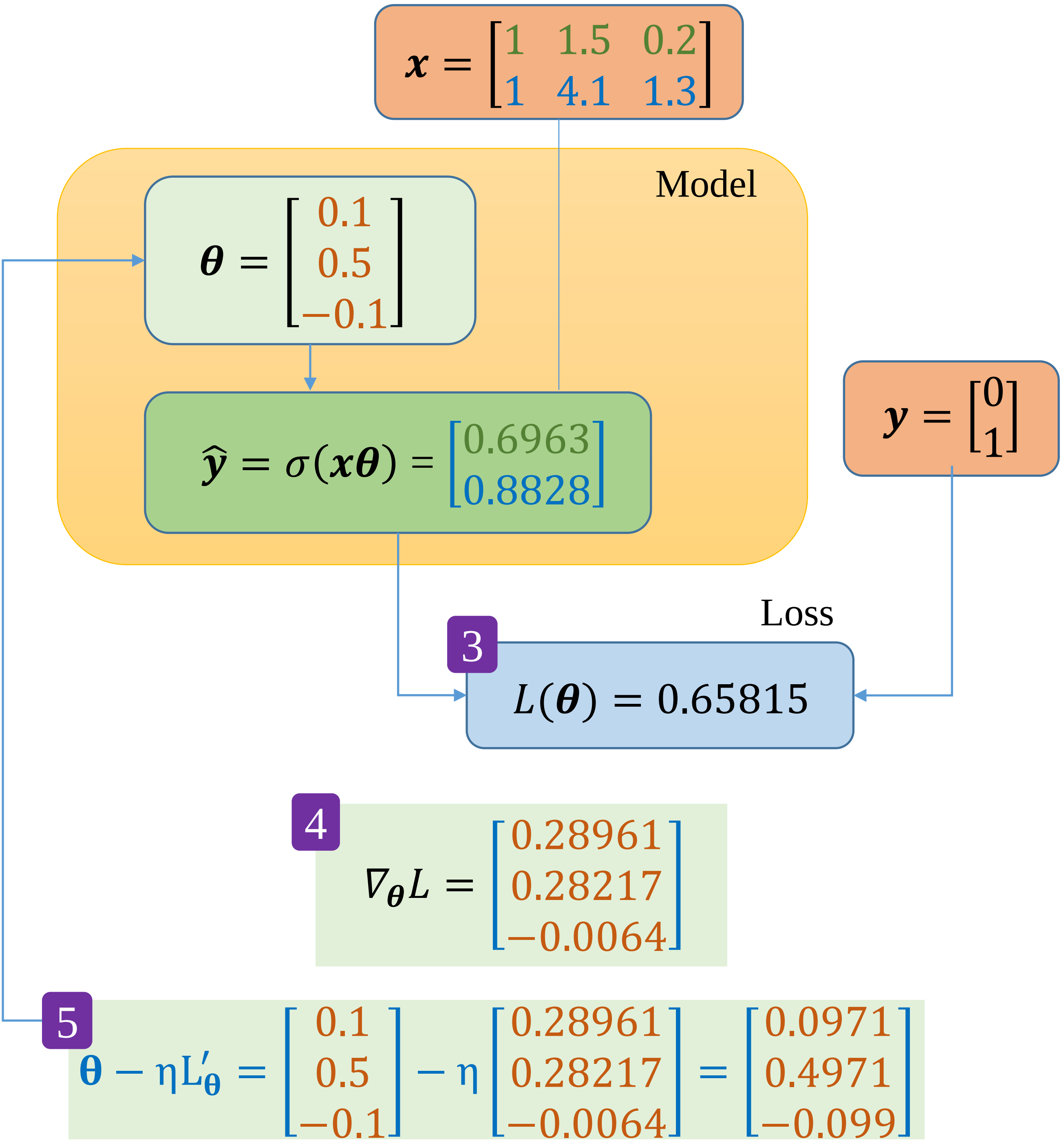



```python
data = np.genfromtxt('iris_1D.csv',
                     delimiter=',', skip_header=1)
X = torch.from_numpy(data[:,0:1]).float()
y = torch.from_numpy(data[:,1:]).float()
```

```python
# Create a linear layer
linear = nn.Linear(1, 1)

# loss and optimizer
loss_fn = torch.nn.BCELoss()
optimizer = torch.optim.SGD(linear.parameters(),
                            lr=0.01)
```

```python
# training
for epoch in range(epochs):
    y_hat = torch.sigmoid(linear(X))
    loss = loss_fn(y_hat, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Logistic Regression




```python
data = np.genfromtxt('iris_2D_demo.csv',
                     delimiter=',', skip_header=1)
X = torch.from_numpy(data[:,0:2]).float()
y = torch.from_numpy(data[:,2:]).float()
```

```python
# Create a linear layer
linear = nn.Linear(2, 1)

# loss and optimizer
loss_fn = torch.nn.BCELoss()
optimizer = torch.optim.SGD(linear.parameters(),
                            lr=0.01)
```

```python
# training
for epoch in range(epochs):
    y_hat = torch.sigmoid(linear(X))
    loss = loss_fn(y_hat, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

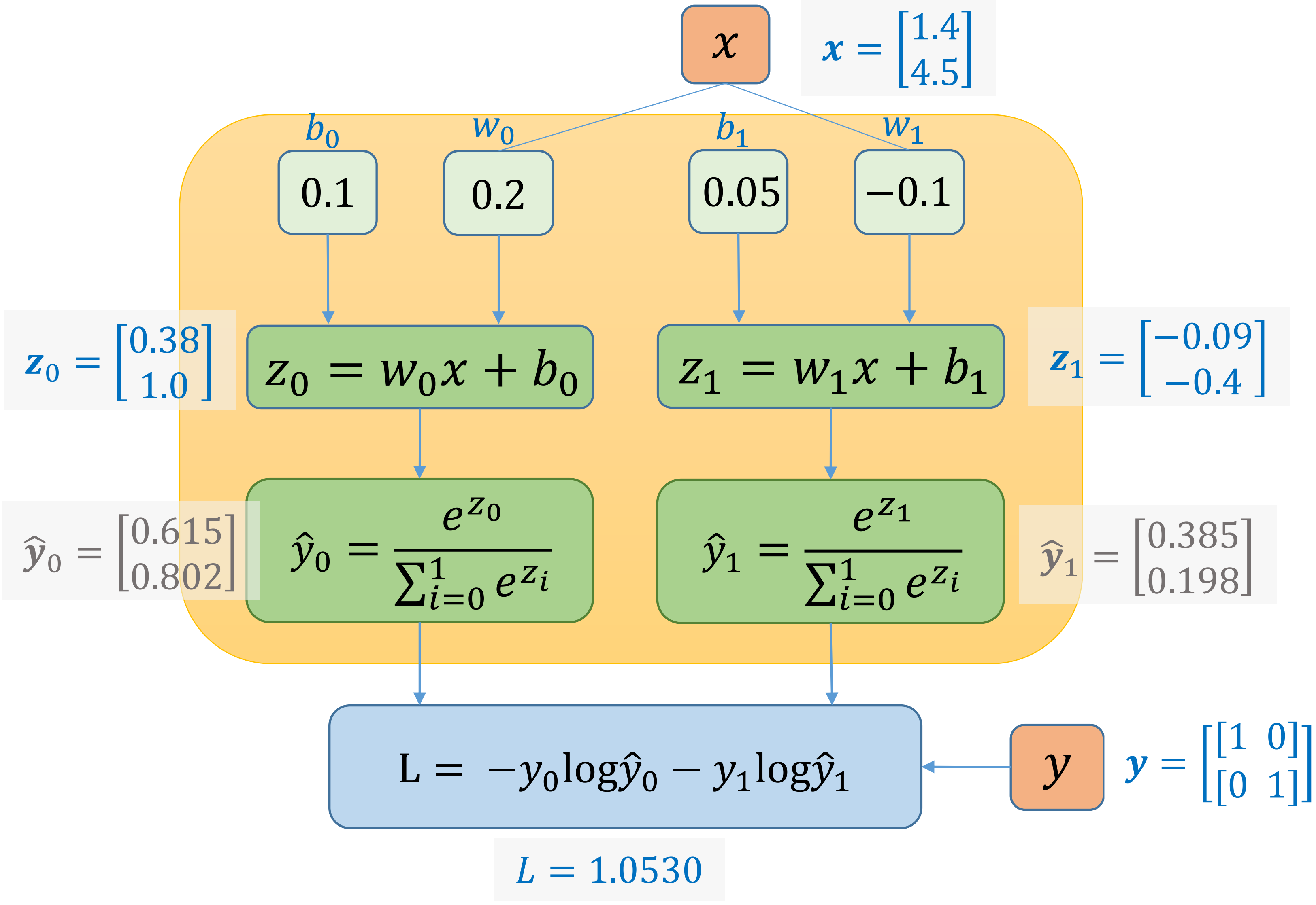
Loss Functions

❖ Cross-entropy

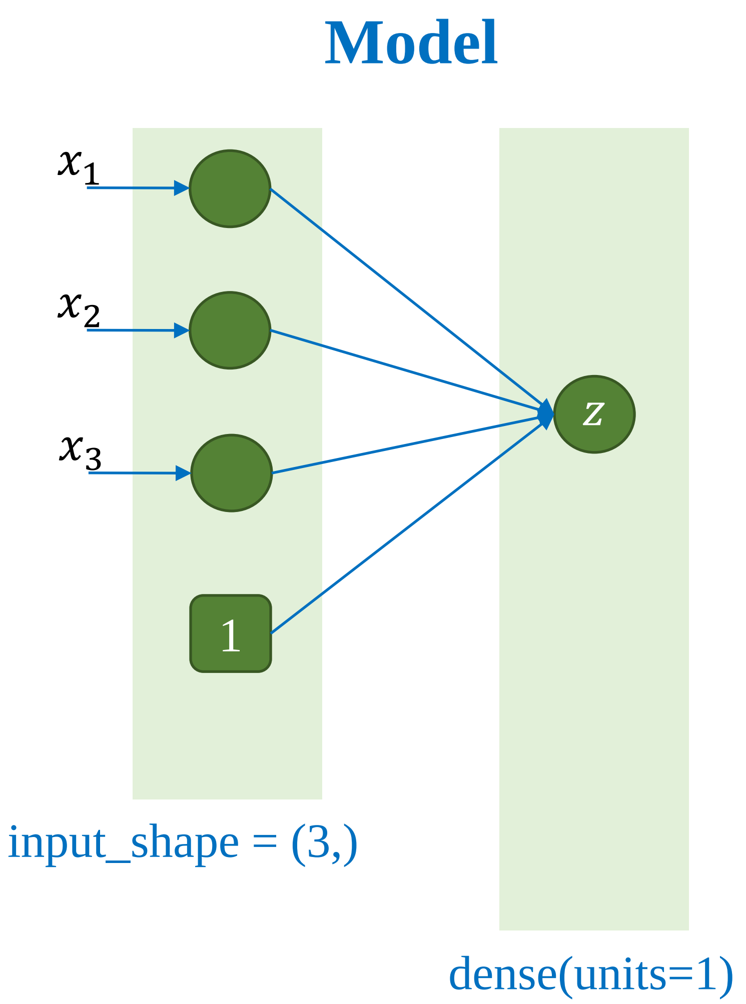
$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_i -y^i \log(\hat{y}^i)$$

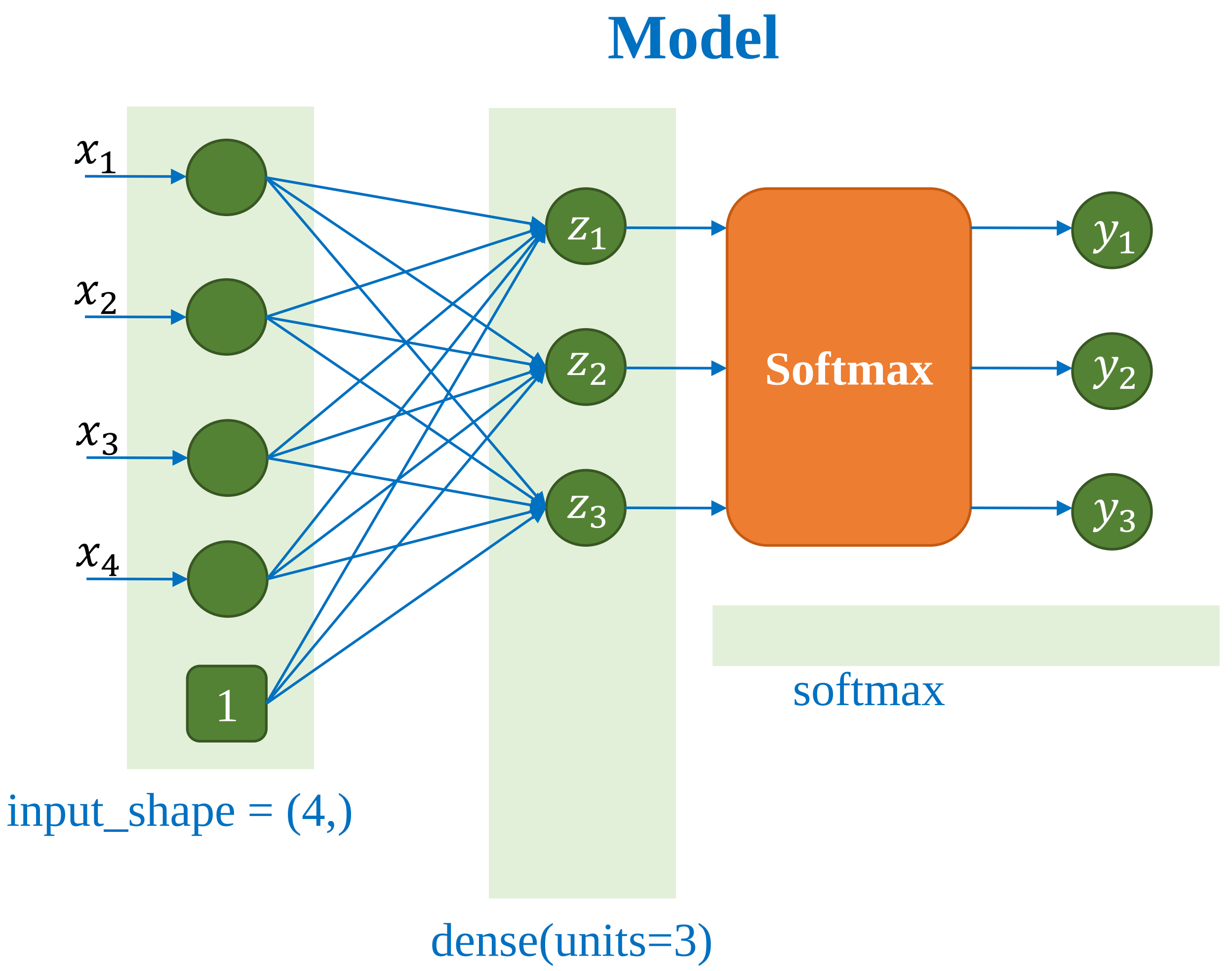
```
import torch.nn as nn
import torch

y_true = torch.Tensor([0, 1]).long()
logit_pred = torch.Tensor([[0.38, -0.09],
                           [1.0, -0.4]])

criterion = nn.CrossEntropyLoss()
loss = criterion(logit_pred, y_true)
print(loss)
```

```
tensor(1.0530)
```



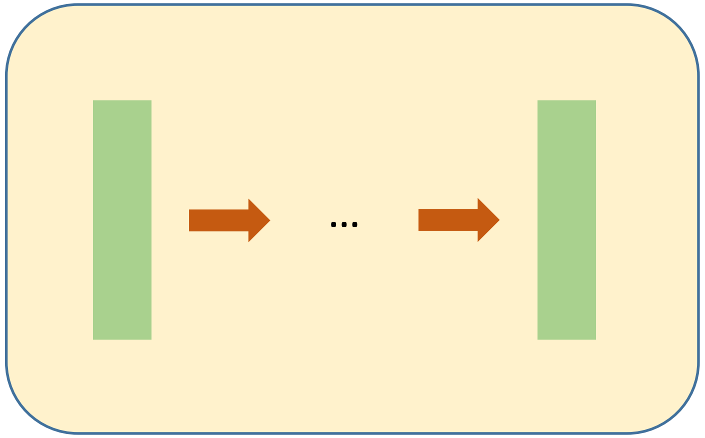
nn.Sequential()



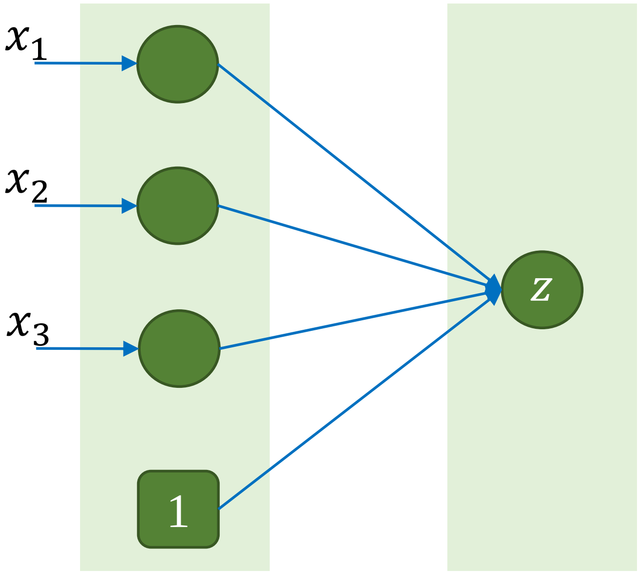


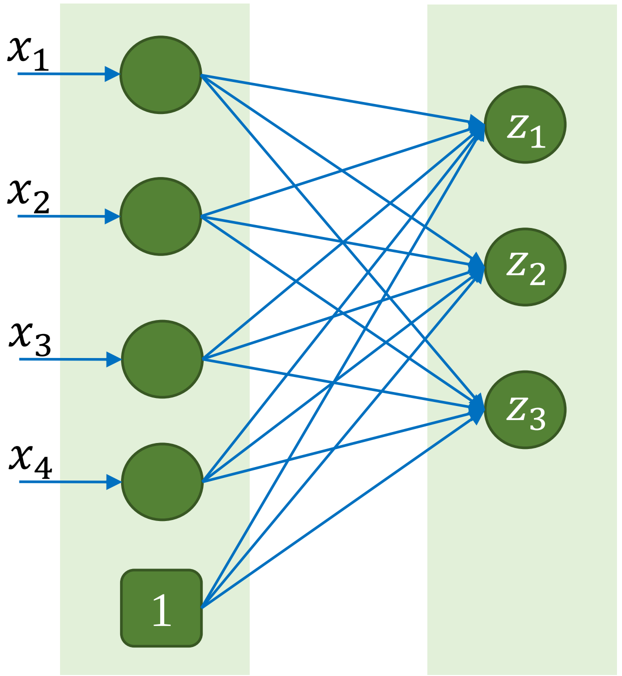
nn.Sequential()

Model



nn.Sequential()






```python
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=1)
)
summary(model, input_size=(3,))
```

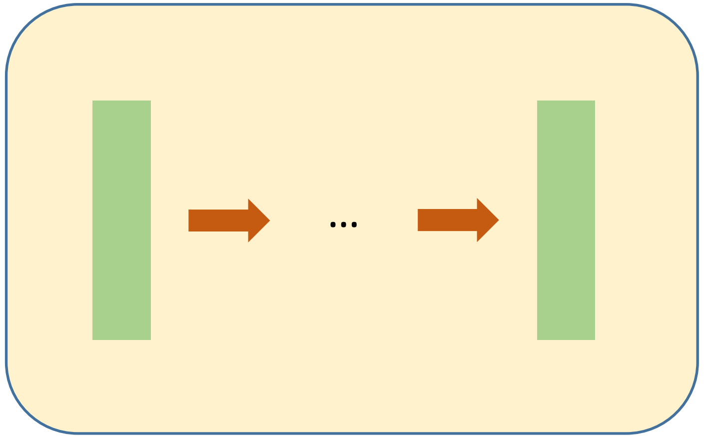
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]               4
================================================================
```

```python
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=4, out_features=3)
)
summary(model, input_size=(4,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 3]              15
================================================================
```

nn.Sequential()

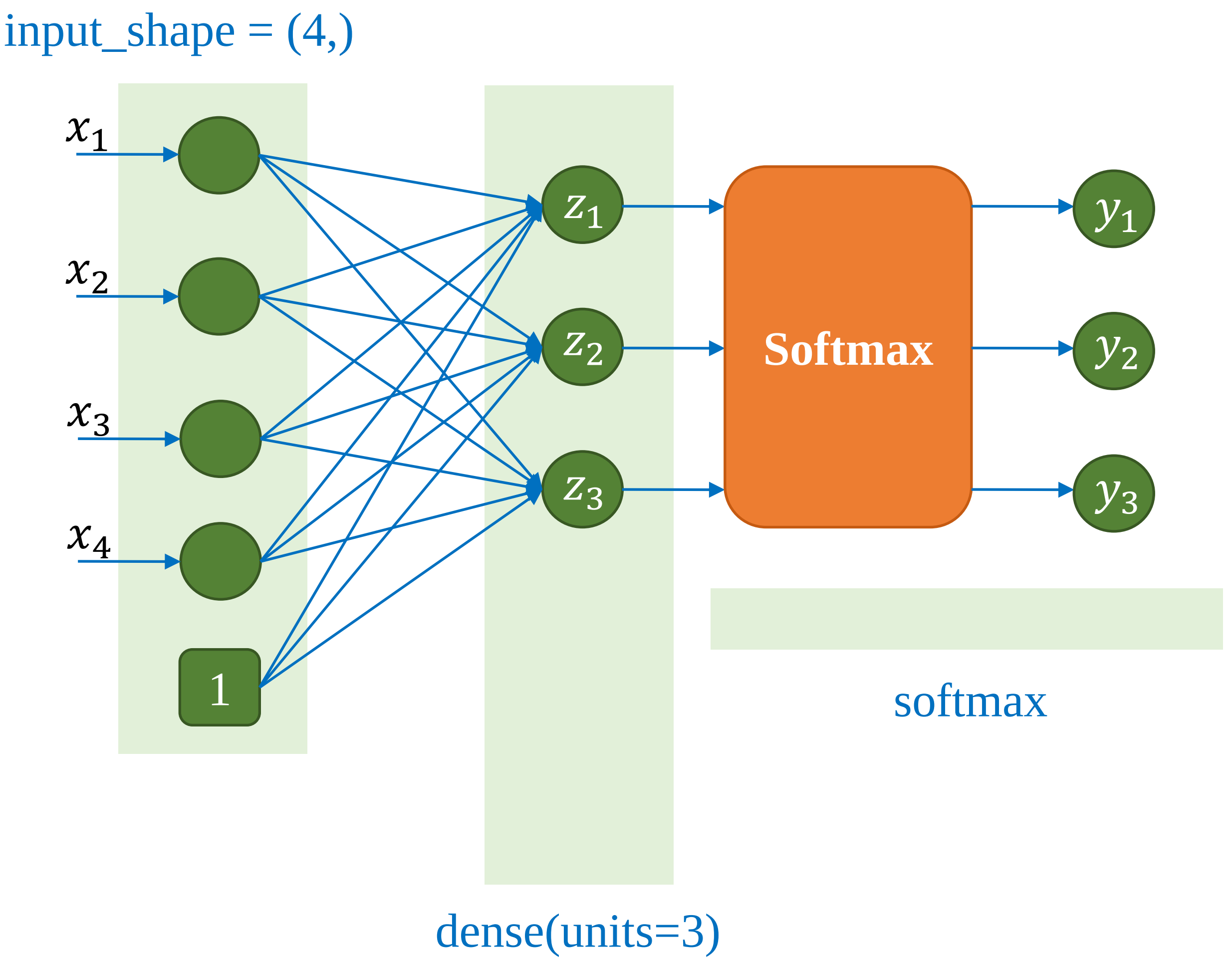
Model



nn.Sequential()

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=4, out_features=3)
    # don't include a softmax layer
)
summary(model, input_size=(4,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 3]              15
================================================================
```



Outline

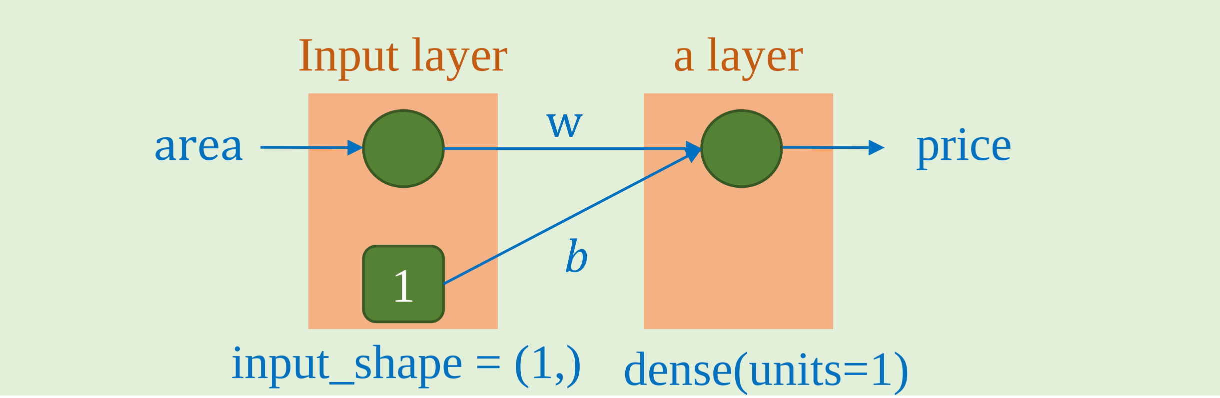
- Introduction to PyTorch
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Model Construction

❖ Linear regression

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data




```python
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=1, out_features=1)
)
summary(model, input_size=(1,))
```

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

Model

$$\text{price} = w * area + b$$

$$\hat{y} = wx + b$$

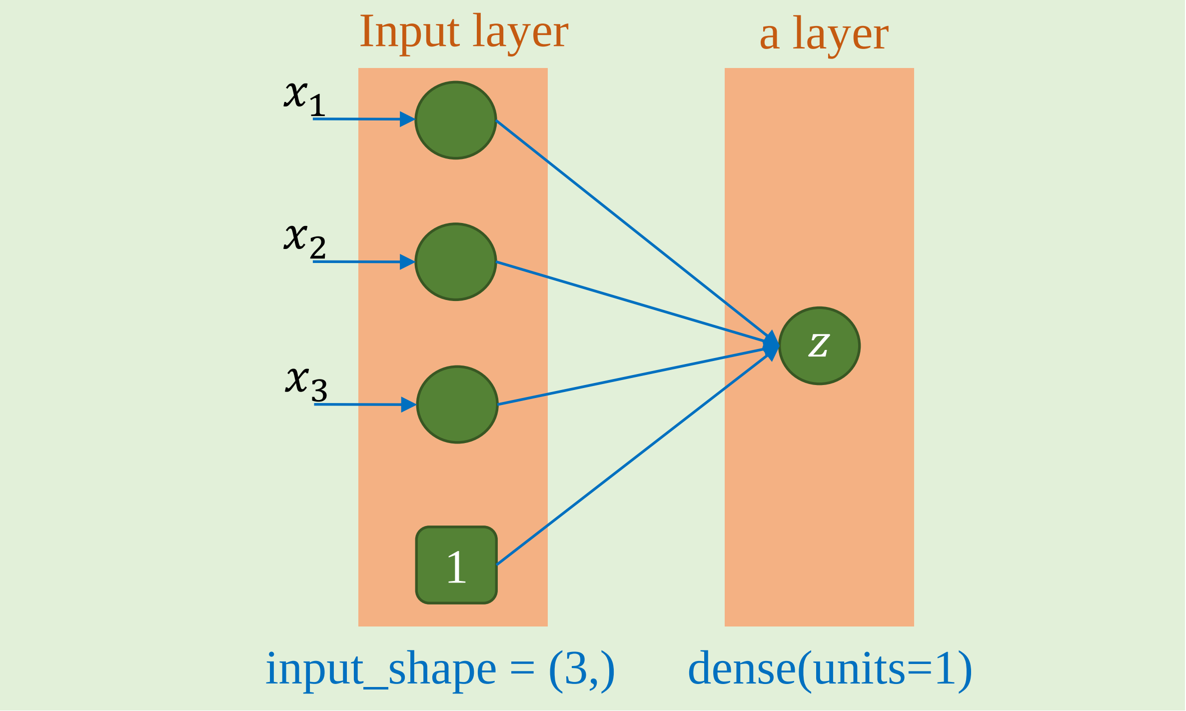
Model Construction

❖ Linear regression

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Features: TV, Radio, Newspaper — Label: Sales

Advertising-based sale data



Model

$$Sale = w_1 * TV + w_2 * Radio + w_3 * Newspaper + b$$

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b$$

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=3, out_features=1)
)
summary(model, input_size=(3,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]               4
================================================================
```

Model Construction

❖ Linear regression

Boston House Price Data

Features | Label

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

Model

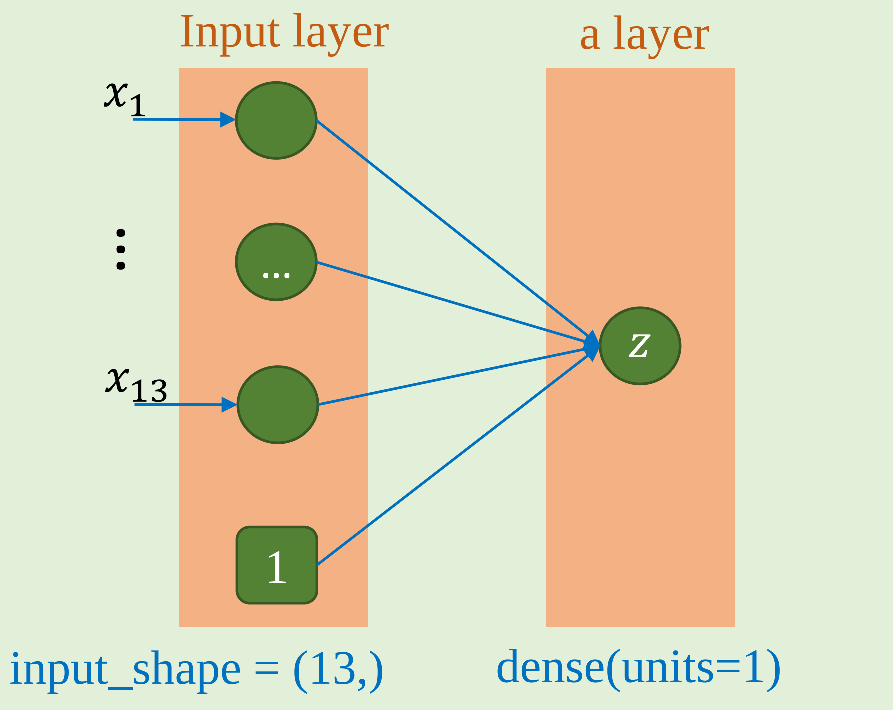
$$\text{medv} = w_1 * x_1 + \cdots + w_{13} * x_{13} + b$$

Model Construction

❖ Linear regression

Model

$$\text{medv} = w_1 * x_1 + \cdots + w_{13} * x_{13} + b$$




```python
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=13, out_features=1)
)
summary(model, input_size=(13,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]              14
================================================================
Total params: 14
Trainable params: 14
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
----------------------------------------------------------------
```

AI VIETNAM
All-in-One Course

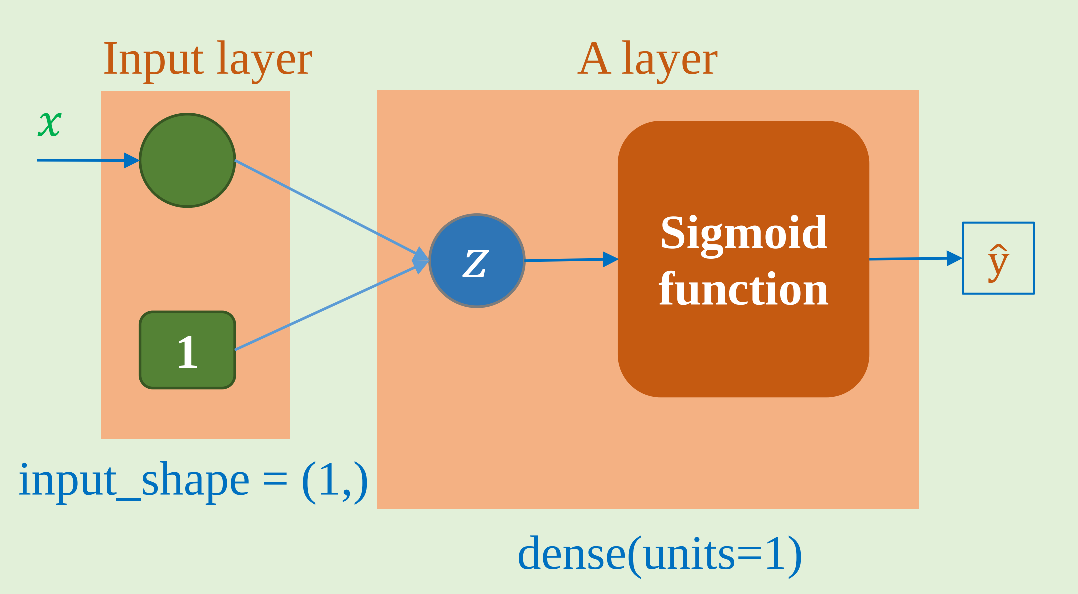
Model Construction

Feature	Label
Petal_Length	Category
1.4	0
1	0
1.5	0
3	1
3.8	1
4.1	1

Model

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

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



❖ Logistic regression

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=1, out_features=1)
    # don't include a sigmoid layer
)
summary(model, input_size=(1,))
```

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

criterion = nn.BCEWithLogitsLoss()

Model Construction

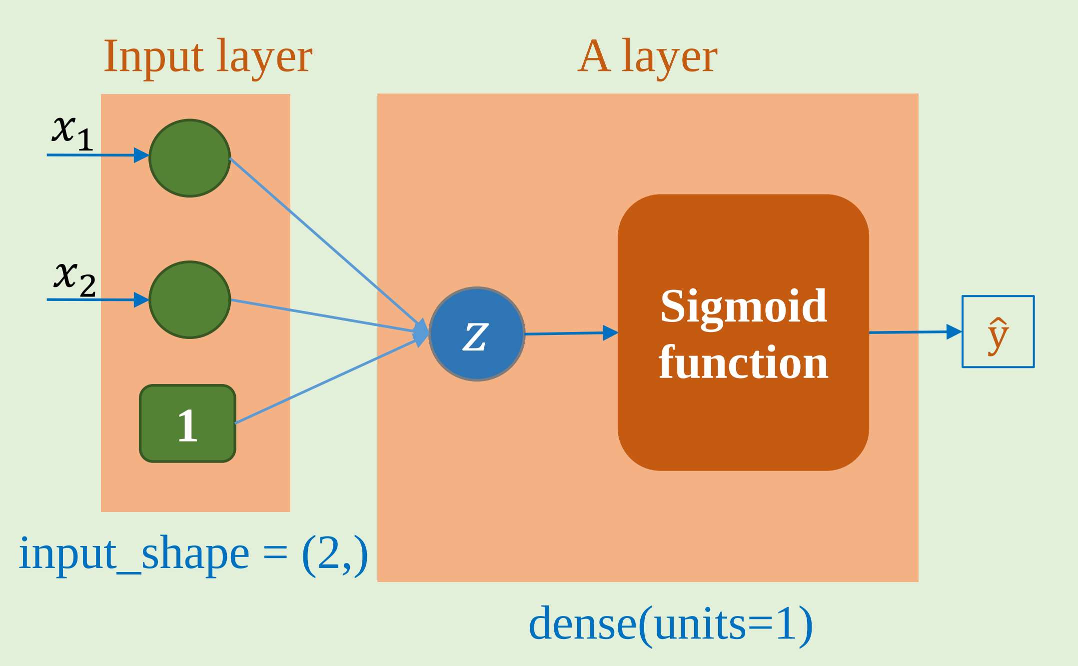
Petal_Length	Petal_Width	Label
1.5	0.2	0
1.4	0.2	0
1.6	0.2	0
4.7	1.6	1
3.3	1.1	1
4.6	1.3	1

Feature / Label

Model

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

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



❖ Logistic regression

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=2, out_features=1)
    # don't include a sigmoid layer
)
summary(model, input_size=(2,))
```

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

criterion = nn.BCEWithLogitsLoss()

Model Construction

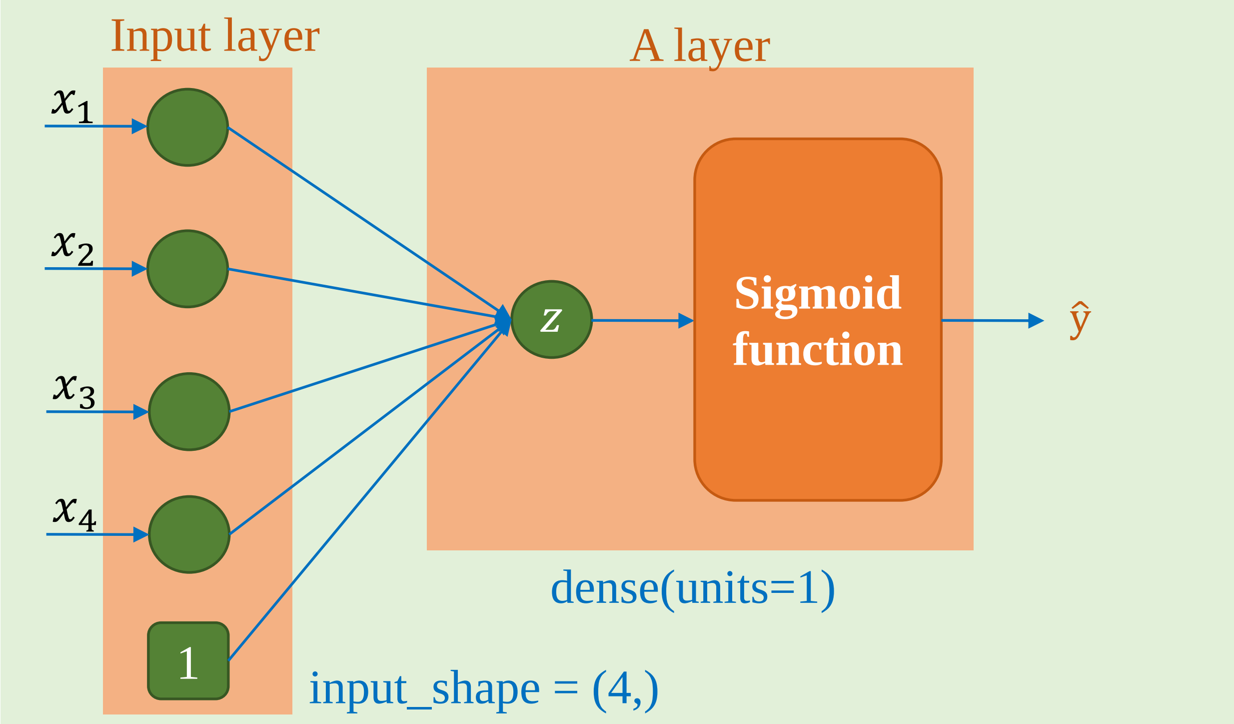
Feature | **Label**

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Label
5.2	3.5	1.5	0.2	0
5.2	3.4	1.4	0.2	0
4.7	3.2	1.6	0.2	0
6.3	3.3	4.7	1.6	1
4.9	2.4	3.3	1.1	1
6.6	2.9	4.6	1.3	1

Model

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

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



❖ Logistic regression

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=4, out_features=1)
    # don't include a sigmoid layer
)
summary(model, input_size=(4,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]               5
================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
----------------------------------------------------------------
```

criterion = nn.BCEWithLogitsLoss()

Model Construction

Feature **Label**

Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2

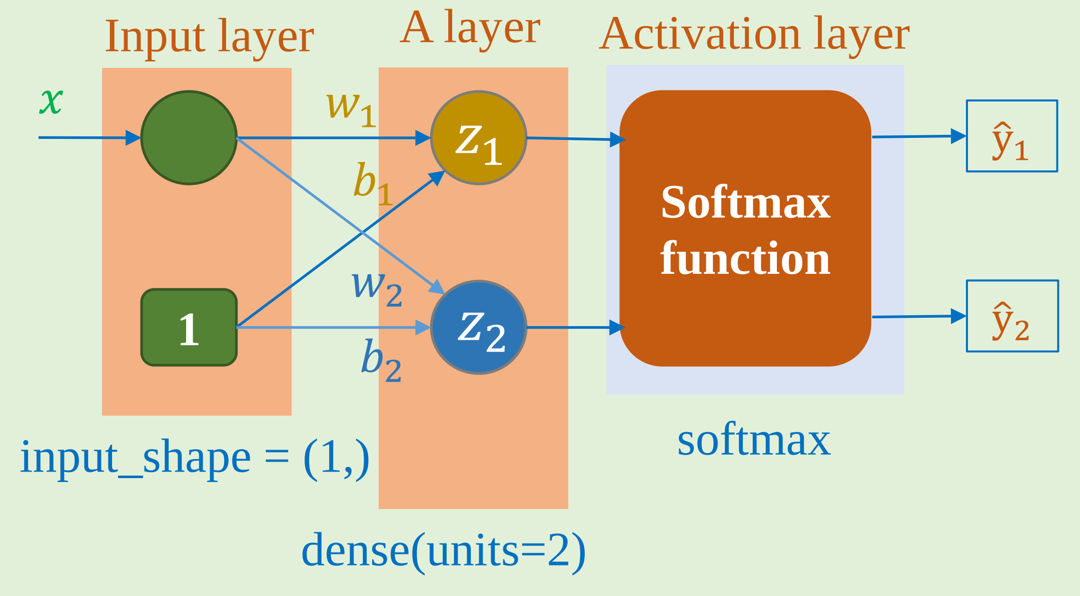
Iris Classification Data

$$z_1 = xw_1 + b_1$$

$$z_2 = xw_2 + b_2$$

$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$

$$\hat{y}_2 = \frac{e^{z_1}}{\sum_{j=1}^{2} e^{z_j}}$$



❖ Softmax regression

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=1, out_features=2)
    # don't include a softmax layer
)
summary(model, input_size=(1,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 2]               4
================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
----------------------------------------------------------------
```

criterion = nn.CrossEntropyLoss()

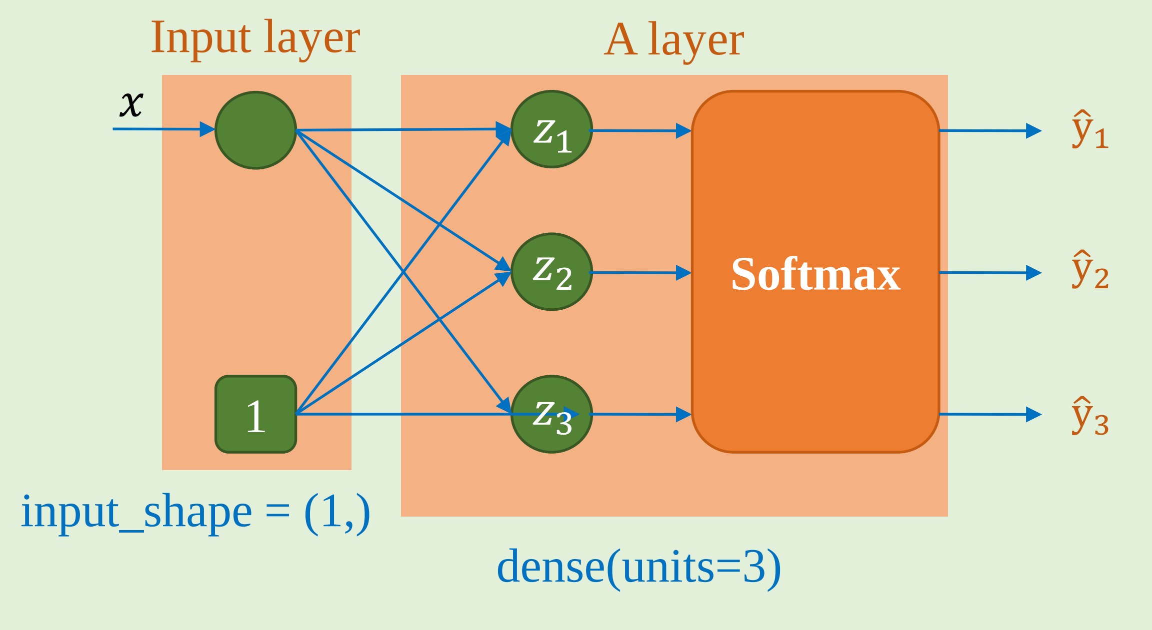
Model Construction

❖ Softmax regression

Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2
5.2	3
5.6	3
5.9	3

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=1, out_features=3)
    # don't include a softmax layer
)
summary(model, input_size=(1,))
```

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



criterion = nn.CrossEntropyLoss()

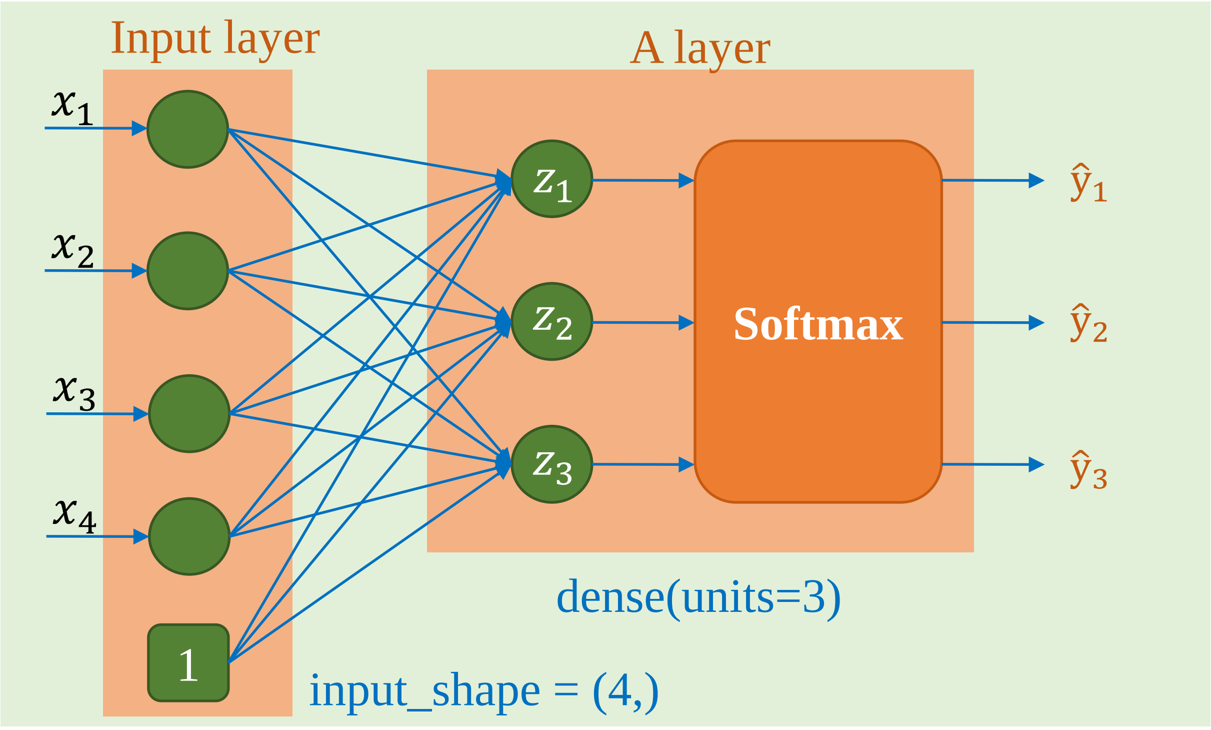
Model Construction

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Label
5.2	3.5	1.5	0.2	1
5.2	3.4	1.4	0.2	1
4.7	3.2	1.6	0.2	1
6.3	3.3	4.7	1.6	2
4.9	2.4	3.3	1.1	2
6.6	2.9	4.6	1.3	2
6.4	2.8	5.6	2.2	3
6.3	2.8	5.1	1.5	3
6.1	2.6	5.6	1.4	3

❖ Softmax regression

Forward computation

$$z = \theta^T x \qquad \hat{y} = \frac{e^{z}}{\sum_{i=1}^{k} e^{z_i}}$$




```python
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(in_features=4, out_features=3)
    # don't include a softmax layer
)
summary(model, input_size=(4,))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 3]              15
================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
----------------------------------------------------------------
```

Outline

- Introduction to PyTorch
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Training

❖ Logistic regression

→ Tính output $\hat{y}$

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

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

→ Tính loss (binary cross-entropy)

$$L(\boldsymbol{\theta}) = \left(-\mathbf{y}^{\mathrm{T}}\log\hat{\mathbf{y}} - (\mathbf{1}-\mathbf{y})^{\mathrm{T}}\log(1-\hat{\mathbf{y}})\right)$$

→ Tính đạo hàm

$$L'_{\boldsymbol{\theta}} = \mathbf{x}^{\mathrm{T}}(\hat{\mathbf{y}} - \boldsymbol{y})$$

→ Cập nhật tham số (Stochastic gradient descent)

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

Declare optimizer and loss function

```
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=learning_rate)
```

Start training

```
for epoch in range(max_epoch):
    # Zero the gradients
    optimizer.zero_grad()

    # Forward pass
    outputs = model(X)

    # Compute loss
    loss = criterion(outputs, y)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

Training

❖ Softmax regression

→ Tính output $\hat{y}$

$$\boldsymbol{z} = \boldsymbol{\theta}^T \boldsymbol{x} \qquad \hat{\mathbf{y}} = \frac{e^{\boldsymbol{z}}}{\sum_{i=1}^{k} e^{z_i}}$$

→ Tính loss (cross-entropy)

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{k} y_i \log \hat{y}_i$$

→ Tính đạo hàm

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \boldsymbol{x}(\hat{y}_i - y_i)$$

→ Cập nhật tham số (Stochastic gradient descent)

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

Delaration and Training

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=learning_rate)
```

```
for epoch in range(max_epoch):
    # Zero the gradients
    optimizer.zero_grad()

    # Forward pass
    outputs = model(X)

    # Compute loss
    loss = criterion(outputs, y)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

Training

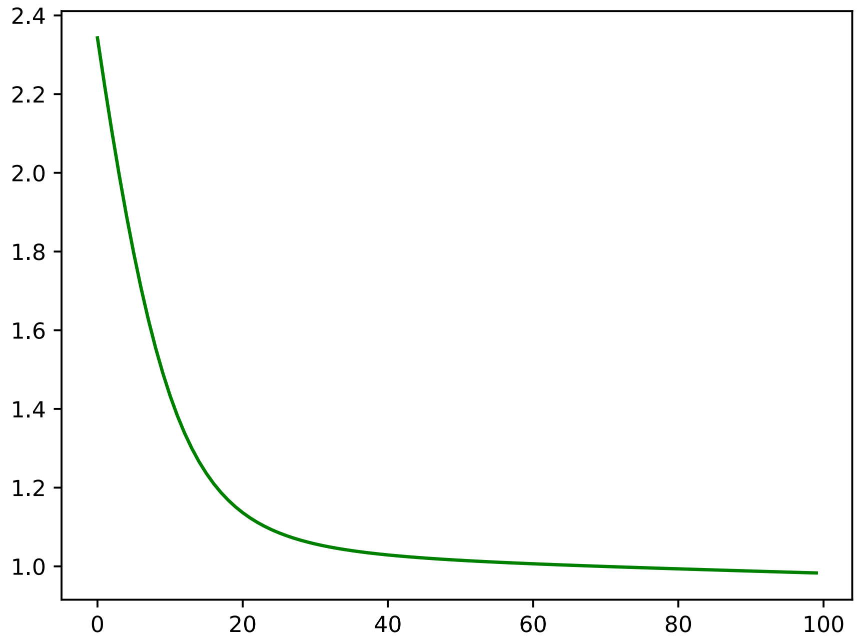
Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2
5.2	3
5.6	3
5.9	3

❖ Softmax regression

Model

$$z = \theta^T x$$

$$\hat{y} = \frac{e^z}{\sum_{i=1}^{k} e^{z_i}}$$




```python
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

# Load data
iris = np.genfromtxt('iris_2D_3c.csv', dtype=None,
                     delimiter=',', skip_header=1)
X = torch.tensor(iris[:, 0:2], dtype=torch.float32)
y = torch.tensor(iris[:, 2], dtype=torch.int64)

# Define a simple Sequential model
input_dim = X.shape[1]
output_dim = len(torch.unique(y))
model = nn.Sequential(
    nn.Linear(in_features=input_dim, out_features=output_dim)
)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Training

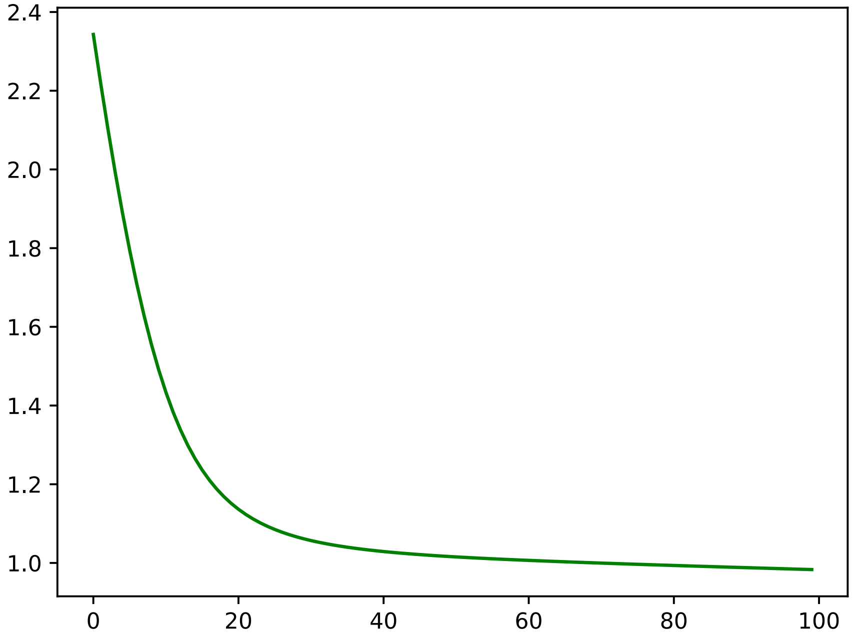
Petal_Length	Label
1.4	1
1.3	1
1.5	1
4.5	2
4.1	2
4.6	2
5.2	3
5.6	3
5.9	3

❖ Softmax regression

Model

$$z = \theta^T x$$

$$\hat{y} = \frac{e^{z}}{\sum_{i=1}^{k} e^{z_i}}$$




```python
# Training loop
max_epoch = 100
losses = []

for epoch in range(max_epoch):
    # Zero the gradients
    optimizer.zero_grad()

    # Forward pass
    outputs = model(X)

    # Compute loss
    loss = criterion(outputs, y)
    losses.append(loss.item())

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

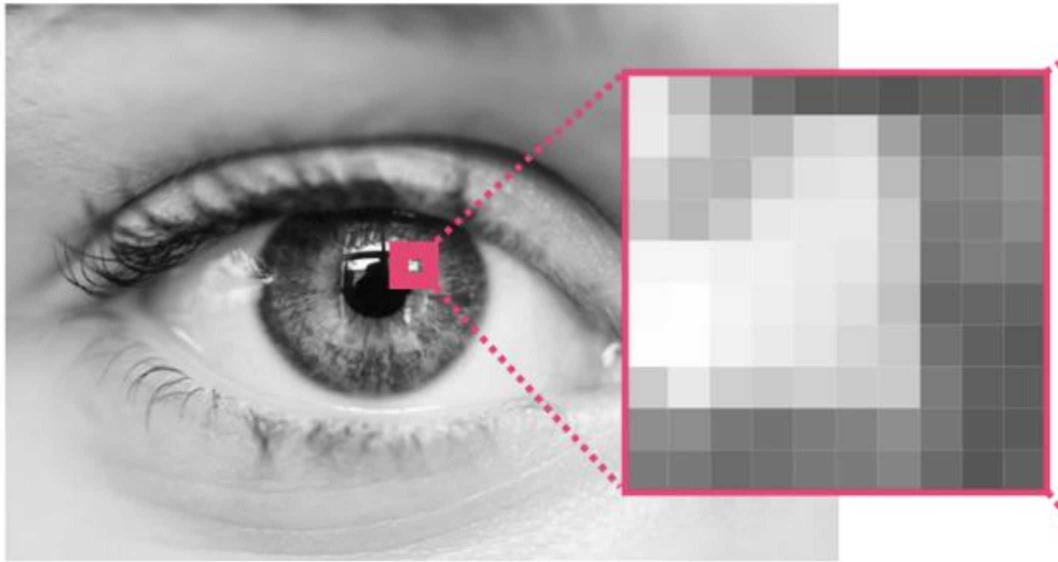
```python
import matplotlib.pyplot as plt
plt.plot(losses, 'g')
```

Outline

- Introduction to PyTorch
- Model Construction
- Model Training and Inference
- Applying Softmax for Image Data

Image Classification: Image Data

❖ Grayscale images



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97

(Height, Width)

Pixel p = scalar

$0 \leq p \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Image Classification: Image Data

❖ Color images

(Channel, Height, Width,)

RGB color image

Pixel $p = \begin{bmatrix} r \\ g \\ b \end{bmatrix}$

$0 \le r,g,b \le 255$

Resolution: #pixels

Resolution = HeightxWidth

Important Packages

❖ Some functions

To download a file

import urllib.request as req
req.urlretrieve(url, name)

To open an image

from PIL import Image
img = Image.open(name)

To show an image

import matplotlib.pyplot as plt
plt.imshow(img)

```python
import urllib.request as req
from PIL import Image
import matplotlib.pyplot as plt

# download an image
req.urlretrieve('https://www.dropbox.com/s/zwy8ddkdm3thatr/nature.jpg?dl=1',
                'image.jpg')

# show the image
img = Image.open('image.jpg')
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f5088018b90>
```

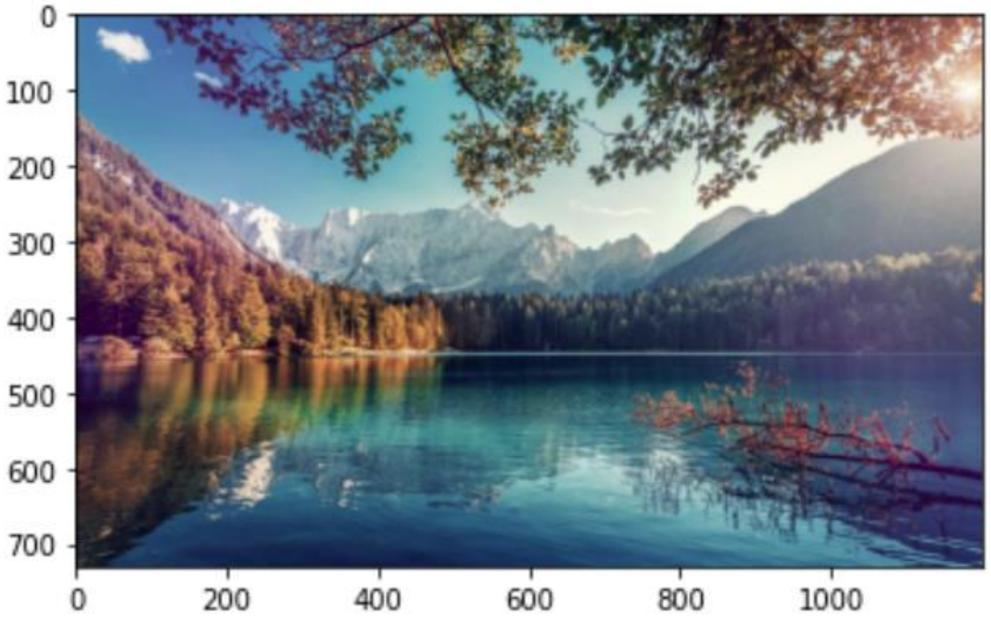


Image Data

MNIST dataset

Grayscale images

Resolution=28x28

Training set: 60000 samples

Testing set: 10000 samples

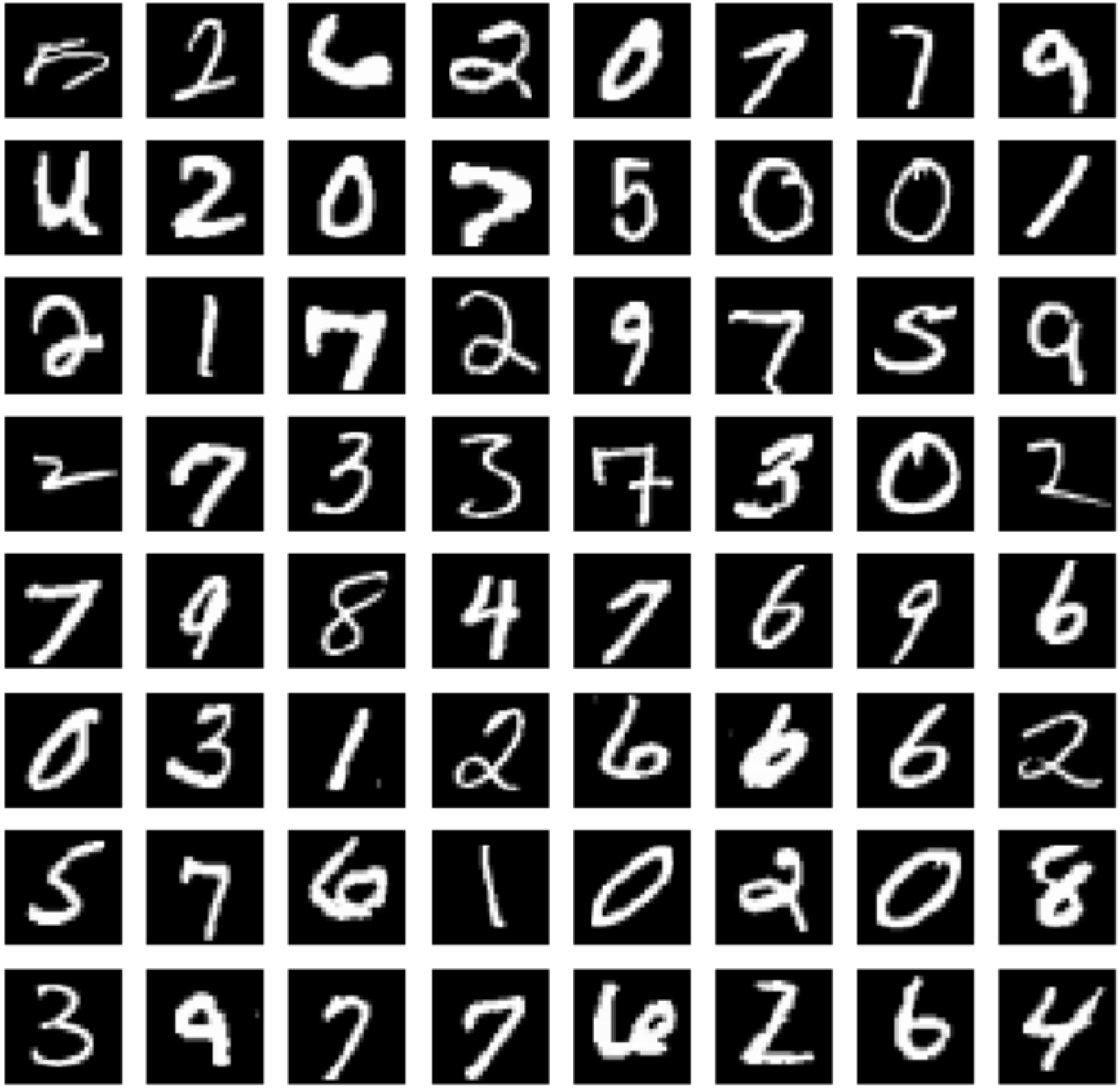


Image Data

Fashion-MNIST dataset

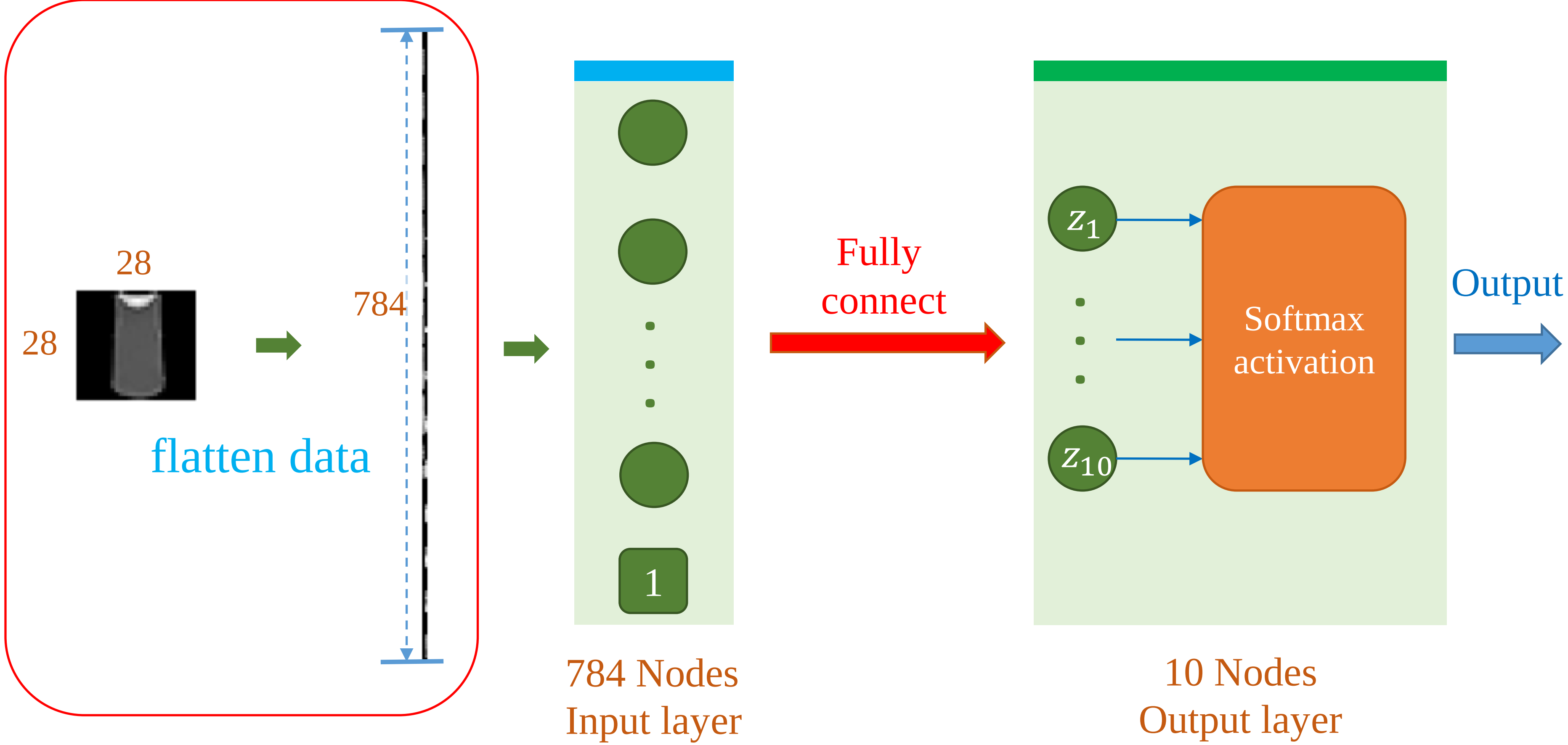
Grayscale images

Resolution=28x28

Training set: 60000 samples

Testing set: 10000 samples

Using Softmax Regression



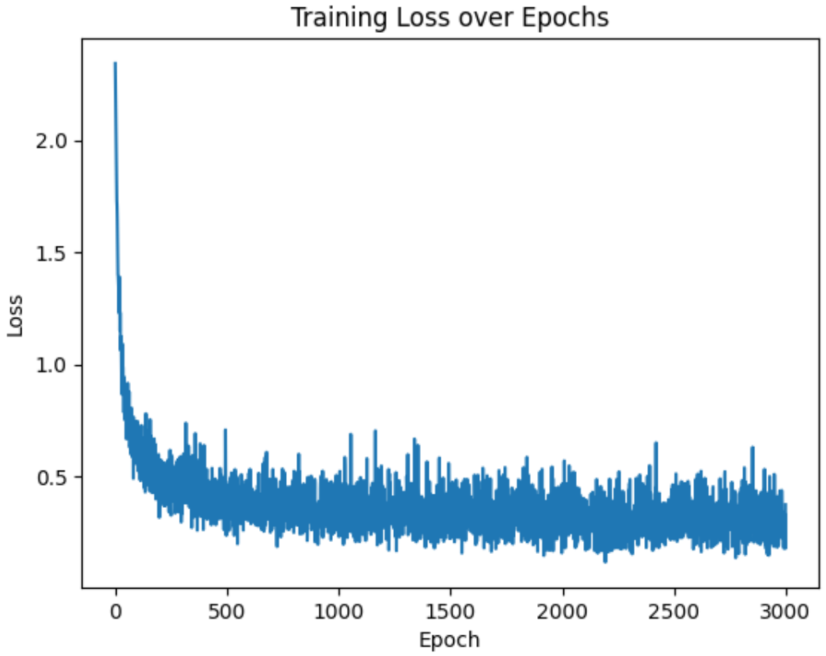

```
X_train: (60000, 784)
y_train: (60000,)
X_test: (10000, 784)
y_test: (10000,)
```

Data Sets

```python
class SoftmaxRegression(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(SoftmaxRegression, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)
        self.flatten = nn.Flatten()

    def forward(self, x):
        x = self.flatten(x)
        return self.linear(x)
```

Demo

```python
# Testing the model
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        outputs = model(images)
        predicted = torch.argmax(outputs.data, dim=1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```



Test Accuracy
91.97%

```python
# Load MNIST dataset
train_loader = ...
test_loader = ...

# Input and output dimensions
input_dim = 28 * 28  # MNIST images are 28x28
output_dim = 10      # 10 classes for MNIST

# Create a Sequential model
model = nn.Sequential(nn.Flatten(),
                      nn.Linear(input_dim, output_dim))

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# Training Loop
num_epochs = 5
losses = []  # to store the average loss for each epoch

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        outputs = model(images)
        loss = criterion(outputs, labels)
        losses.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```