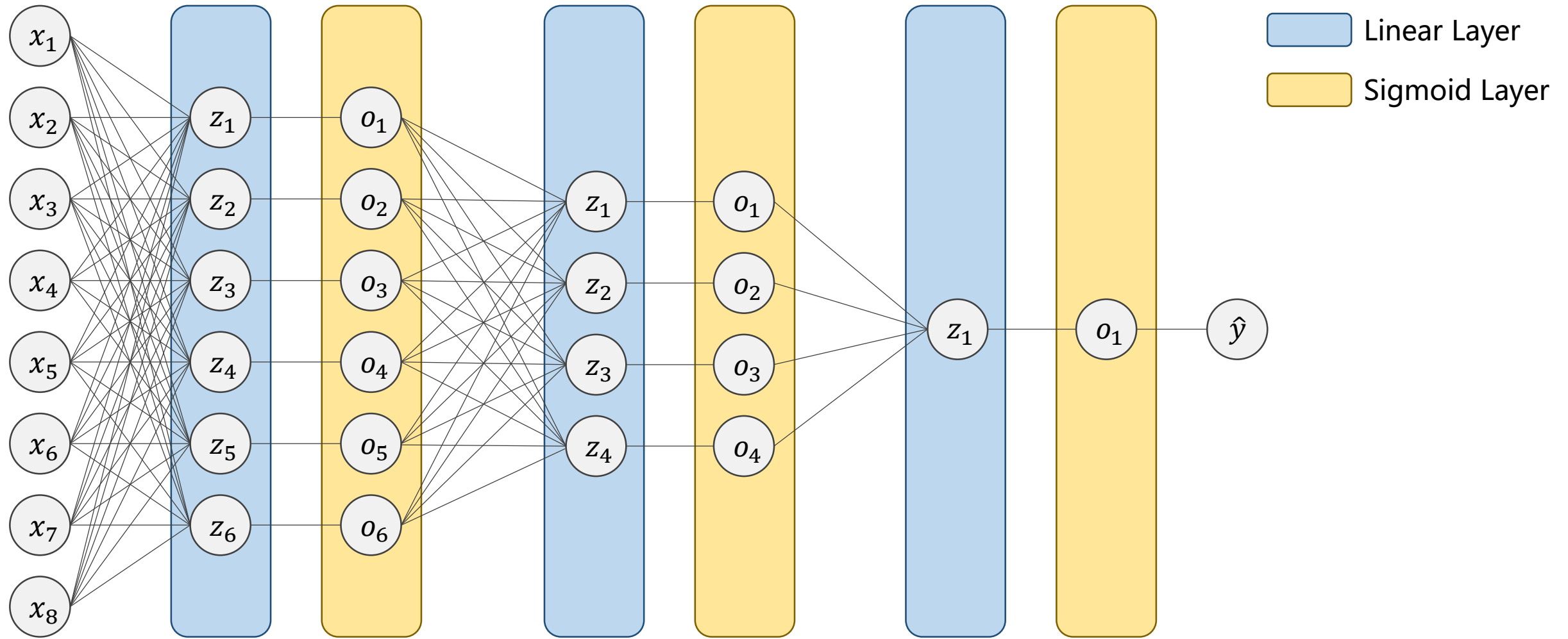




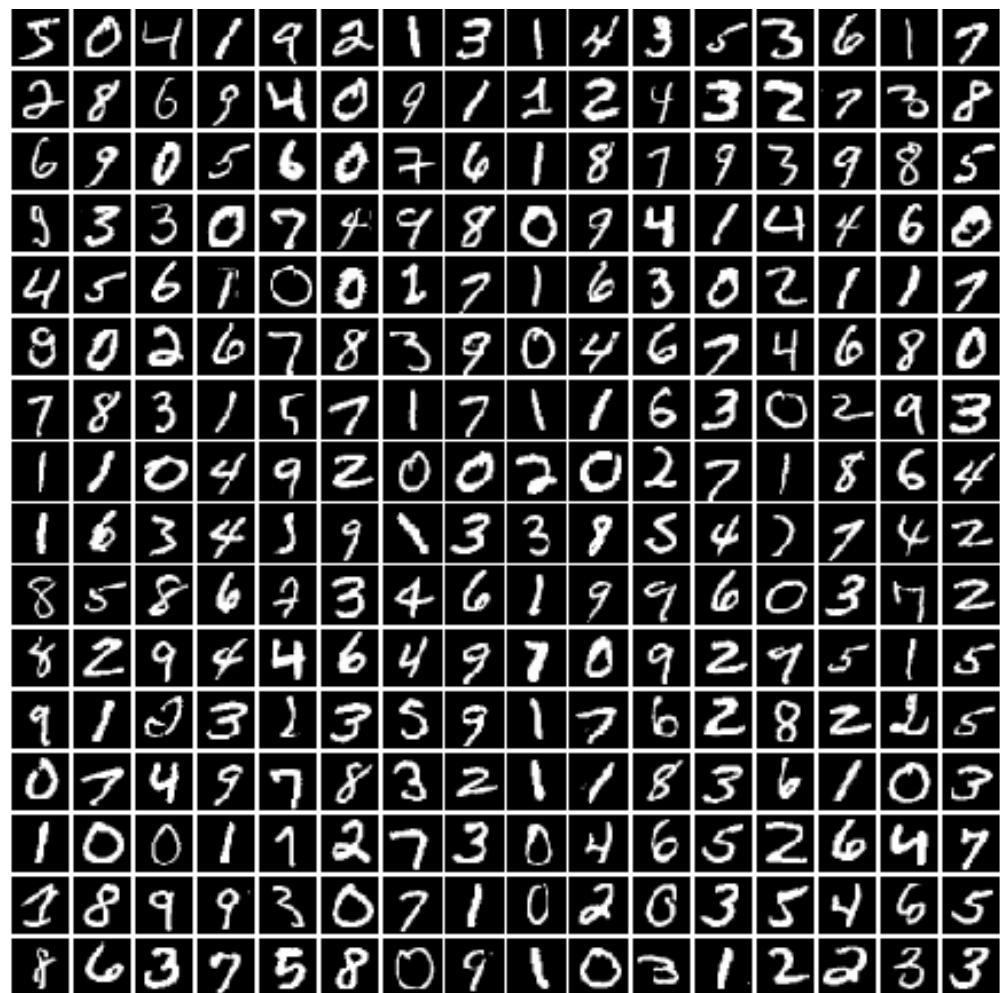
PyTorch Tutorial

09. Softmax Classifier

Revision: Diabetes dataset



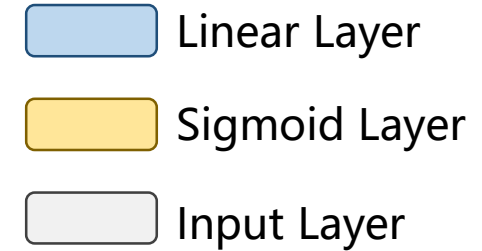
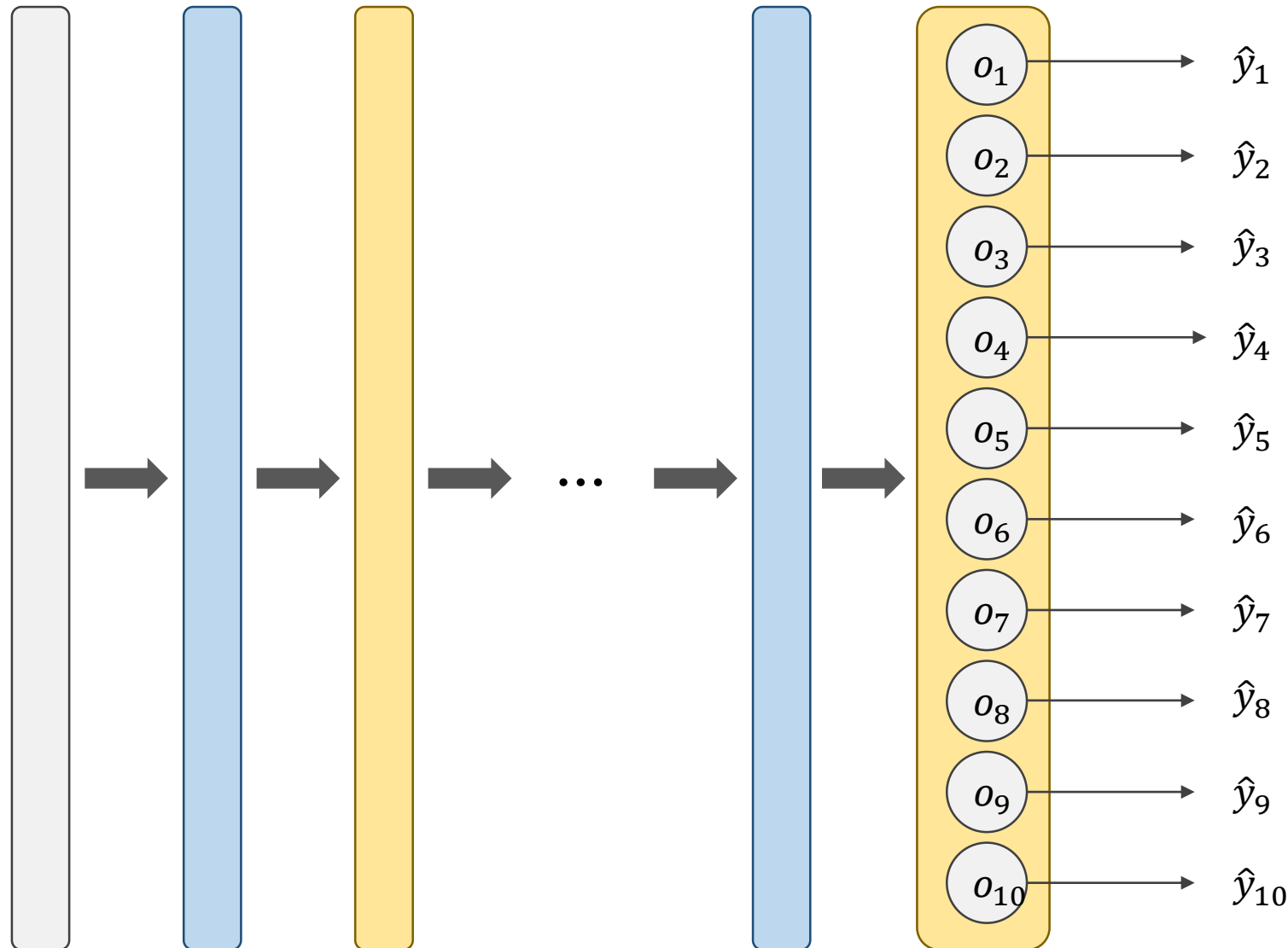
Revision: MNIST Dataset



There are **10** labels in MNIST dataset.

How to design the neural network?

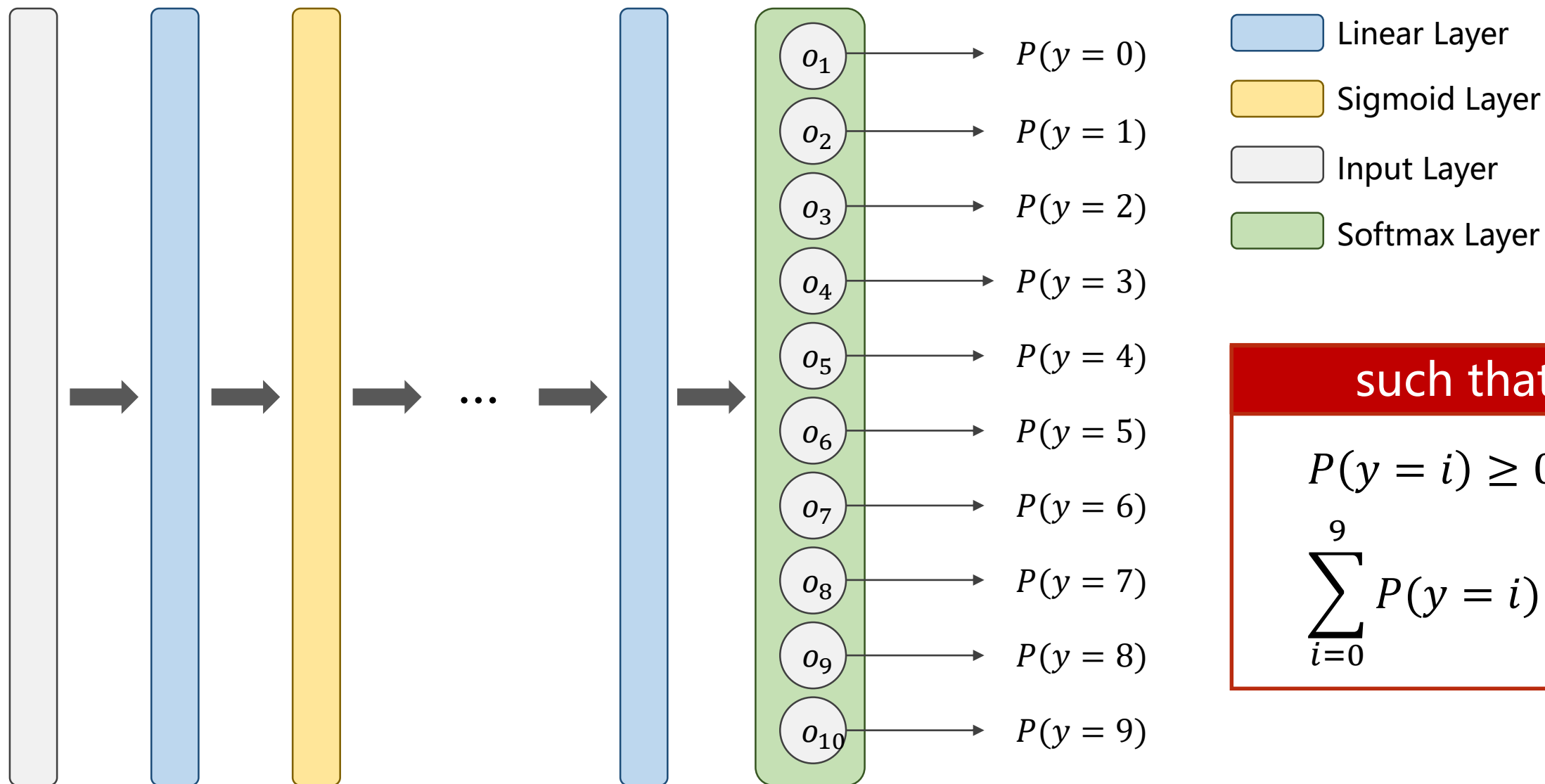
Design 10 outputs using Sigmoid?



What is wrong?

We hope the outputs is competitive!
Actually we hope the neural network
outputs a **distribution**.

Output a Distribution of prediction with Softmax



such that

$$P(y = i) \geq 0$$

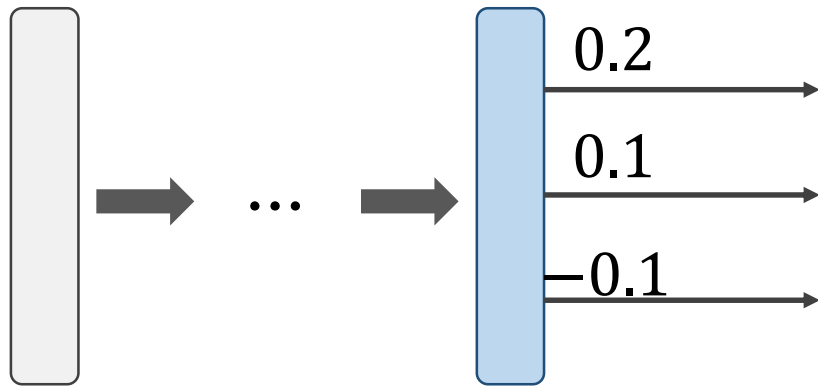
$$\sum_{i=0}^9 P(y = i) = 1$$

Softmax Layer

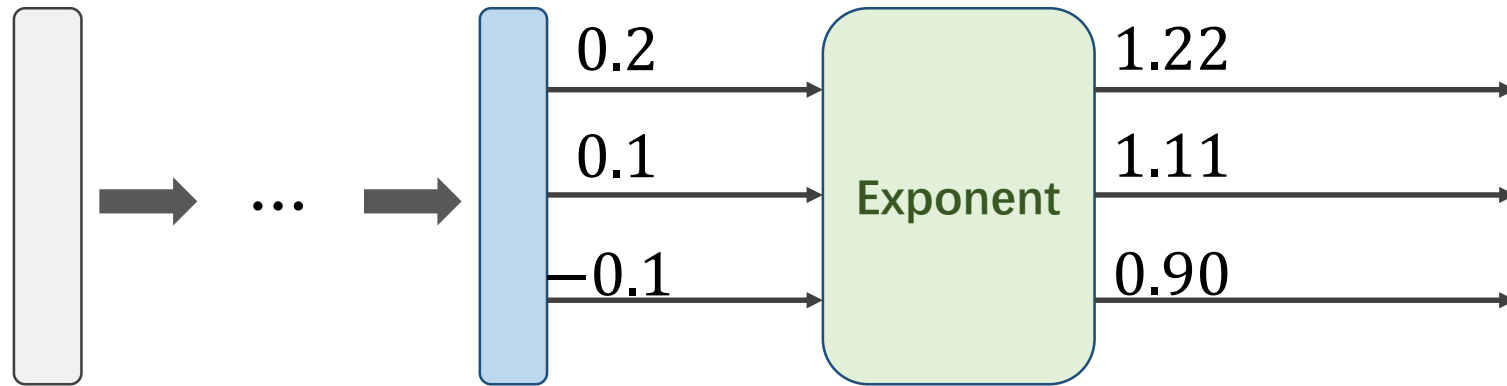
Suppose $Z^l \in \mathbb{R}^K$ is the output of the last linear layer, the Softmax function:

$$P(y = i) = \frac{e^{z_i} \text{ 使 } z_i > 0}{\sum_{j=0}^{K-1} e^{z_j}}, i \in \{0, \dots, K - 1\}$$

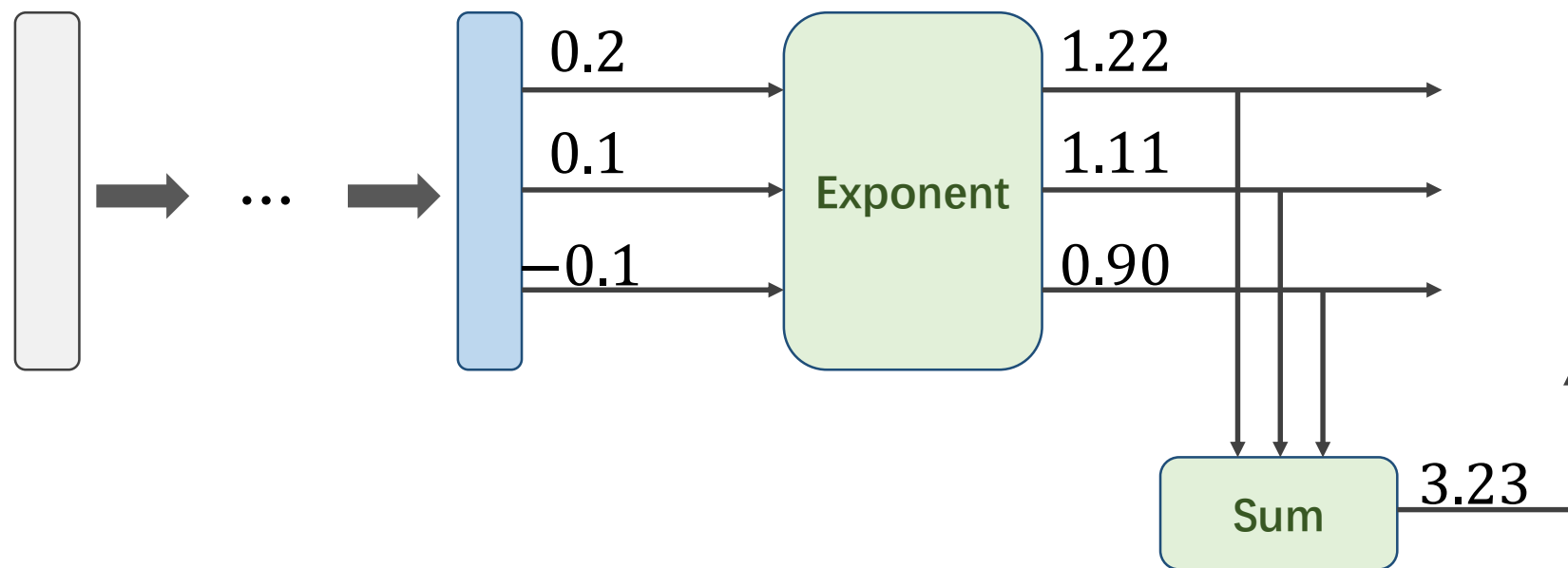
Softmax Layer - Example



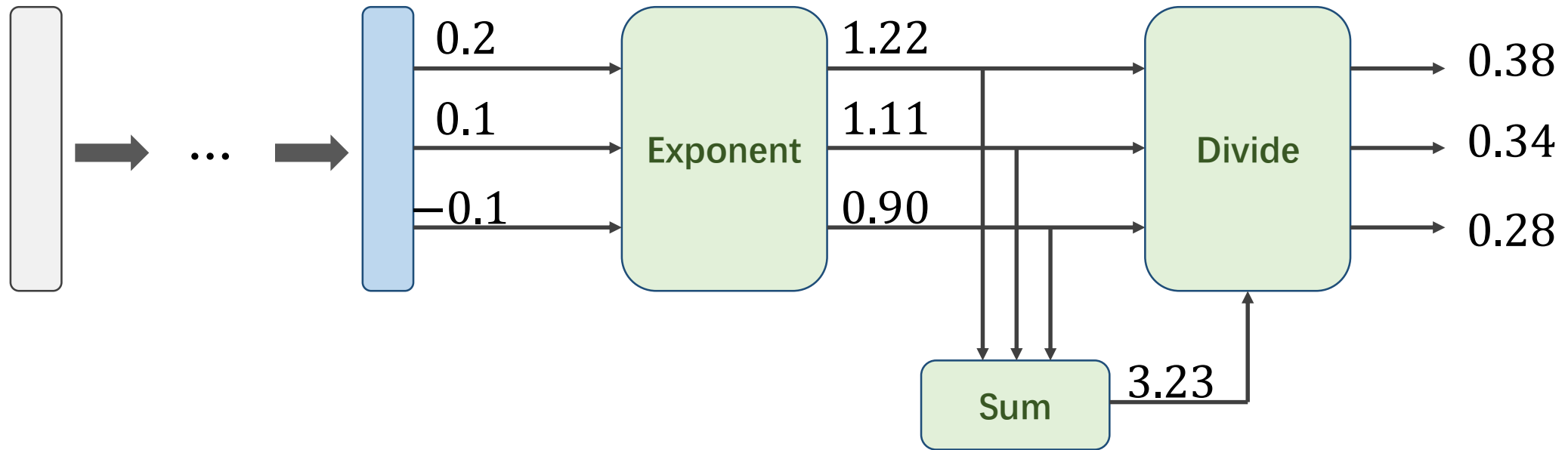
Softmax Layer - Example



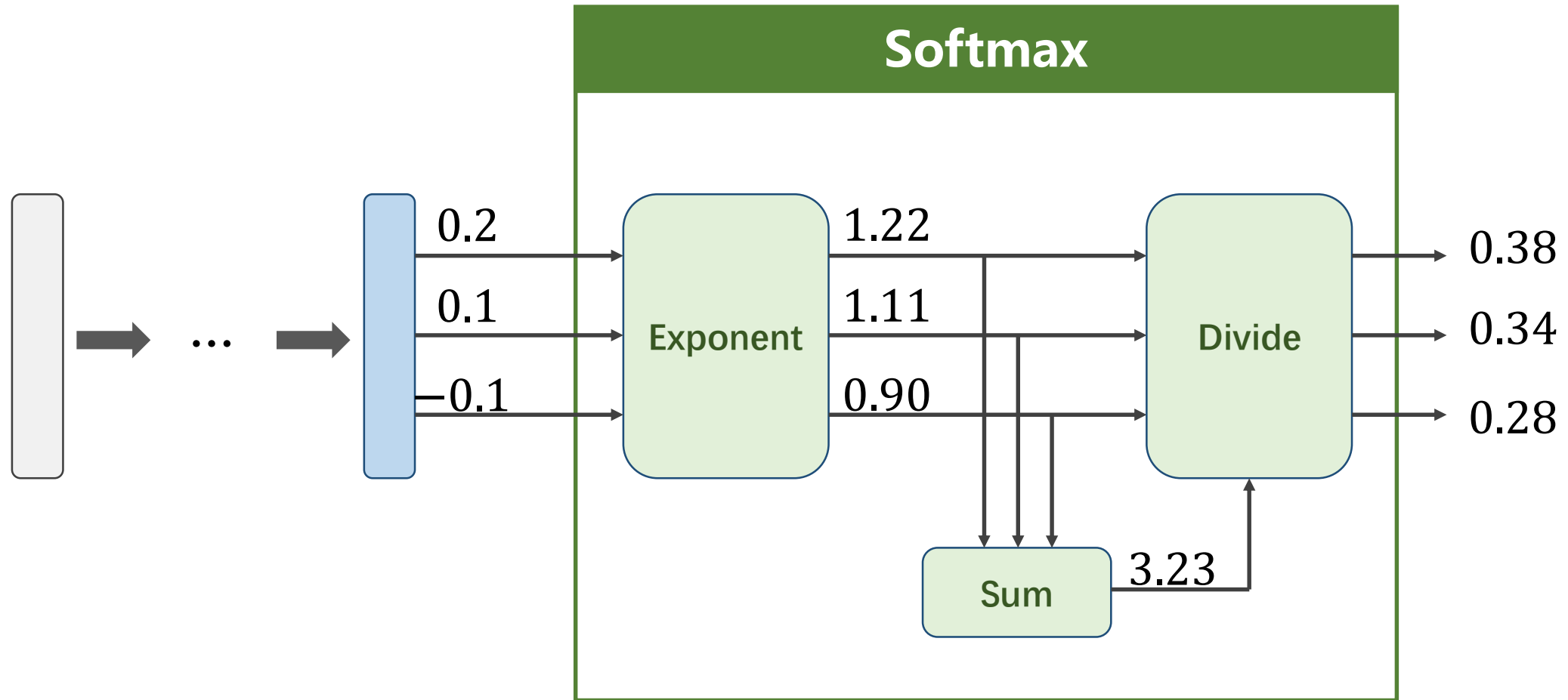
Softmax Layer - Example



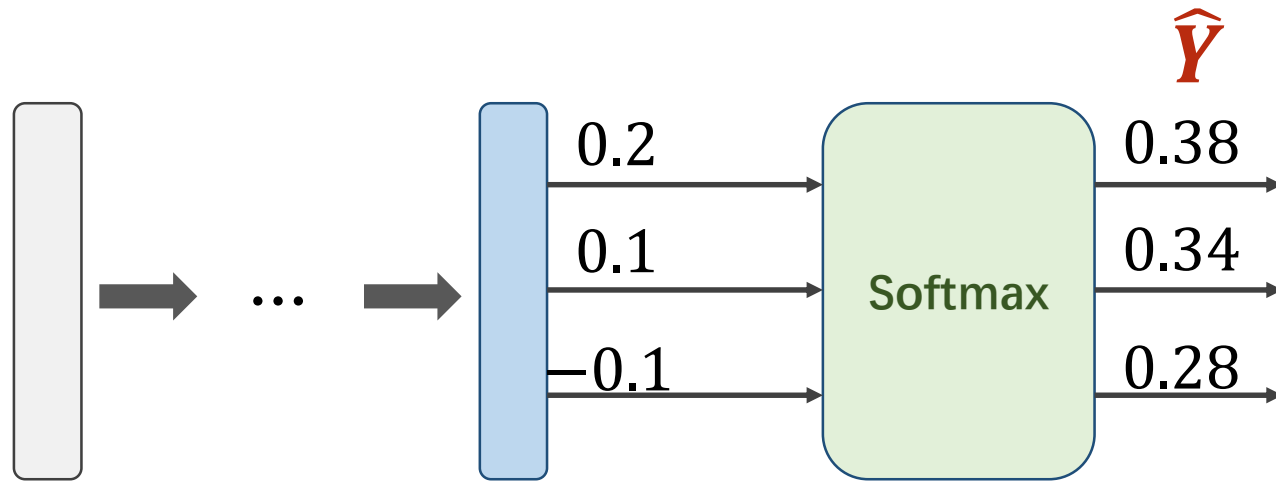
Softmax Layer - Example



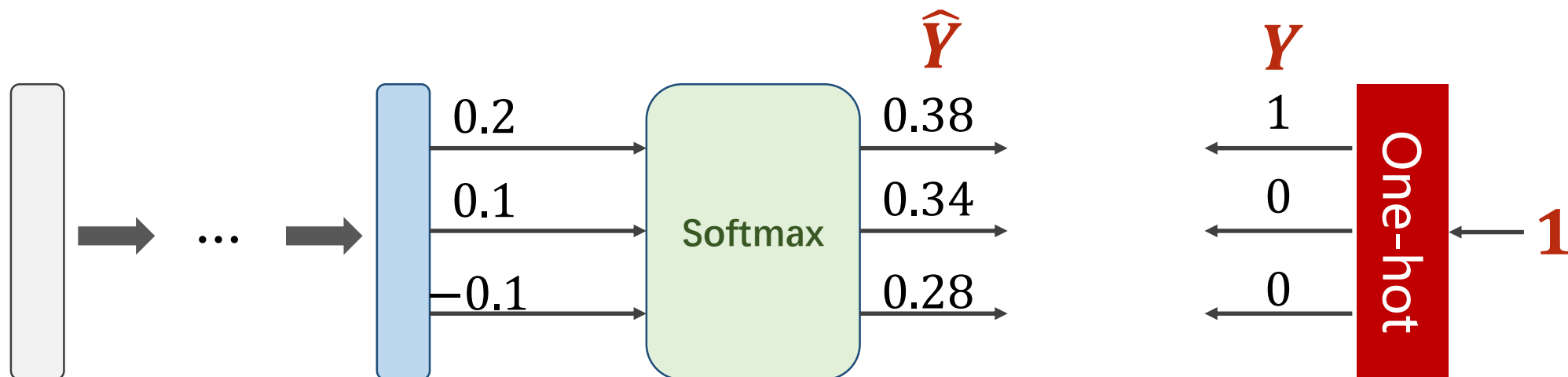
Softmax Layer - Example



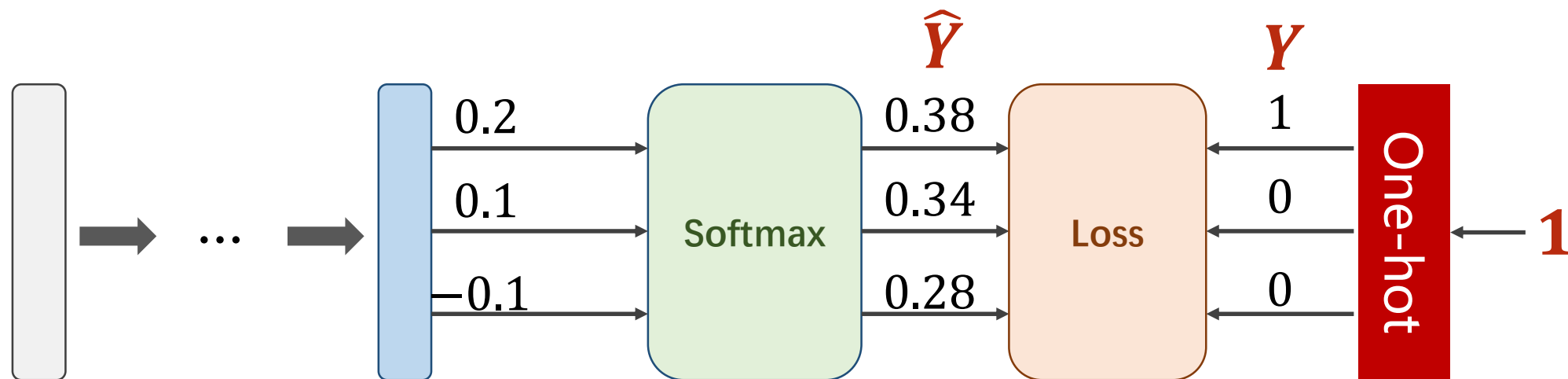
Loss function - Cross Entropy



Loss function - Cross Entropy

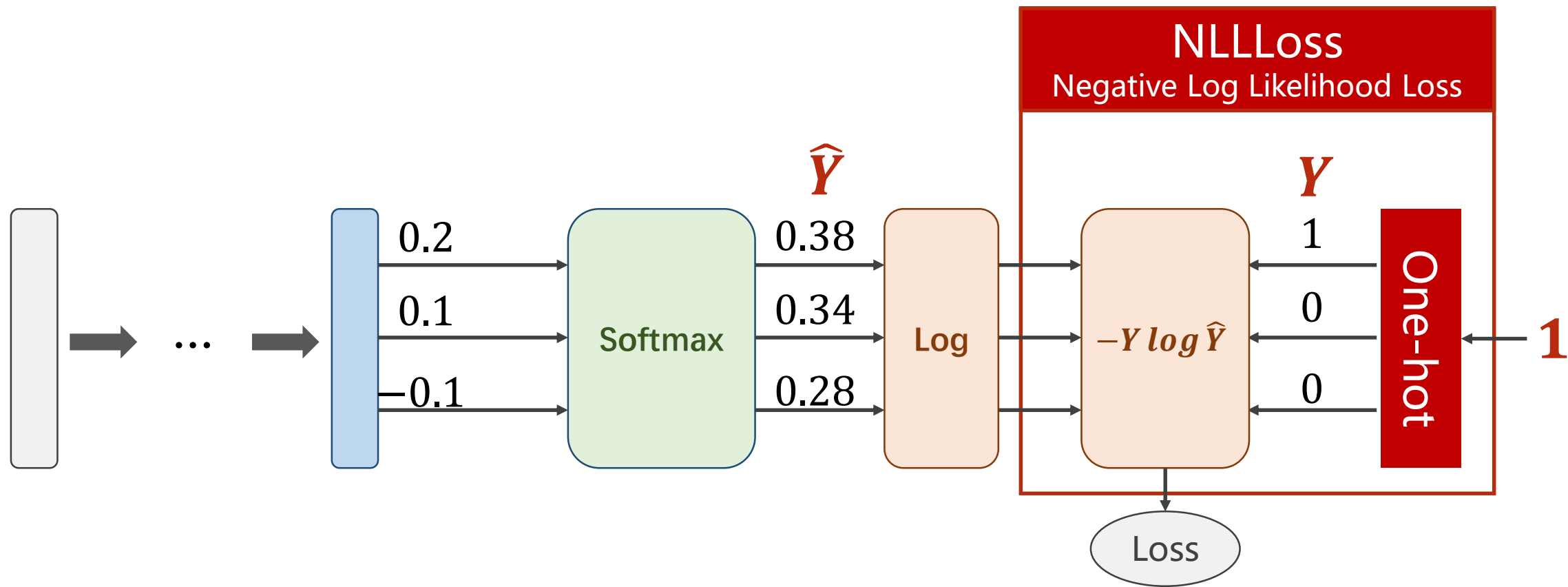


Loss function - Cross Entropy



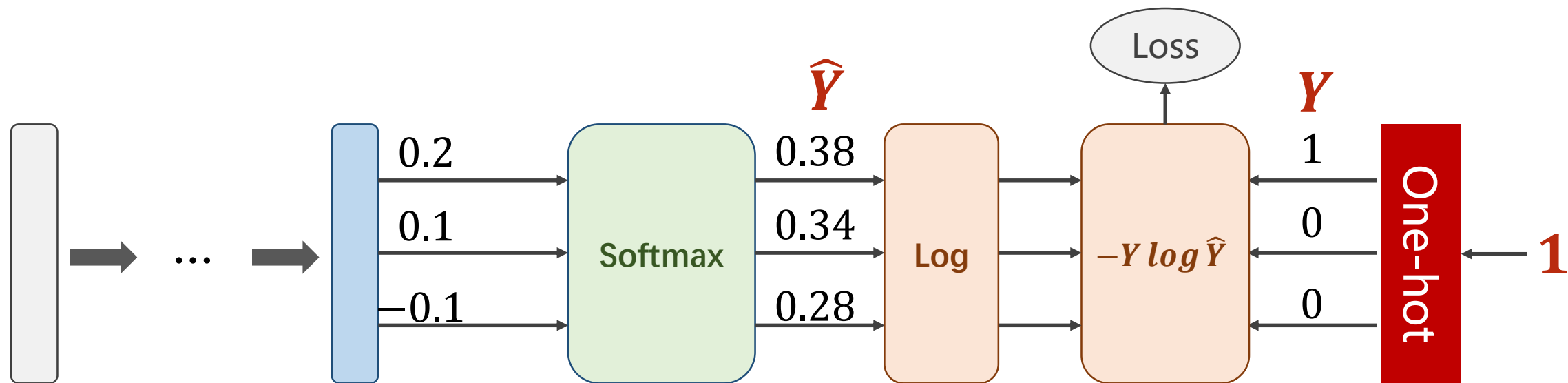
$$Loss(\hat{Y}, Y) = -Y \log \hat{Y}$$

Loss function - Cross Entropy



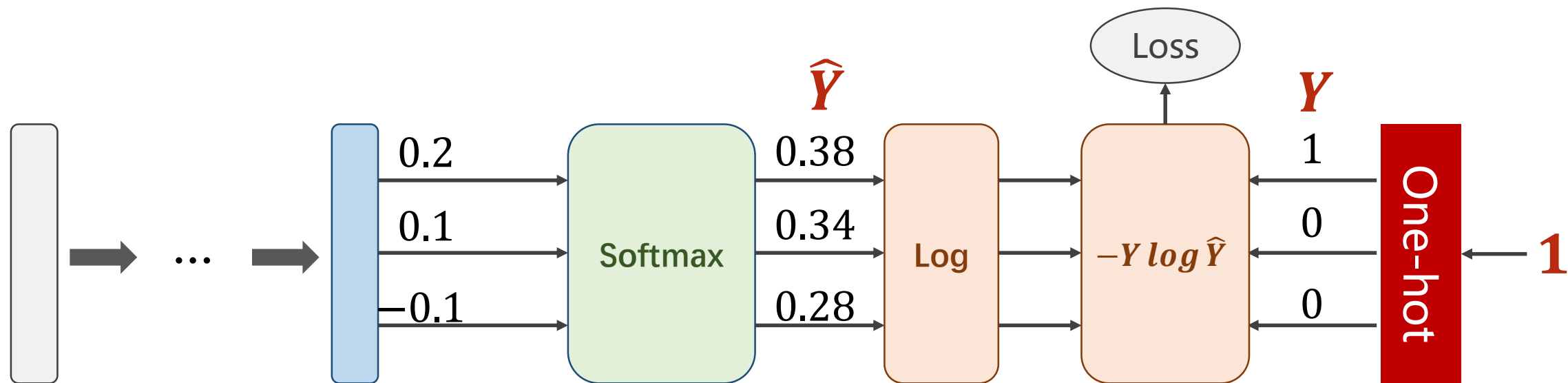
$$Loss(\hat{Y}, Y) = -Y \log \hat{Y}$$

Cross Entropy in Numpy



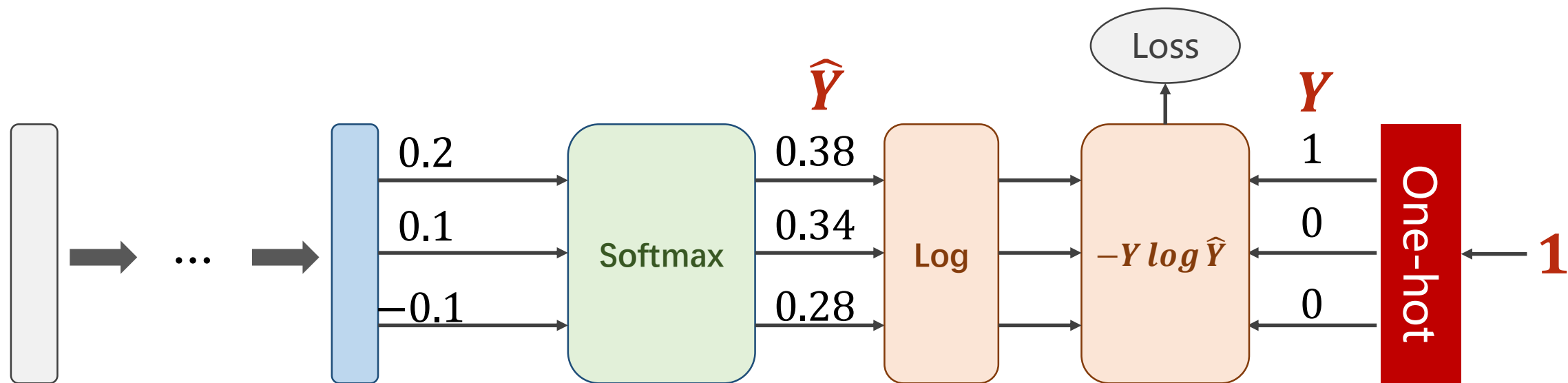
```
import numpy as np
y = np.array([1, 0, 0])
z = np.array([0.2, 0.1, -0.1])
y_pred = np.exp(z) / np.exp(z).sum()
loss = (- y * np.log(y_pred)).sum()
print(loss)
```


Cross Entropy in Numpy



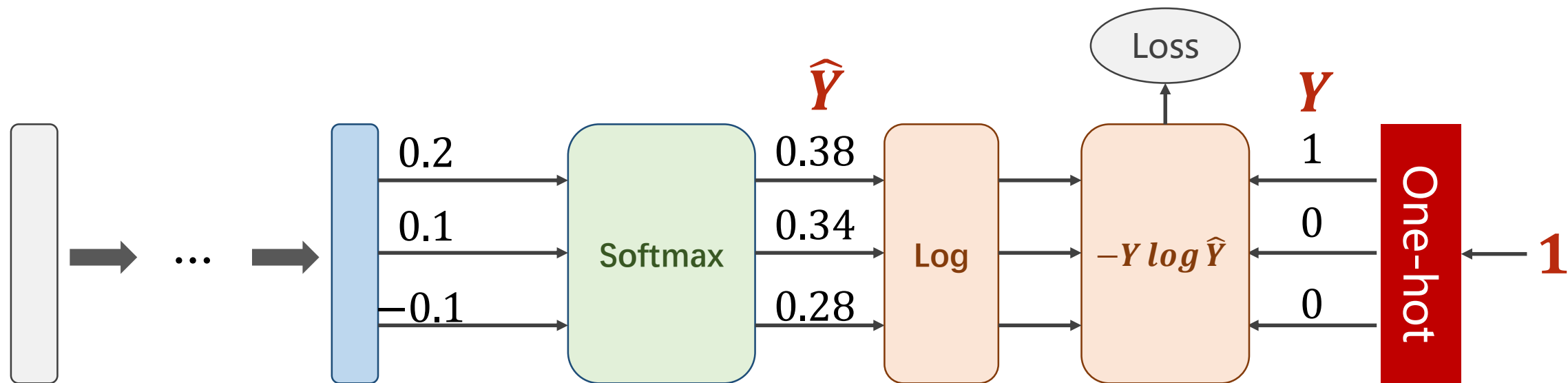
```
import numpy as np
y = np.array([1, 0, 0])
z = np.array([0.2, 0.1, -0.1])
y_pred = np.exp(z) / np.exp(z).sum()
loss = (- y * np.log(y_pred)).sum()
print(loss)
```

Cross Entropy in Numpy



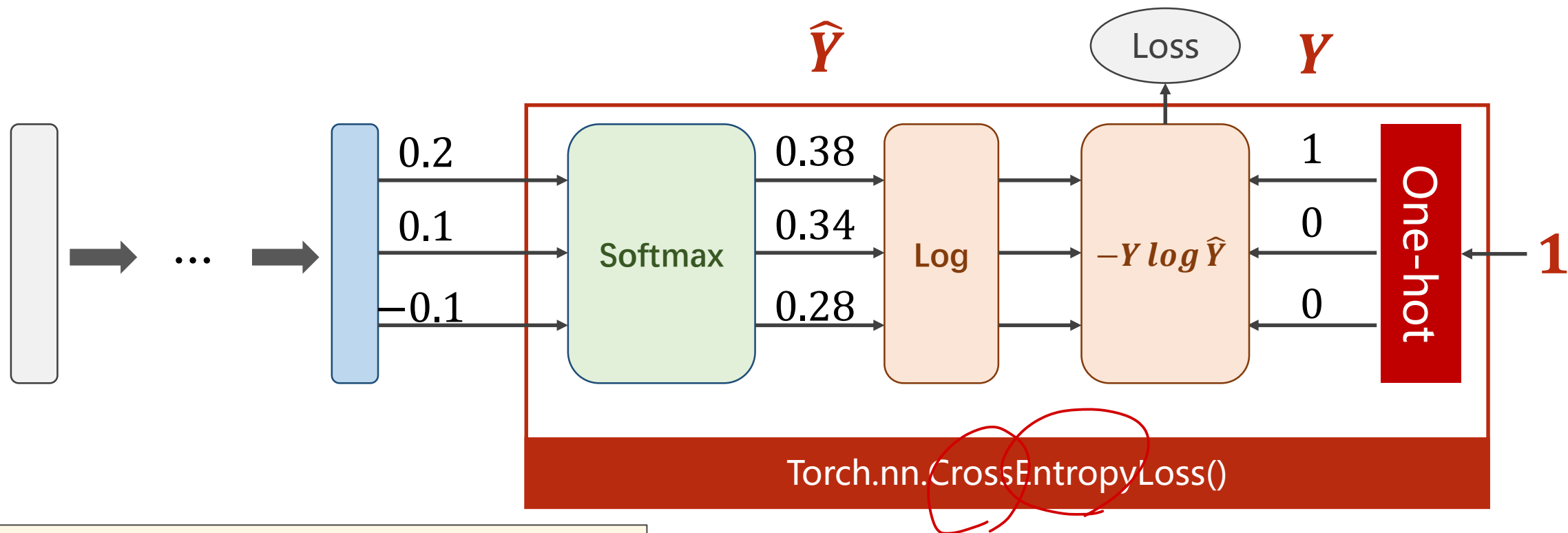
```
import numpy as np
y = np.array([1, 0, 0])
z = np.array([0.2, 0.1, -0.1])
y_pred = np.exp(z) / np.exp(z).sum()
loss = (- y * np.log(y_pred)).sum()
print(loss)
```

Cross Entropy in Numpy



```
import numpy as np
y = np.array([1, 0, 0])
z = np.array([0.2, 0.1, -0.1])
y_pred = np.exp(z) / np.exp(z).sum()
loss = (- y * np.log(y_pred)).sum()
print(loss)
```

Cross Entropy in PyTorch



```
import torch
y = torch.LongTensor([0])
z = torch.Tensor([[0.2, 0.1, -0.1]])
criterion = torch.nn.CrossEntropyLoss()
loss = criterion(z, y)
print(loss)
```

Mini-Batch: *batch_size=3*

```
import torch
criterion = torch.nn.CrossEntropyLoss()
Y = torch.LongTensor([2, 0, 1])

Y_pred1 = torch.Tensor([[0.1, 0.2, 0.9],
                        [1.1, 0.1, 0.2],
                        [0.2, 2.1, 0.1]])
Y_pred2 = torch.Tensor([[0.8, 0.2, 0.3],
                        [0.2, 0.3, 0.5],
                        [0.2, 0.2, 0.5]])

l1 = criterion(Y_pred1, Y)
l2 = criterion(Y_pred2, Y)
print("Batch Loss1 = ", l1.data, "\nBatch Loss2=", l2.data)
```

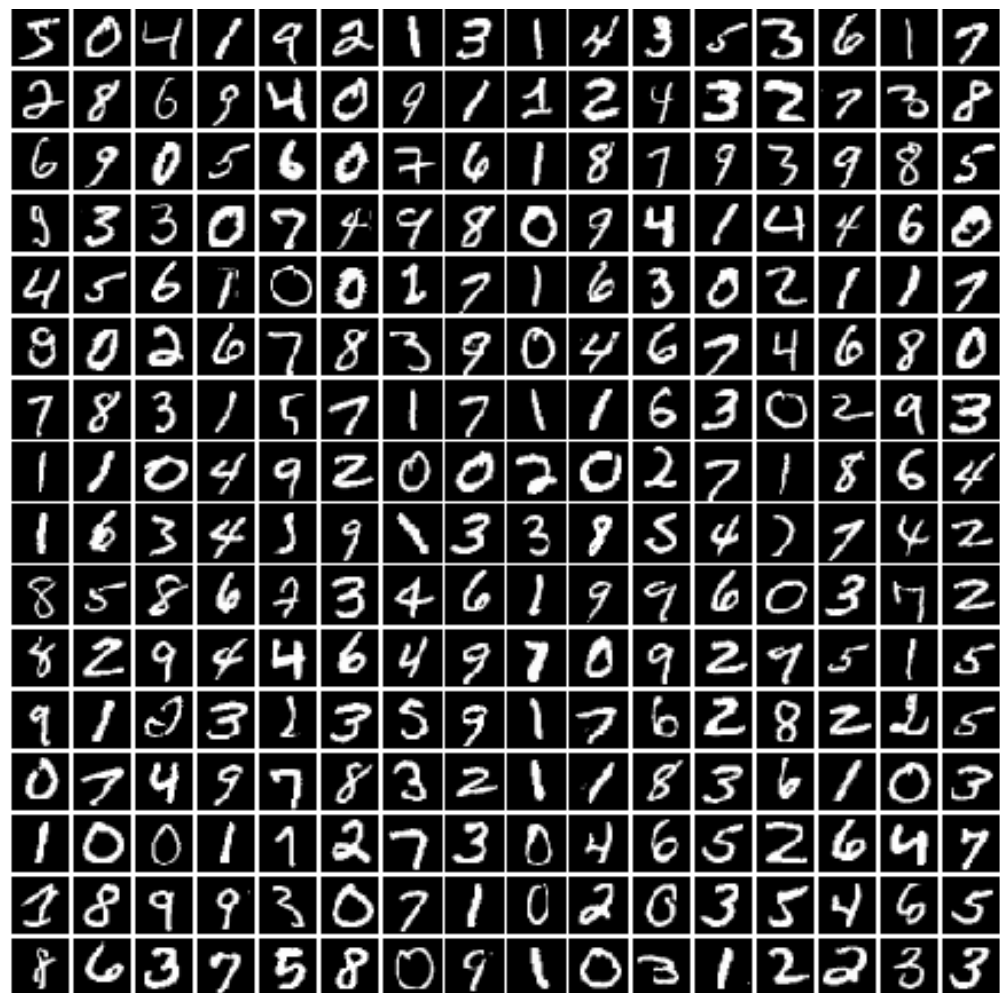
Batch Loss1 = tensor(0.4966)

Batch Loss2 = tensor(1.2389)

Exercise 9-1: CrossEntropyLoss vs NLLLoss

- What are the differences?
- Reading the document:
 - <https://pytorch.org/docs/stable/nn.html#crossentropyloss>
 - <https://pytorch.org/docs/stable/nn.html#nllloss>
- Try to know why:
 - $\text{CrossEntropyLoss} \iff \text{LogSoftmax} + \text{NLLLoss}$

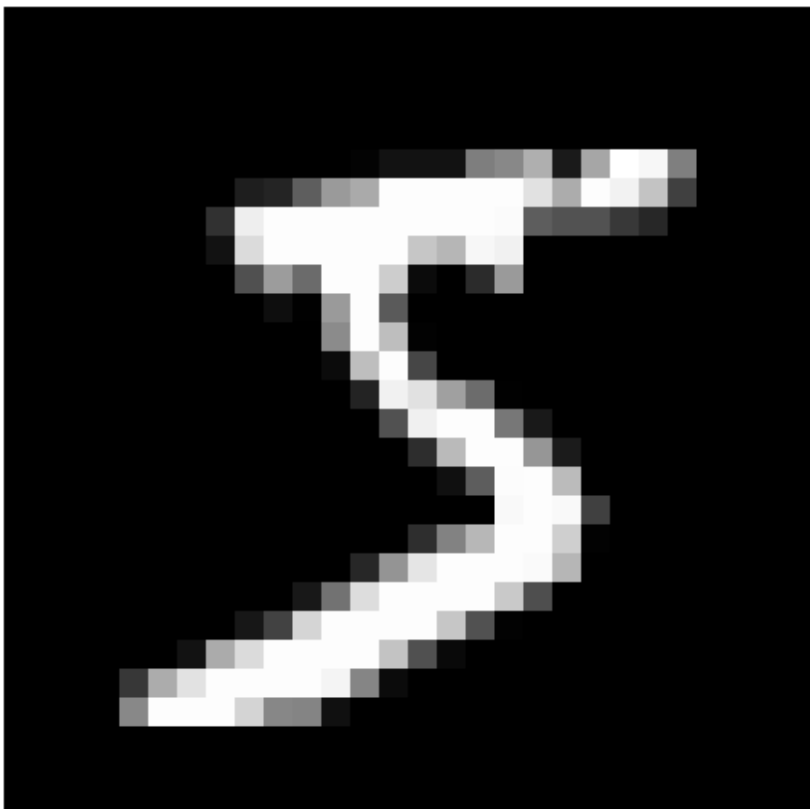
Back to MNIST Dataset



There are **10** labels in MNIST dataset.

How to design the neural network?

MNIST Dataset



$$28 * 28 = 784$$

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.5 | 0.5 | 0.7 | 0.1 | 0.7 | 1.0 | 1.0 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.4 | 0.6 | 0.7 | 1.0 | 1.0 | 1.0 | 1.0 | 0.9 | 0.7 | 1.0 | 1.0 | 0.8 | 0.3 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.4 | 0.3 | 0.3 | 0.2 | 0.2 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.8 | 0.7 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.6 | 0.4 | 1.0 | 1.0 | 0.8 | 0.0 | 0.0 | 0.2 | 0.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.6 | 1.0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.6 | 1.0 | 0.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.8 | 1.0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 1.0 | 0.9 | 0.6 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 0.9 | 1.0 | 1.0 | 0.5 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.7 | 1.0 | 1.0 | 0.6 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.4 | 1.0 | 1.0 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.5 | 0.7 | 1.0 | 1.0 | 0.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.6 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.5 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 0.8 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.3 | 0.8 | 1.0 | 1.0 | 1.0 | 1.0 | 0.8 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.7 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 0.8 | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.7 | 0.9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.5 | 1.0 | 1.0 | 1.0 | 0.8 | 0.5 | 0.5 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

Implementation of classifier to MNIST dataset

1

Prepare dataset
Dataset and Dataloader

2

Design model using Class
inherit from nn.Module

3

Construct loss and optimizer
using PyTorch API

4

Training cycle
forward, backward, update

Implementation of classifier to MNIST dataset

1

Prepare dataset
Dataset and Dataloader

2

Design model using Class
inherit from nn.Module

3

Construct loss and optimizer
using PyTorch API

4

Training cycle + **Test**
forward, backward, update

Implementation – 0. Import Package

```
import torch
from torchvision import transforms
from torchvision import datasets
from torch.utils.data import DataLoader
import torch.nn functional as F
import torch.optim as optim
```

For constructing DataLoader

图像

ReLU

Implementation – 0. Import Package

```
import torch
from torchvision import transforms
from torchvision import datasets
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.optim as optim
```

For using function relu()

Implementation – 0. Import Package

```
import torch
from torchvision import transforms
from torchvision import datasets
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.optim as optim
```

→ For constructing Optimizer

Implementation – 1. Prepare Dataset

```
batch_size = 64
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307, ), (0.3081, ))
])

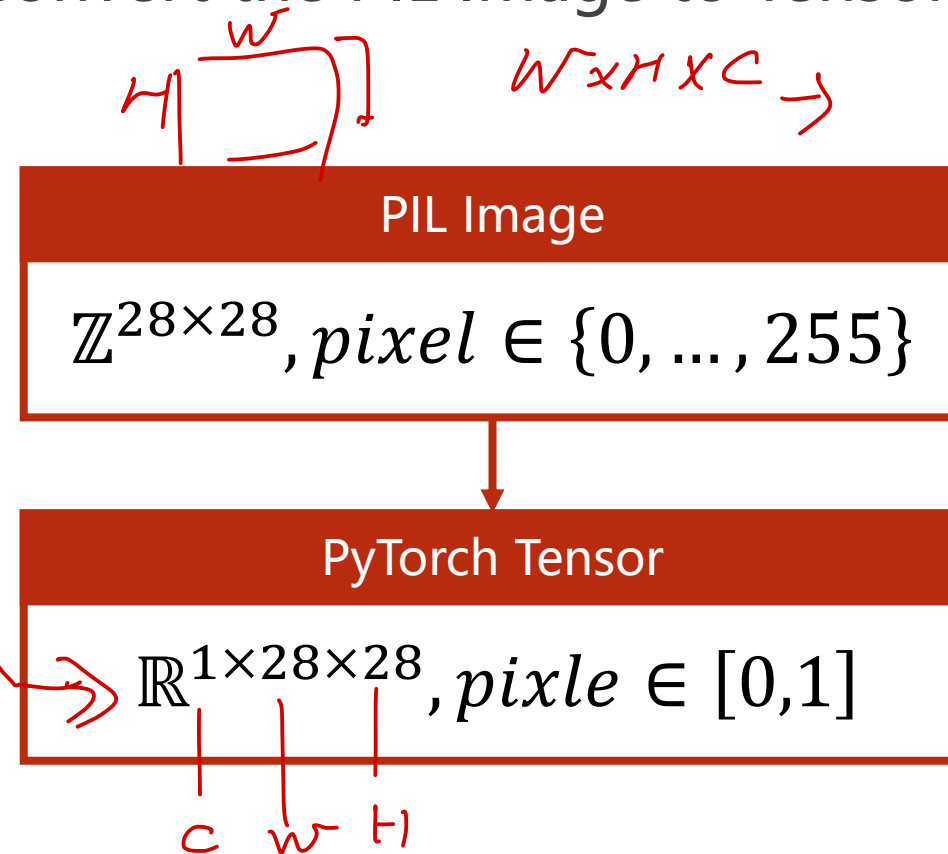
train_dataset = datasets.MNIST(root='../dataset/mnist/',
                               train=True,
                               download=True,
                               transform=transform)

train_loader = DataLoader(train_dataset,
                          shuffle=True,
                          batch_size=batch_size)

test_dataset = datasets.MNIST(root='../dataset/mnist/',
                              train=False,
                              download=True,
                              transform=transform)

test_loader = DataLoader(test_dataset,
                        shuffle=False,
                        batch_size=batch_size)
```

Convert the PIL Image to Tensor.



Implementation – 1. Prepare Dataset

```
batch_size = 64
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307, ), (0.3081, ))
])

train_dataset = datasets.MNIST(root='../dataset/mnist',
                               train=True,
                               download=True,
                               transform=transform)

train_loader = DataLoader(train_dataset,
                          shuffle=True,
                          batch_size=batch_size)

test_dataset = datasets.MNIST(root='../dataset/mnist',
                              train=False,
                              download=True,
                              transform=transform)

test_loader = DataLoader(test_dataset,
                        shuffle=False,
                        batch_size=batch_size)
```

The parameters are *mean* and *std* respectively. It use formulation below:

$$Pixel_{norm} = \frac{Pixel_{origin} - mean}{std}$$

Implementation – 1. Prepare Dataset

```
batch_size = 64
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307, ), (0.3081, ))
])

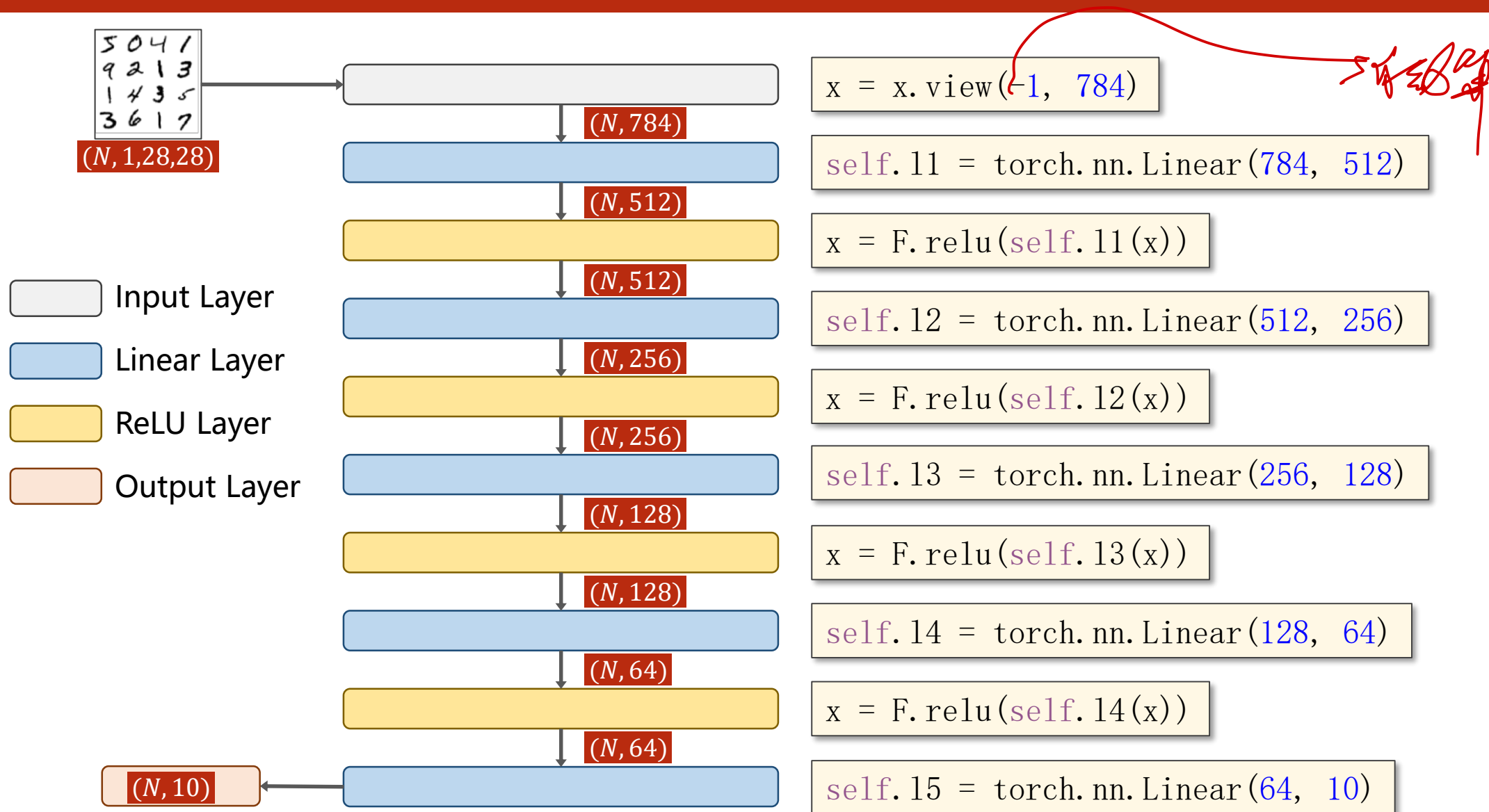
train_dataset = datasets.MNIST(root='../dataset/mnist/',
                               train=True,
                               download=True,
                               transform=transform)

train_loader = DataLoader(train_dataset,
                          shuffle=True,
                          batch_size=batch_size)

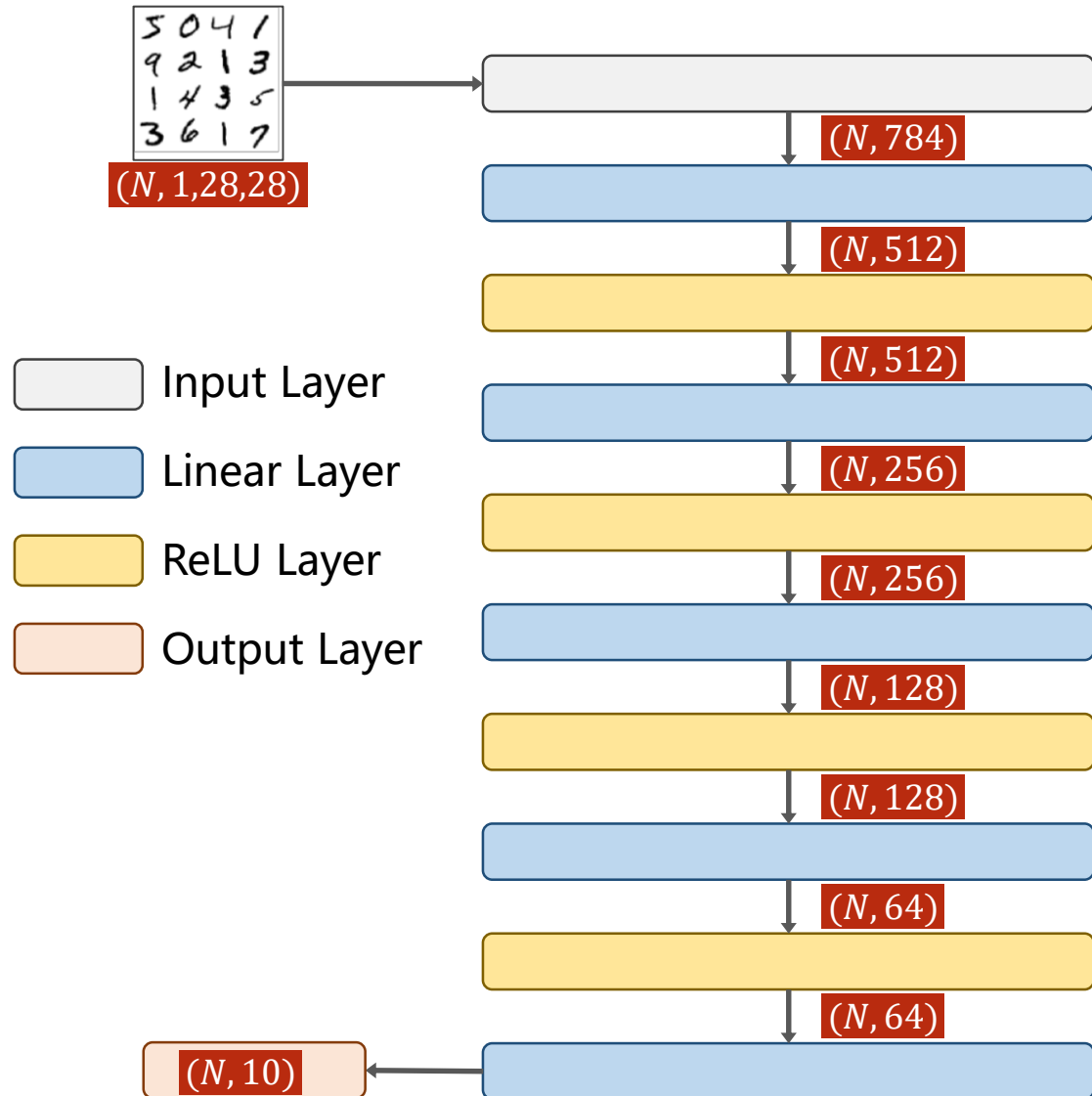
test_dataset = datasets.MNIST(root='../dataset/mnist/',
                              train=False,
                              download=True,
                              transform=transform)

test_loader = DataLoader(test_dataset,
                        shuffle=False,
                        batch_size=batch_size)
```


Implementation – 2. Design Model



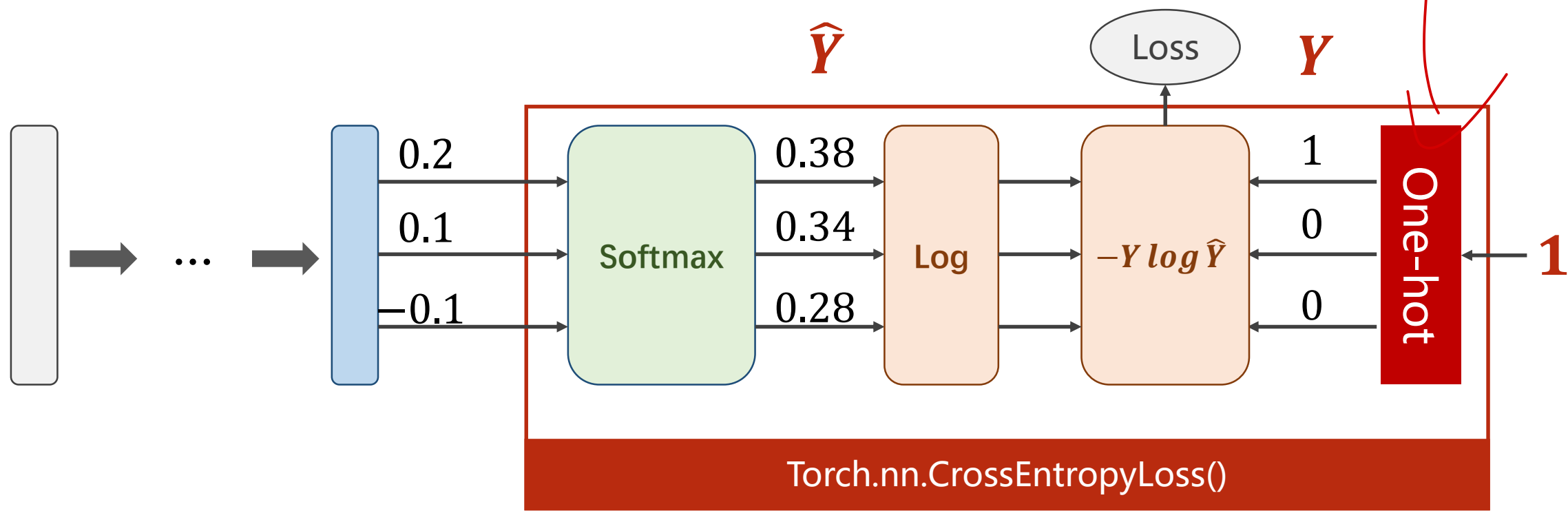
Implementation – 2. Design Model



```
class Net(torch.nn.Module):  
    def __init__(self):  
        super(Net, self).__init__()  
        self.l1 = torch.nn.Linear(784, 512)  
        self.l2 = torch.nn.Linear(512, 256)  
        self.l3 = torch.nn.Linear(256, 128)  
        self.l4 = torch.nn.Linear(128, 64)  
        self.l5 = torch.nn.Linear(64, 10)  
  
    def forward(self, x):  
        x = x.view(-1, 784)  
        x = F.relu(self.l1(x))  
        x = F.relu(self.l2(x))  
        x = F.relu(self.l3(x))  
        x = F.relu(self.l4(x))  
        return self.l5(x)  
  
model = Net()
```

Implementation – 3. Construct Loss and Optimizer

```
criterion = torch.nn.CrossEntropyLoss()  
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```



Implementation – 4. Train and Test

```
def train(epoch):  
    running_loss = 0.0  
    for batch_idx, data in enumerate(train_loader, 0):  
        inputs, target = data  
        optimizer.zero_grad()  
  
        # forward + backward + update  
        outputs = model(inputs)  
        loss = criterion(outputs, target)  
        loss.backward()  
        optimizer.step()  
  
        running_loss += loss.item()  
        if batch_idx % 300 == 299:  
            print('[%d, %5d] loss: %.3f' % (epoch + 1, batch_idx + 1, running_loss / 300))  
            running_loss = 0.0
```

Implementation – 4. Train and Test

```
def train(epoch):
    running_loss = 0.0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        optimizer.zero_grad()

        # forward + backward + update
        outputs = model(inputs)
        loss = criterion(outputs, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if batch_idx % 300 == 299:
            print('[%d, %5d] loss: %.3f' % (epoch + 1, batch_idx + 1, running_loss / 300))
            running_loss = 0.0
```

Implementation – 4. Train and Test

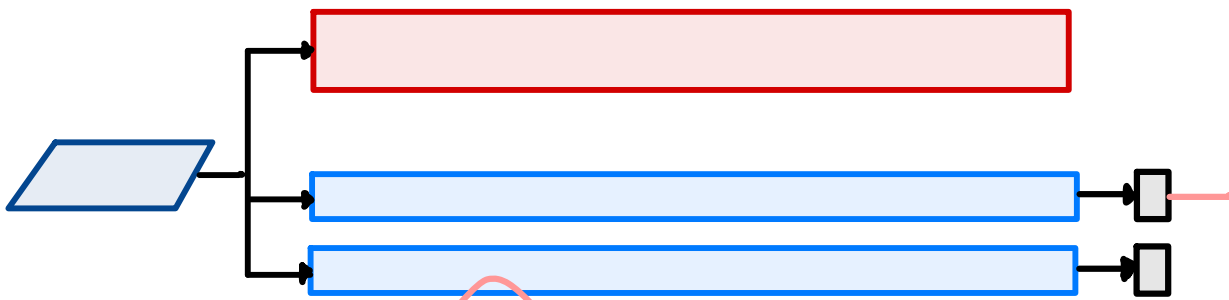
```
def train(epoch):  
    running_loss = 0.0  
    for batch_idx, data in enumerate(train_loader, 0):  
        inputs, target = data  
        optimizer.zero_grad()  
  
        # forward + backward + update  
        outputs = model(inputs)  
        loss = criterion(outputs, target)  
        loss.backward()  
        optimizer.step()  
  
        running_loss += loss.item()  
        if batch_idx % 300 == 299:  
            print('[%d, %5d] loss: %.3f' % (epoch + 1, batch_idx + 1, running_loss / 300))  
            running_loss = 0.0
```

Implementation – 4. Train and Test

```
def train(epoch):
    running_loss = 0.0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        optimizer.zero_grad()

        # forward + backward + update
        outputs = model(inputs)
        loss = criterion(outputs, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if batch_idx % 300 == 299:
            print('[%d, %5d] loss: %.3f' % (epoch + 1, batch_idx + 1, running_loss / 300))
            running_loss = 0.0
```



NCZ
 正 T
 不 F
 T 正
 真阳 TP
 假阴 FN
 F 不
 假阳 FP
 真阴 TN
 331
 20.73

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2PR}{P + R}$$

$$\text{Sen: } \frac{TP}{TP + FN}$$

NC1 + NC2

Implementation – 4. Train and Test

```
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print('Accuracy on test set: %d %%' % (100 * correct / total))
```

Implementation – 4. Train and Test

```
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print('Accuracy on test set: %d %%' % (100 * correct / total))
```

Implementation – 4. Train and Test

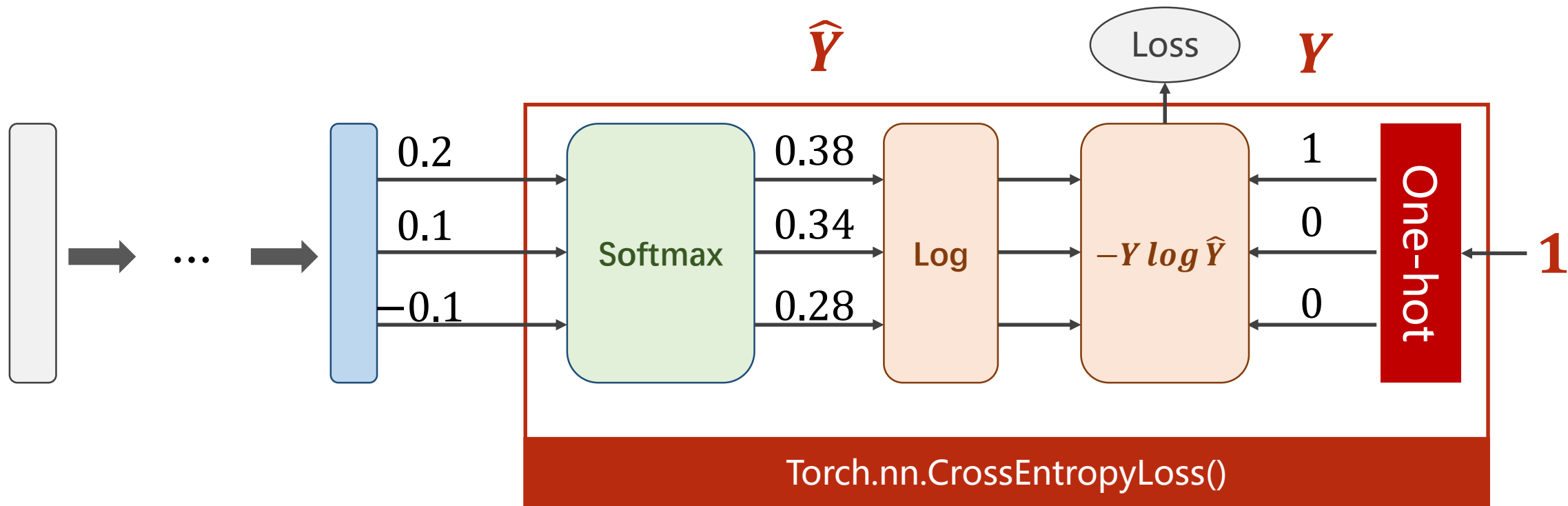
```
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print('Accuracy on test set: %d %%' % (100 * correct / total))
```

Implementation – 4. Train and Test

```
if __name__ == '__main__':  
    for epoch in range(10):  
        train(epoch)  
        test()
```

```
[1, 300] loss: 0.335  
[1, 600] loss: 0.154  
[1, 900] loss: 0.067  
Accuracy on test set: 90 %  
[2, 300] loss: 0.048  
[2, 600] loss: 0.040  
[2, 900] loss: 0.035  
Accuracy on test set: 93 %  
.....  
[9, 300] loss: 0.005  
[9, 600] loss: 0.006  
[9, 900] loss: 0.007  
Accuracy on test set: 97 %  
[10, 300] loss: 0.005  
[10, 600] loss: 0.005  
[10, 900] loss: 0.005  
Accuracy on test set: 97 %
```


Softmax and CrossEntropyLoss



Exercise 9-2: Classifier Implementation

- Try to implement a classifier for:
 - Otto Group Product Classification Challenge
 - Dataset: <https://www.kaggle.com/c/otto-group-product-classification-challenge/data>

Overview

Data

Kernels

Discussion

Leaderboard

Rules

Late Submission

Data

API

?

Download All

Data Sources

sampleSubmission....

144k x 10

test.csv

144k x 94

train.csv

61.9k x 95

About this file

No description yet

Columns

id

feat_1

feat_2

feat_3

feat_4

feat_5

feat_6



PyTorch Tutorial

09. Softmax Classifier