AI VIET NAM
@aivietnam.edu.vn

Multi-layer Perceptron

Quang-Vinh Dinh
Ph.D. in Computer Science

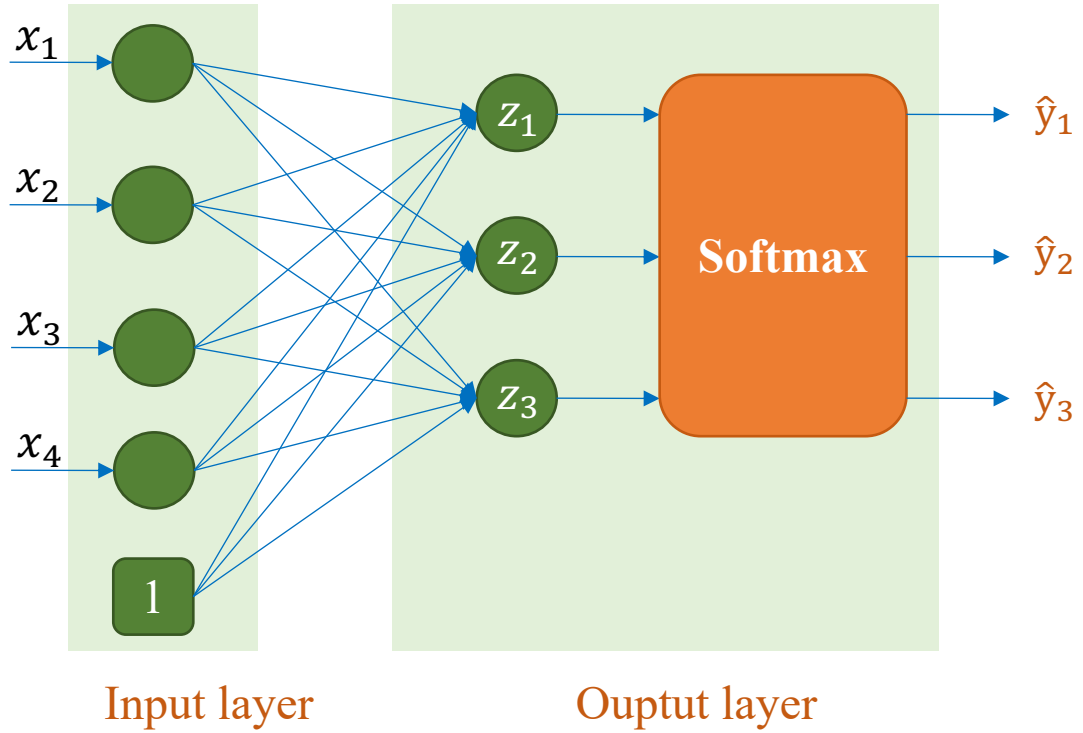
Year 2024

code&data

Motivation

❖ Performance of softmax regression for tabular data

Feature				Label
Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Label
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
5.5	2.3	4	1.3	1
4.9	2.5	4.5	1.7	2
7.3	2.9	6.3	1.8	2
6.7	2.5	5.8	1.8	2

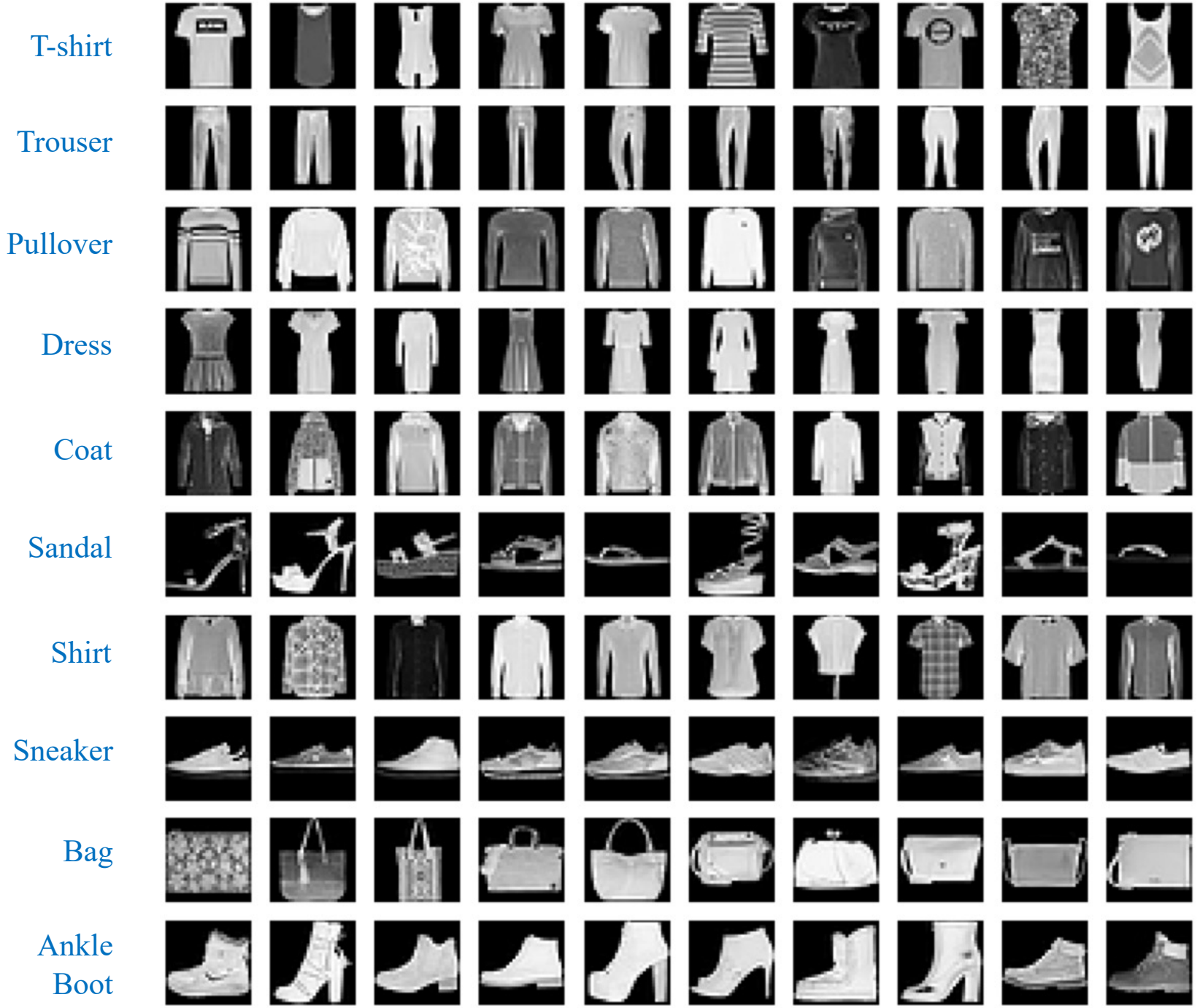


Achieve very high training accuracies (>98%)

Complex Data

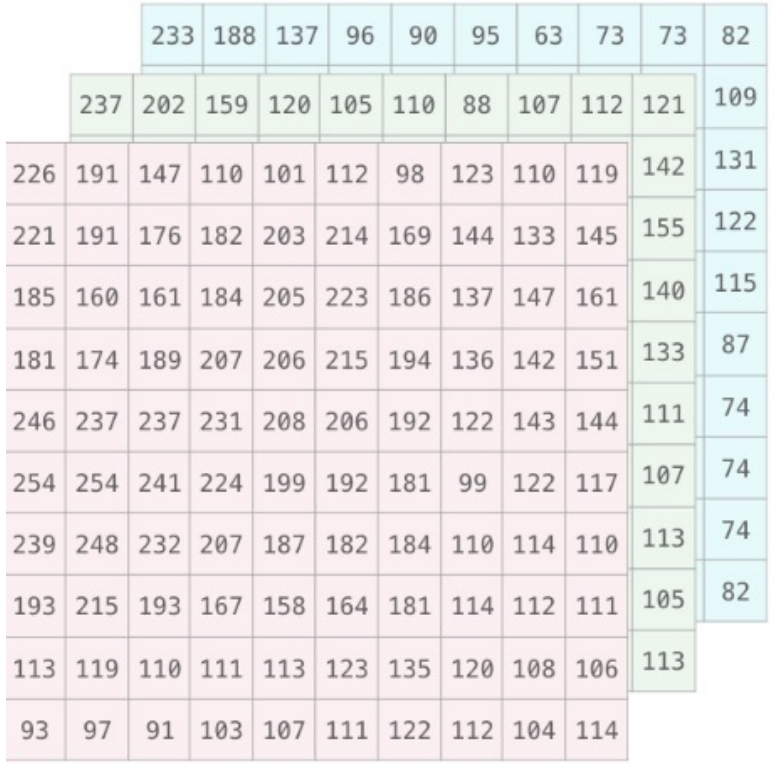
Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples



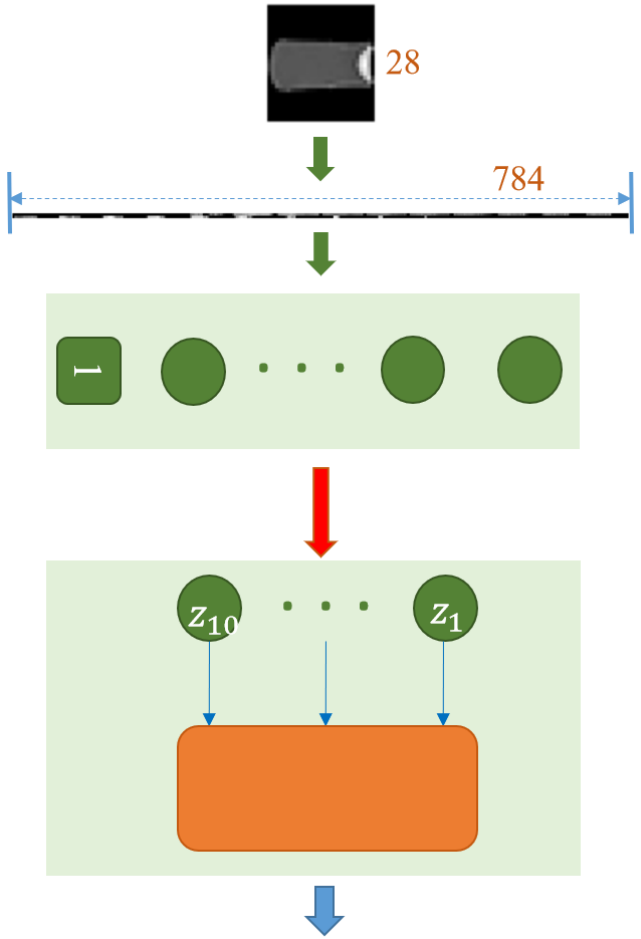
Objectives

Image Data

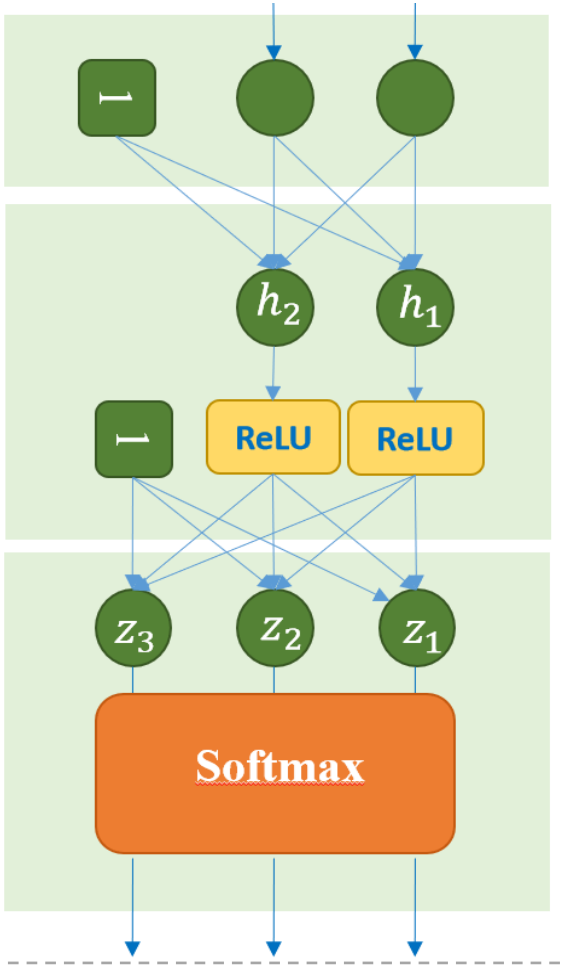


	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

S. R. for Image Data



Multilayer Perceptron



Outline

SECTION 1

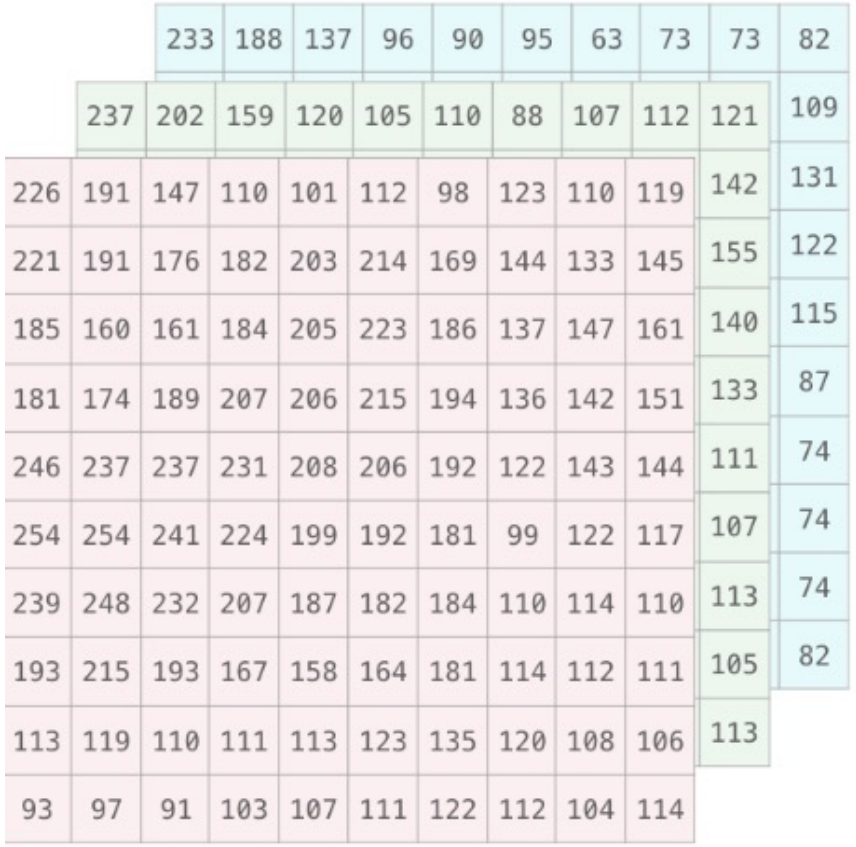
Image Data

SECTION 2

S. R. for Image Data

SECTION 3

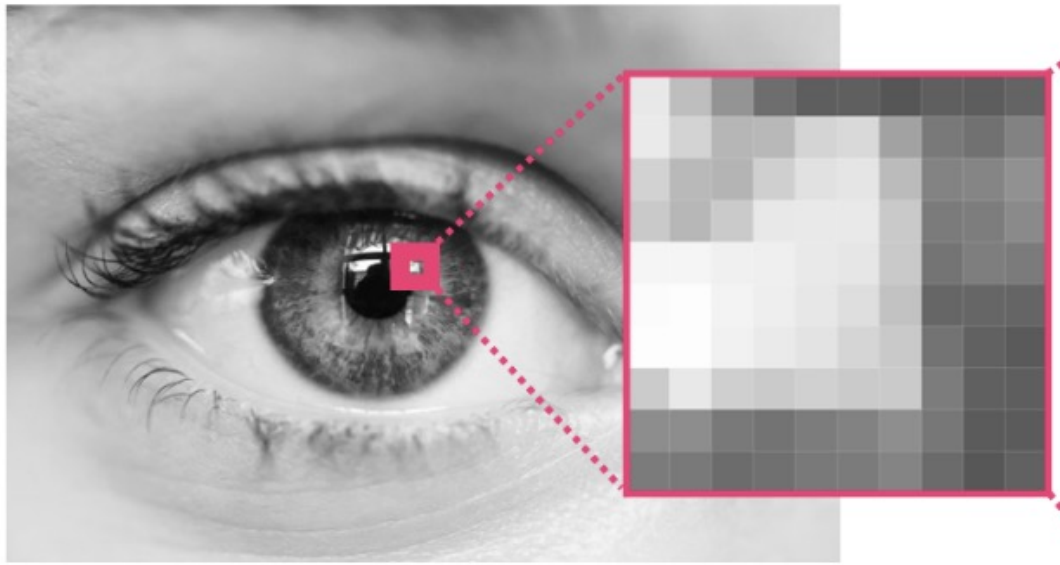
MLP for Image Data



	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

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)

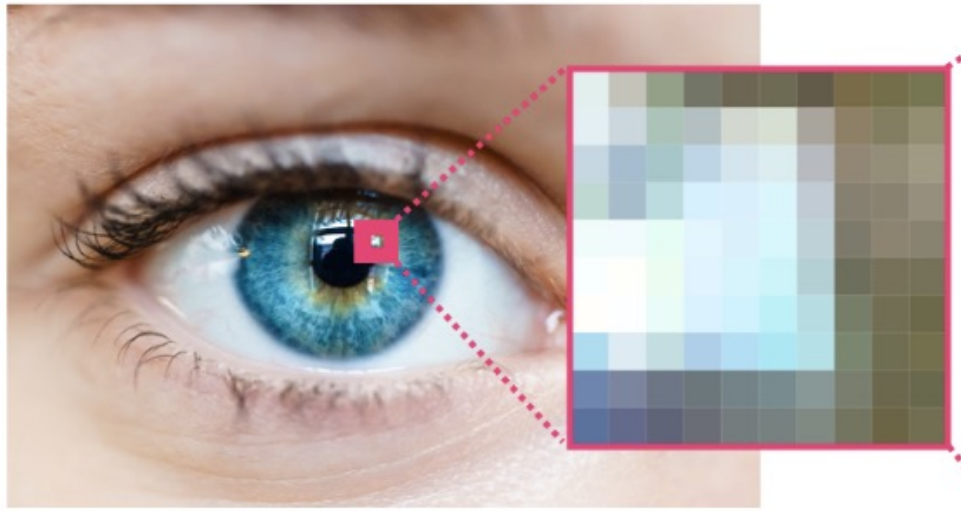
Pixel p = scalar

$0 \leq p \leq 255$

Resolution: #pixels
Resolution = HeightxWidth

Image Classification: Image Data

❖ Color images



	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

(Height, Width, channel)

RGB color image

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

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

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)

```
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>
```

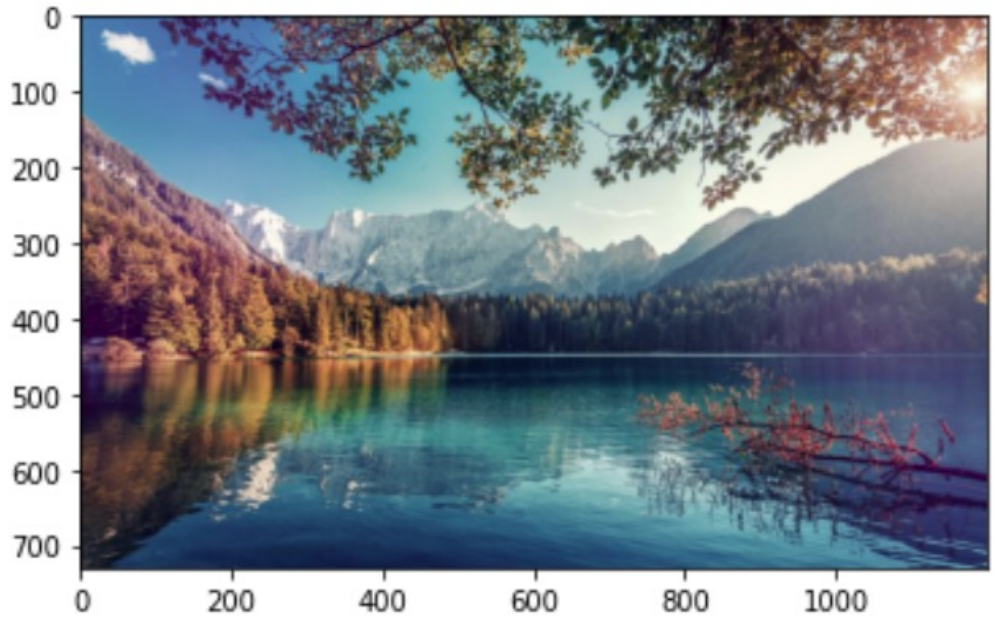


Image Data

MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

How many bytes needed to store a 28x28 grayscale image?

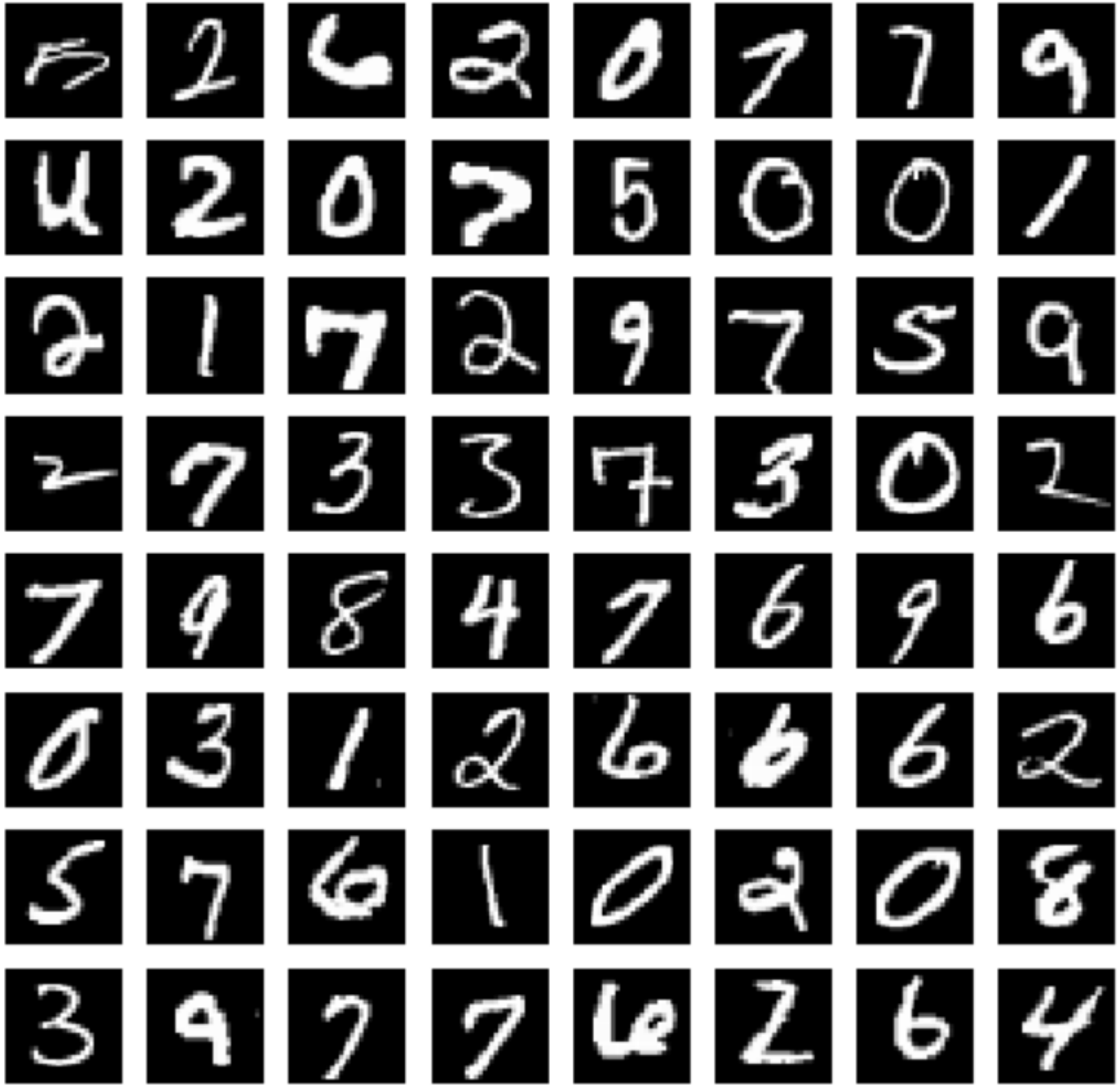


Image Data

MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

http://yann.lecun.com/exdb/mnist/

Image Data

Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

Image Classification

Fashion-MNIST dataset

- Grayscale images
- Resolution=28x28
- Training set: 60000 samples
- Testing set: 10000 samples

```
from urllib import request

filenames = ["train-images-idx3-ubyte.gz",
             "train-labels-idx1-ubyte.gz",
             "t10k-images-idx3-ubyte.gz",
             "t10k-labels-idx1-ubyte.gz"]

folder = 'data_fashion_mnist/'
base_url = "http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/"
for name in filenames:
    print("Downloading " + name + "...")

    # lưu vào folder data_fashion_mnist
    request.urlretrieve(base_url + name, folder + name)
print("Download complete.")
```

Download data

Name	Size
t10k-images-idx3-ubyte.gz	4.4 MB
t10k-labels-idx1-ubyte.gz	5.1 kB
train-images-idx3-ubyte.gz	26.4 MB
train-labels-idx1-ubyte.gz	29.5 kB

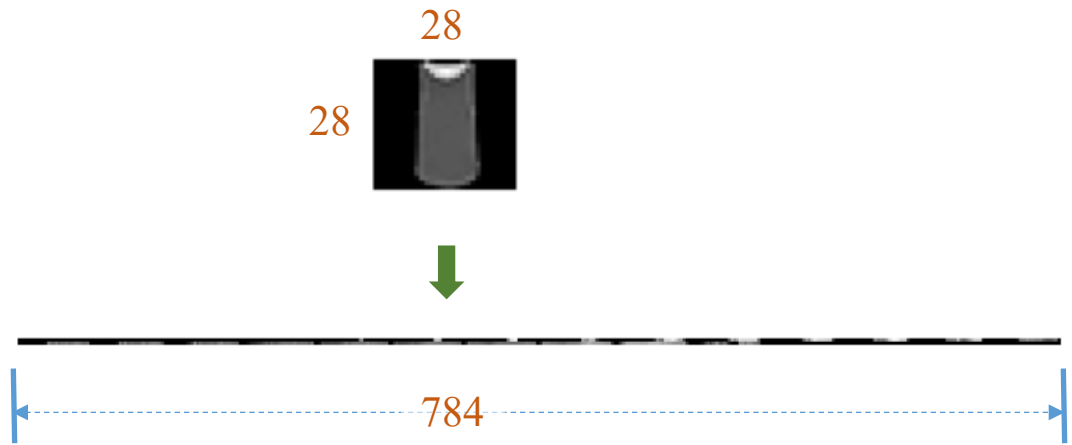
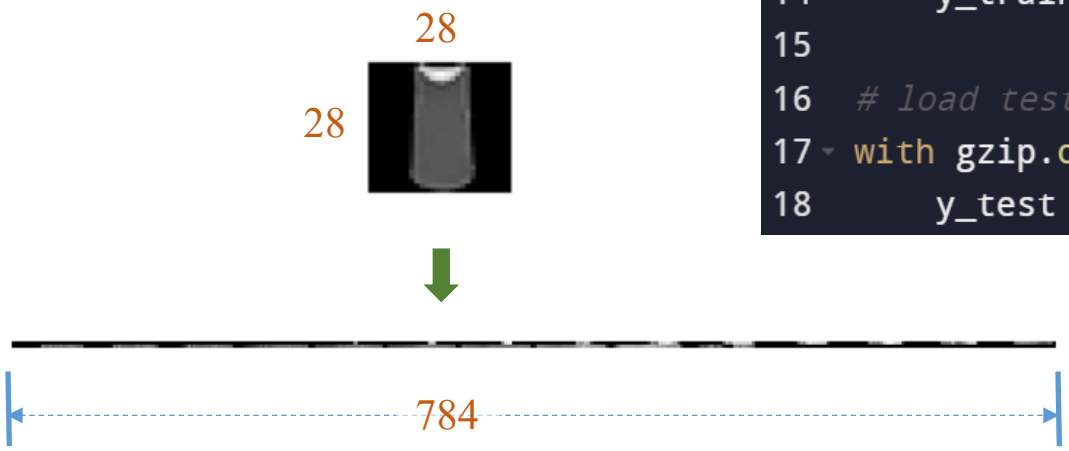


Image Data

❖ Fashion-MNIST data

Download data

Name	Size
t10k-images-idx3-ubyte.gz	4.4 MB
t10k-labels-idx1-ubyte.gz	5.1 kB
train-images-idx3-ubyte.gz	26.4 MB
train-labels-idx1-ubyte.gz	29.5 kB




```python
import numpy as np
import gzip

# load training images
with gzip.open('data_fashion_mnist/train-images-idx3-ubyte.gz', 'rb') as f:
    X_train = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1, 28*28)

# load testing images
with gzip.open('data_fashion_mnist/t10k-images-idx3-ubyte.gz', 'rb') as f:
    X_test = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1, 28*28)

# load training labels
with gzip.open('data_fashion_mnist/train-labels-idx1-ubyte.gz', 'rb') as f:
    y_train = np.frombuffer(f.read(), np.uint8, offset=8)

# load testing labels
with gzip.open('data_fashion_mnist/t10k-labels-idx1-ubyte.gz', 'rb') as f:
    y_test = np.frombuffer(f.read(), np.uint8, offset=8)
```

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

Demo

Using PyTorch

Image Data

❖ Using Pytorch

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

data
(ndarray, tensor)

Size

…

Mode

…

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

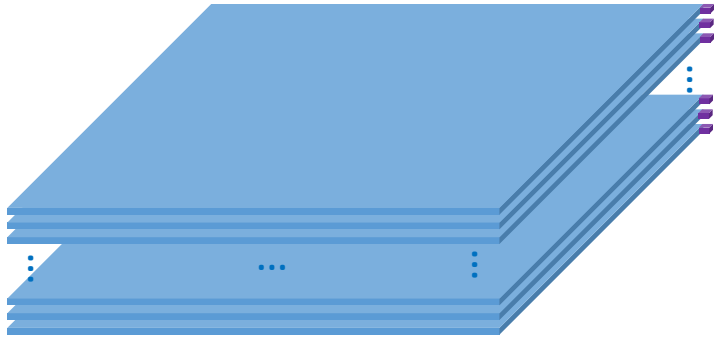
data
(ndarray, tensor)

Fashion MNIST

❖ Using Pytorch

Each sample is a tuple (PIL-image, label)

60000 samples



```
from torchvision.datasets import FashionMNIST
trainset = FashionMNIST(root='data',
                        train=True,
                        download=True)

img, label = trainset[0]
print(type(img), label)
```

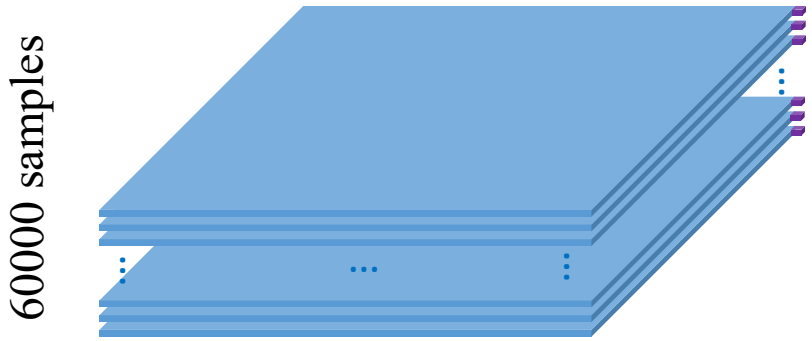
```
<class 'PIL.Image.Image'> 9
```

```
import matplotlib.pyplot as plt

img, _ = trainset[0]

plt.figure(figsize=(2,2))
plt.imshow(img, cmap='gray')
plt.axis('off')  # Hide axis
plt.show()
```

Fashion MNIST

❖ Using Pytorch

Each sample is a tuple (image-tensor, label)




```
from torch.utils.data import DataLoader
from torchvision import transforms

transform = transforms.Compose([transforms.ToTensor()])
trainset = FashionMNIST(root='data', train=True,
                        download=True, transform=transform)

img, label = trainset[0]
print(type(img), label)
```

```
<class 'torch.Tensor'> 9
```

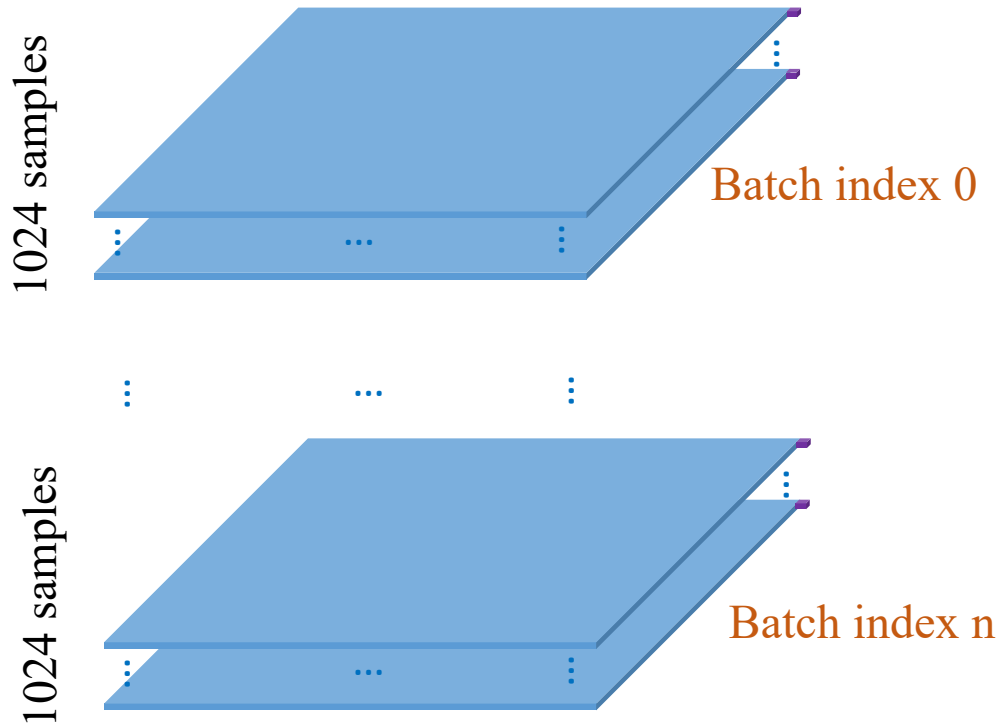
```
import matplotlib.pyplot as plt

img, _ = trainset[0]
np_img = img.numpy()
np_img = np.transpose(np_img, (1, 2, 0))

plt.figure(figsize=(2,2))
plt.imshow(np_img, cmap='gray')
plt.axis('off')
plt.show()
```

Fashion MNIST

approximately

❖ Using Pytorch

Each sample is a tuple (image-tensor, label)

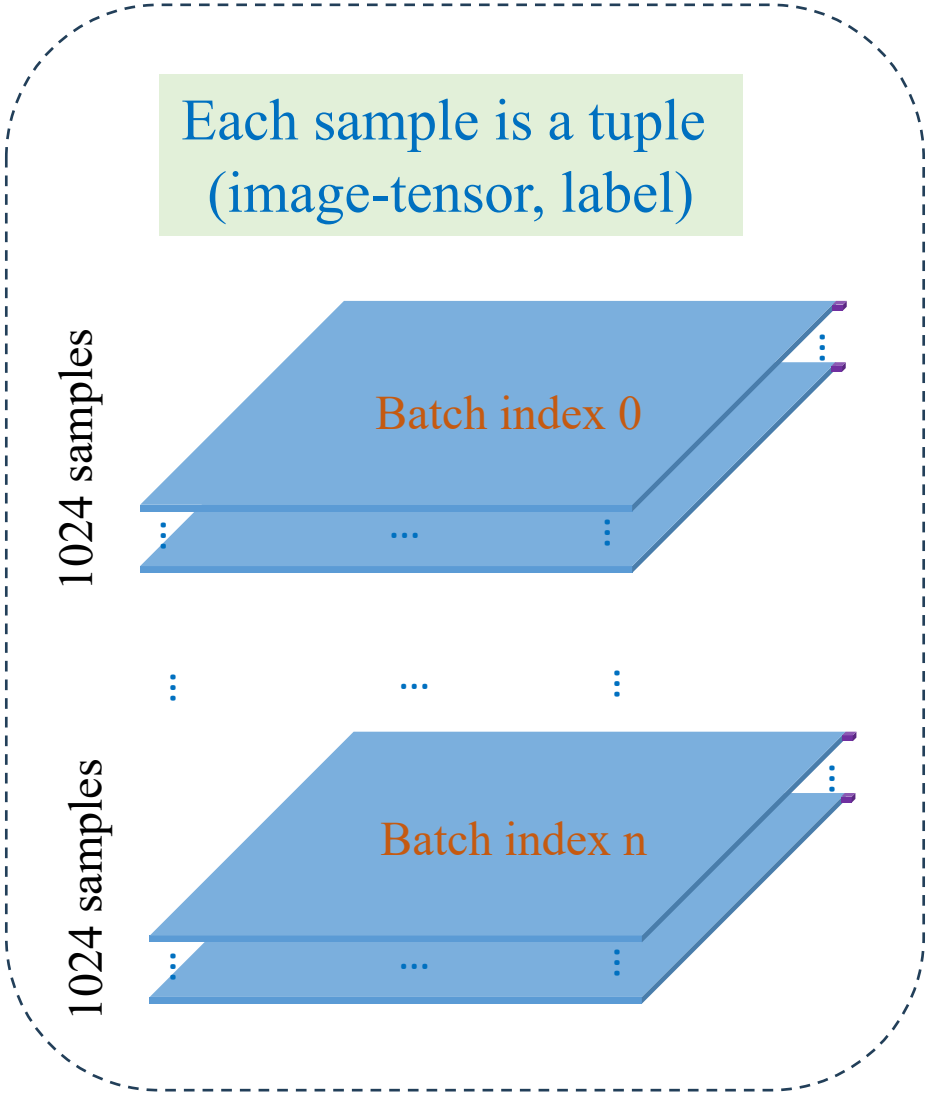



```python
from torchvision.datasets import FashionMNIST
from torch.utils.data import DataLoader
from torchvision import transforms

transform = transforms.Compose([transforms.ToTensor()])
trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)

trainloader = DataLoader(trainset,
                         batch_size=1024,
                         num_workers=2,
                         shuffle=True)
print(len(trainloader))
```

```
59
```

Fashion MNIST

❖ Using Pytorch




```python
# batch_size=3500
for i, (inputs, labels) in enumerate(trainloader, 0):
    print(f'Batch index {i} -- {inputs.shape} -- {labels.shape}')
```

```
Batch index 0 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 1 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 2 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 3 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 4 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 5 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 6 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 7 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 8 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 9 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 10 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 11 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 12 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 13 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 14 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 15 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 16 -- torch.Size([3500, 1, 28, 28]) -- torch.Size([3500])
Batch index 17 -- torch.Size([500, 1, 28, 28]) -- torch.Size([500])
```

Outline

SECTION 1

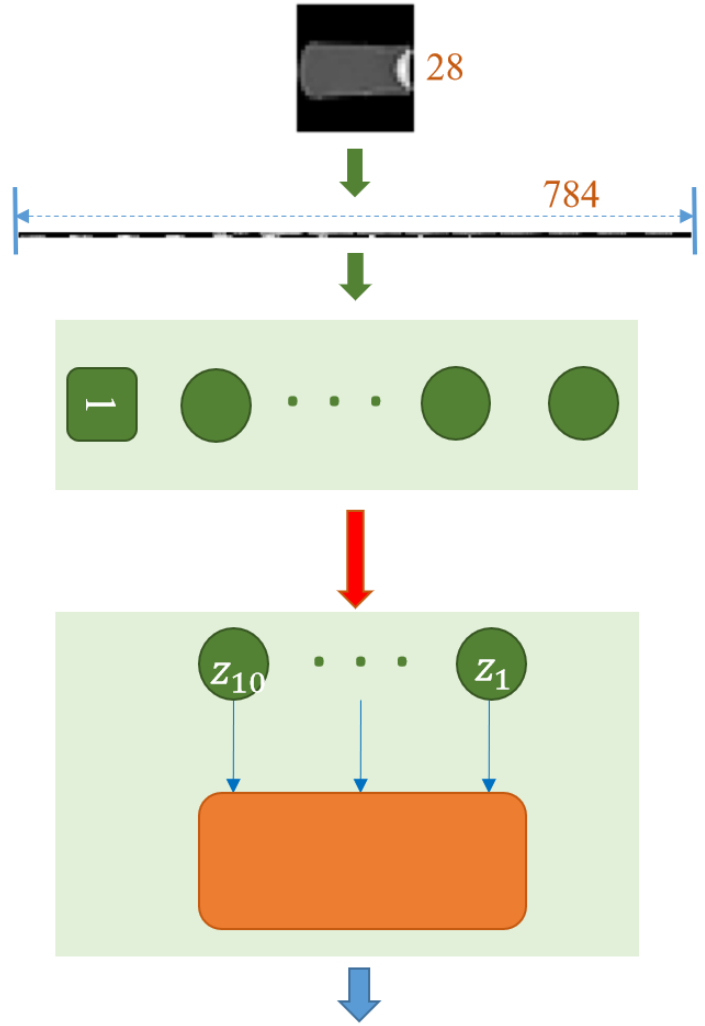
Image Data

SECTION 2

S. R. for Image Data

SECTION 3

MLP for Image Data



Softmax Regression

Petal_Length	Label
1.4	0
1.3	0
1.5	0
4.5	1
4.1	1
4.6	1

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

$$\boldsymbol{\theta} = \begin{bmatrix} b_0 & b_1 \\ w_0 & w_1 \end{bmatrix}$$

One-hot encoding for label

$$y = 0 \rightarrow \boldsymbol{y}^T = \begin{matrix} y_0 & y_1 \\ [1 & 0] \end{matrix}$$

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

scalar vector

$$z_0 = xw_0 + b_0$$

$$z_1 = xw_1 + b_1$$

$$\hat{y}_0 = \frac{e^{z_0}}{\sum_{j=0}^{1} e^{z_j}}$$

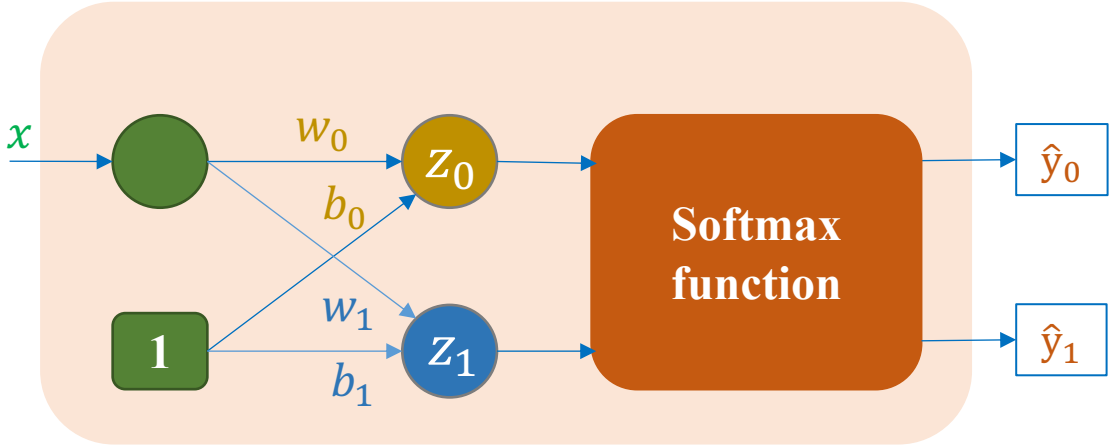
$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{j=0}^{1} e^{z_j}}$$

$$\boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} b_0 & w_0 \\ b_1 & w_1 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta}_0^T \\ \boldsymbol{\theta}_1^T \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \end{bmatrix} = \frac{1}{\sum_{j=0}^{1} e^{z_j}} \begin{bmatrix} e^{z_0} \\ e^{z_1} \end{bmatrix} = \frac{e^{\boldsymbol{z}}}{\sum_{j=0}^{1} e^{z_j}}$$

$$L(\boldsymbol{\theta}) = -\sum_{i=0}^{1} y_i \log \hat{y}_i = -\boldsymbol{y}^T log \hat{\boldsymbol{y}}$$

Model



Derivative

$$\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$$

$$\frac{\partial \hat{y}_i}{\partial z_j} = \begin{cases} \hat{y}_i(1 - \hat{y}_i) & if\ i = j \\ -\hat{y}_i \hat{y}_j & if\ i \neq j \end{cases}$$

$$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i$$

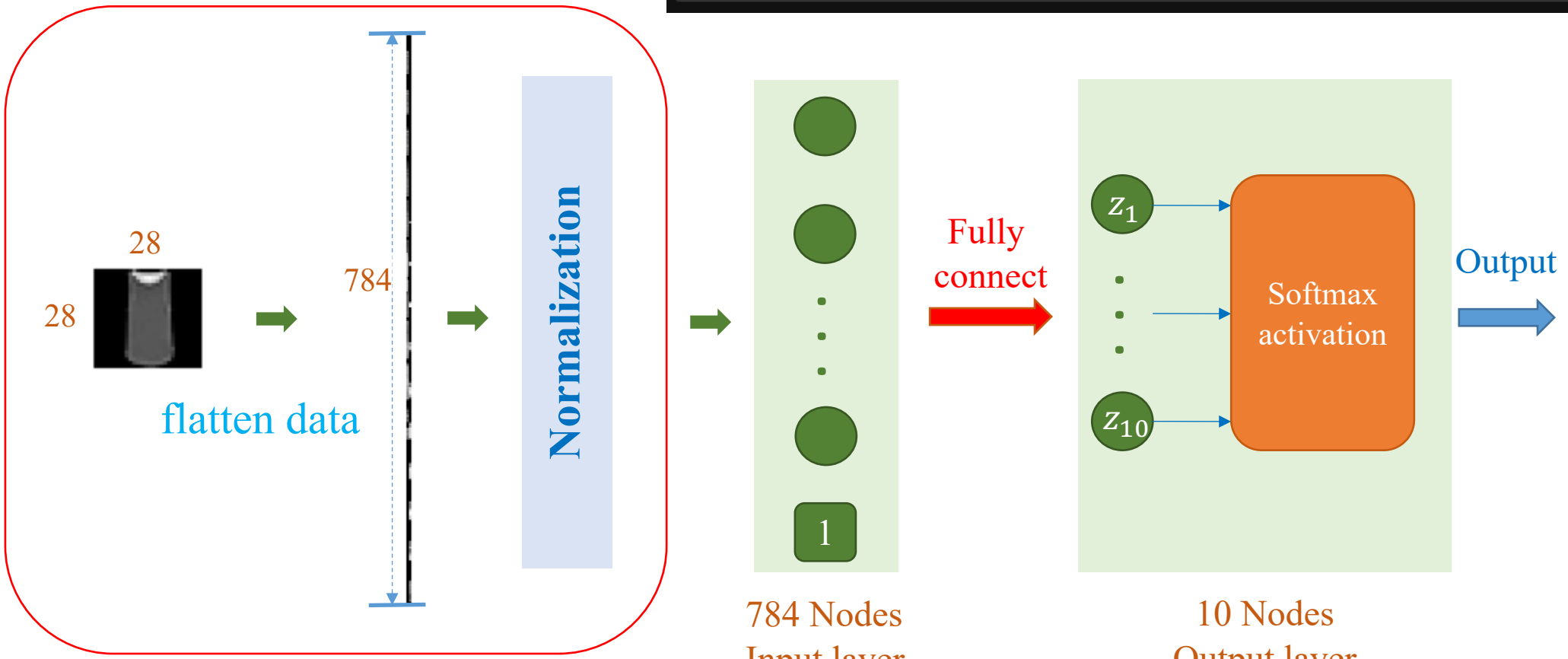
$$\frac{\partial L}{\partial w_i} = x(\hat{y}_i - y_i)$$

$$\frac{\partial L}{\partial b_i} = \hat{y}_i - y_i$$

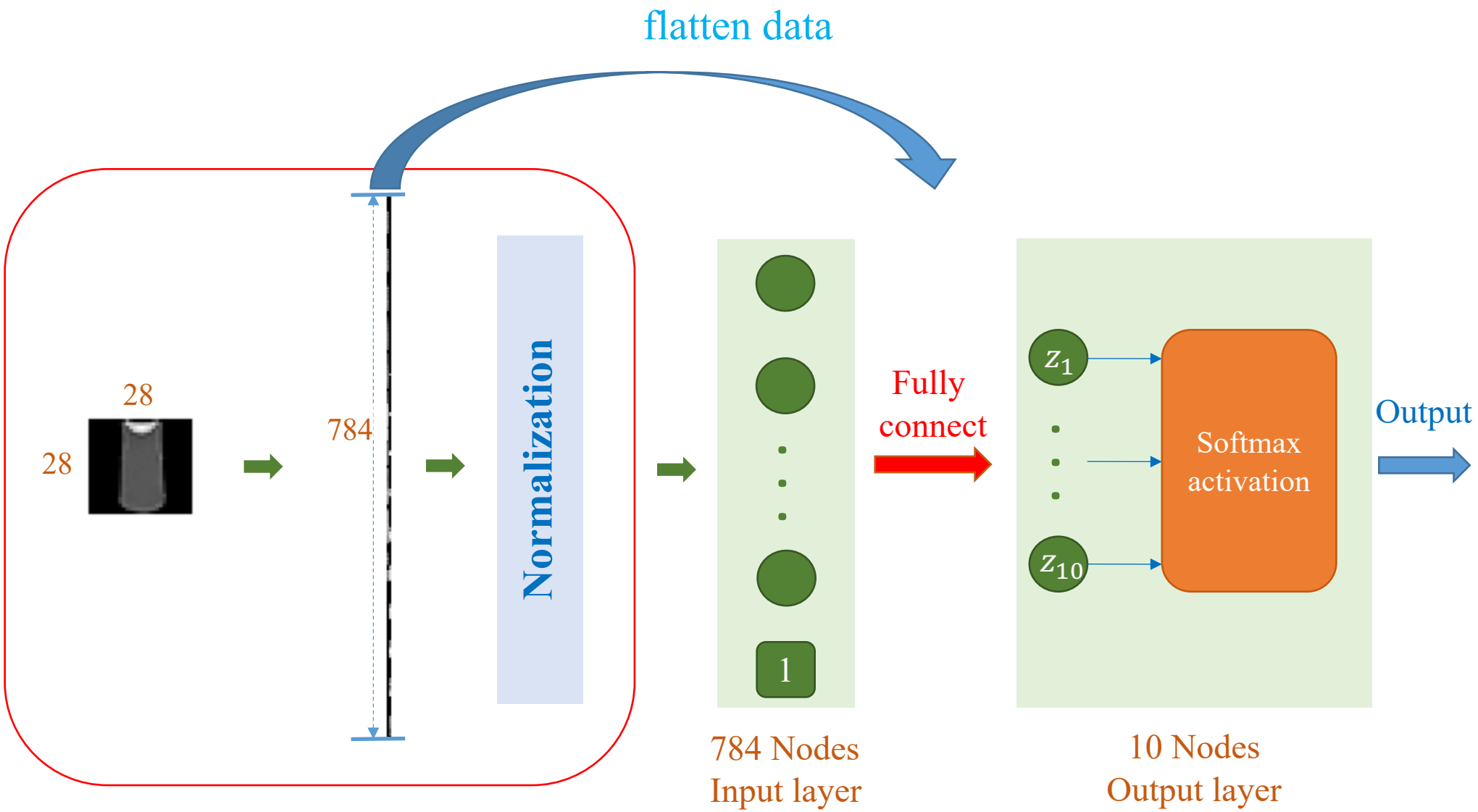
Where to put Flatten

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Lambda(torch.flatten)])
trainset = FashionMNIST(root='data', train=True,
                        download=True, transform=transform)

img, _ = trainset[0]
print(img.shape)
```

```
torch.Size([784])
```



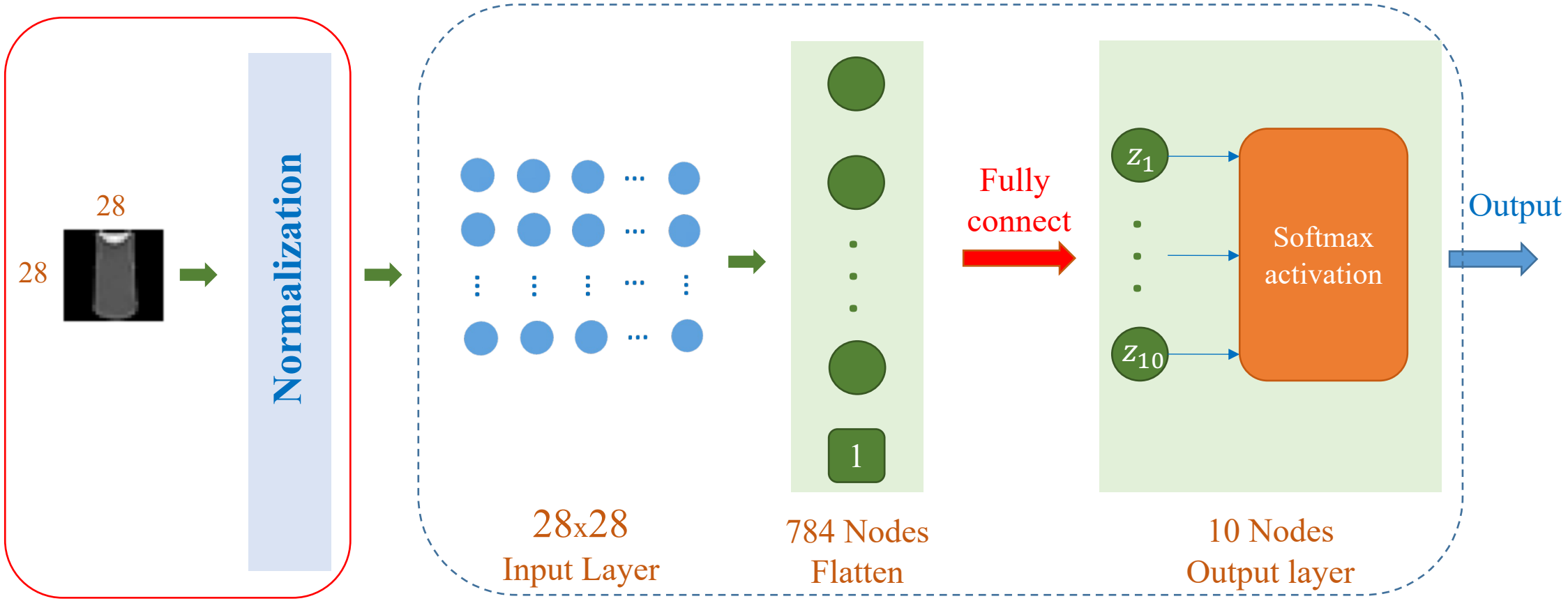
Where to put Flatten



Where to put Flatten

```
transform = transforms.Compose([transforms.ToTensor()])
trainset = FashionMNIST(root='data', train=True,
                        download=True, transform=transform)

img, label = trainset[0]
print(img.shape)
```

```
torch.Size([1, 28, 28])
```

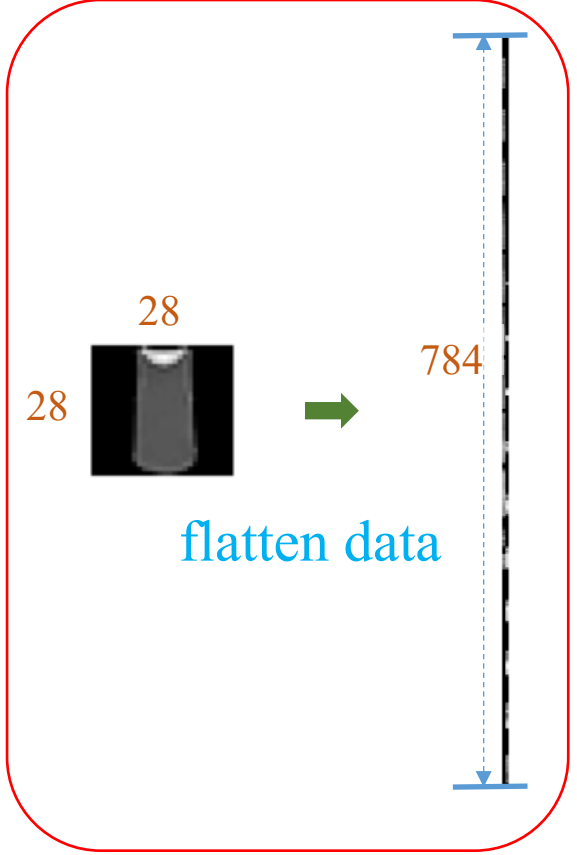


Case 1

Softmax Regression

without normalization

learning rate = 0.01



784 Nodes
Input layer

10 Nodes
Output layer

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

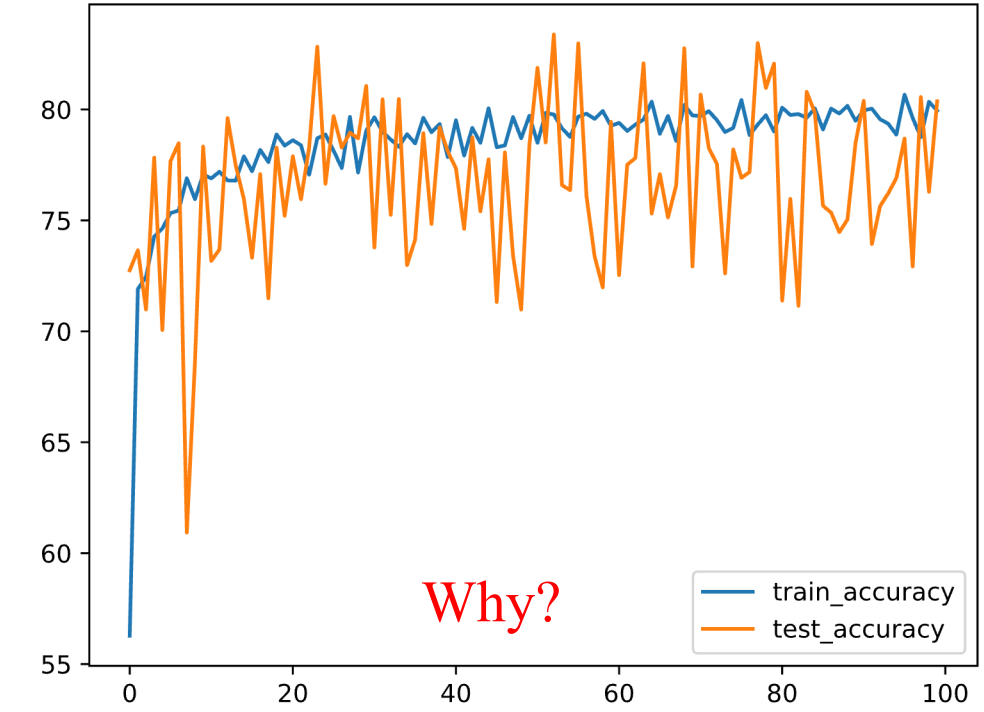
Data Sets

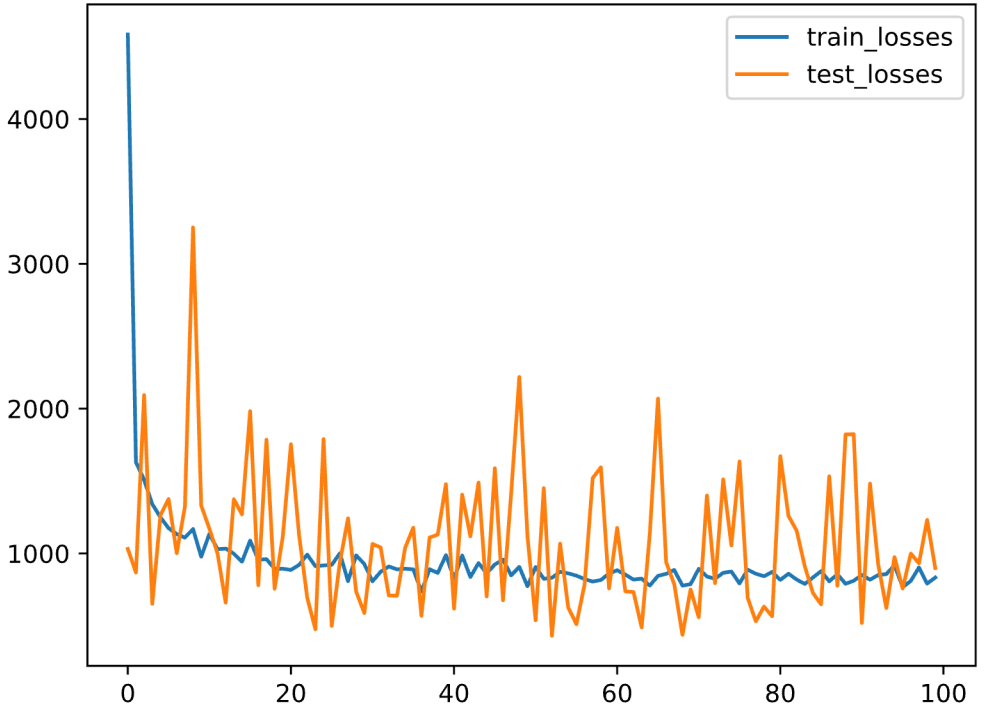
```python
import torch.nn as nn

model = nn.Sequential(
    nn.Flatten(), nn.Linear(784, 10)
)
print(model)
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=10, bias=True)
)
```

```python
# Generating a random tensor
input_tensor = torch.rand(5, 28, 28)

# Feeding the tensor into the model
output = model(input_tensor)
print(output.shape)
```

```
torch.Size([5, 10])
```





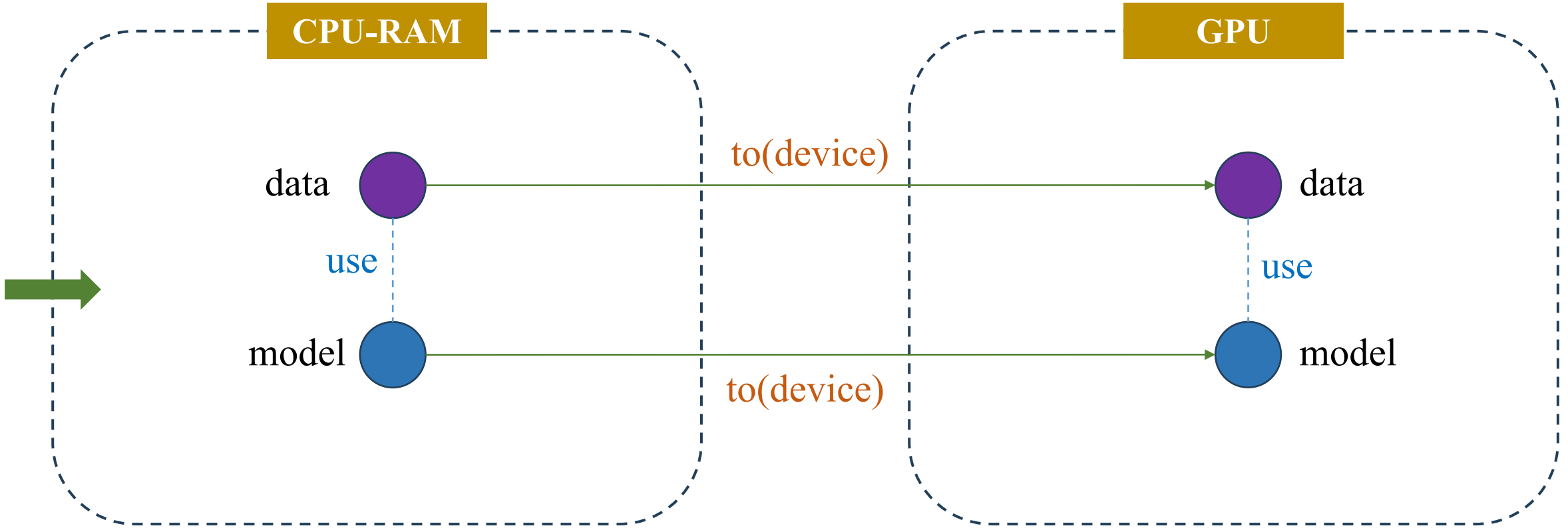

```python
model = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
model = model.to(device)
```

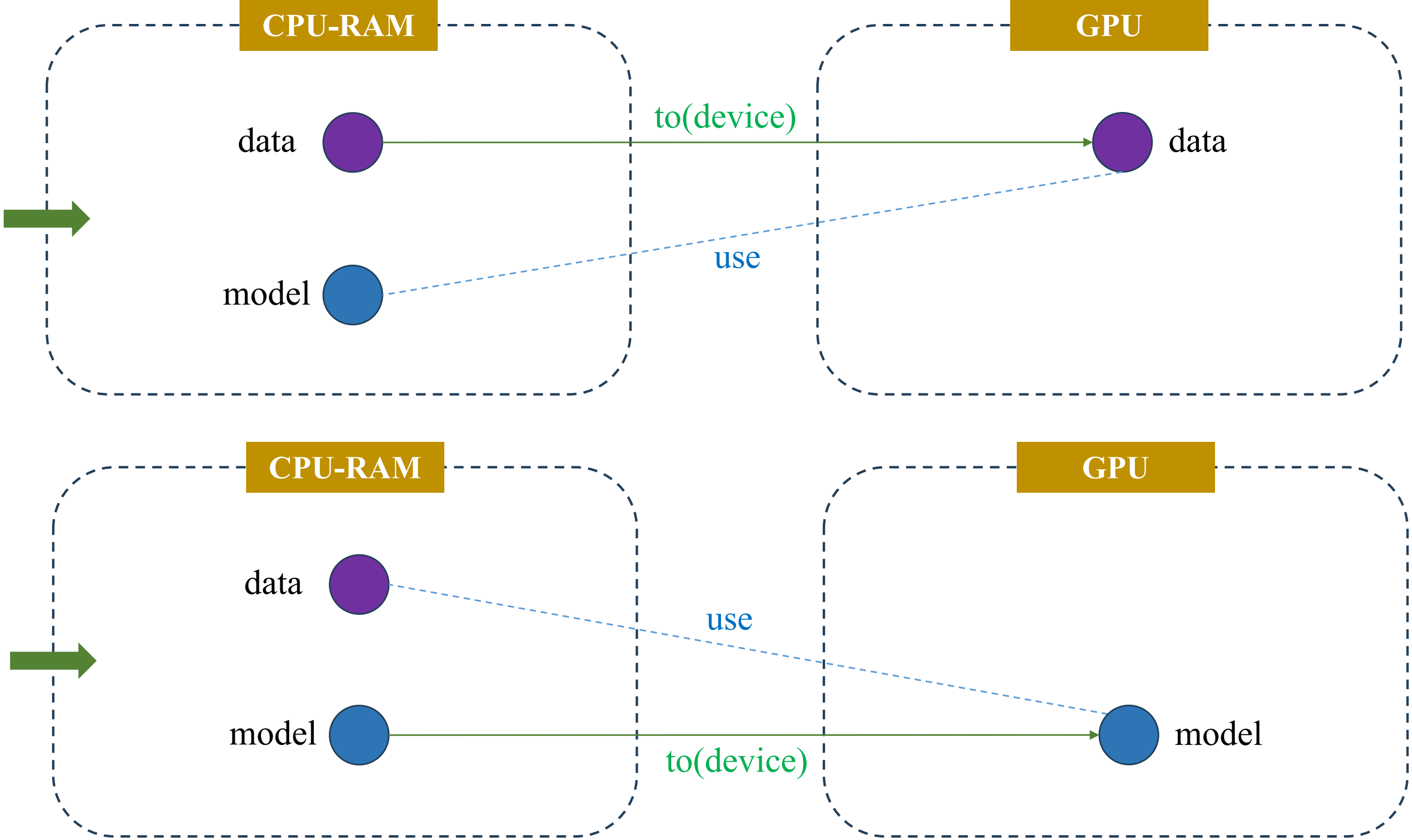
```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

```python
# train
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()

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

Softmax Regression - Implementation

❖ to(device) function



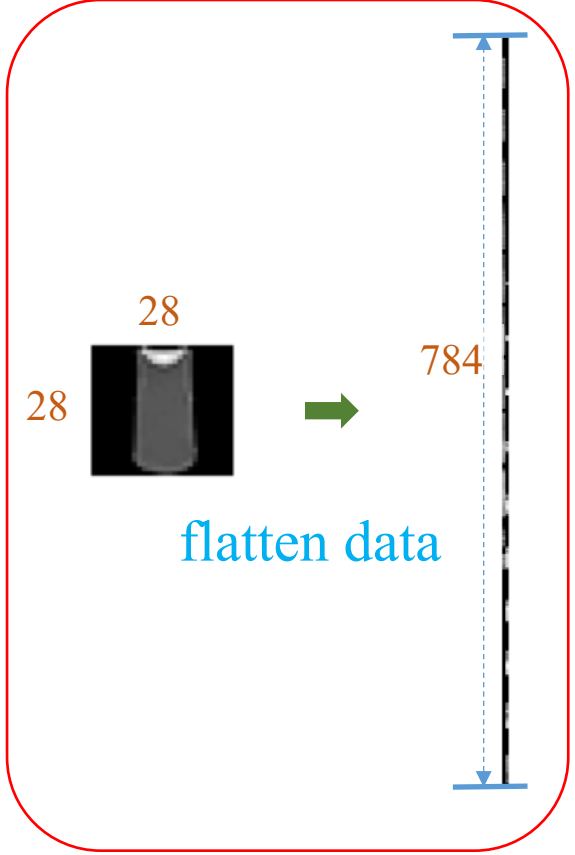

CPU-RAM
GPU
data
to(device)
data
use
model
CPU-RAM
GPU
data
use
model
model
to(device)

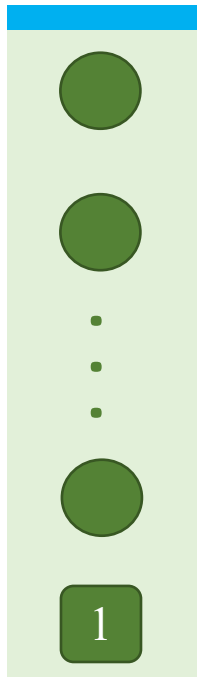
Case 2

Softmax Regression

without normalization

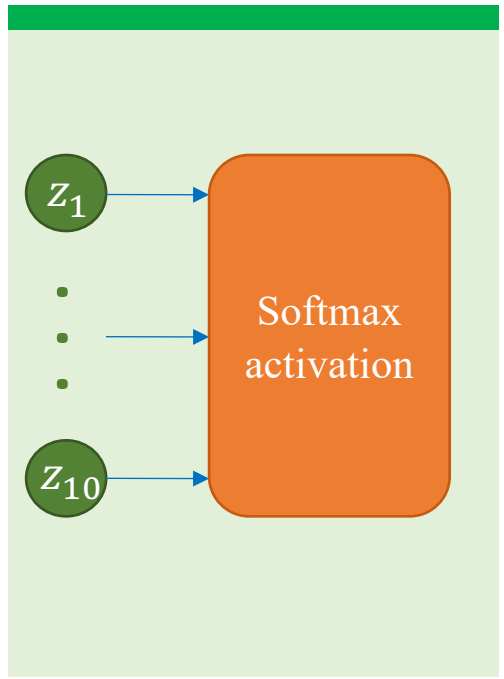
learning rate = 0.00001





784 Nodes
Input layer

Fully connect



Output

10 Nodes
Output layer

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

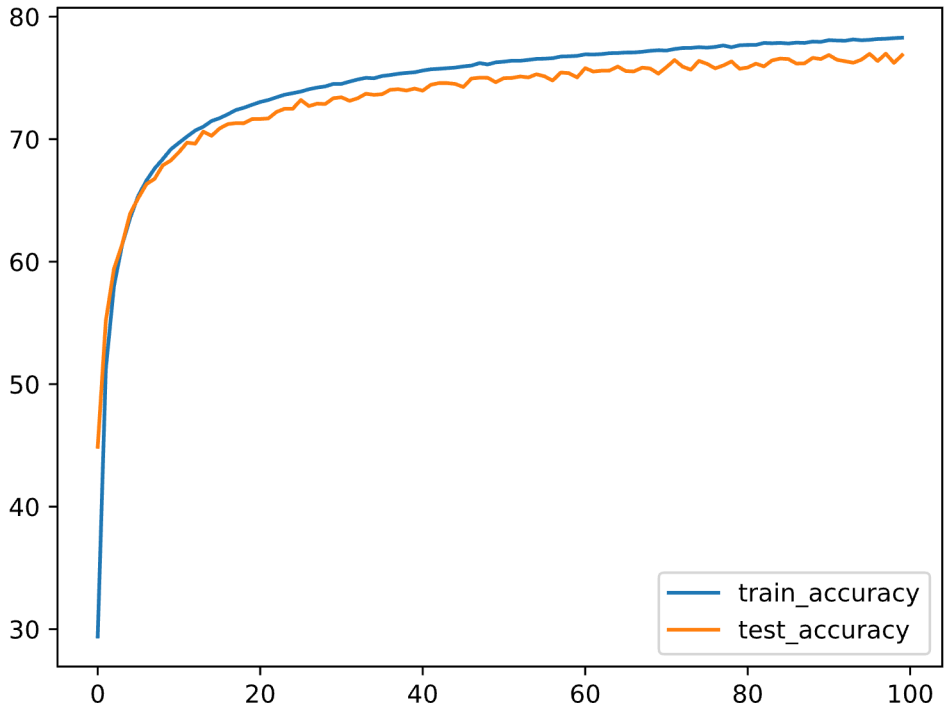
Data Sets

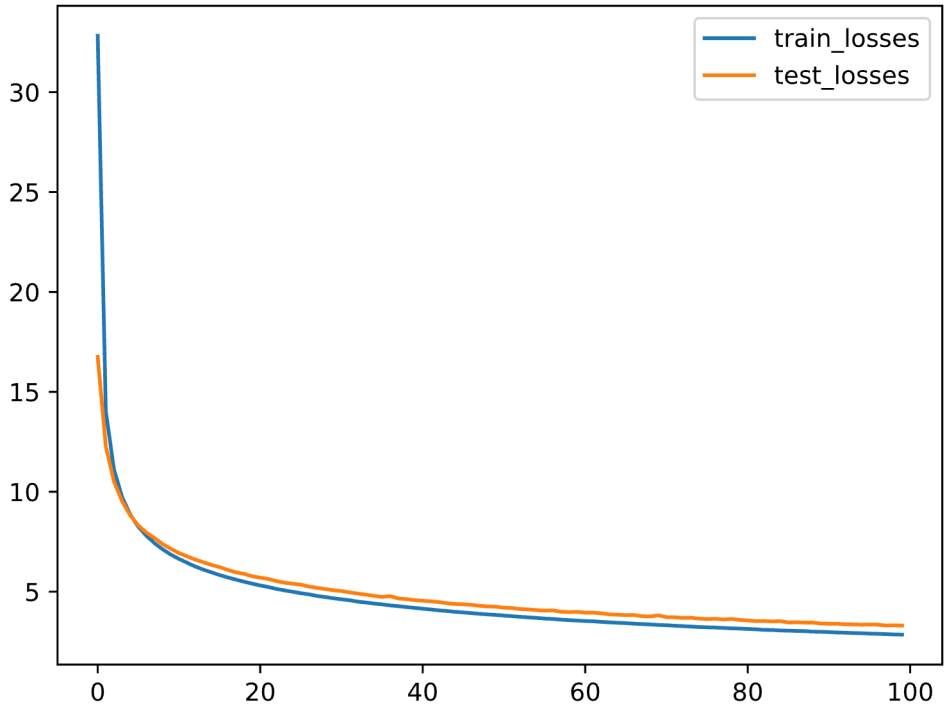
```python
import torch.nn as nn

model = nn.Sequential(
    nn.Flatten(), nn.Linear(784, 10)
)
print(model)
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=10, bias=True)
)
```

```python
# Generating a random tensor
input_tensor = torch.rand(5, 28, 28)

# Feeding the tensor into the model
output = model(input_tensor)
print(output.shape)
```

```
torch.Size([5, 10])
```






```
model = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
model = model.to(device)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.00001)
```

```
# train
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()

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

Implementation

In Theory

$X \in [0, 255]$

Convert to the range [0,1]

$$\text{Image} = \frac{\text{Image}}{255}$$

Convert to the range [-1,1]

$$\text{Image} = \frac{\text{Image}}{127.5} - 1$$

Z-score normalization

$$\text{Image} = \frac{\text{Image} - \mu}{\sigma}$$

In Pytorch

$X \in [0, 1]$

Normalize($mean$, std)

$$\text{Image} = \frac{\text{Image} - \text{mean}}{\text{std}}$$

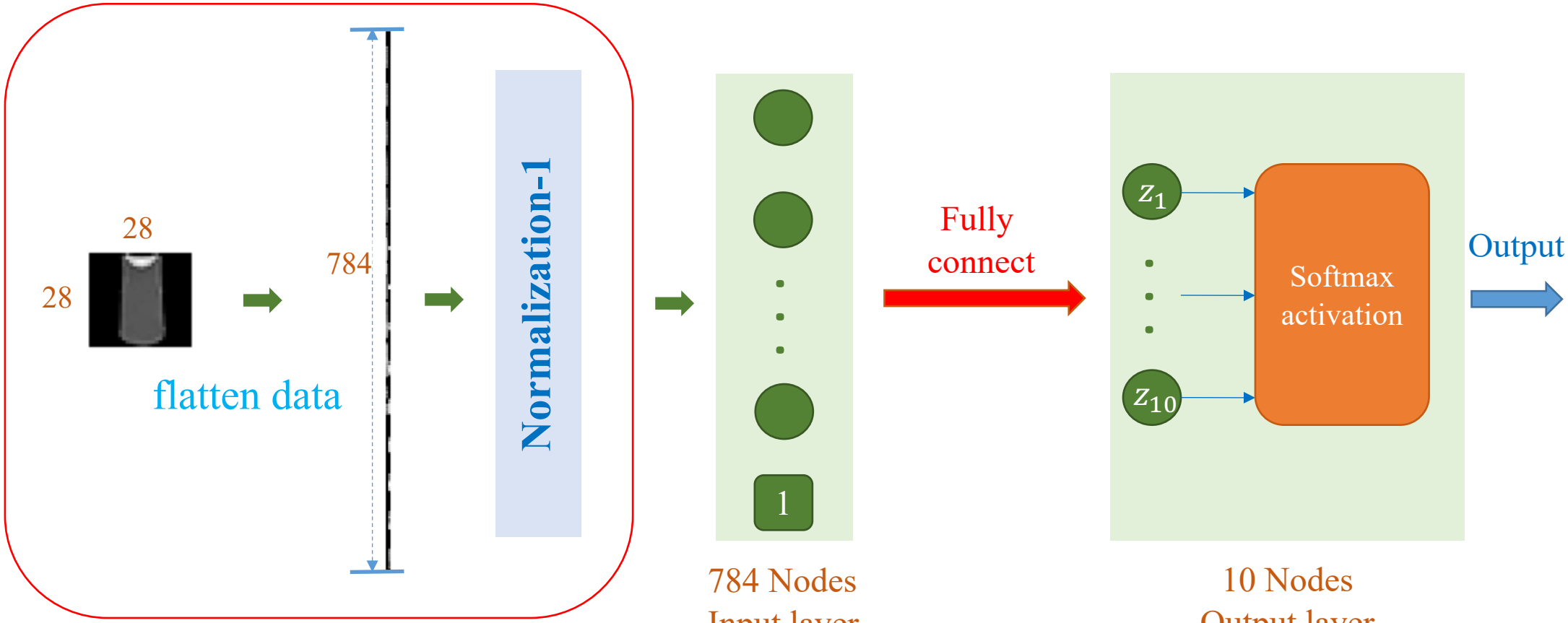
[0,1]	mean = 0 ; std = 1
[-1,1]	mean = 0.5; std = 0.5

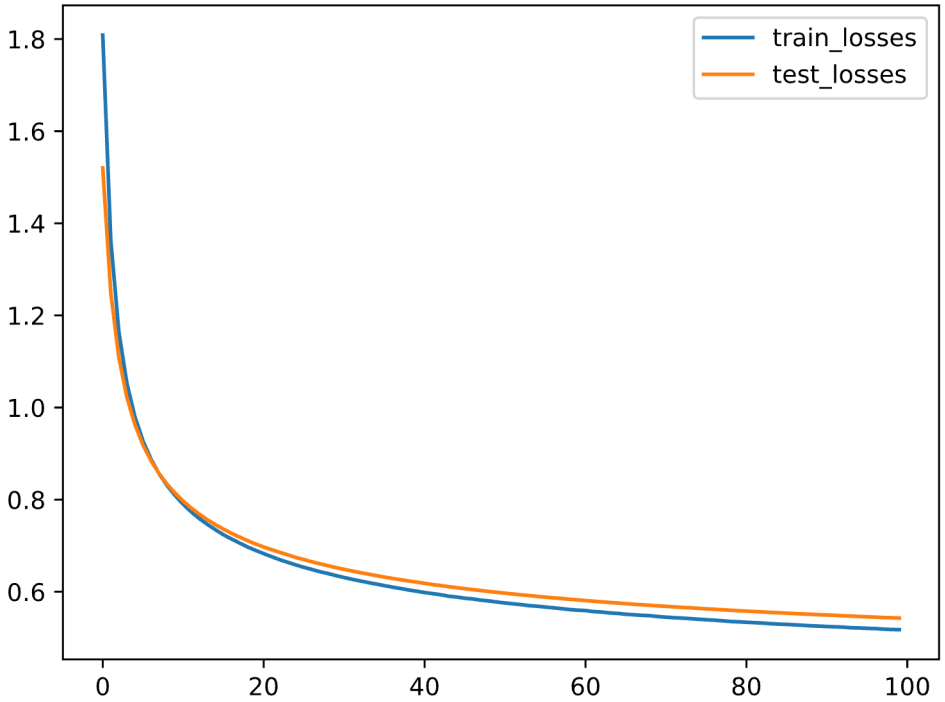
Compute mean and std from data

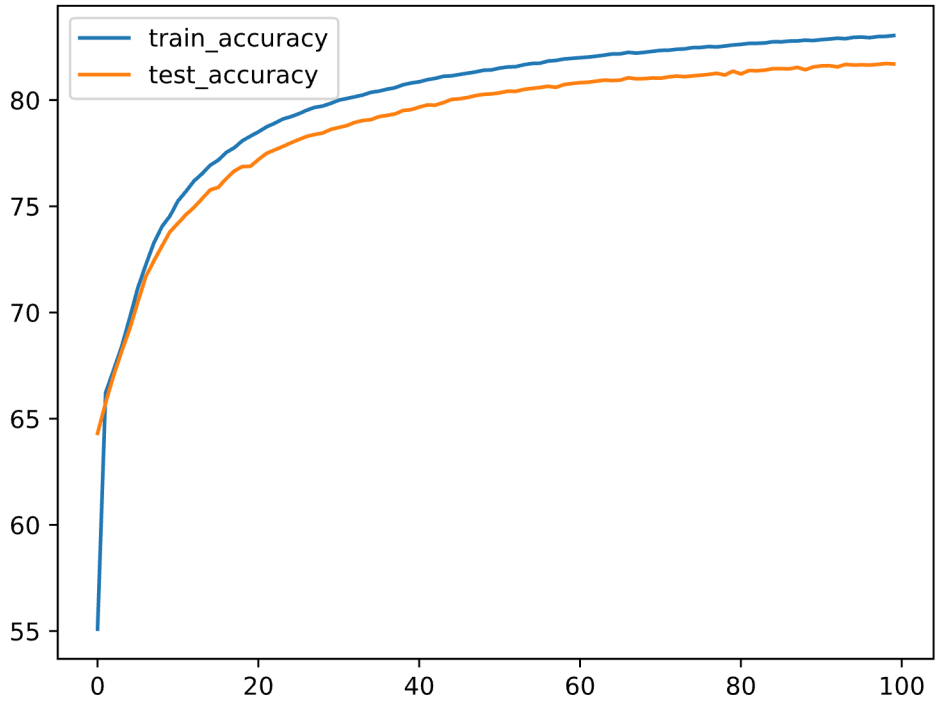
Softmax Regression + Normalization

$$\text{Image} = \frac{\text{Image}}{255}$$

```
import torchvision.transforms as transforms

transform = transforms.Compose([transforms.ToTensor()])
trainset = torchvision.datasets.FashionMNIST(root='data',
                                             train=True,
                                             download=True,
                                             transform=transform)
```








```python
model = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
model = model.to(device)
```

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

```python
# train
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()

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

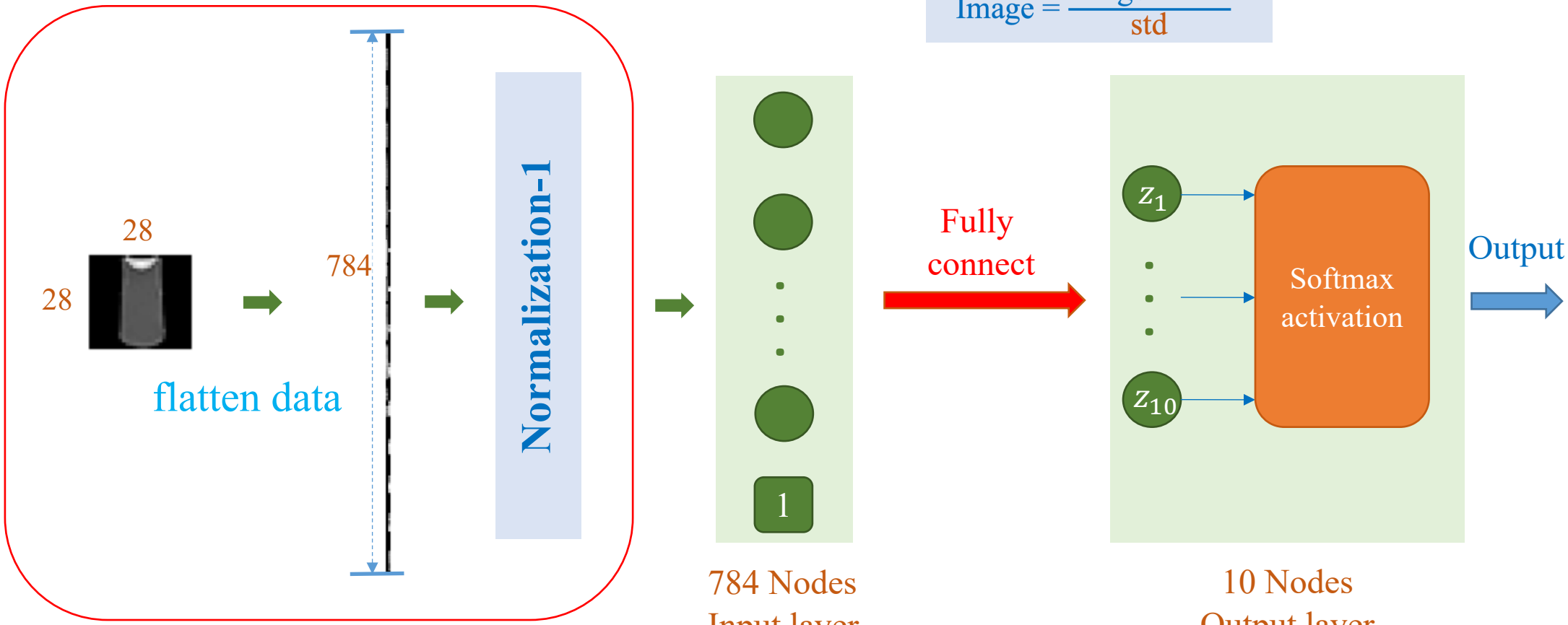
Softmax Regression + Normalization

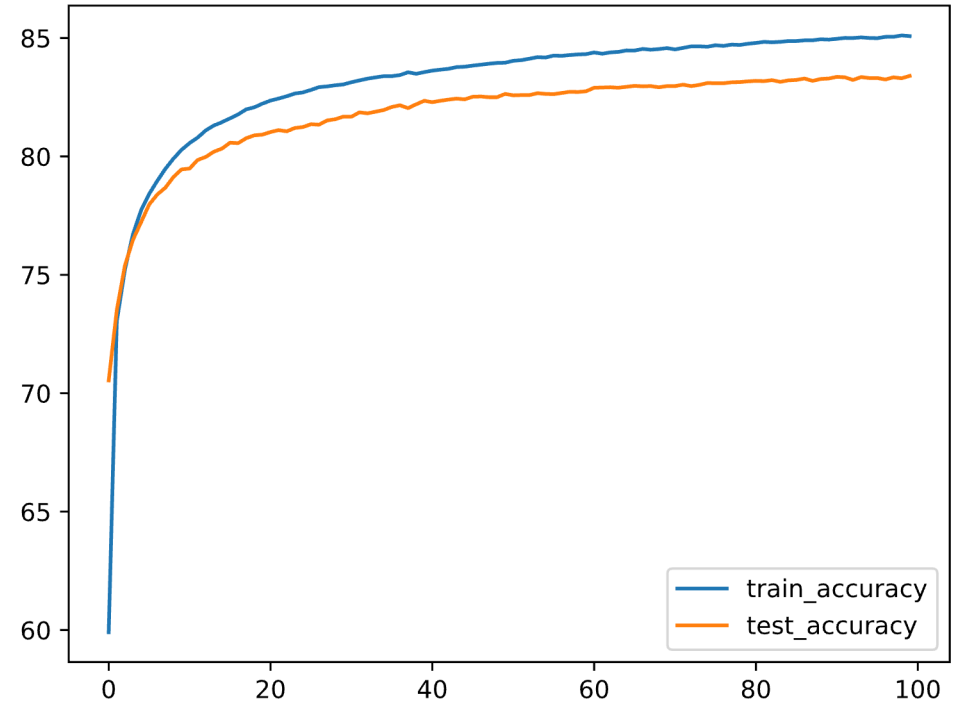
$$\text{Image} = \frac{\text{Image}}{127.5} - 1$$

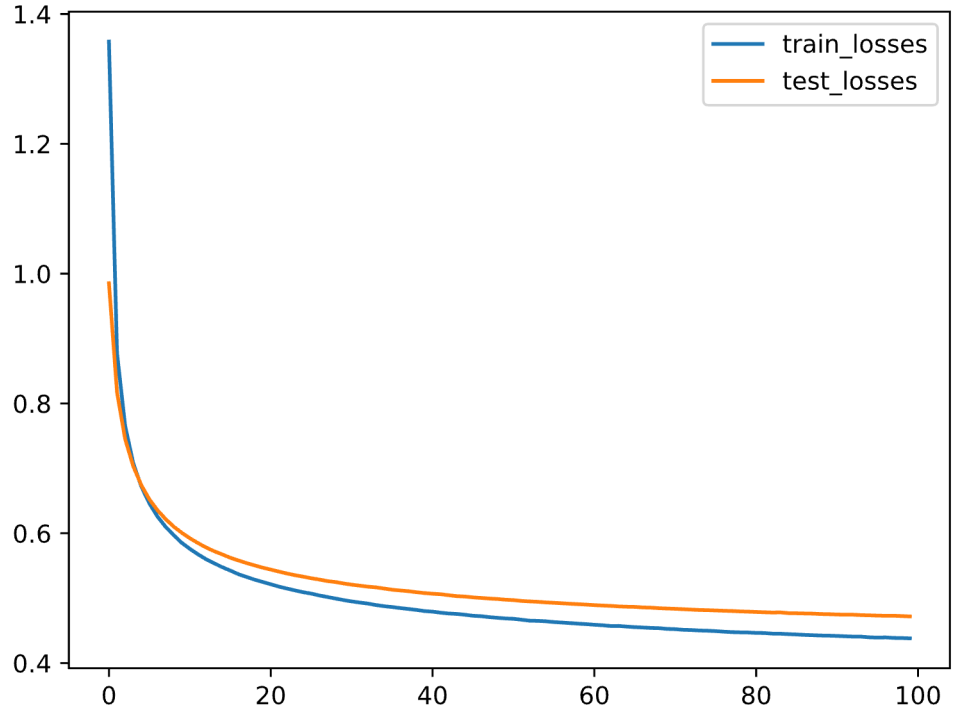
```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,),
                                                     (0.5,))])
trainset = torchvision.datasets.FashionMNIST(root='data',
                                             train=True,
                                             download=True,
                                             transform=transform)
```

Normalize(*mean*, std)

$$\text{Image} = \frac{\text{Image} - \text{mean}}{\text{std}}$$








```python
model = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
model = model.to(device)
```

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

```python
# train
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()

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

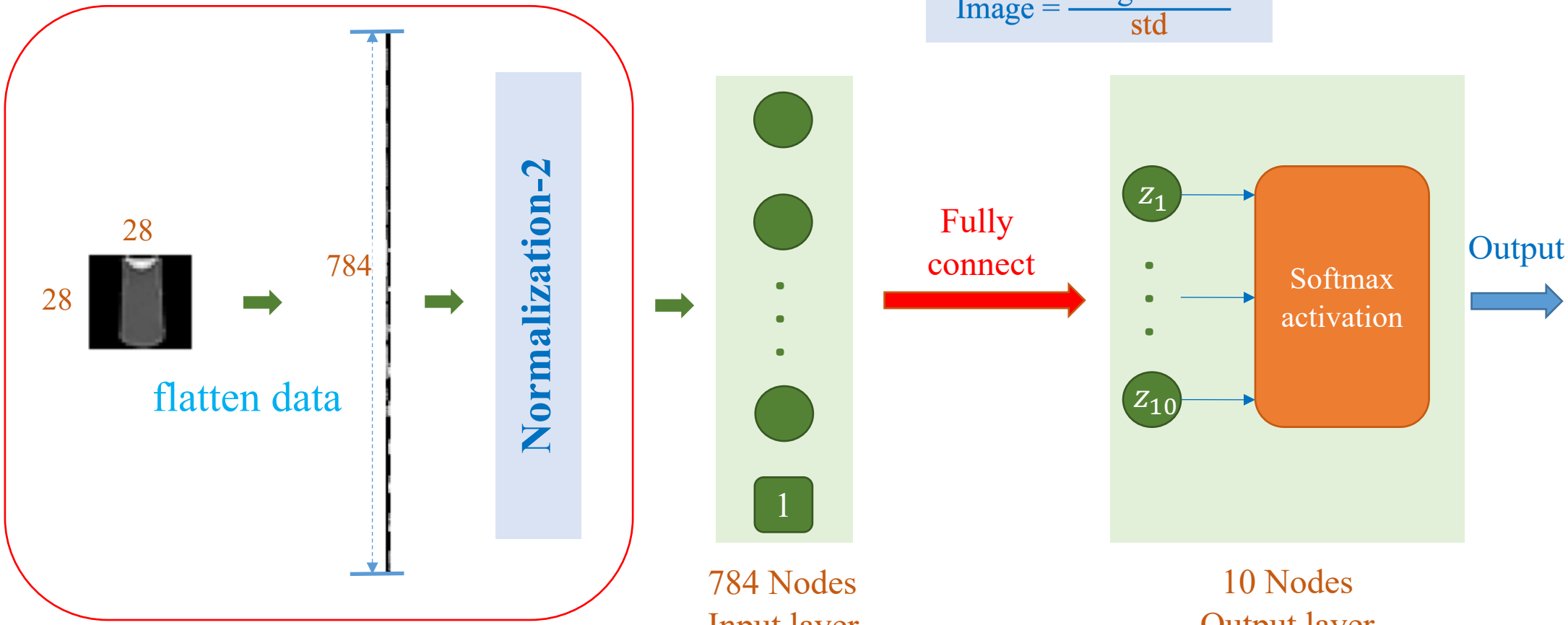
Softmax Regression + Normalization

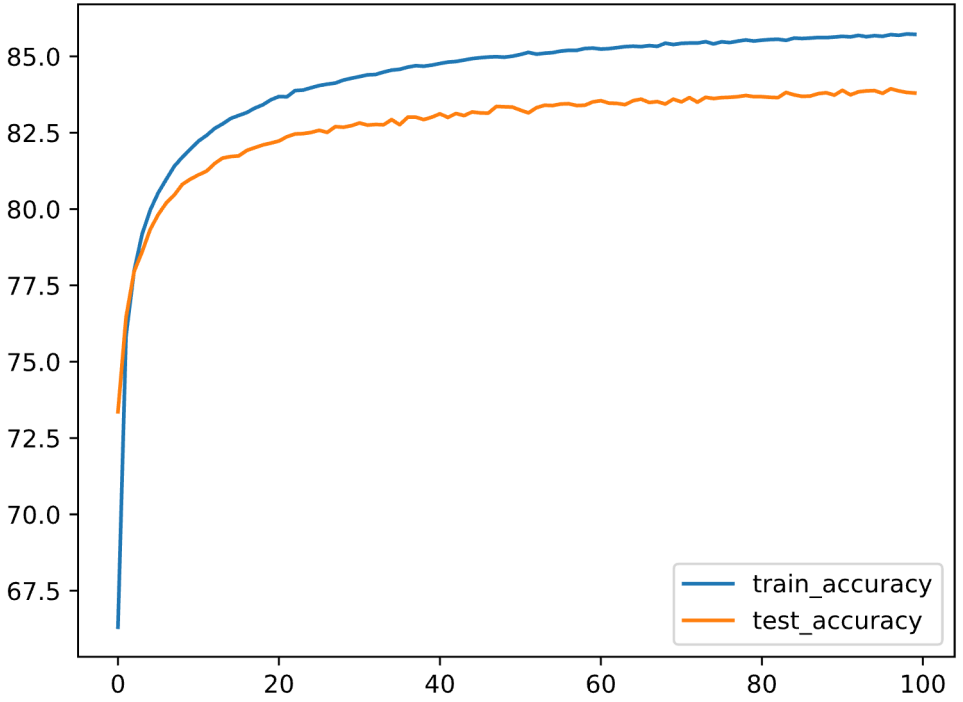
$$\text{Image} = \frac{\text{Image} - \mu}{\sigma}$$

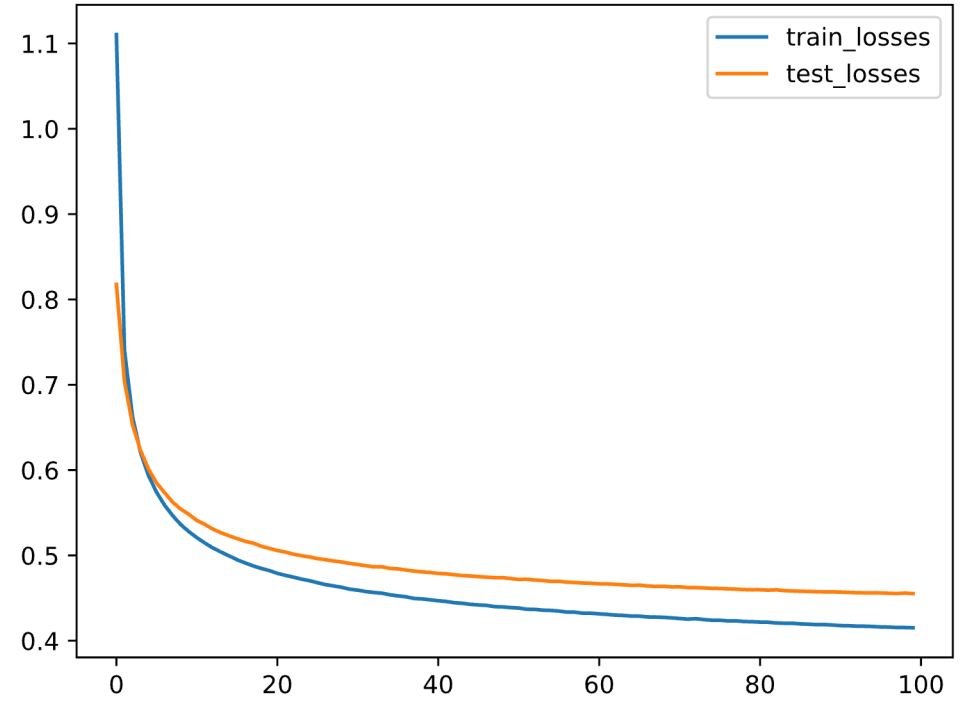
```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((mean,),
                                                     (std,))])
trainset = torchvision.datasets.FashionMNIST(root='data',
                                             train=True,
                                             download=True,
                                             transform=transform)
```

Normalize(*mean*, std)

$$\text{Image} = \frac{\text{Image} - \text{mean}}{\text{std}}$$








```python
model = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
model = model.to(device)
```

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

```python
# train
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()

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

Outline

SECTION 1

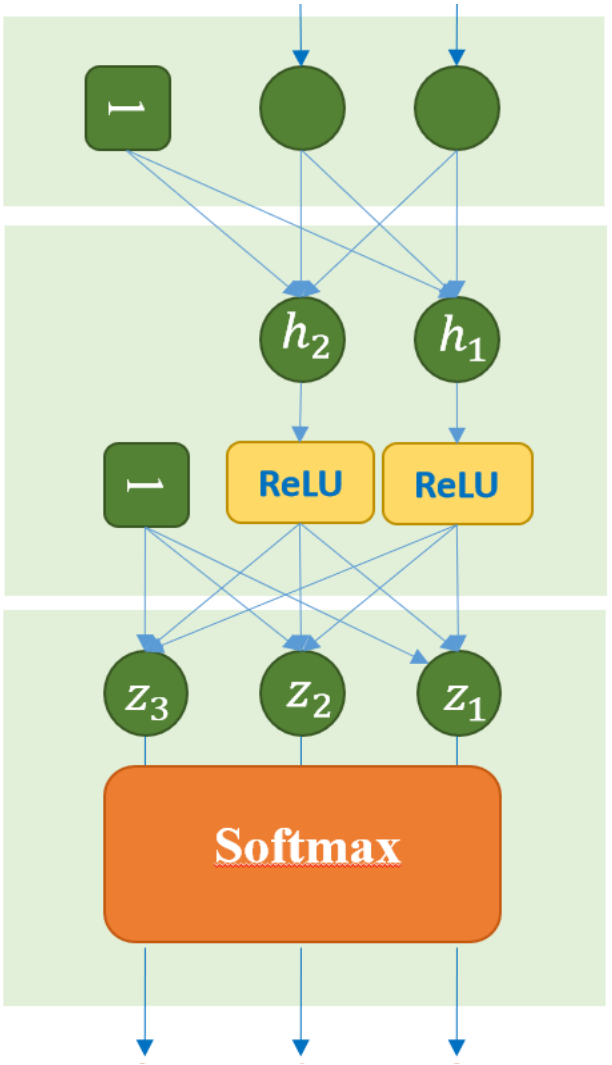
Image Data

SECTION 2

S. R. for Image Data

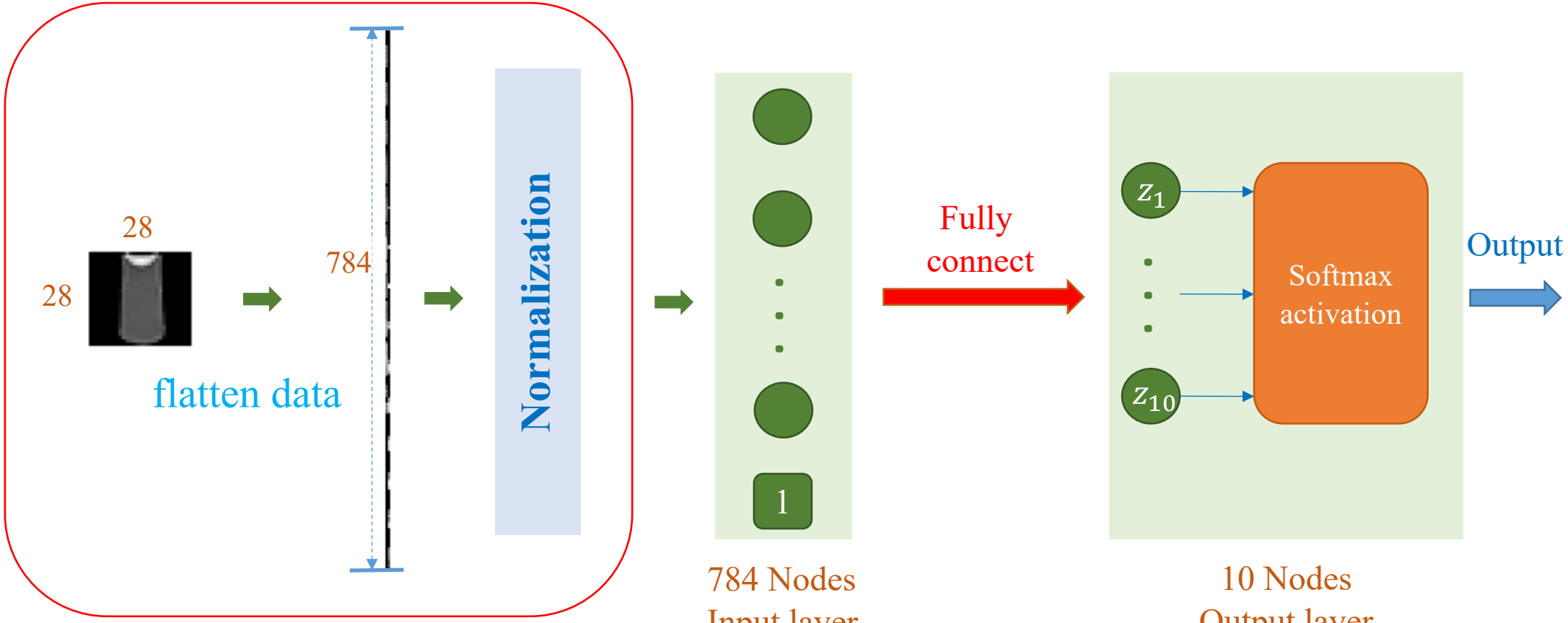
SECTION 3

MLP for Image Data



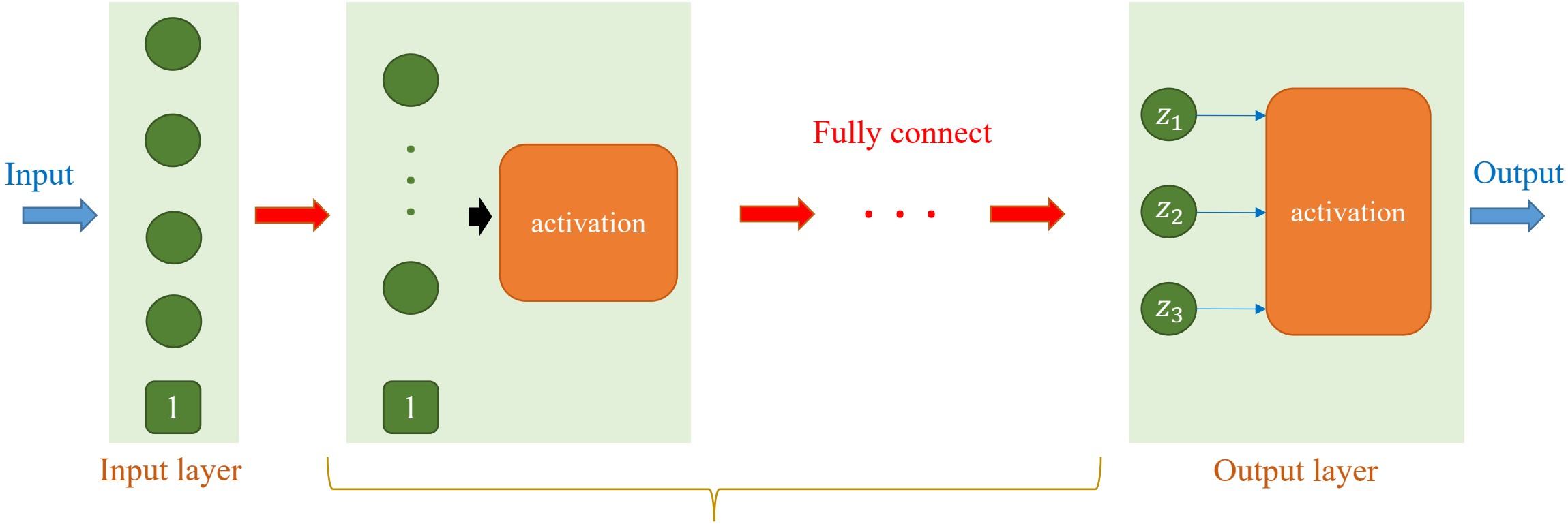
MLP - Motivation

- John Von Neumann's quote "with four parameters I can fit an elephant, with five I can make him wiggle his trunk"
- More parameters → better capacity (~stronger model)



Multi-layer Perceptron

- An idea: More parameters → better capacity (~stronger model)
  - Adding more layers



#hidden layers is arbitrary
#nodes in a hidden layer is arbitrary

Multi-layer Perceptron

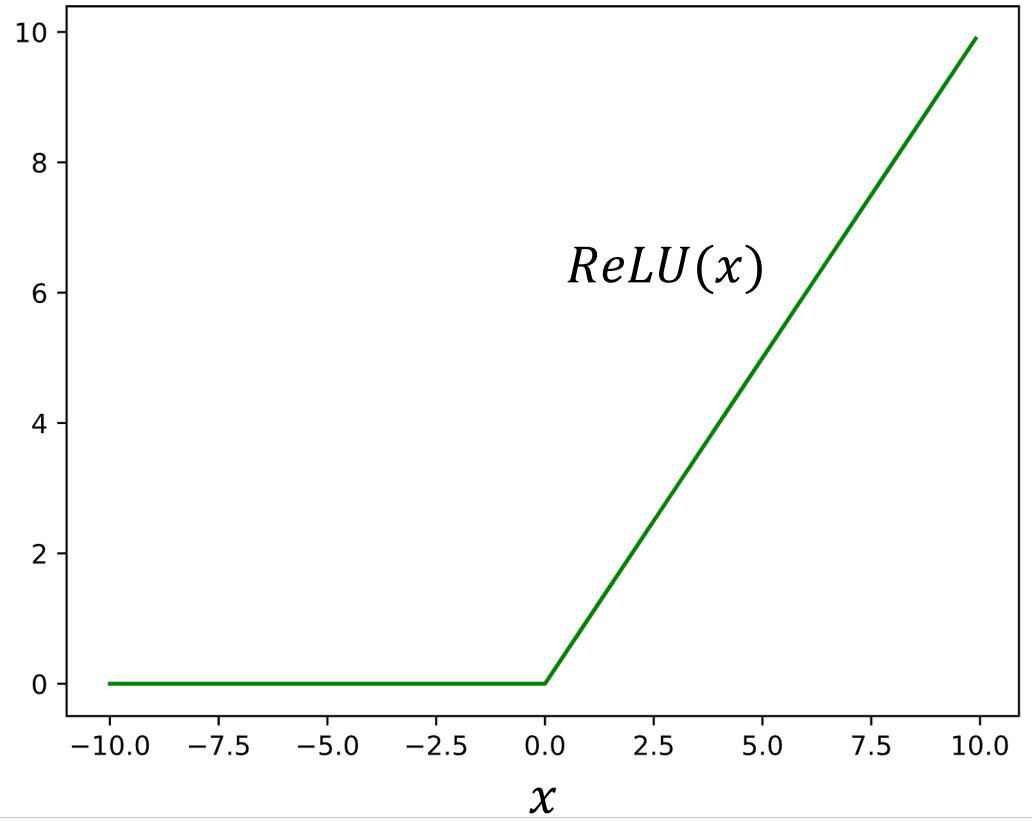
❖ ReLU function

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

data = | 1 | 5 | -4 | 3 | -2 |

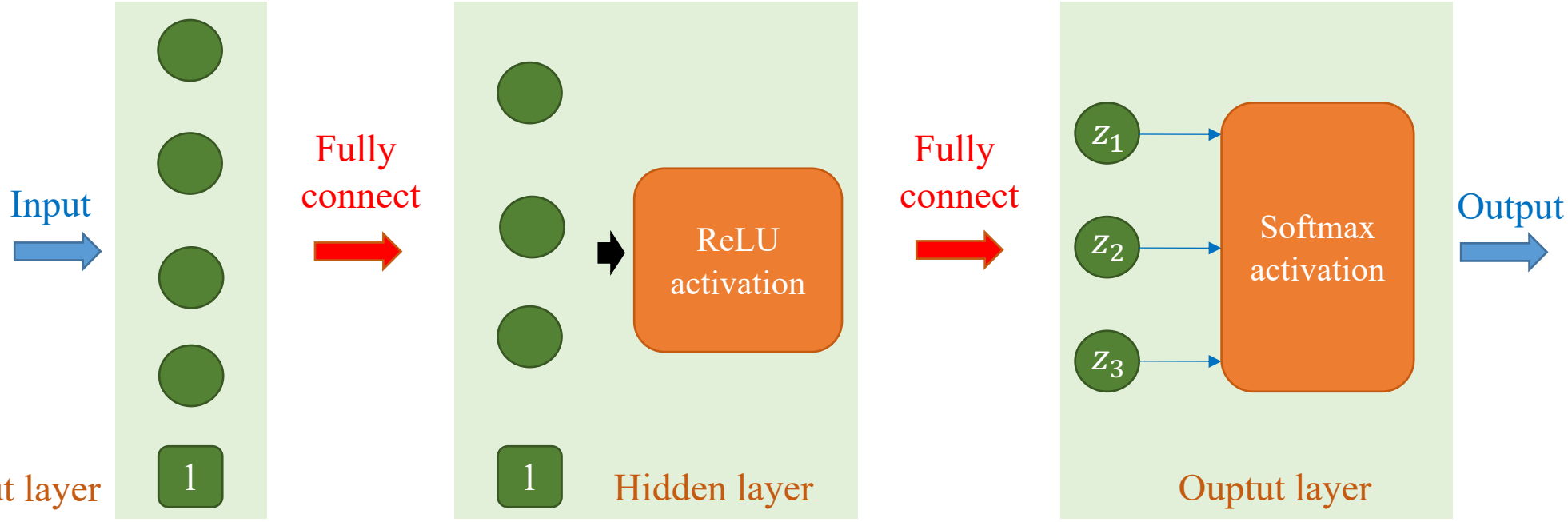
data_a = ReLU(data)

data_a = | 1 | 5 | 0 | 3 | 0 |



Multi-layer Perceptron

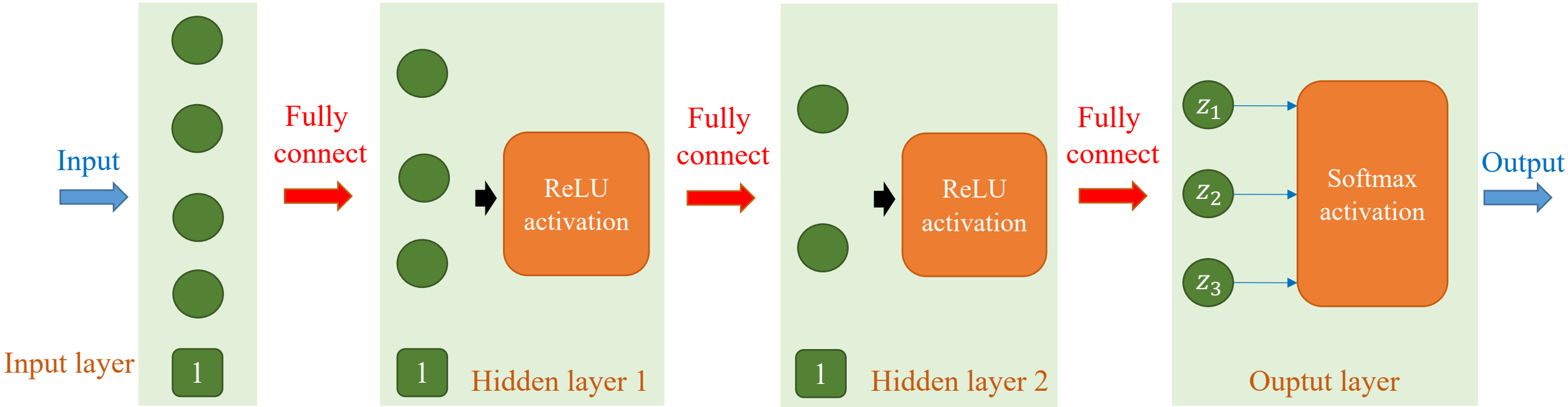
❖ An instance




```python
import torch.nn as nn

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

```
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 3]              15
              ReLU-2                [-1, 3]               0
            Linear-3                [-1, 3]              12
================================================================
Total params: 27
Trainable params: 27
Non-trainable params: 0
```

Multi-layer Perceptron




```python
import torch.nn as nn

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

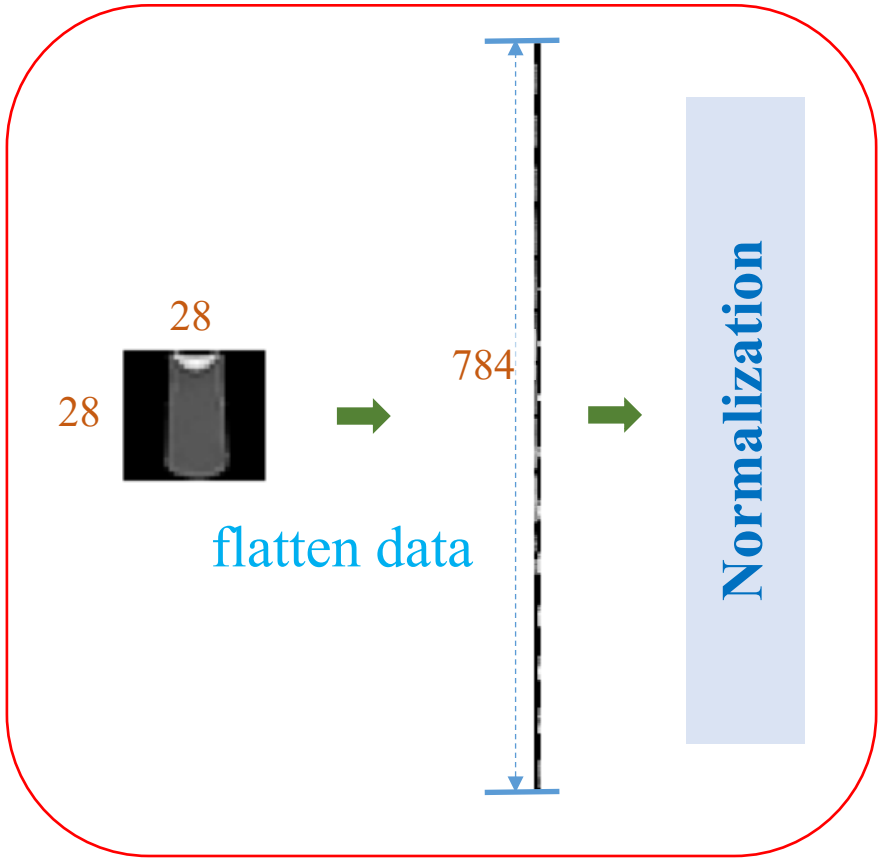
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 3]              15
              ReLU-2                [-1, 3]               0
            Linear-3                [-1, 2]               8
              ReLU-4                [-1, 2]               0
            Linear-5                [-1, 3]               9
================================================================
Total params: 32
```

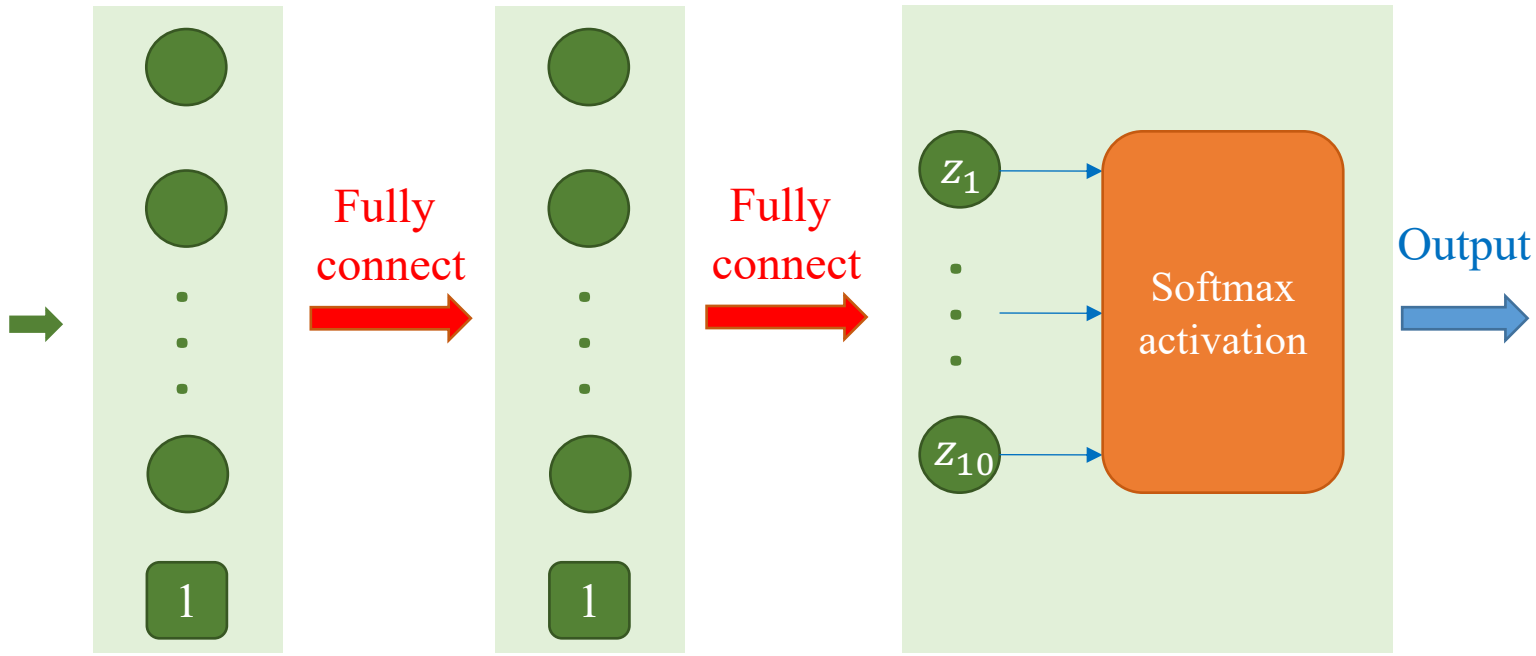
Back to Fashion-MNIST

$$\text{Image} = \frac{\text{Image}}{255.0}$$

```
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
print(model)
```

```
Sequential(
  (0): Linear(in_features=784, out_features=256, bias=True)
  (1): ReLU()
  (2): Linear(in_features=256, out_features=10, bias=True)
)
```





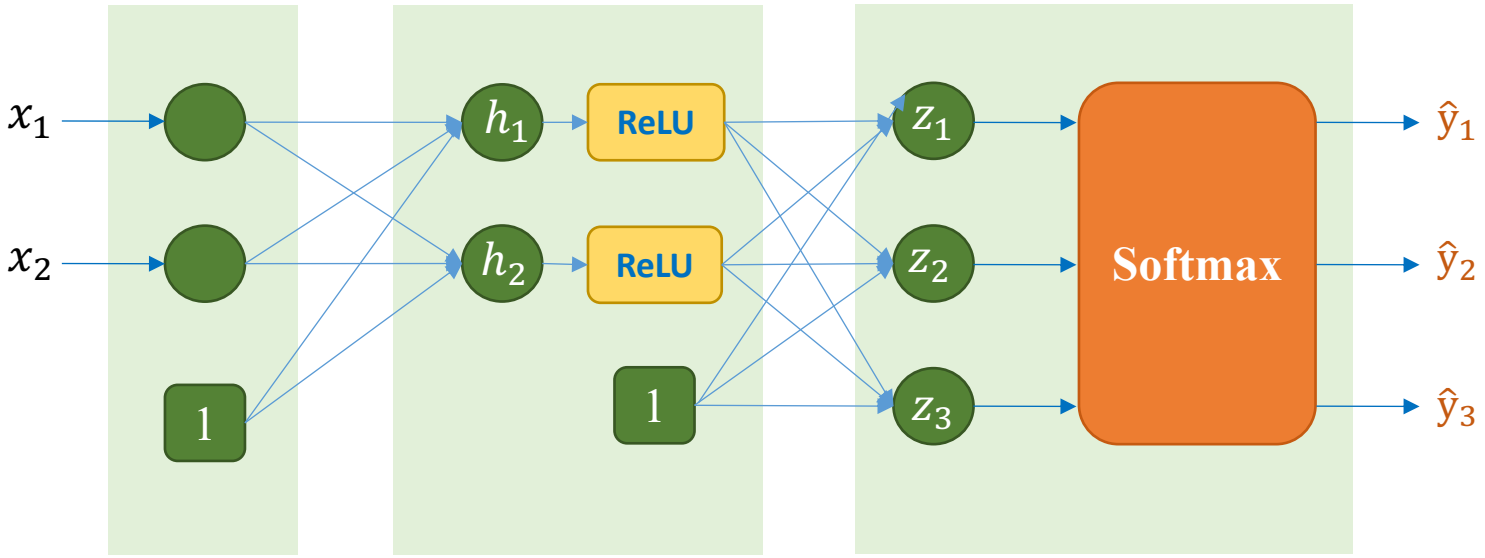
784 Nodes
Input layer

256 Nodes
Hidden layer

10 Nodes
Output layer

MLP Example

Feature		Label
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
5.6	2.2	2
5.1	1.5	2
5.6	1.4	2



$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} 1.5 & 0.2 \\ 4.7 & 1.6 \\ 5.6 & 2.2 \end{bmatrix}$$

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

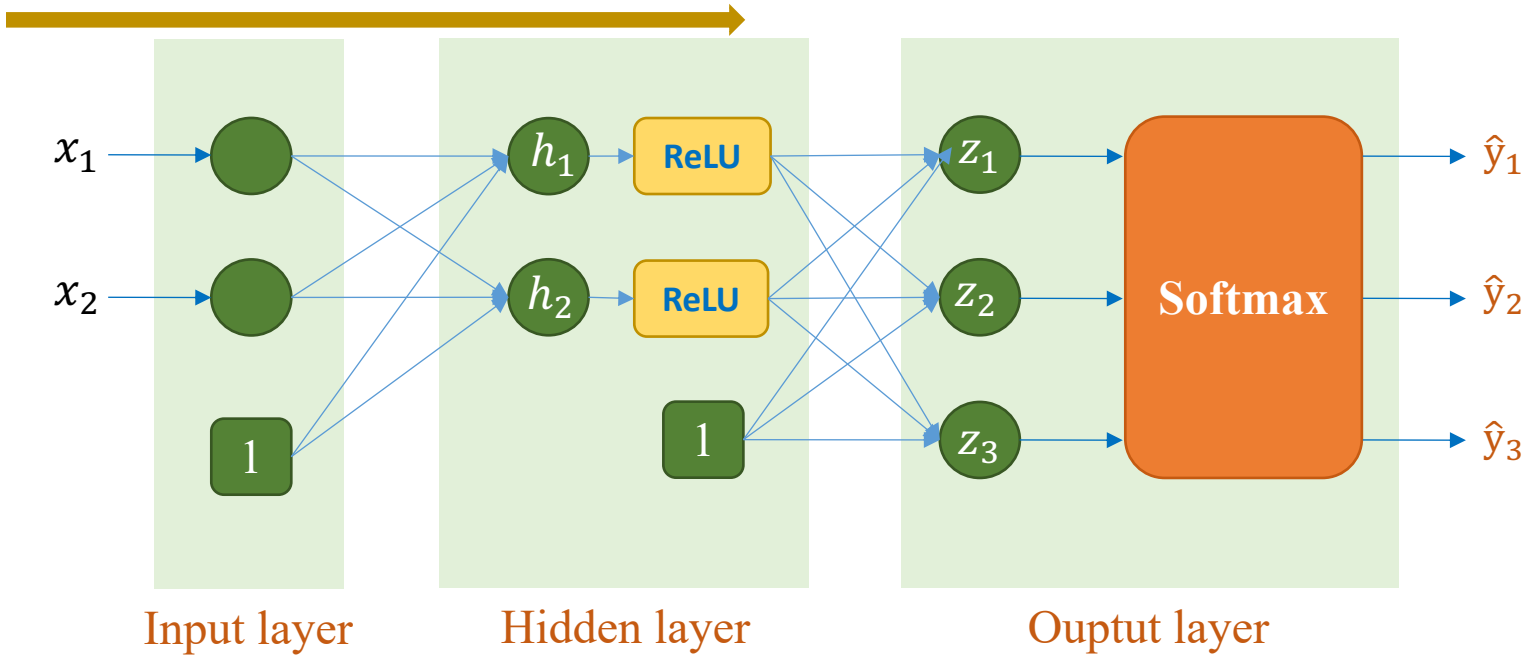
$$\boldsymbol{W}_h = [\boldsymbol{W}_{h1} \quad \boldsymbol{W}_{h2}] = \begin{bmatrix} 0.0 & 0.0 \\ 0.86 & -1.04 \\ 0.41 & -0.65 \end{bmatrix}$$

$$\boldsymbol{W}_z = [\boldsymbol{W}_{z1} \quad \boldsymbol{W}_{z2} \quad \boldsymbol{W}_{z3}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.32 & 0.25 & 0.14 \\ -0.47 & -1.06 & 0.063 \end{bmatrix}$$

$$\boldsymbol{h} = \boldsymbol{x}\boldsymbol{W}_h = \begin{bmatrix} 1 & 1.5 & 0.2 \\ 1 & 4.7 & 1.6 \\ 1 & 5.6 & 2.2 \end{bmatrix} \begin{bmatrix} 0.0 & 0.0 \\ 0.86 & -1.04 \\ 0.41 & -0.65 \end{bmatrix} = \begin{bmatrix} 1.373 & -1.696 \\ 4.708 & -5.951 \\ 5.731 & -7.281 \end{bmatrix}$$

$$\text{ReLU}(\boldsymbol{h}) = \begin{bmatrix} 1.373 & 0 \\ 4.708 & 0 \\ 5.731 & 0 \end{bmatrix}$$

Feature		Label
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
5.6	2.2	2
5.1	1.5	2
5.6	1.4	2



$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} 1 & 1.5 & 0.2 \\ 1 & 4.7 & 1.6 \\ 1 & 5.6 & 2.2 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$\boldsymbol{W}_h = [\boldsymbol{W}_{h1} \quad \boldsymbol{W}_{h2}] \qquad \boldsymbol{W}_z = [\boldsymbol{W}_{z1} \quad \boldsymbol{W}_{z2} \quad \boldsymbol{W}_{z3}]$$

$$= \begin{bmatrix} 0.0 & 0.0 \\ 0.86 & -1.04 \\ 0.41 & -0.65 \end{bmatrix} \qquad = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.32 & 0.25 & 0.14 \\ -0.47 & -1.06 & 0.063 \end{bmatrix}$$

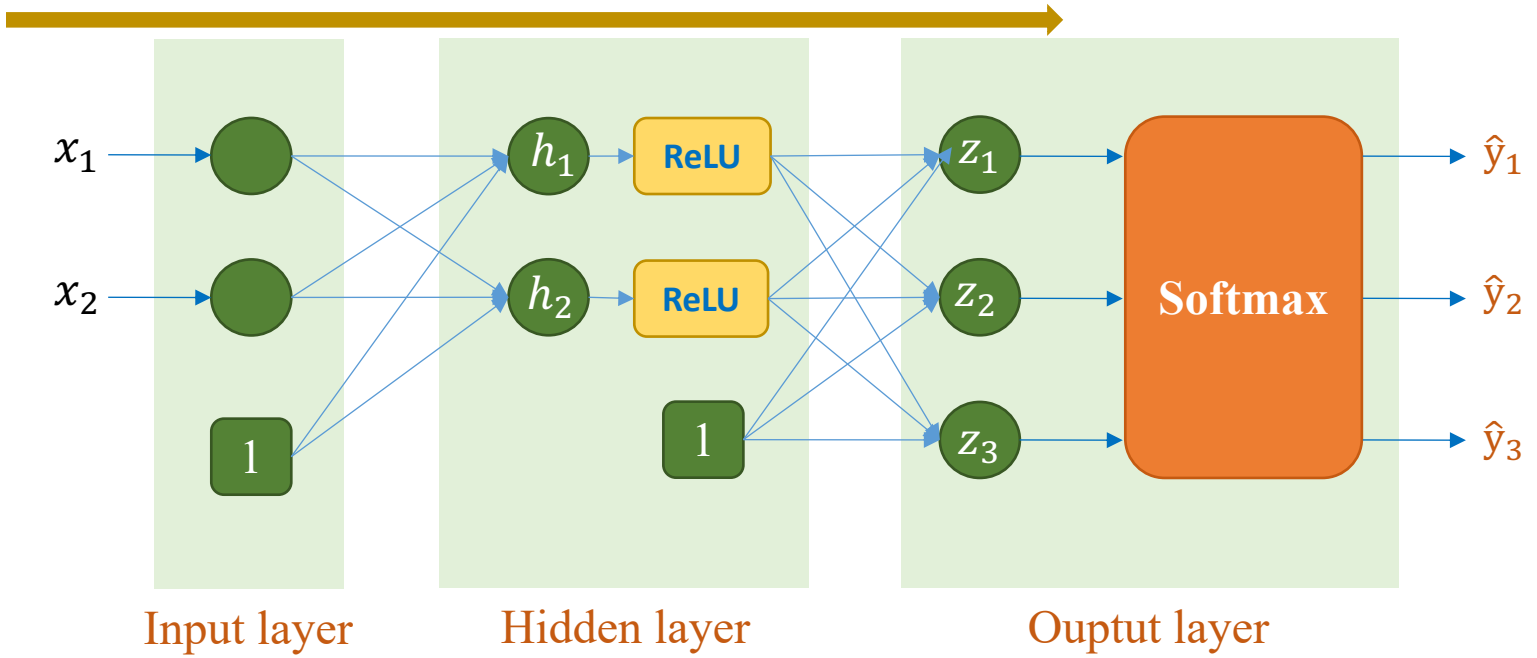
$$\text{ReLU}(\boldsymbol{h}) = \begin{bmatrix} 1.373 & 0 \\ 4.708 & 0 \\ 5.731 & 0 \end{bmatrix}$$

$$[\mathbf{1} \quad \text{ReLU}(\boldsymbol{h})] = \begin{bmatrix} 1 & 1.373 & 0 \\ 1 & 4.708 & 0 \\ 1 & 5.731 & 0 \end{bmatrix}$$

$$\boldsymbol{z} = [\mathbf{1} \quad \text{ReLU}(\boldsymbol{h})]\boldsymbol{W}_z = \begin{bmatrix} 1 & 1.373 & 0 \\ 1 & 4.708 & 0 \\ 1 & 5.731 & 0 \end{bmatrix} \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.32 & 0.25 & 0.14 \\ -0.47 & -1.06 & 0.063 \end{bmatrix}$$

$$= \begin{bmatrix} 0.439 & 0.356 & 0.195 \\ 1.507 & 1.220 & 0.670 \\ 1.835 & 1.485 & 0.816 \end{bmatrix}$$

Feature		Label
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
5.6	2.2	2
5.1	1.5	2
5.6	1.4	2



$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \\ \boldsymbol{x}^{(3)} \end{bmatrix} = \begin{bmatrix} 1 & 1.5 & 0.2 \\ 1 & 4.7 & 1.6 \\ 1 & 5.6 & 2.2 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$\boldsymbol{W}_h = [\boldsymbol{W}_{h1} \quad \boldsymbol{W}_{h2}] \qquad \boldsymbol{W}_z = [\boldsymbol{W}_{z1} \quad \boldsymbol{W}_{z2} \quad \boldsymbol{W}_{z3}]$$

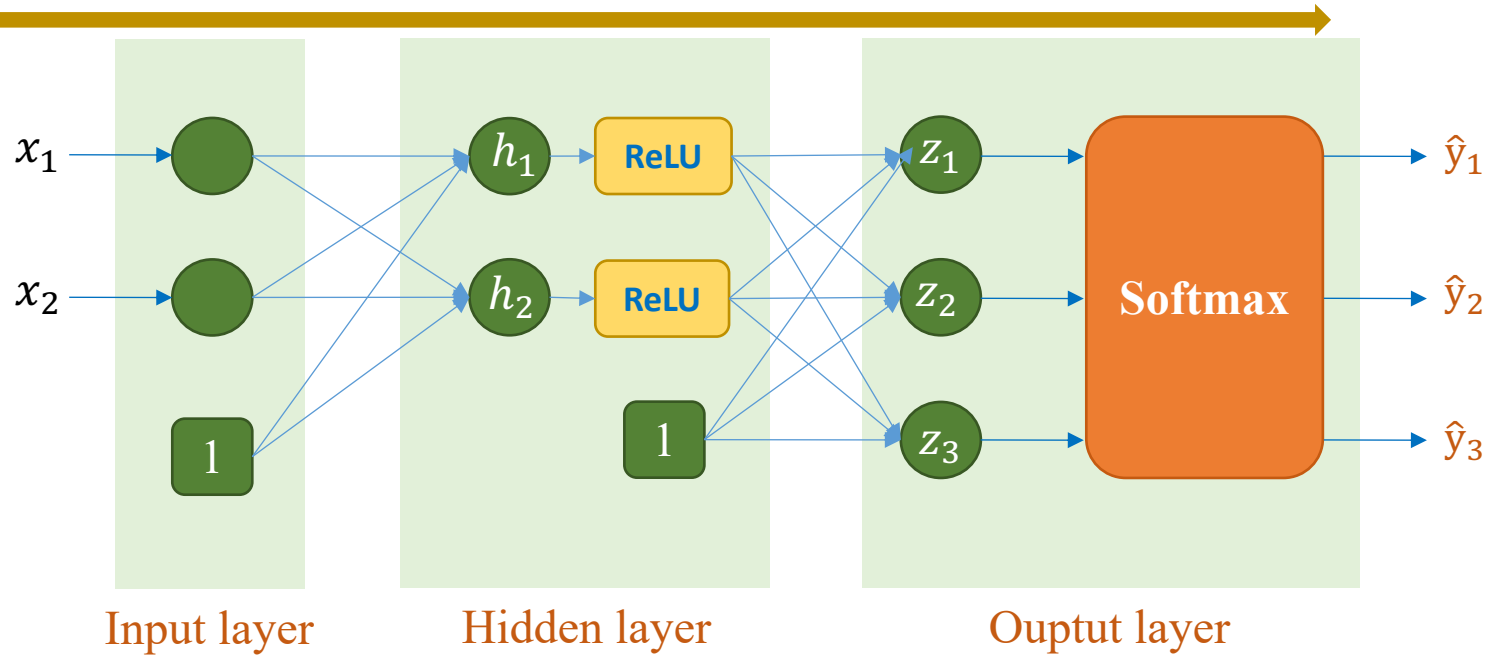
$$= \begin{bmatrix} 0.0 & 0.0 \\ 0.86 & -1.04 \\ 0.41 & -0.65 \end{bmatrix} \qquad = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.32 & 0.25 & 0.14 \\ -0.47 & -1.06 & 0.063 \end{bmatrix}$$

$$z = \begin{bmatrix} 0.439 & 0.356 & 0.195 \\ 1.507 & 1.220 & 0.670 \\ 1.835 & 1.485 & 0.816 \end{bmatrix}$$

$$\hat{y} = \text{softmax}(z) = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \end{bmatrix} = \begin{bmatrix} 0.369 & 0.340 & 0.289 \\ 0.458 & 0.343 & 0.198 \\ 0.484 & 0.341 & 0.174 \end{bmatrix}$$

loss = 1.269

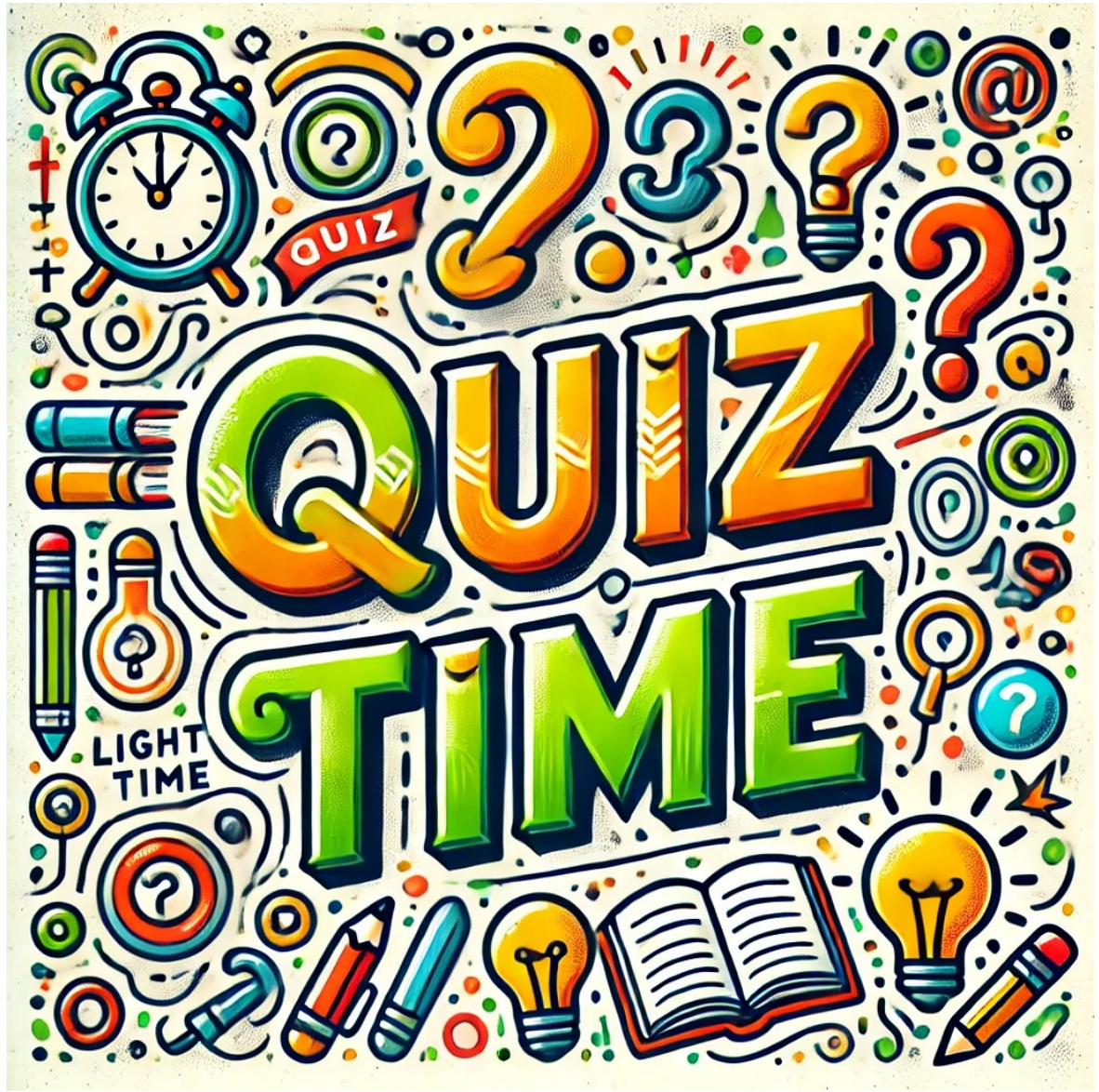
Feature		Label
Petal Length	Petal Width	Label
1.5	0.2	0
1.4	0.2	0
1.6	0.2	0
4.7	1.6	1
3.3	1.1	1
4.6	1.3	1
5.6	2.2	2
5.1	1.5	2
5.6	1.4	2



$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix} = \begin{bmatrix} 1 & 1.5 & 0.2 \\ 1 & 4.7 & 1.6 \\ 1 & 5.6 & 2.2 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$W_h = [W_{h1} \quad W_{h2}] = \begin{bmatrix} 0.0 & 0.0 \\ 0.86 & -1.04 \\ 0.41 & -0.65 \end{bmatrix}$$

$$W_z = [W_{z1} \quad W_{z2} \quad W_{z3}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.32 & 0.25 & 0.14 \\ -0.47 & -1.06 & 0.063 \end{bmatrix}$$


QUIZ
QUIZ
TIME
LIGHT
TIME

Question 1

❖ Model sau có bao nhiêu tham số?

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

a) Có 24 tham số

b) Có 28 tham số

c) Có 32 tham số

d) Có 36 tham số

Question 2

❖ Cho dữ liệu ảnh ở dạng 2D hoặc 3D, mô hình sau có nhận trực tiếp dữ liệu ảnh để train được không?

```
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```

a) Không được

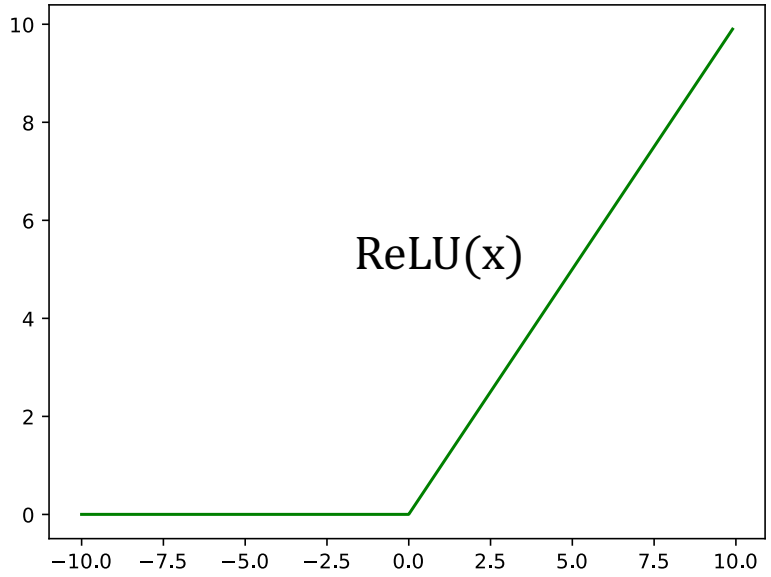
b) Nhận được cả ảnh 2D và 3D

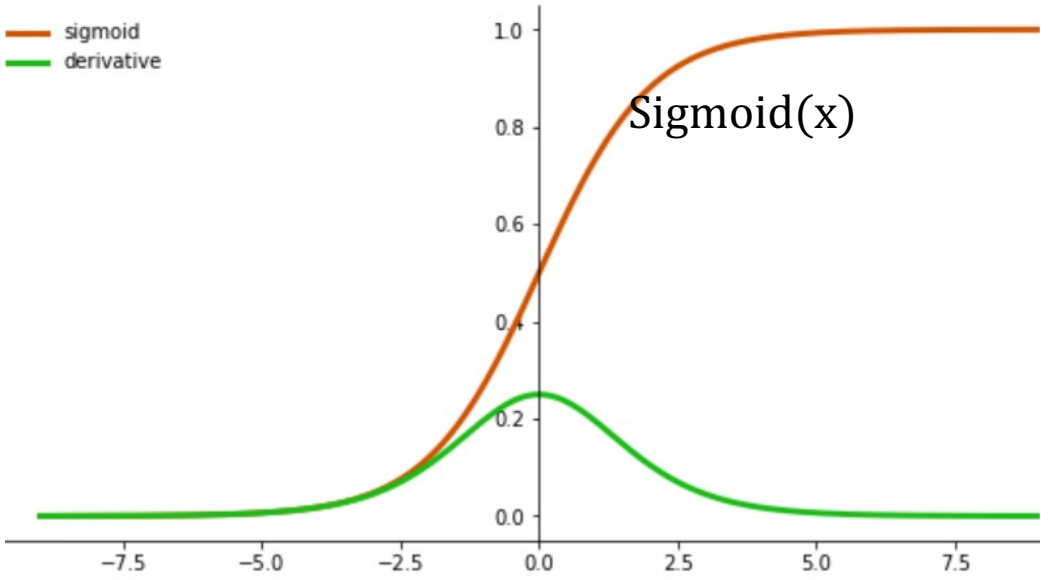
c) Chỉ nhận được ảnh 2D

d) Chỉ nhận được ảnh 3D

Question 3

❖ Hàm Sigmoid(.) hay ReLU(.) có giá trị đạo hàm lớn hơn khi xét trong cùng điều kiện input>0?





a) Hàm sigmoid(.)

b) Hàm ReLU(.)

c) Tùy thuộc từng đoạn

d) Không xác định

Question 4

❖ Data normalization có ảnh hưởng đến bước cập nhật trọng số không (bước update w)?

a) Có ảnh hưởng

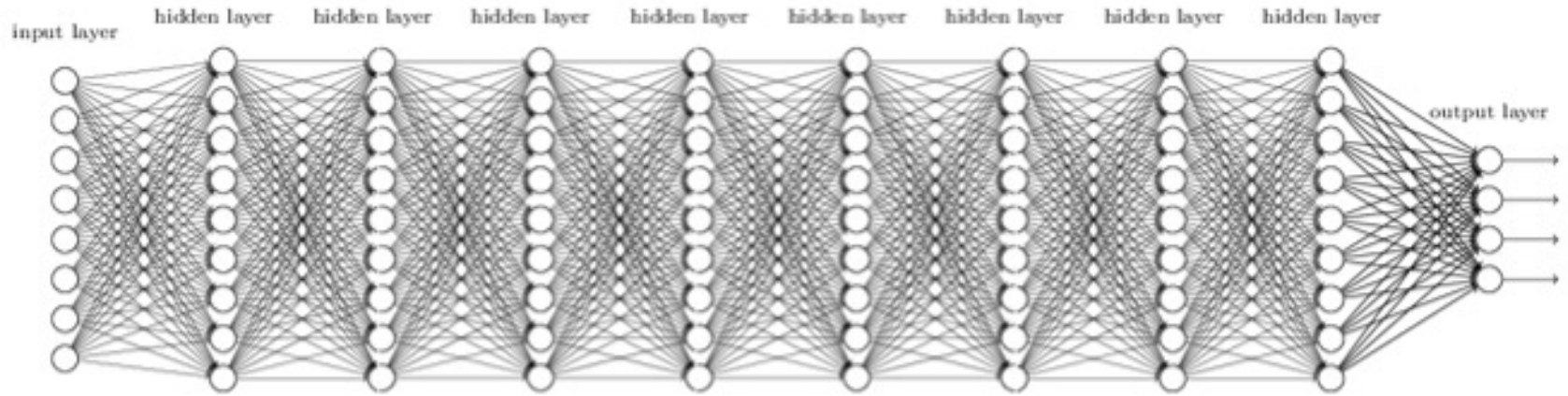
b) Không ảnh hưởng

c) Không xác định

d) Không biết nữa

Question 5

❖ Cho network như hình dưới, dùng activation sigmoid sẽ có vấn đề liên quan nhất đến bước nào?



a) Liên quan đến normalization

b) Liên quan đến số lượng category

c) … việc chọn số node trong 1 layer

d) … bước cập nhật trọng số w

Question 6

❖ MLP với 5 hidden layer có thể dùng được cho bài toán nào (với thiết kế layer cuối tùy ý)?

a) Phân loại nhị phân (2 class)

b) Phân loại đa lớp (2+ class)

c) Bài regression (như bài dự đoán giá nhà)

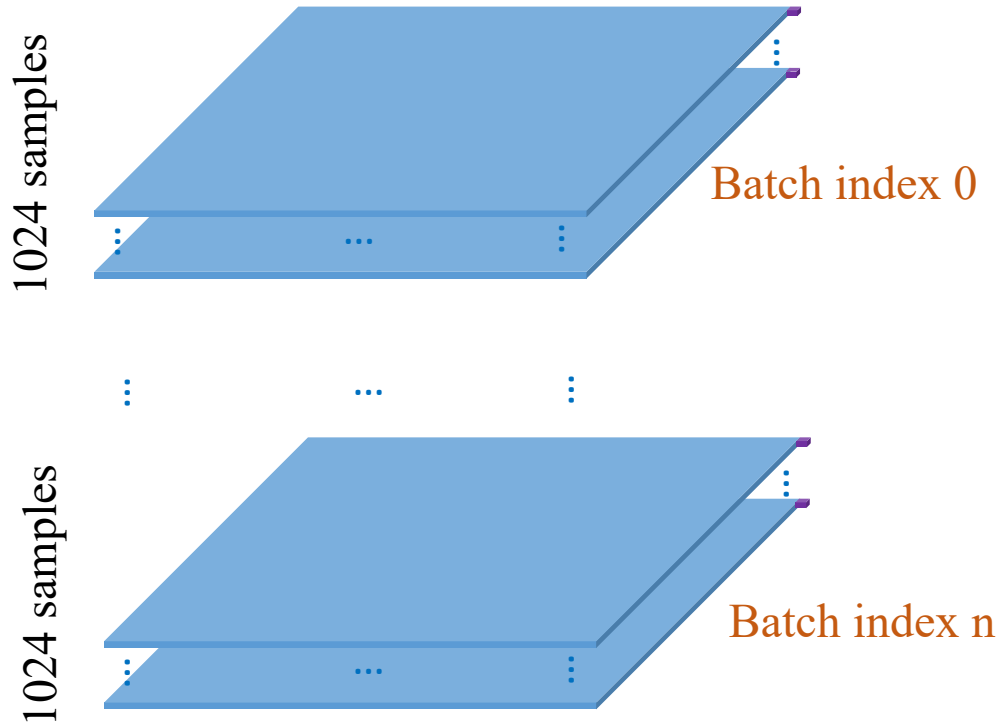
d) Các bài trên đều không áp dụng được

Step-by-Step Implementation

Step-by-Step Implementation

❖ 1. Data Preparation

Each sample is a tuple (image tensor, label)



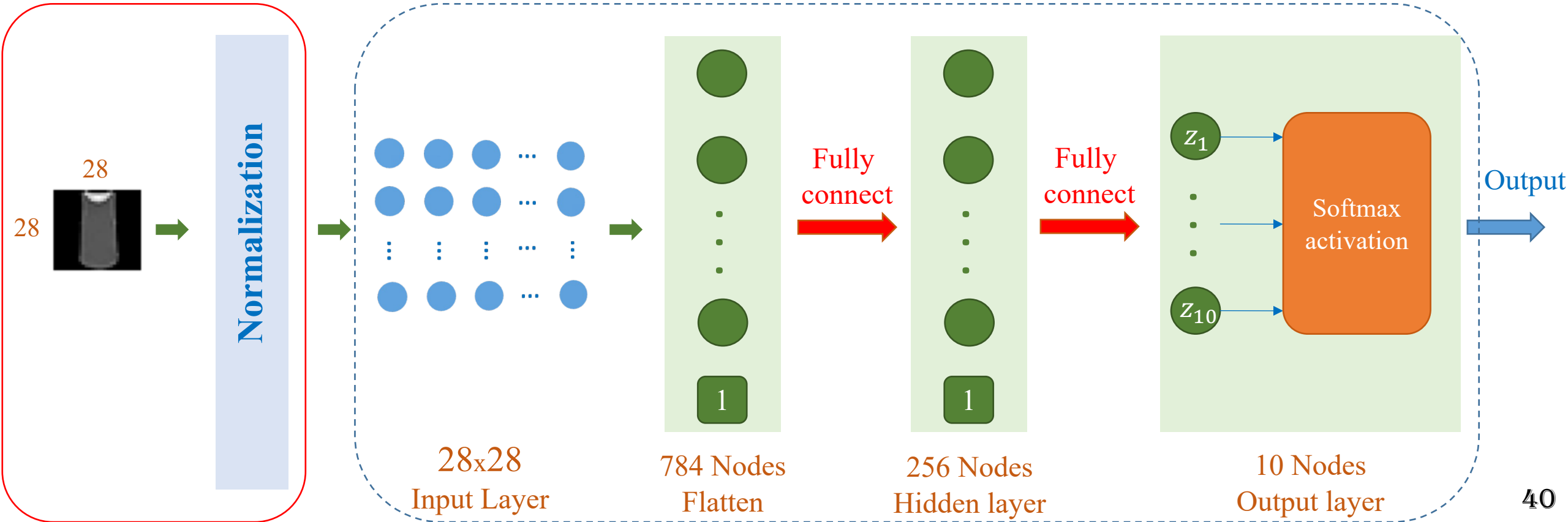

```
transform = T.Compose([T.ToTensor(),
                       T.Normalize((0.5,),
                                   (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                         batch_size=64,
                         shuffle=True)

testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=64,
                        shuffle=False)
```

Step-by-Step Implementation

❖ 2. Model, loss and optimizer

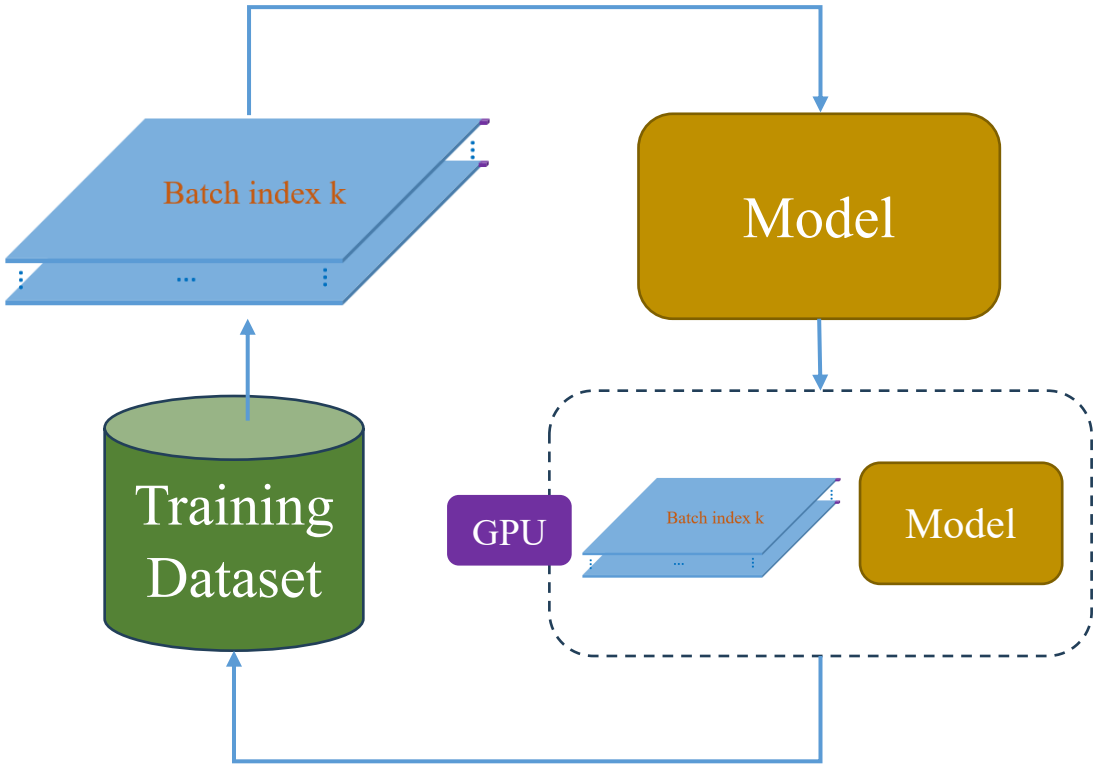
```
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

```
# Define the MLP model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```



Step-by-Step Implementation

❖ 3. Training

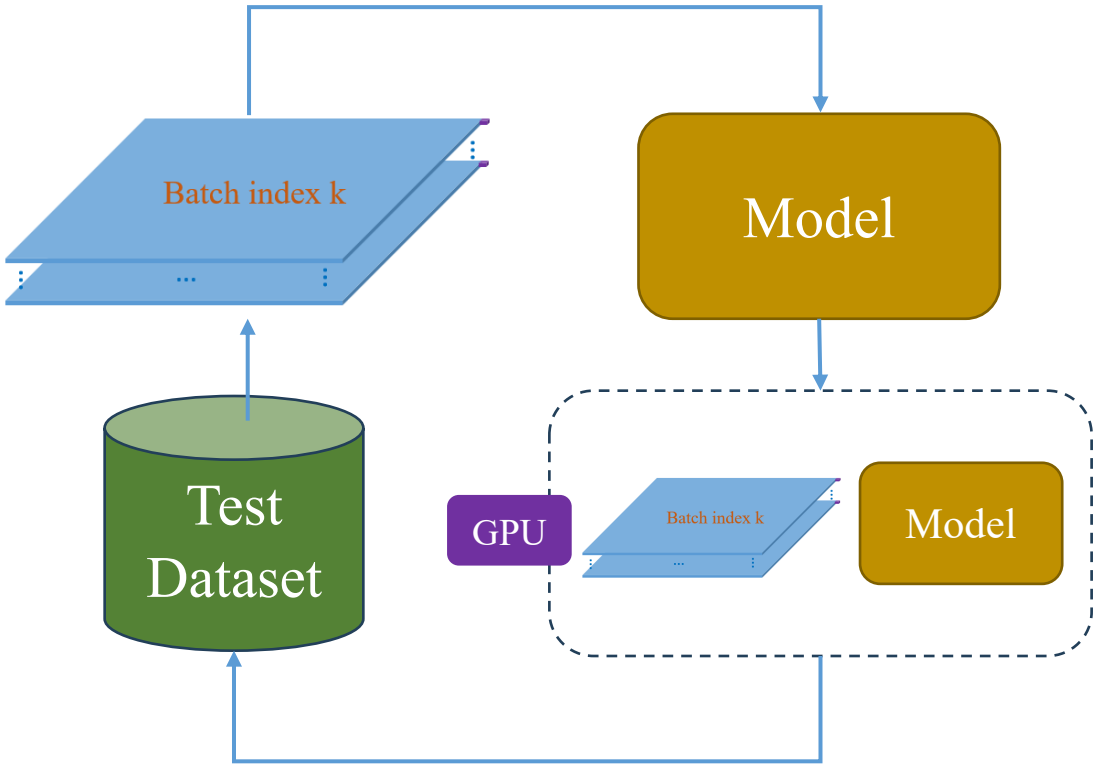



```python
# Training the model
max_epoch = 5
for epoch in range(max_epoch):
    for i, (inputs, labels) in enumerate(trainloader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

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

    print(f"Epoch [{epoch + 1}/{max_epoch}]")
```

```
Epoch [1/5]
Epoch [2/5]
Epoch [3/5]
Epoch [4/5]
Epoch [5/5]
```

Step-by-Step Implementation

❖ 4. Inference




```
correct = 0
total = 0
with torch.no_grad():
    for images, labels in testloader:
        # Move inputs and labels to the device
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"accuracy: {accuracy}")
```

Step-by-Step Implementation

❖ Addition 1: Compute Training Loss and Accuracy

```
Epoch [1/5], Loss: 0.7866, Accuracy: 74.73%
Epoch [2/5], Loss: 0.5205, Accuracy: 81.59%
Epoch [3/5], Loss: 0.4706, Accuracy: 83.28%
Epoch [4/5], Loss: 0.4432, Accuracy: 84.25%
Epoch [5/5], Loss: 0.4232, Accuracy: 85.13%
```

```python
for epoch in range(5):
    running_loss = 0.0
    correct = 0   # to track number of correct predictions
    total = 0     # to track total number of samples

    for i, (inputs, labels) in enumerate(trainloader, 0):
        # see comments from the previous example
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # accumulate loss
        running_loss += loss.item()

    epoch_accuracy = 100 * correct / total
    running_loss = running_loss / (i + 1)
```

Step-by-Step Implementation

❖ Addition 2: Compute Test Loss and Accuracy

```python
def evaluate(model, testloader, criterion):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in testloader:
            # Move inputs and labels to the device
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return test_loss / len(testloader), accuracy
```

```python
for epoch in range(5):
    running_loss = 0.0
    correct = 0   # to track number of correct predictions
    total = 0     # to track total number of samples

    for i, (inputs, labels) in enumerate(trainloader, 0):
        # see comments from the previous example
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # accumulate loss
        running_loss += loss.item()

    epoch_accuracy = 100 * correct / total
    running_loss = running_loss / (i + 1)
    test_loss, test_accuracy = evaluate(model,
                                        testloader,
                                        criterion)
```

Step-by-Step Implementation

❖ Addition 2: Compute Test Loss and Accuracy

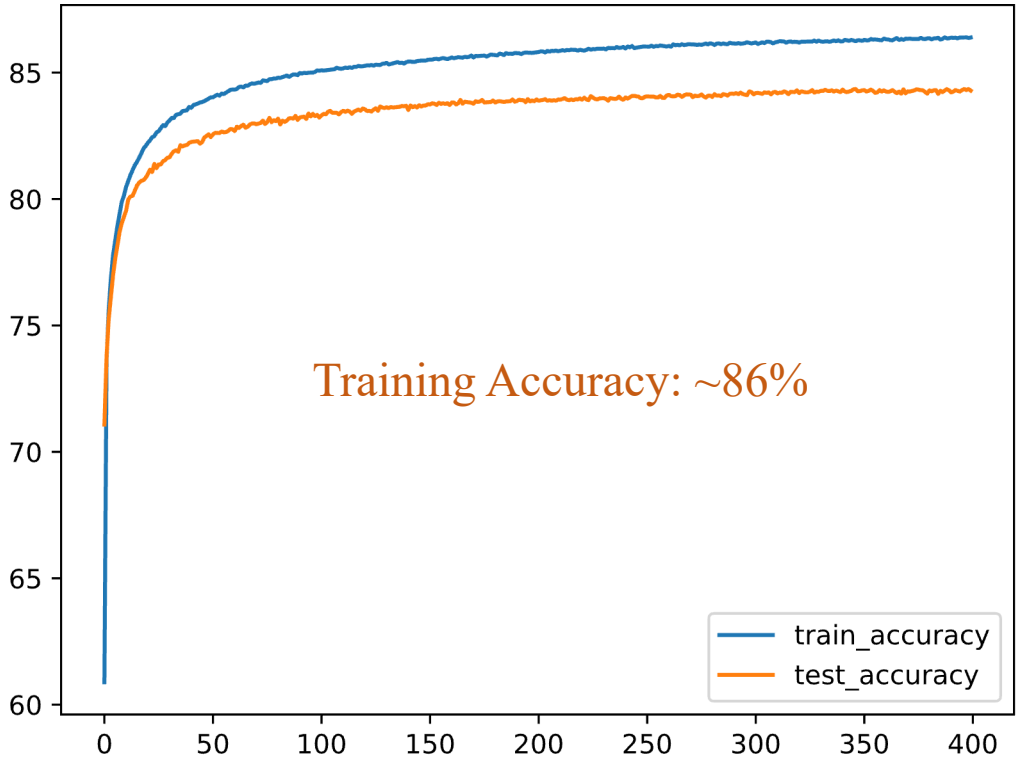
```
Epoch [1/5], Test Loss: 0.5817, Test Accuracy: 79.18%
Epoch [2/5], Test Loss: 0.5139, Test Accuracy: 81.26%
Epoch [3/5], Test Loss: 0.4818, Test Accuracy: 82.80%
Epoch [4/5], Test Loss: 0.4602, Test Accuracy: 83.40%
Epoch [5/5], Test Loss: 0.4479, Test Accuracy: 83.92%
```

```python
for epoch in range(5):
    running_loss = 0.0
    correct = 0   # to track number of correct predictions
    total = 0     # to track total number of samples

    for i, (inputs, labels) in enumerate(trainloader, 0):
        # see comments from the previous example
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # Determine class predictions and track accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # accumulate loss
        running_loss += loss.item()

    epoch_accuracy = 100 * correct / total
    running_loss = running_loss / (i + 1)
    test_loss, test_accuracy = evaluate(model,
                                        testloader,
                                        criterion)
```

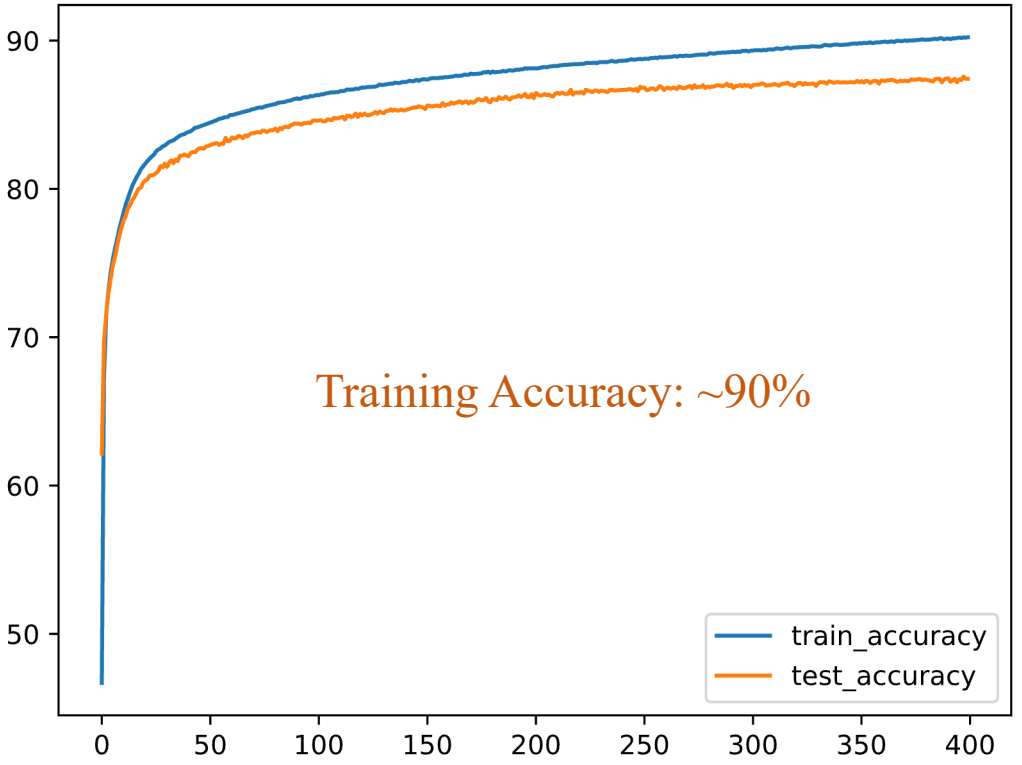
Softmax and MLP

❖ Comparison

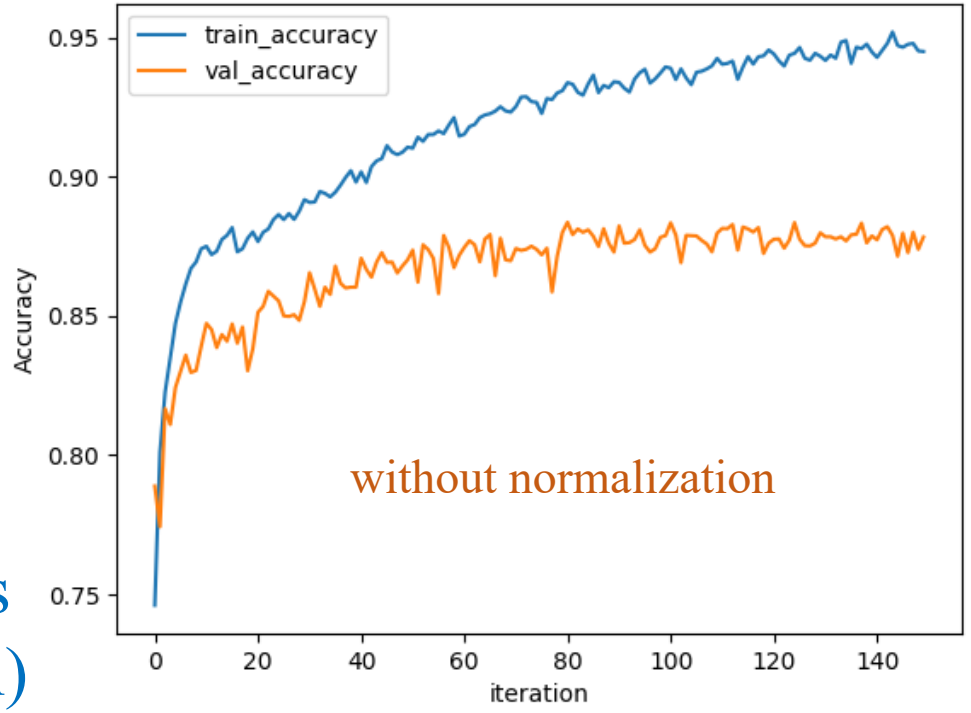
```
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 10)
)
model = model.to(device)
```

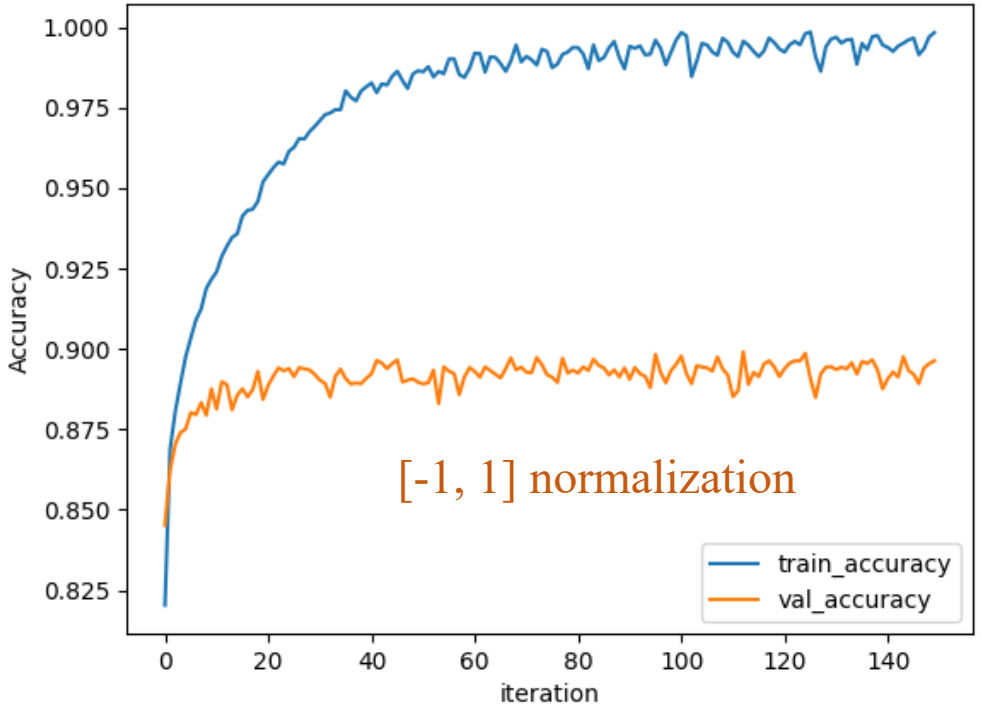


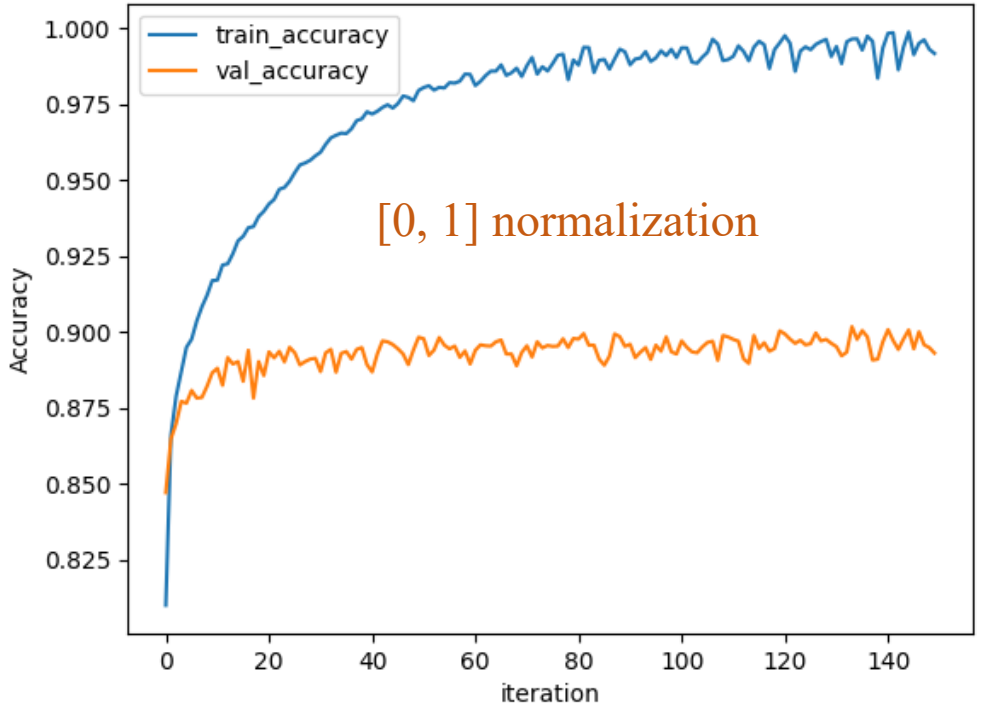

```
model = nn.Sequential(
    nn.Flatten(), nn.Linear(784, 256),
    nn.ReLU(), nn.Linear(256, 10)
)
model = model.to(device)
```

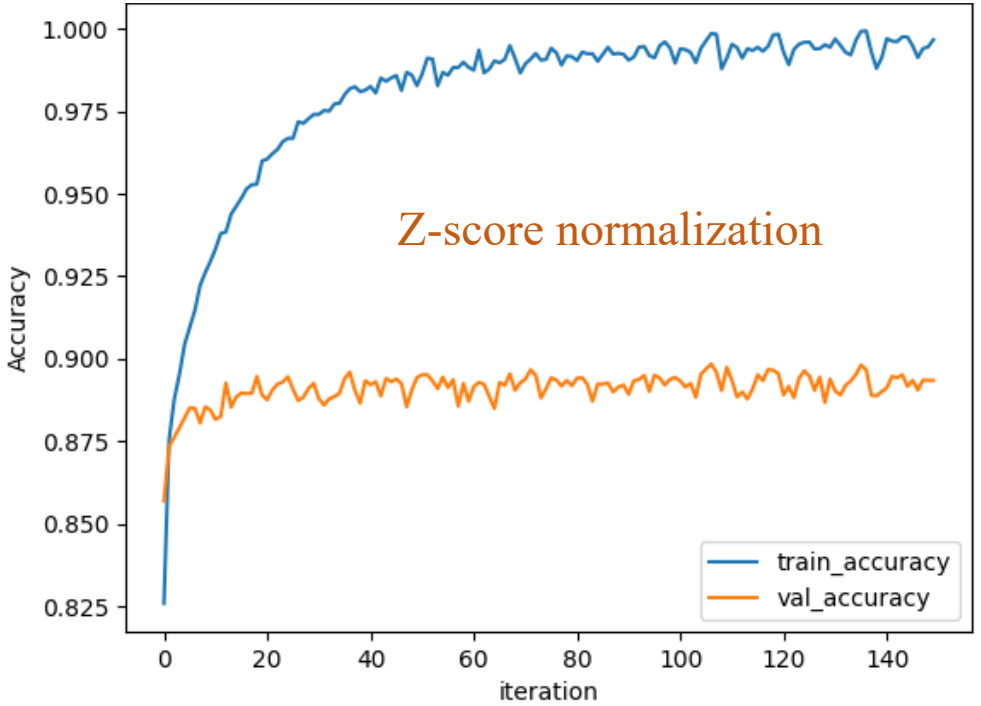


Two hidden layers
(256 nodes for each)



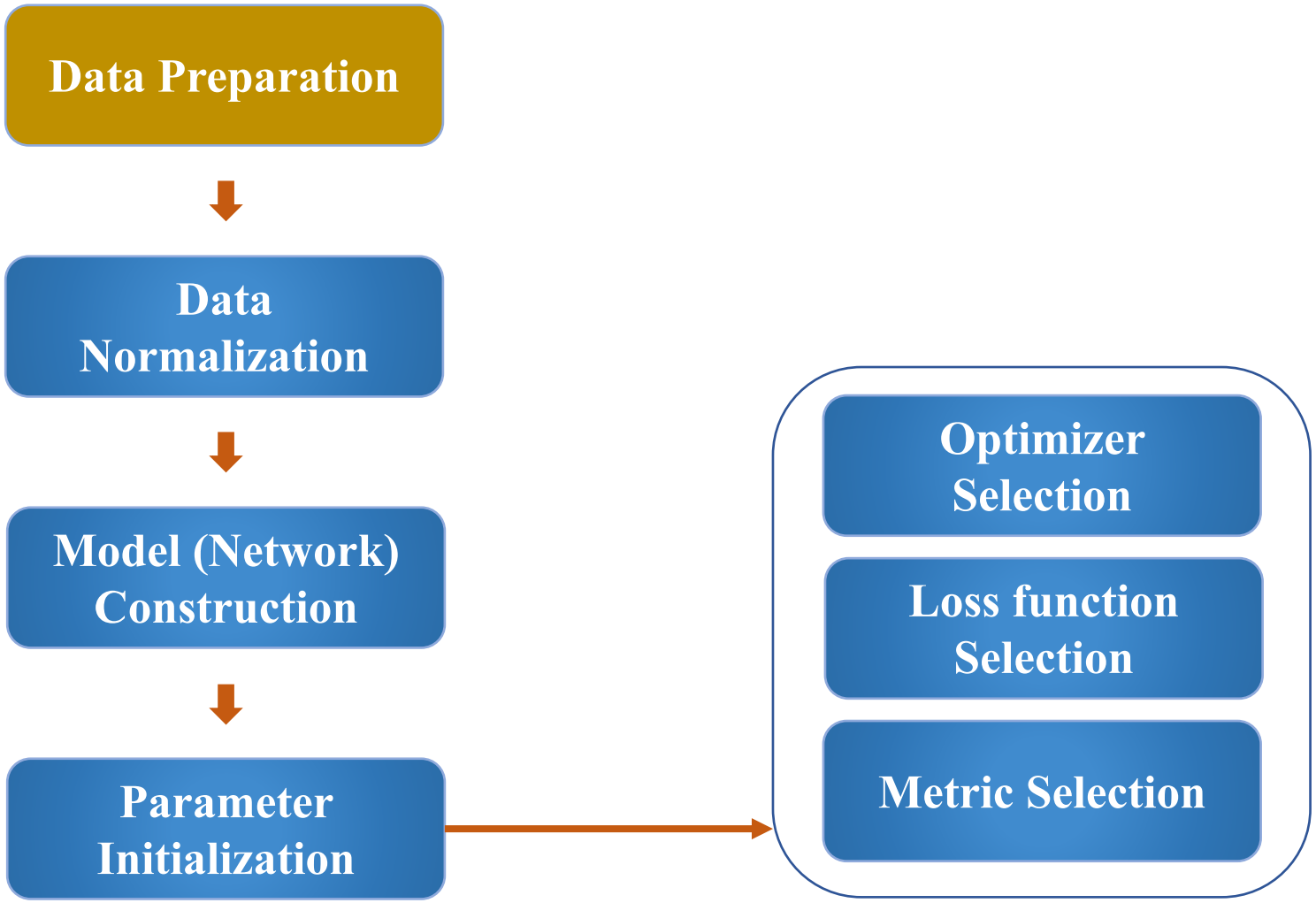






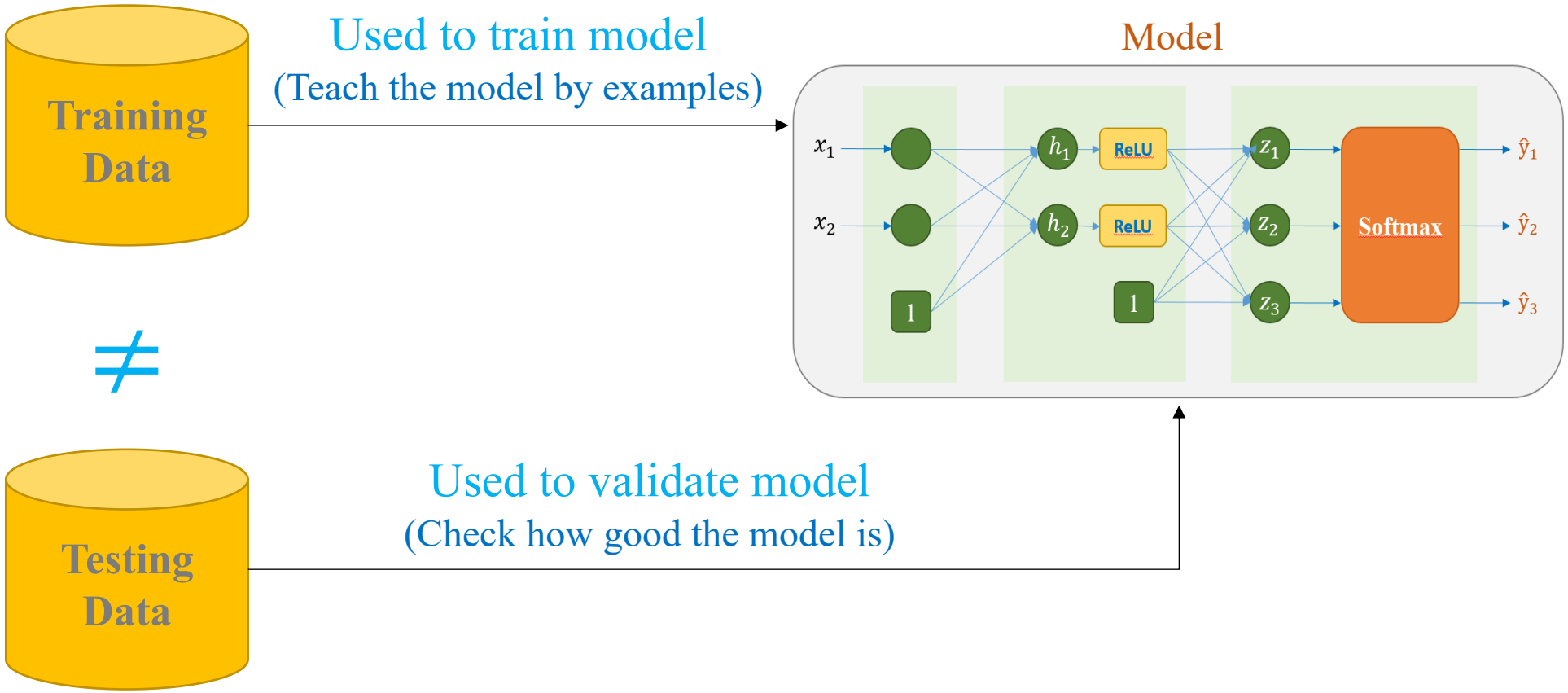
Summary and Next Steps

Summary

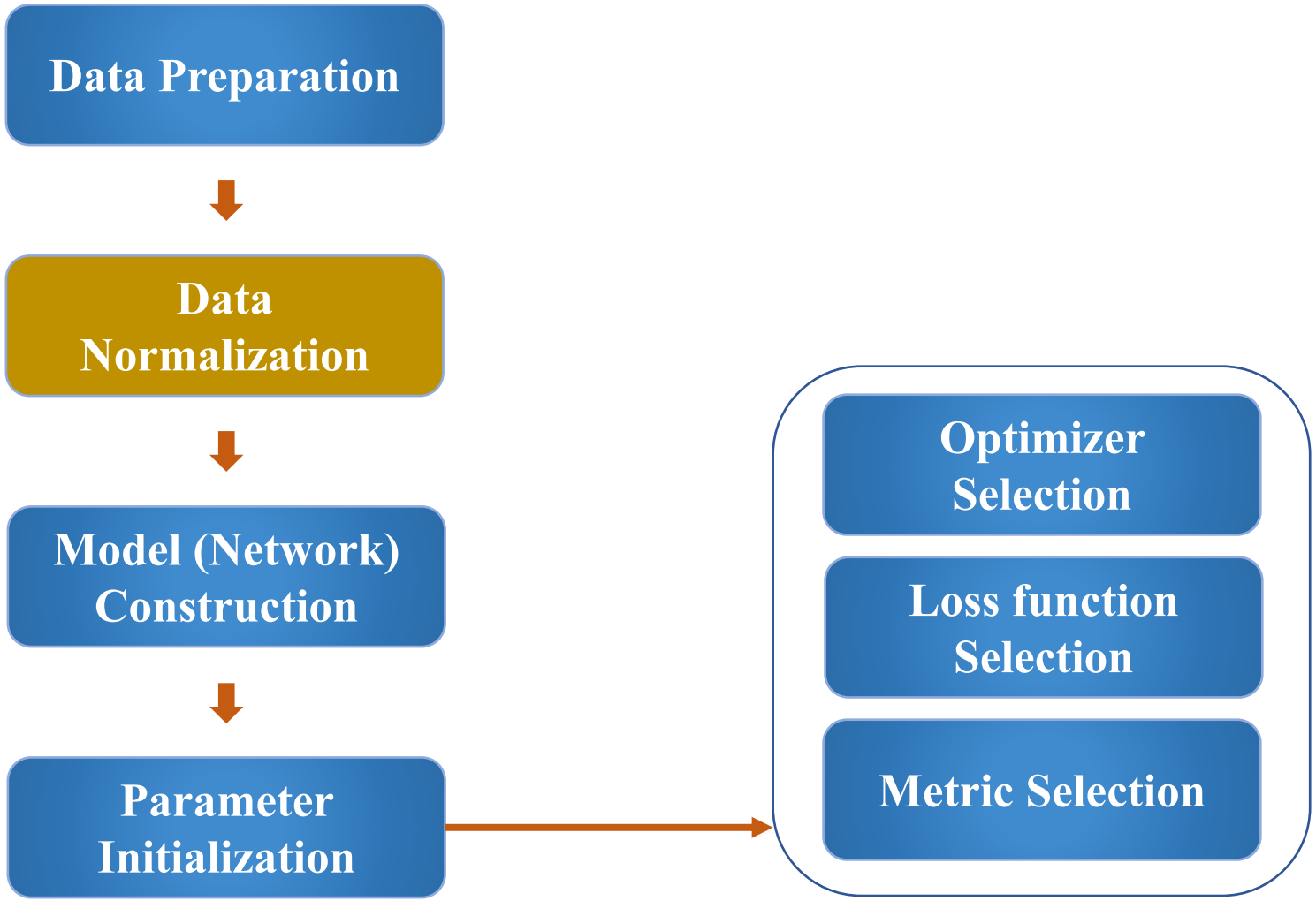

Data Preparation
Data Normalization
Model (Network) Construction
Parameter Initialization
Optimizer Selection
Loss function Selection
Metric Selection

Summary

❖ Data Preparation

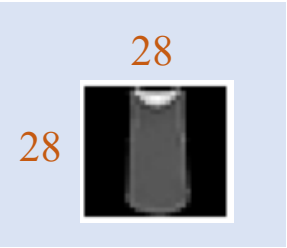


Summary


Data Preparation
Data Normalization
Model (Network) Construction
Parameter Initialization
Optimizer Selection
Loss function Selection
Metric Selection

Summary

❖ Data Normalization



Convert to the range [0,1]

$$\text{Image} = \frac{\text{Image}}{255}$$

Convert to the range [-1,1]

$$\text{Image} = \frac{\text{Image}}{127.5} - 1$$

Z-score normalization

$$\text{Image} = \frac{\text{Image} - \mu}{\sigma}$$

μ is the mean of
the image (or training data)

σ is thestandard deviation
of the image (or training data)

Implementation

In Theory

$X \in [0, 255]$

Convert to the range [0,1]

$$\text{Image} = \frac{\text{Image}}{255}$$

Convert to the range [-1,1]

$$\text{Image} = \frac{\text{Image}}{127.5} - 1$$

Z-score normalization

$$\text{Image} = \frac{\text{Image} - \mu}{\sigma}$$

In Pytorch

$X \in [0, 1]$

Normalize($mean$, std)

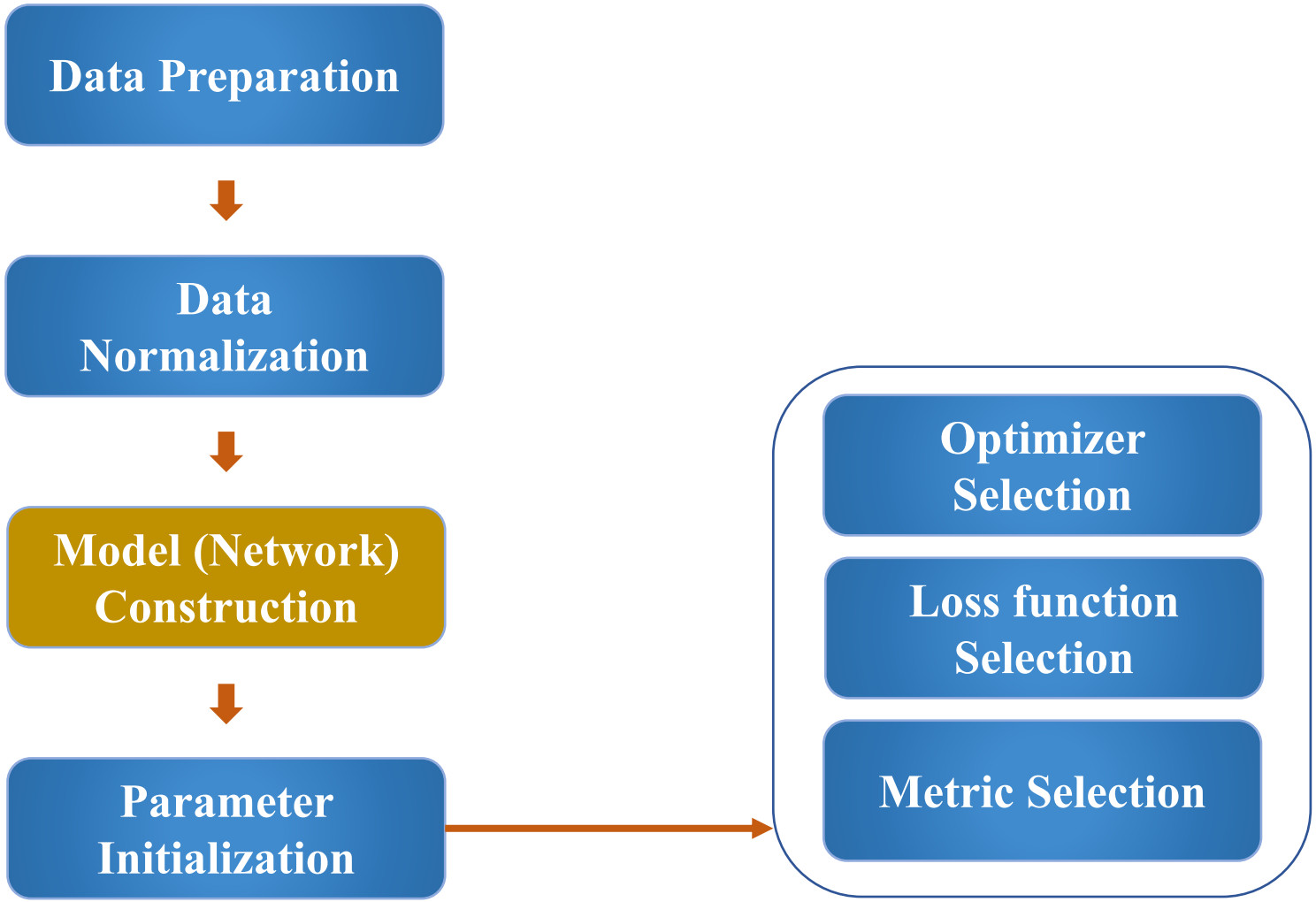
$$\text{Image} = \frac{\text{Image} - mean}{\text{std}}$$

[0,1]	mean = 0 ; std = 1
[-1,1]	mean = 0.5; std = 0.5

Compute mean and std from data

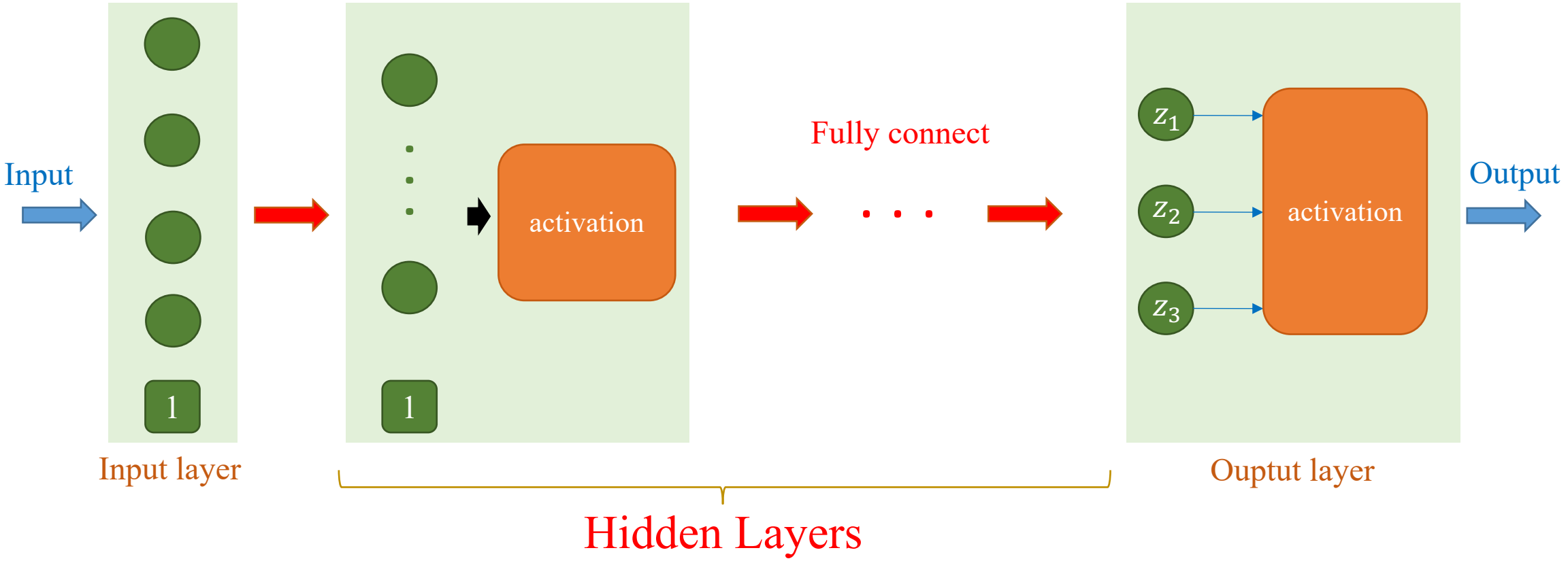
Summary

❖ Training pipeline



Summary

❖ To-do list

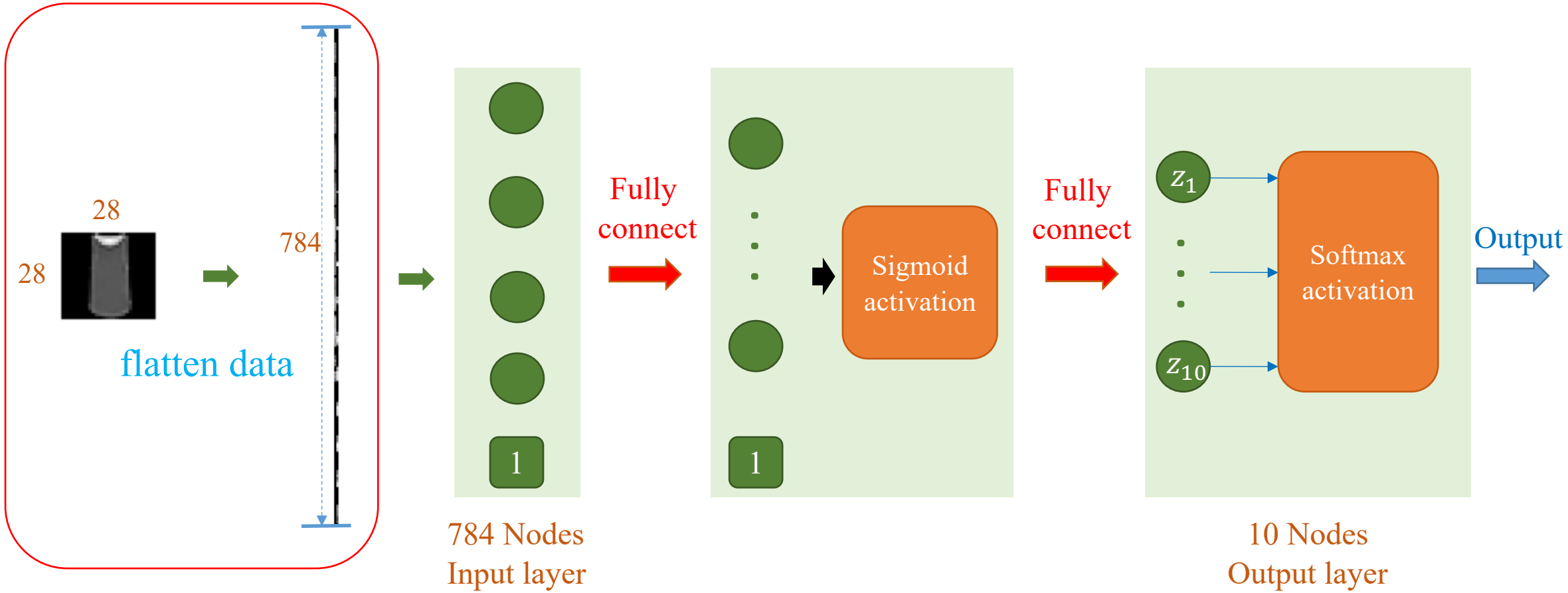


How many hidden layers?
How many nodes in a hidden layer?

Which activation function?
Which network components?

Summary

❖ How many nodes?



How many nodes?

Summary

❖ How many nodes?

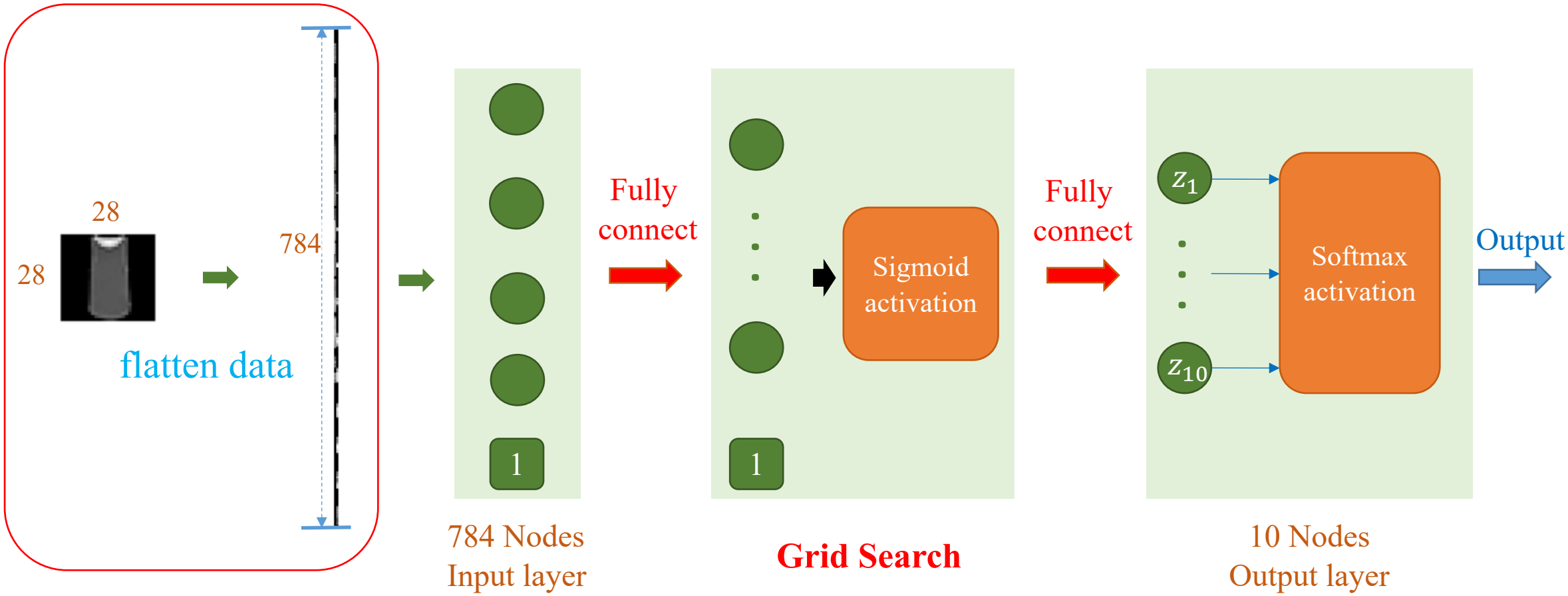


Image Classification

Cifar-10 dataset

- Color images
- Resolution=32x32
- Training set: 50000 samples
- Testing set: 10000 samples

Cifar-10 Data