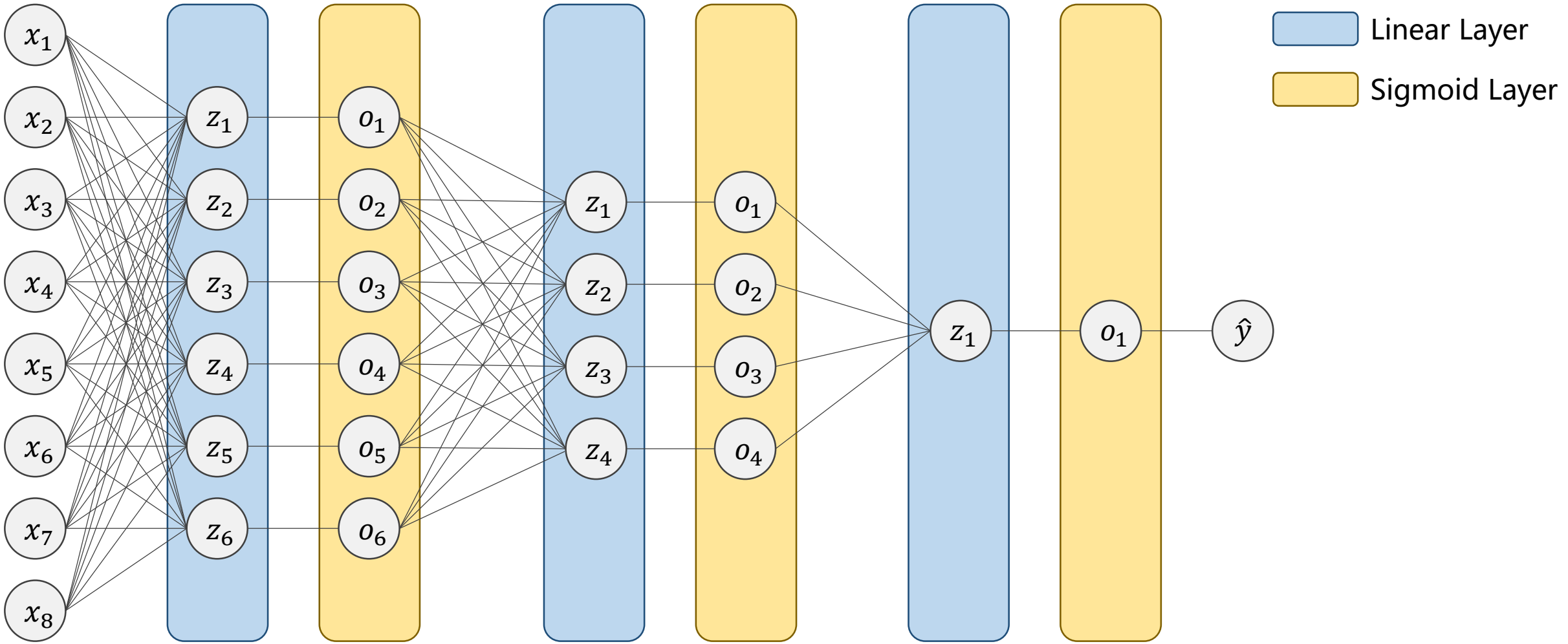
PyTorch Tutorial

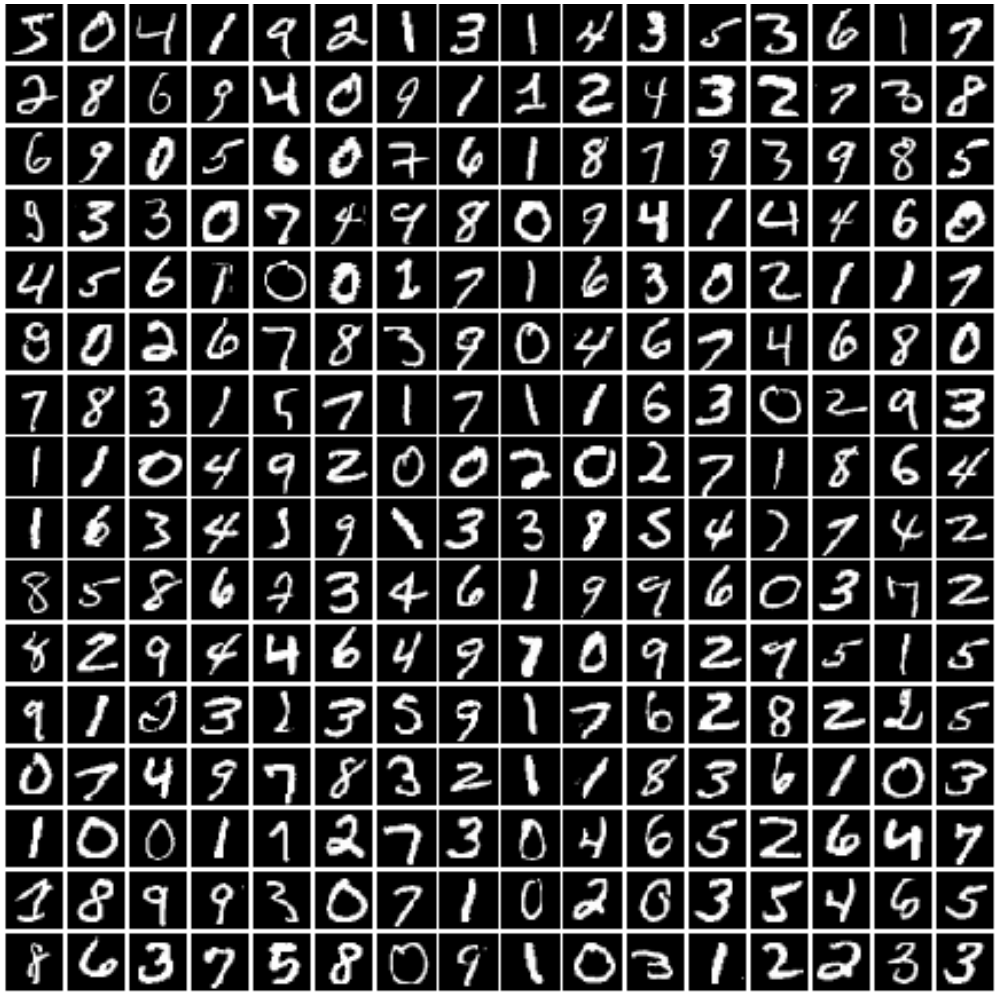
09. Softmax Classifier

Softmax 分类器

Revision: Diabetes dataset



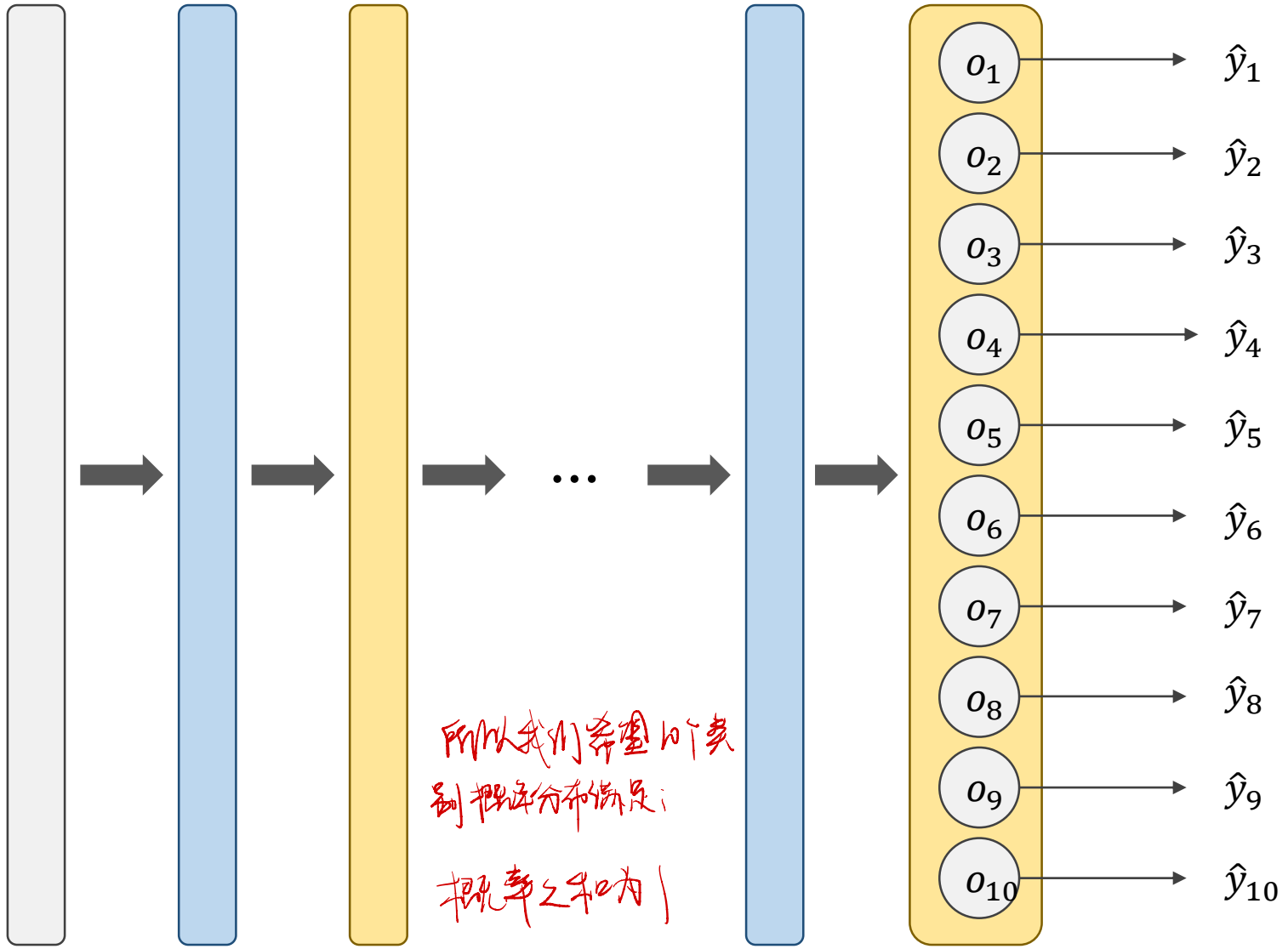
Revision: MNIST Dataset



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

How to design the neural network?

Design 10 outputs using Sigmoid?



Linear Layer

Sigmoid Layer

Input Layer

第一种设计：将输出直接表成10个输出

即将每个类别都看作一个二分类问题（是or不是）

What is wrong?

We hope the outputs is competitive!

Actually we hope the neural network outputs a **distribution**.

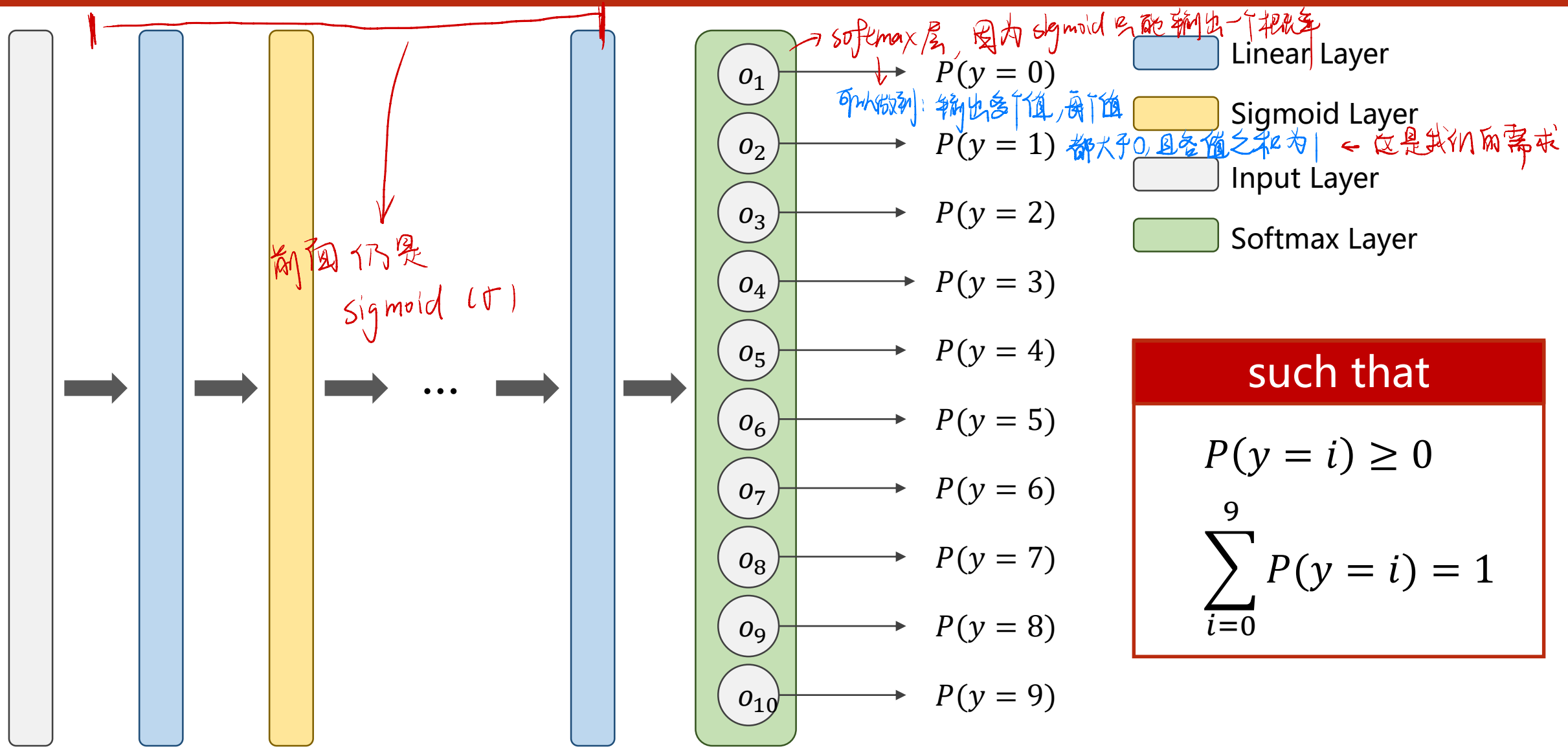
竞争性的

那时每个输出都应用二分类损失函数

$-(y\log\hat{y}+(1-y)\log(1-\hat{y}))$

但，每个类别之间有相互抑制作用，这种方式没有考虑

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:

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

为什么用指数呢？因为指数可以保证大于0

(softmax层前输出K个值x_i

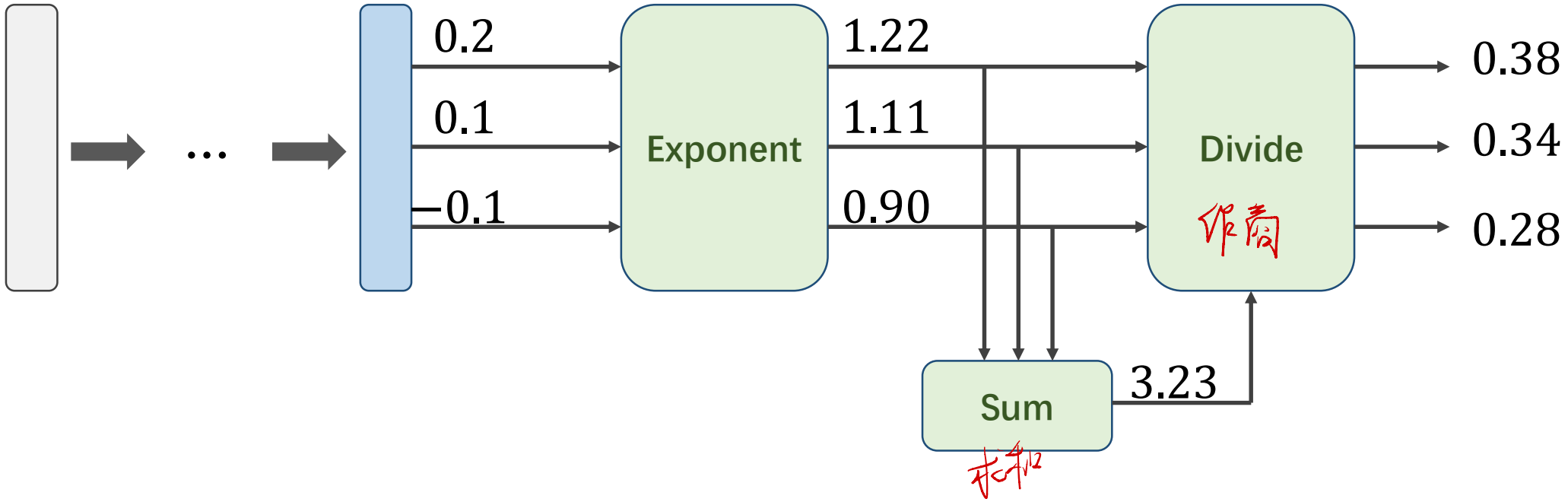
x_i是标量，

$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix}$ 是向量

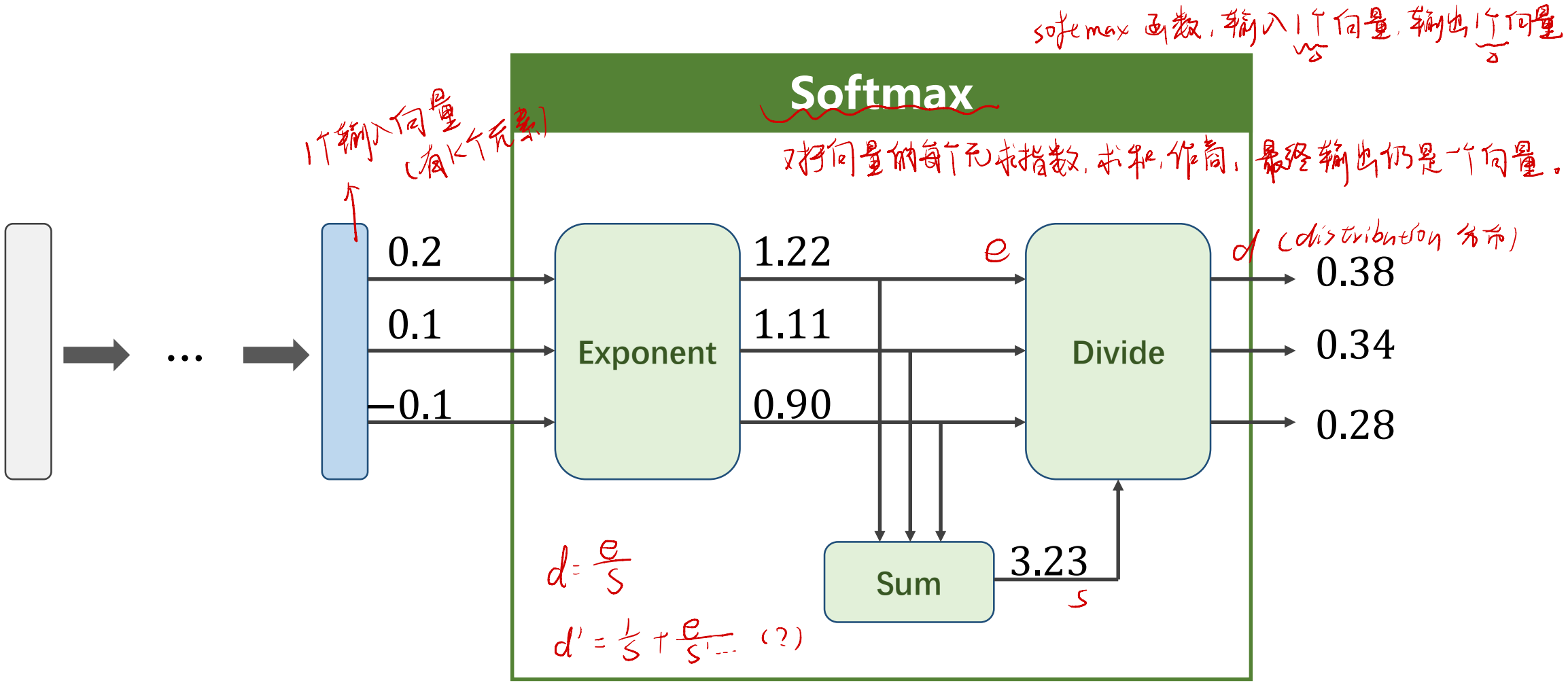
$Z_i = X_i W + b$，

分母为求和，保证概率和为1

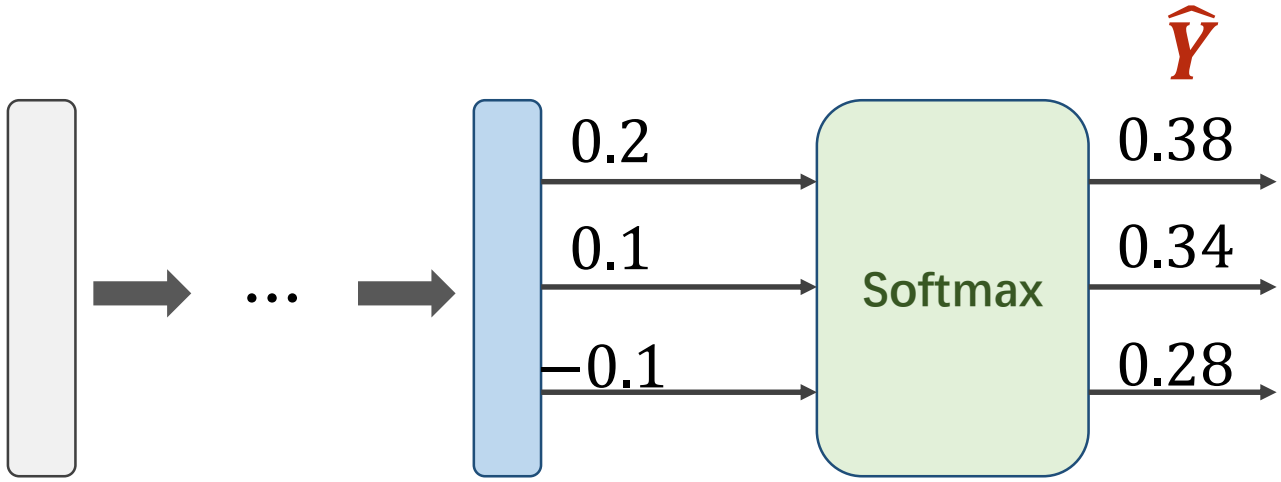
Softmax Layer - Example



Softmax Layer - Example



Loss function - Cross Entropy

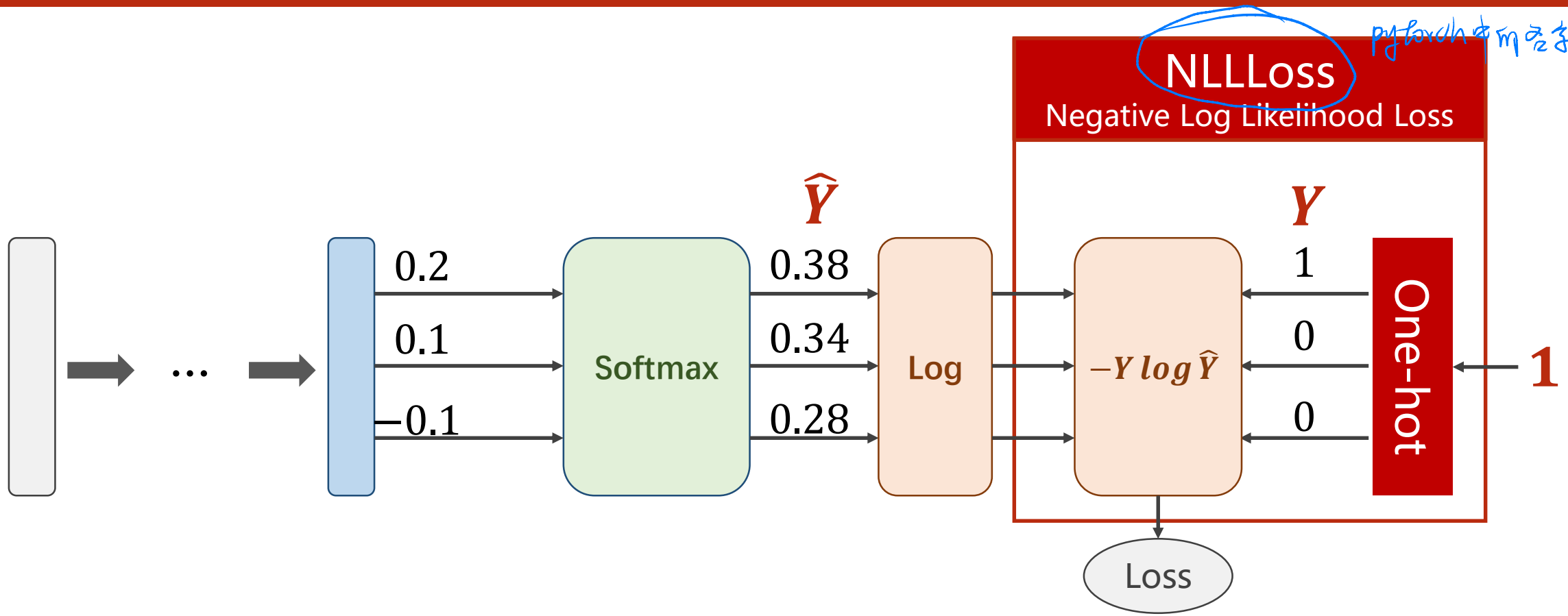


若分为小于0.5的那一方，则1-分大于0.5
那么分对应0，(1-分)对应1
也和分对应1，1-分对应0是一样的
分和1-分只是二分类中两个预测值，和多分类
中分1，分2…分n地位相同

我们看2分类的损失函数：
(交叉熵)
$-(y\log分 + (1-y)\log(1-分))$，在任何时刻，两项中只能有1项非0

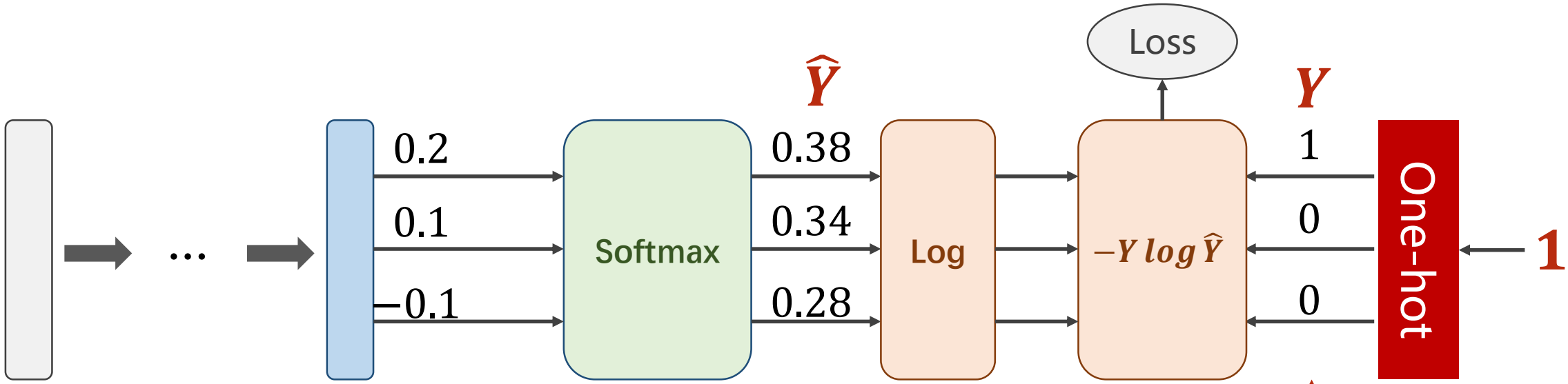
就是对于	实际分布	预测分布	两者相乘再相加，对于多分类仍相同
	1	log分	只是2项变为N项，但同一时刻
	0	log(1-分)	只能有1项为0

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)
```

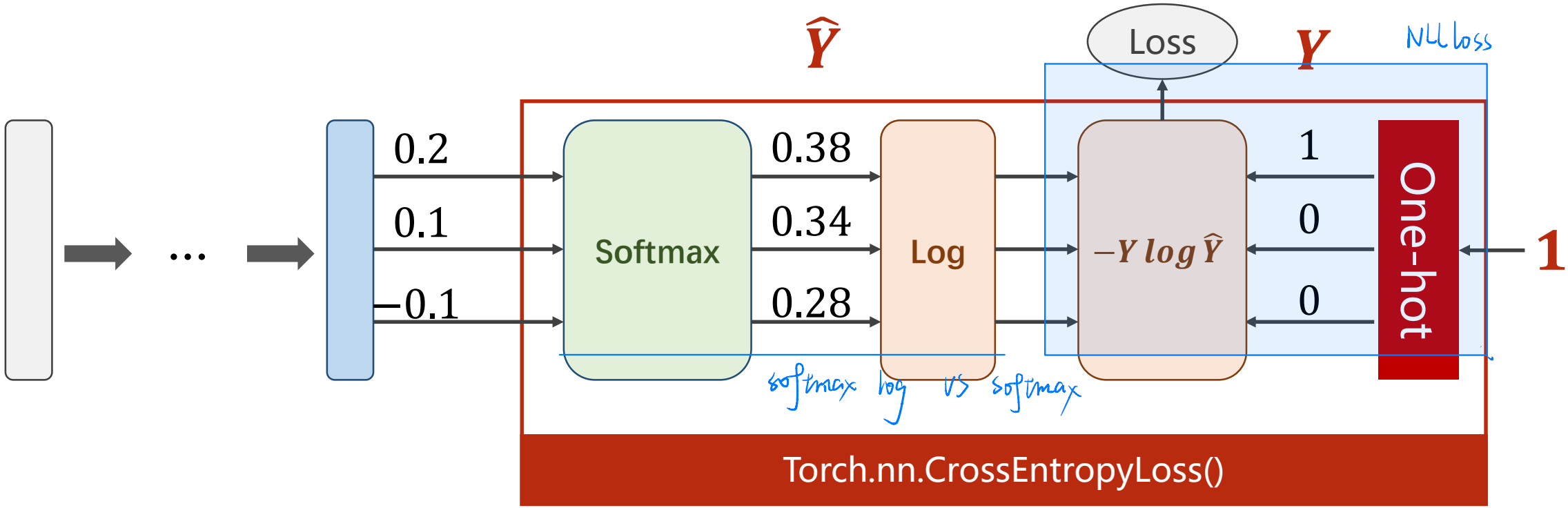
多分类交叉熵函数的实现（用 numpy）

求指数

求指数和

其中有很多的0项

Cross Entropy in PyTorch




```
import torch
y = torch.LongTensor([0])  // y是长整型张量
z = torch.Tensor([[0.2, 0.1, -0.1]])
criterion = torch.nn.CrossEntropyLoss()  //直接使用
loss = criterion(z, y)
print(loss)
```

CrossEntropyLoss 交叉熵损失

已经包括了 softmax 非线性变换.

* 所以在使用交叉熵损失时，神经网络最后一层不要作激活 (不作非线性变换)

Mini-Batch: *batch_size=3*

例：

```
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)
```

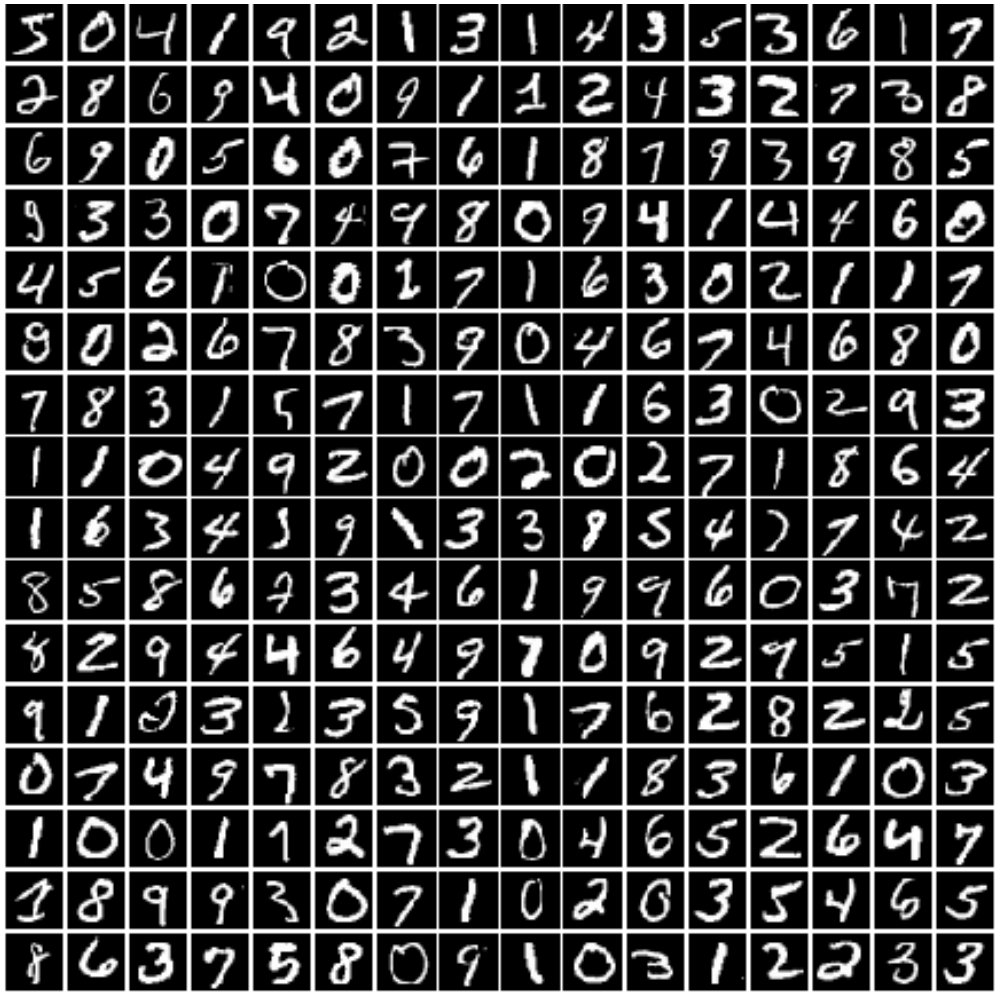
Batch Loss1 = tensor(0.4966)

Batch Loss2 = tensor(1.2389)

Exercise 9-1: CrossEntropyLoss vs NLLLoss

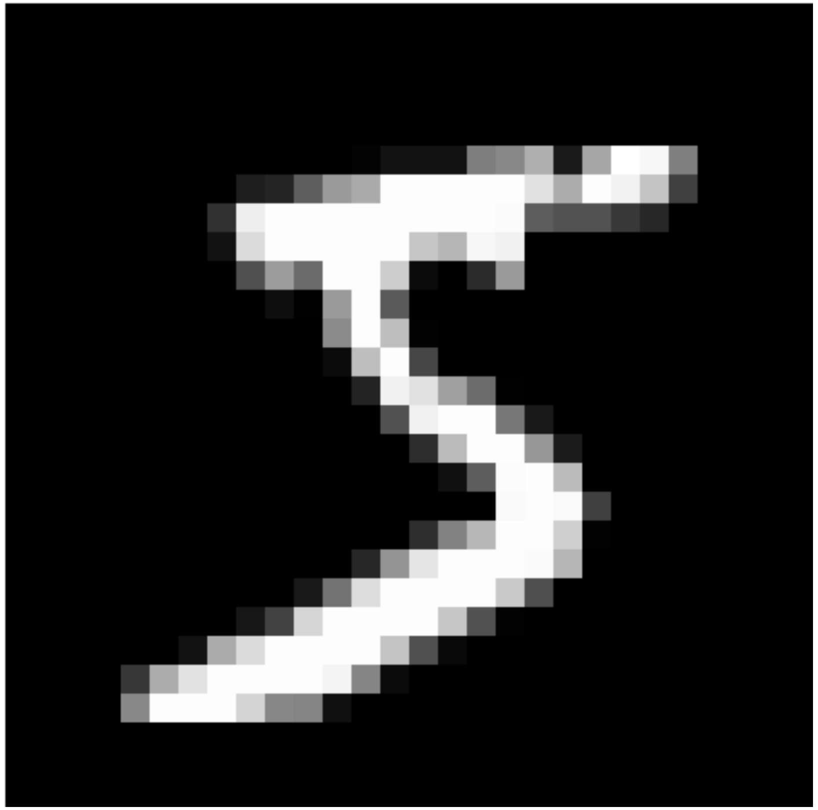
- What are the differences?
- Reading the document:
  - https://pytorch.org/docs/stable/nn.html#crossentropyloss
  - https://pytorch.org/docs/stable/nn.html#nllloss
- Try to know why:
  - CrossEntropyLoss <==> LogSoftmax + NLLLoss

Back to MNIST Dataset



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

How to design the neural network?



$28 * 28 = 784$

图像用矩阵保存

数字越大越亮

数字越小越深

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	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	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	0.0	0.0	0.2	0.9	1.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	0.0	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	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 + **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

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

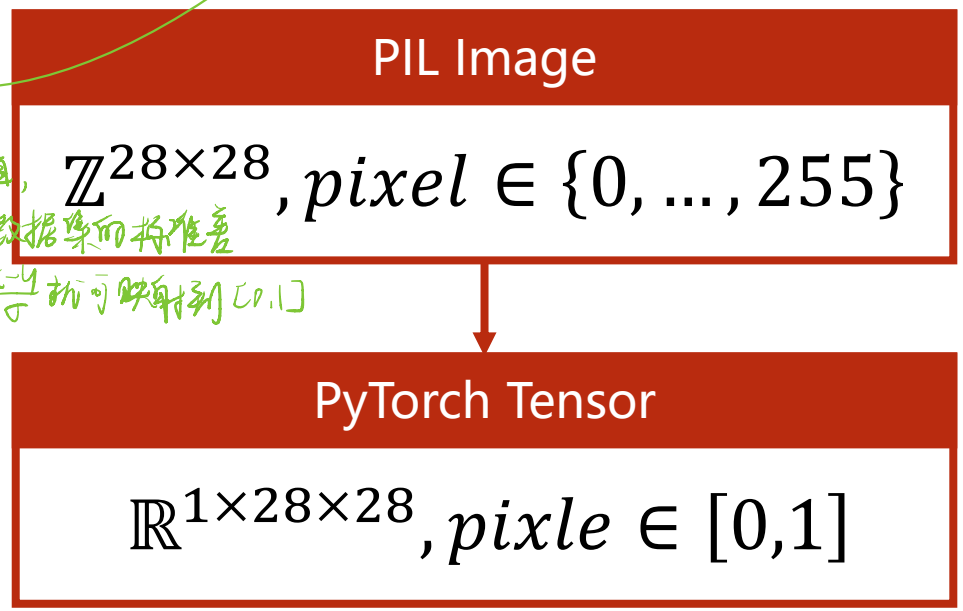
```
import torch
from torchvision import transforms
from torchvision import datasets
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.optim as optim
```

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.



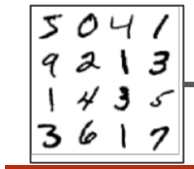
Implementation – 1. Prepare Dataset

```
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 – 2. Design Model



$(N, 1, 28, 28)$

$(N, 784)$

$(N, 512)$

$(N, 512)$

$(N, 256)$

$(N, 256)$

$(N, 128)$

$(N, 128)$

$(N, 64)$

$(N, 64)$

$(N, 10)$

Input Layer

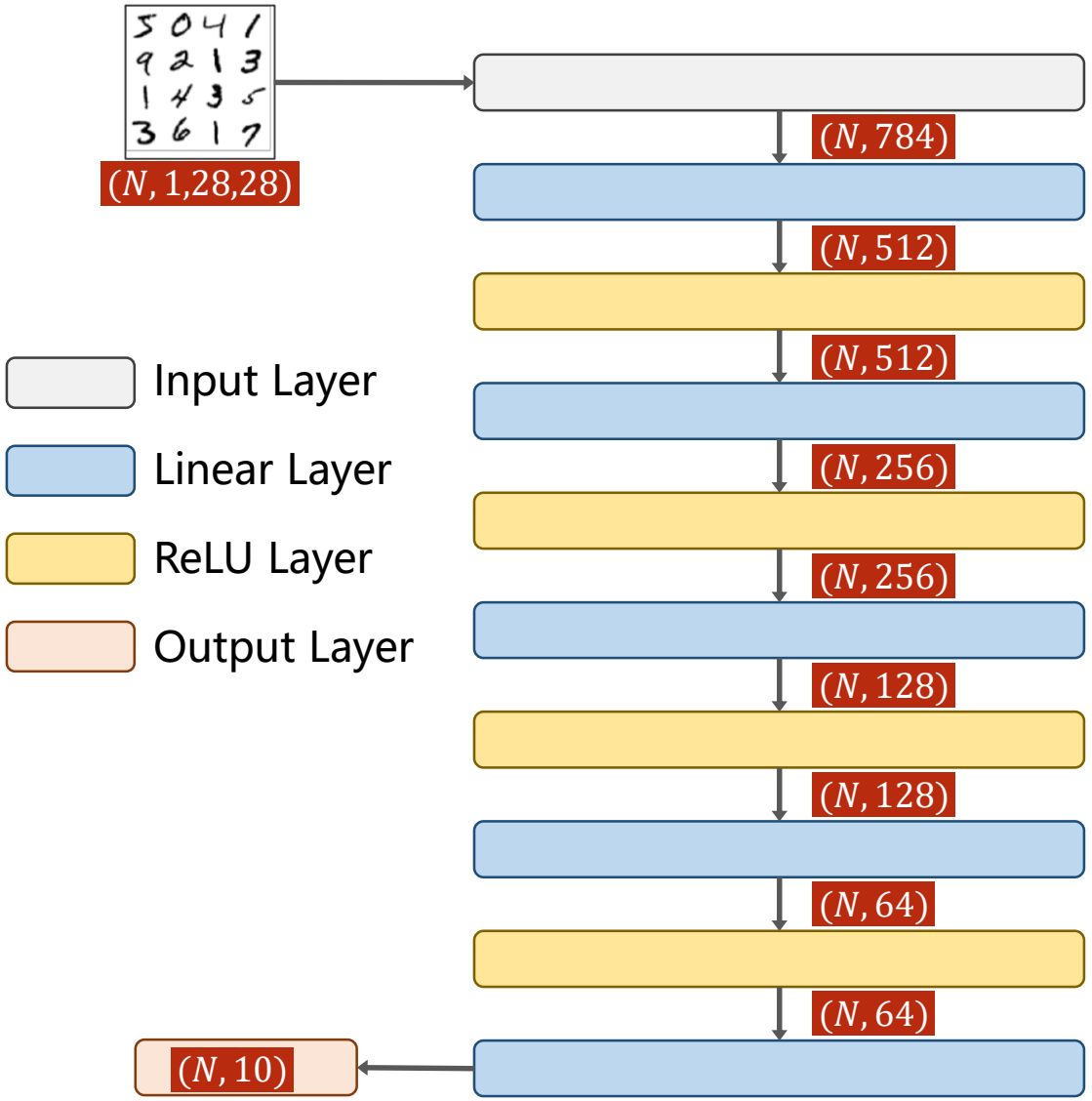
Linear Layer

ReLU Layer

Output Layer

```
x = x.view(-1, 784)
self.l1 = torch.nn.Linear(784, 512)
x = F.relu(self.l1(x))
self.l2 = torch.nn.Linear(512, 256)
x = F.relu(self.l2(x))
self.l3 = torch.nn.Linear(256, 128)
x = F.relu(self.l3(x))
self.l4 = torch.nn.Linear(128, 64)
x = F.relu(self.l4(x))
self.l5 = torch.nn.Linear(64, 10)
```

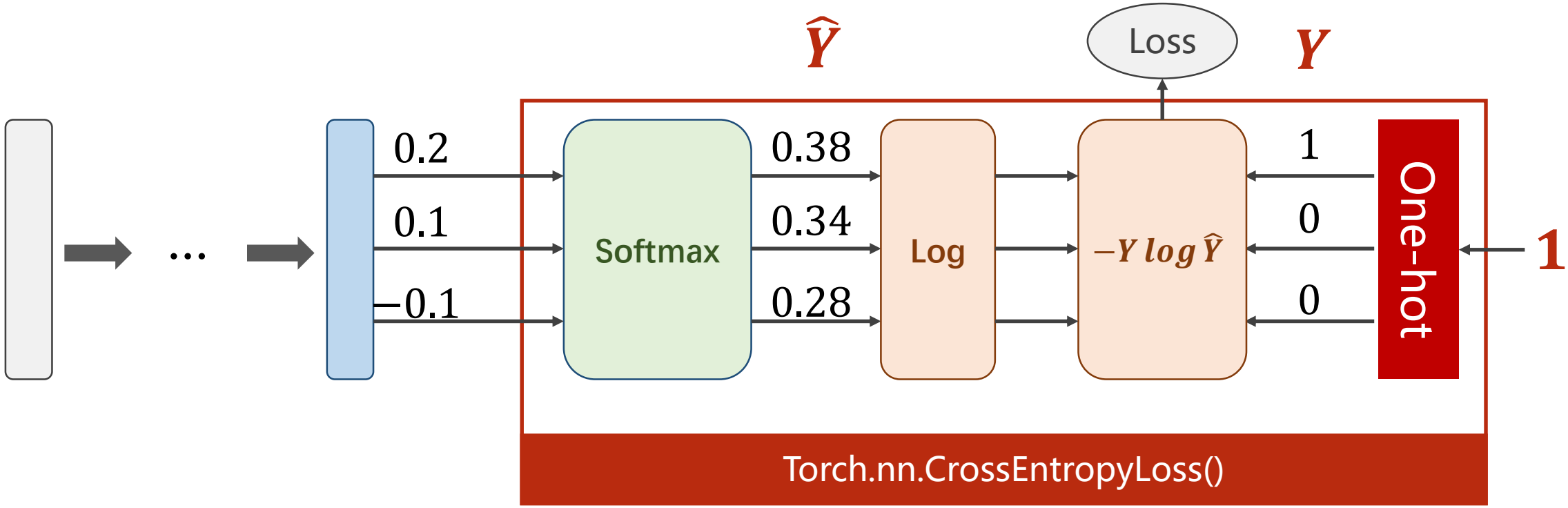
Implementation – 2. Design Model




```
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() //不构建计算图
        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 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()
```

一轮训练一轮测试

全连接：把所有数据（有用 and 没用）都用来训练

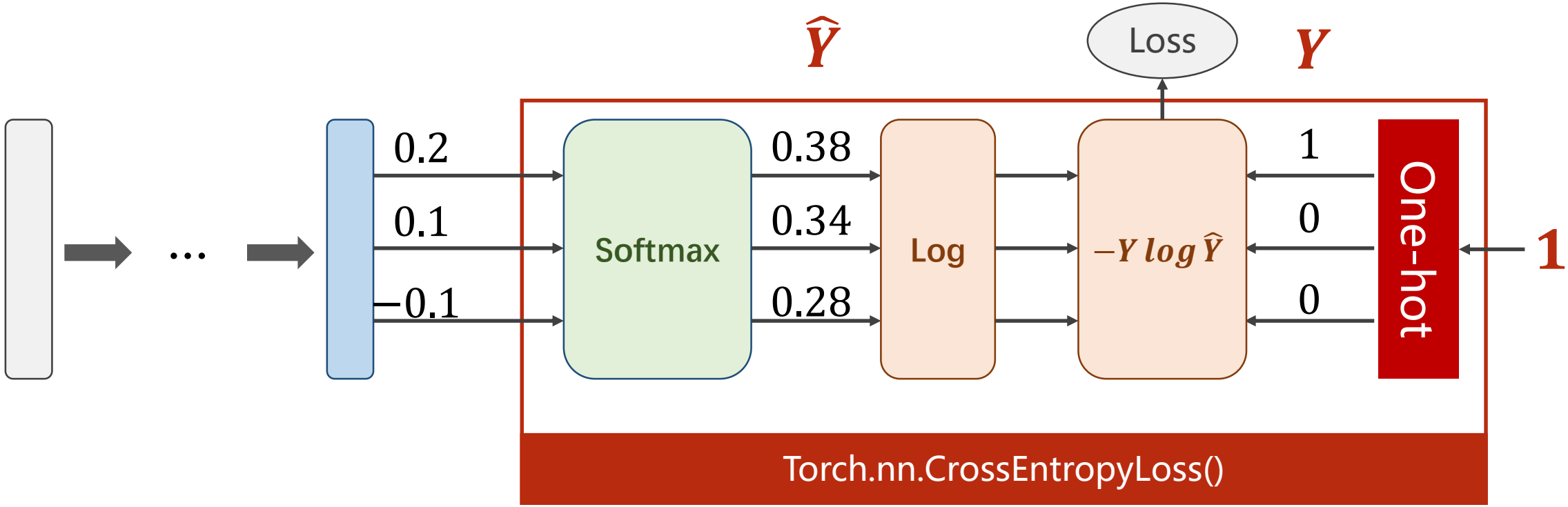
特征提取，只用特征训练 e.g. CNN

e.g. FFT，小波 ...

自动：CNN

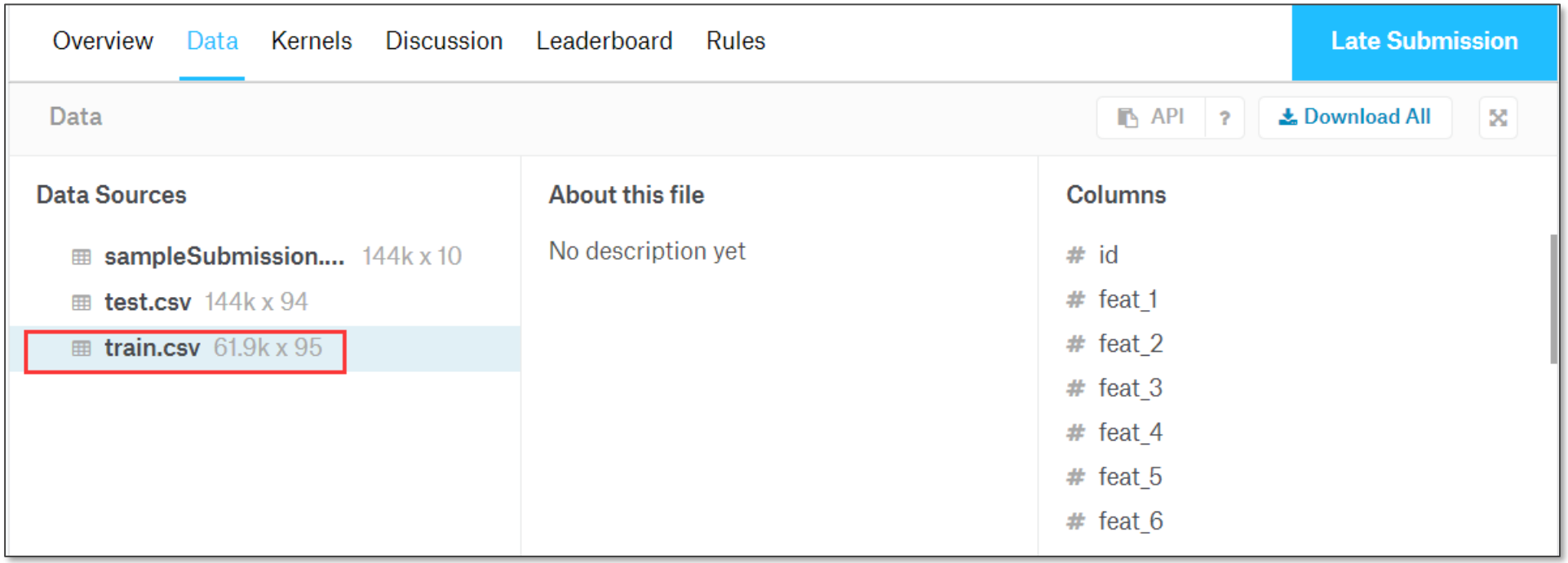
```
[1,   300] loss: 0.335
[1,   600] loss: 0.154
[1,   900] loss: 0.067
Accuracy on test set: 90 %
[2,   300] loss: 0.048
[2,   600] loss: 0.040
[2,   900] loss: 0.035
Accuracy on test set: 93 %
.....................................
[9,   300] loss: 0.005
[9,   600] loss: 0.006
[9,   900] loss: 0.007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



PyTorch Tutorial

09. Softmax Classifier

PyTorch Tutorial

09. Softmax Classifier

PyTorch Tutorial

09. Softmax Classifier