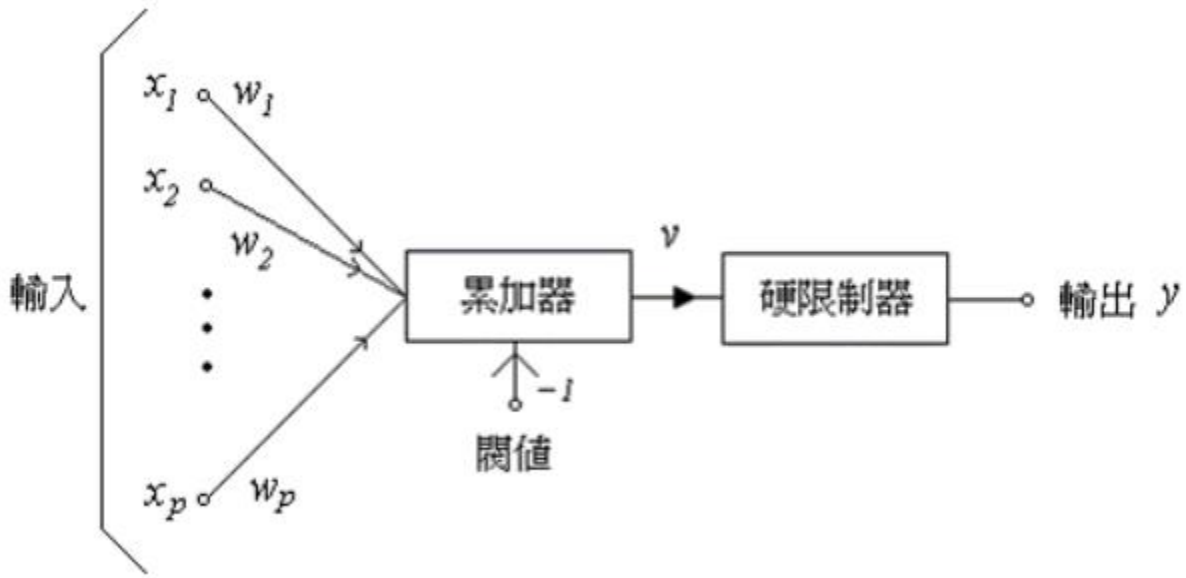
機器學習及其應用

MLP

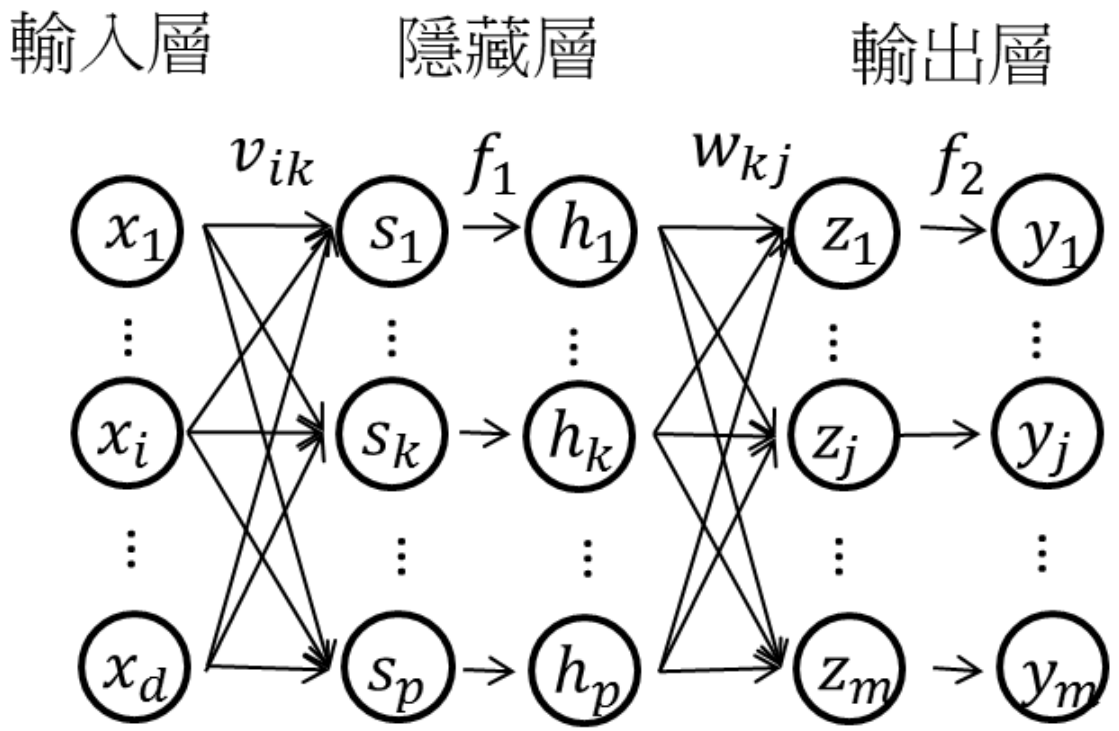
感知機

- 感知機的基本組成元件為一個具有線性組合功能的累加器，後接一個硬限制器 (hard limiter) 而成
- 累加器->神經網路訓練
- 硬限制器->分類
- 分類的判斷以感知機的輸出



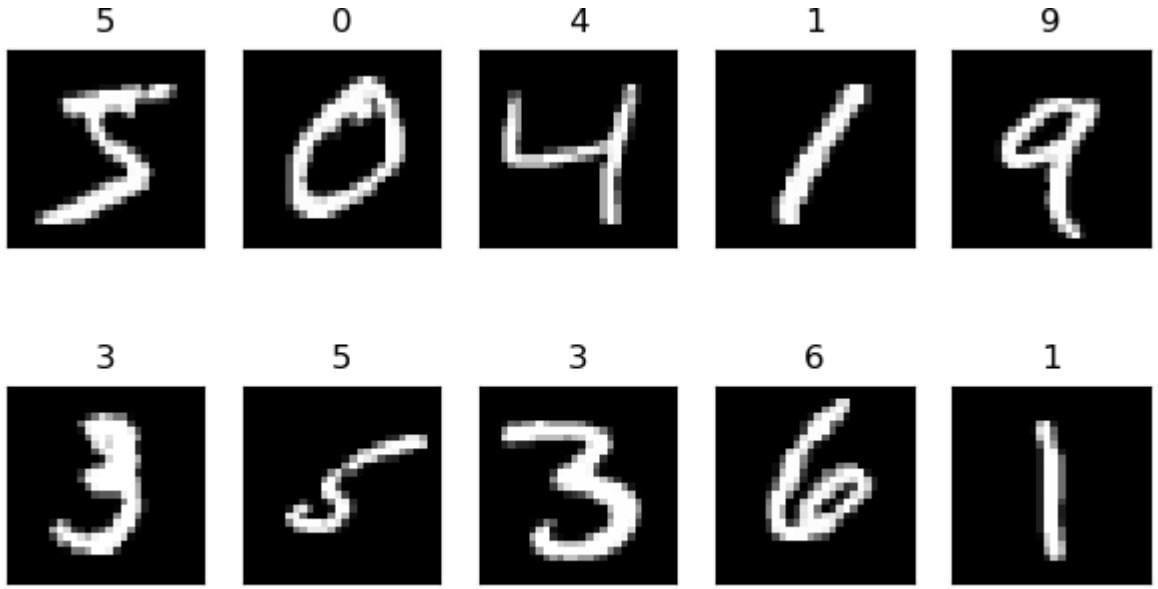
多層感知機（MLP）

- 為多層架構
- 採用監督式學習
- 輸出端包含一個非線性之活化函數
- 須包含一層以上的隱藏層
- 具高度連結性

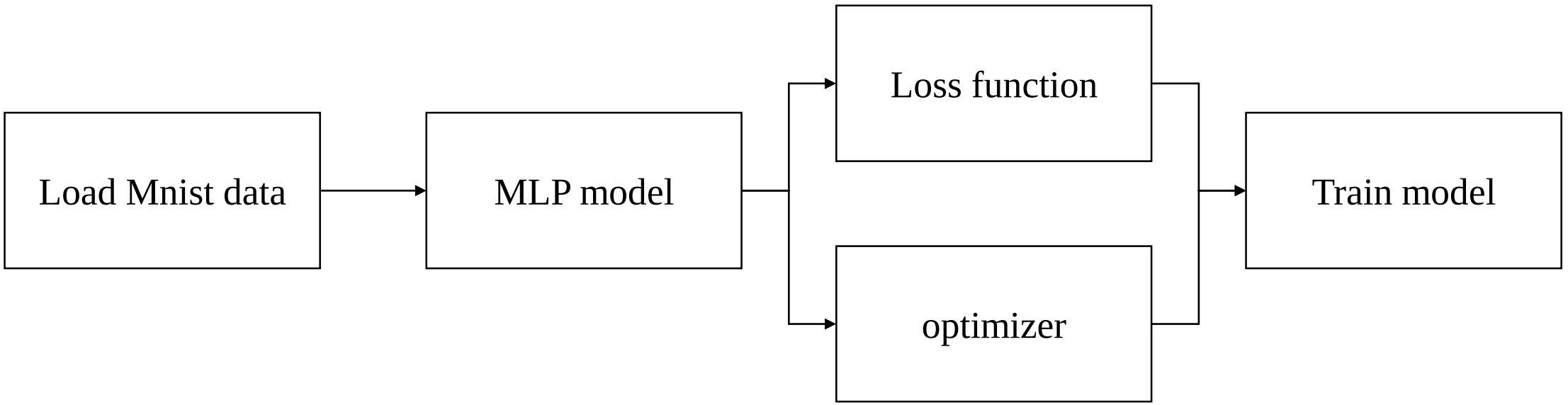


Mnist

- 手寫數字資料集



流程


Load Mnist data
MLP model
Loss function
optimizer
Train model

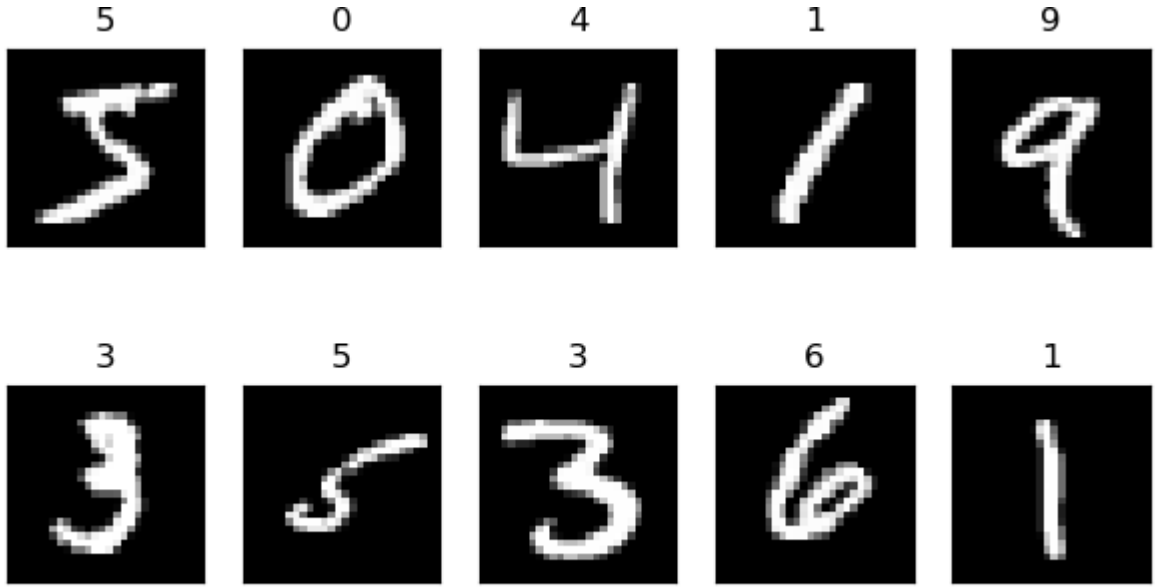
Load Mnist data

```
train_data = mnist.MNIST('data/mnist', train = True, transform = data_transform, download=True)
test_data = mnist.MNIST('data/mnist', train=False, transform= data_transform, download=True)

train_loader = DataLoader(train_data, batch_size = 64, shuffle = True)
test_loader = DataLoader(test_data, batch_size = 64, shuffle = True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to data/mnist\MNIST\raw\train-images-idx3-ubyte.gz
100.1%Extracting data/mnist\MNIST\raw\train-images-idx3-ubyte.gz to data/mnist\MNIST\raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to data/mnist\MNIST\raw\train-labels-idx1-ubyte.gz
113.5%Extracting data/mnist\MNIST\raw\train-labels-idx1-ubyte.gz to data/mnist\MNIST\raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to data/mnist\MNIST\raw\t10k-images-idx3-ubyte.gz
100.4%Extracting data/mnist\MNIST\raw\t10k-images-idx3-ubyte.gz to data/mnist\MNIST\raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to data/mnist\MNIST\raw\t10k-labels-idx1-ubyte.gz
180.4%Extracting data/mnist\MNIST\raw\t10k-labels-idx1-ubyte.gz to data/mnist\MNIST\raw
Processing...
```

Load Mnist data



Torch.tensor

```
def data_transform(x):
    x = np.array(x, dtype='float32')
    x = (x - 0.5)/0.5
    x = x.reshape((-1,))
    x = torch.from_numpy(x)

    return x
```

MLP model

```python
class MLP(nn.Module):
    def __init__(self, input_dim, out_dim):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(input_dim, 500)
        self.linear2 = nn.Linear(500, 250)
        self.linear3 = nn.Linear(250, 125)
        self.linear4 = nn.Linear(125, out_dim)
        self.fc = nn.Sigmoid()

    def forward(self, x):
        x = self.linear1(x)
        x = self.linear2(x)
        x = self.linear3(x)
        x = self.linear4(x)
        x = self.fc(x)

        return x
```

```
MLP(
  (linear1): Linear(in_features=784, out_features=500, bias=True)
  (linear2): Linear(in_features=500, out_features=250, bias=True)
  (linear3): Linear(in_features=250, out_features=125, bias=True)
  (linear4): Linear(in_features=125, out_features=10, bias=True)
  (fc): Sigmoid()
)
```

Loss function & optimizer

```
model = MLP(784, 10)
print(model)
loss_f = nn.CrossEntropyLoss()
opt = optim.Adam(model.parameters(), lr = 1e-3)
epoch = 5
```

Training data: $(x^n, \hat{y}^n)$

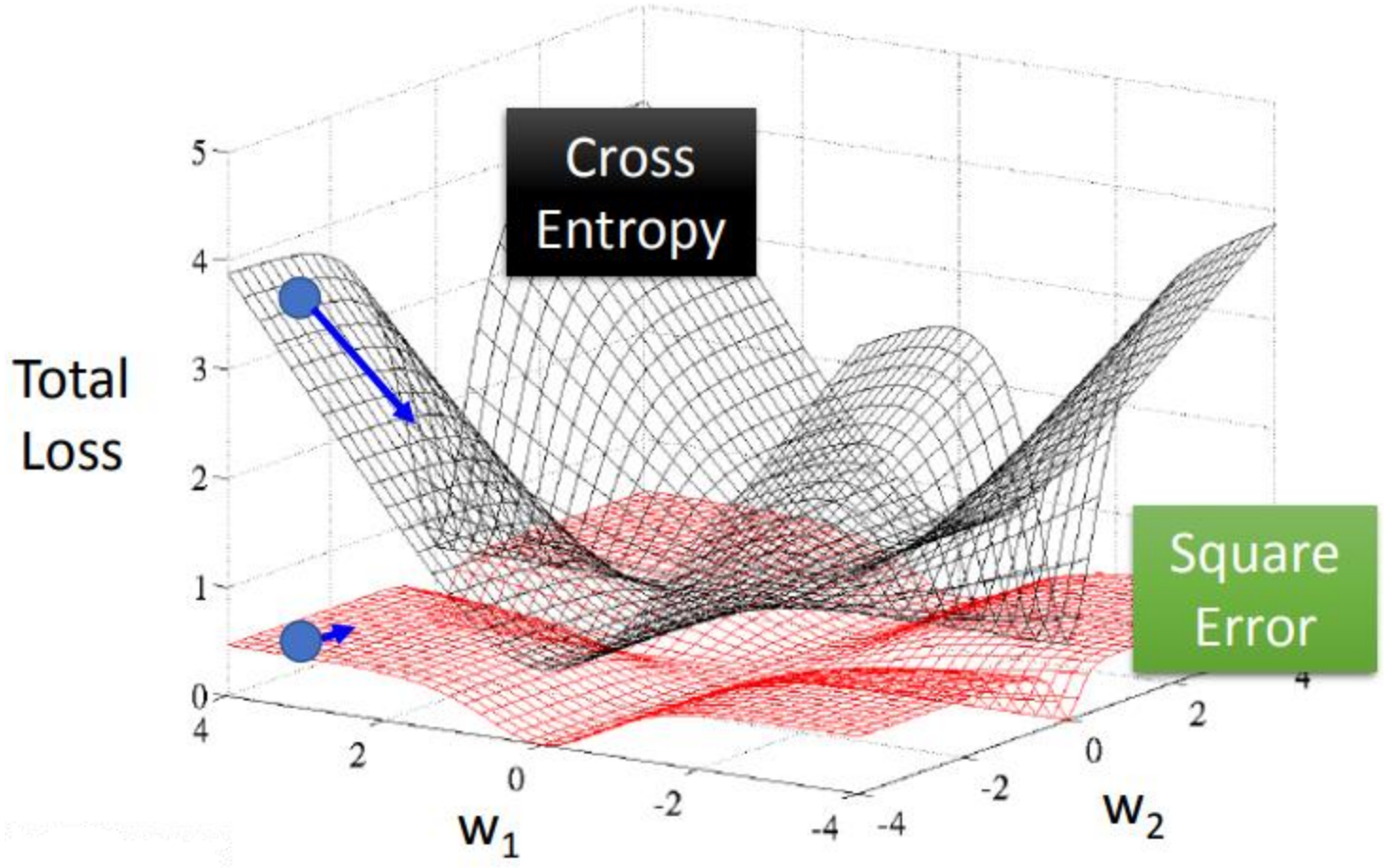
$\hat{y}^n$: 1 for class 1, 0 for class 2

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

Cross entropy:

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n ln f(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Loss function & optimizer



Train model

```python
def train():
    train_loss = 0
    train_acc = 0
    model.train()
    for im, label in train_loader:
        im = Variable(im)
        label = Variable(label)
        pred = model(im)
        loss = loss_f(pred, label)
        opt.zero_grad()
        loss.backward()
        opt.step()

        train_loss += loss
        _, y = pred.max(1)
        correct = (y == label).sum().item()
        acc = correct / im.shape[0]
        train_acc += acc

    return train_loss, train_acc
```

Train model

```python
def test():
    test_loss = 0
    test_acc = 0
    model.eval()
    for im, label in test_loader:
        im = Variable(im)
        label = Variable(label)
        out = model(im)
        loss = loss_f(out, label)
        test_loss += loss.item()
        _, pred = out.max(1)
        num_correct = (pred == label).sum().item()
        acc = num_correct / im.shape[0]
        test_acc += acc

    return test_loss, test_acc
```

```python
for i in range(epoch+1):
    train_loss, train_acc = train()
    test_loss, test_acc = test()

    losses.append(train_loss/len(train_loader))
    acces.append(train_acc/len(train_loader))
    test_losses.append(test_loss/len(test_loader))
    test_acces.append(test_acc/len(test_loader))
    print('epoch: {}, Train Loss: {:.6f}, Train Acc: {:.6f}, test Loss: {:.6f}, test Acc: {:.6f}'
          .format(i, train_loss / len(train_loader), train_acc / len(train_loader),
          test_loss / len(test_loader), test_acc / len(test_loader)))
```

Result

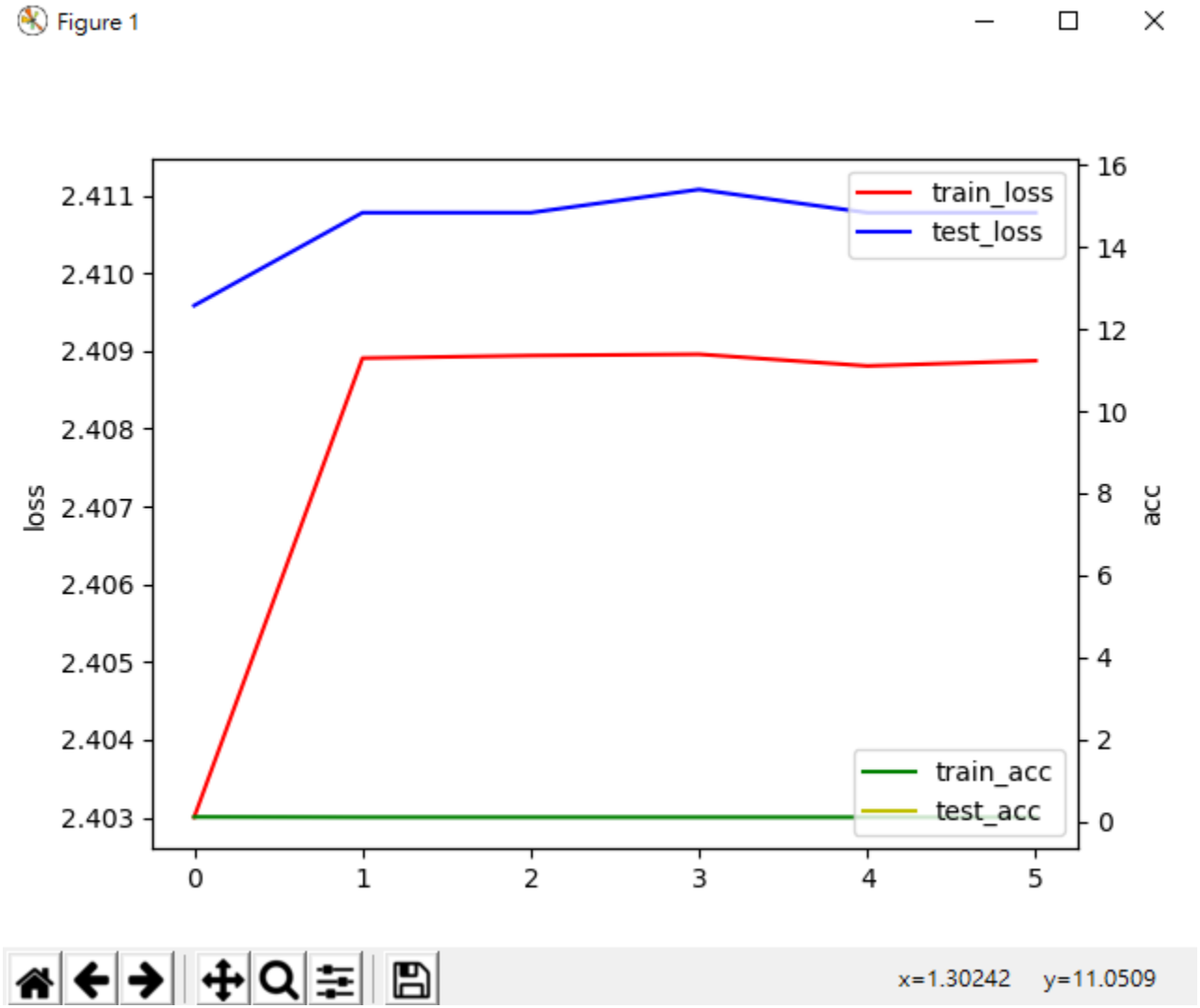


Figure 1
train_loss
test_loss
train_acc
test_acc
loss
acc
2.411
2.410
2.409
2.408
2.407
2.406
2.405
2.404
2.403
16
14
12
10
8
6
4
2
0
0
1
2
3
4
5
x=1.30242
y=11.0509

Exercise: 579

提示

- Label 需轉換為 One hot
- Loss 計算 y 需轉為一維

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

```
y = pd.get_dummies(df[2]).to_numpy()
y = torch.from_numpy(y)
```

Result

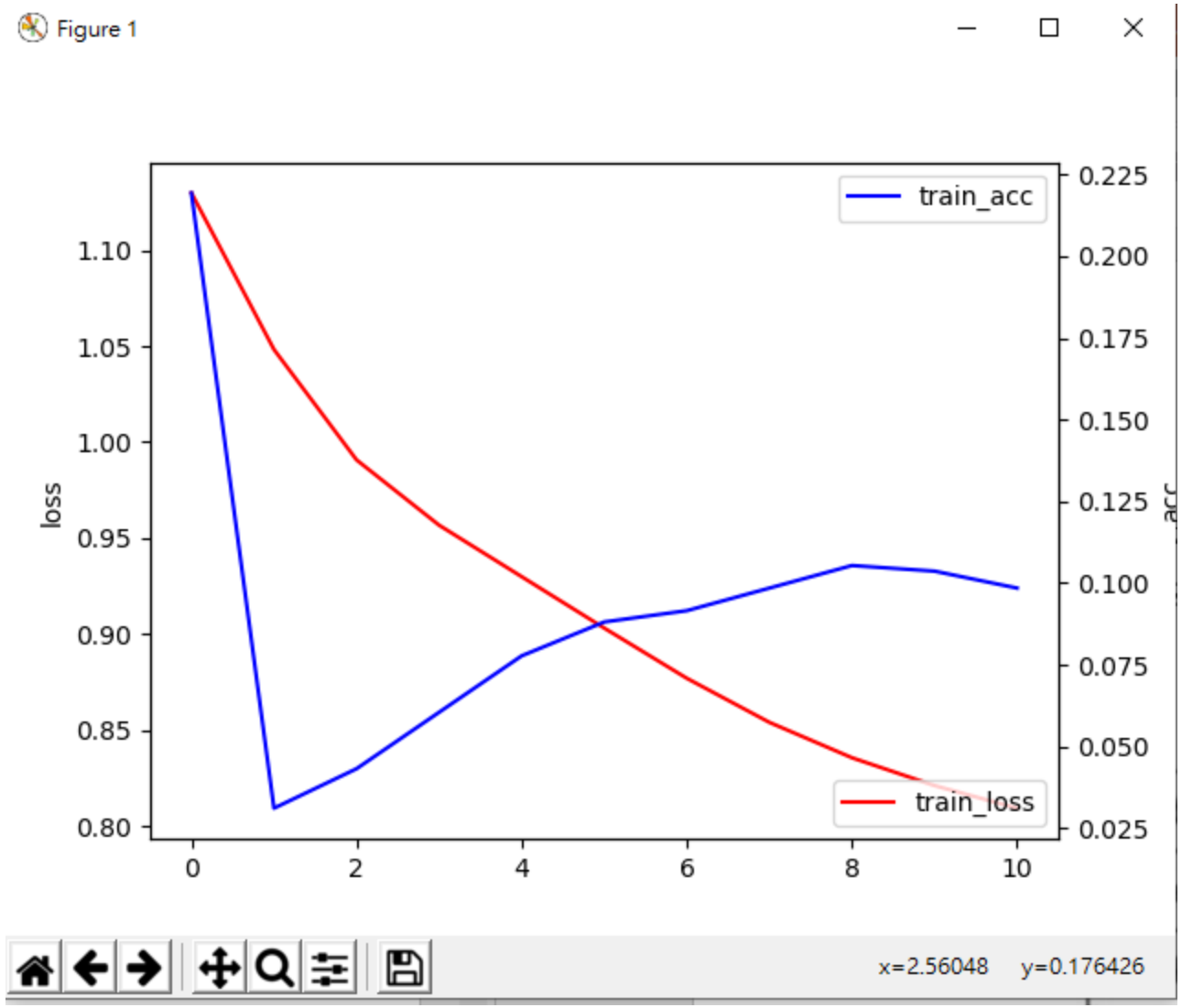


Figure 1
train_acc
train_loss
loss
acc
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
2
4
6
8
10
x=2.56048
y=0.176426