```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, TensorDataset
import pandas as pd
```

Solve the following problems in Pytorch_multinomial_regression.ipynb

1> Solve Exercise 2.3.4 (Iris data)

2> Solve Exercise 2.3.5 (Fashion Mnist data)

```
#1 exercise 2.3.4(iris data)

```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()

# Convert the data to PyTorch tensors
X = torch.tensor(iris.data, dtype=torch.float32)[:,:2] #x는 [n,4]4가지 설명변수 2가지만 가지고 하
y = torch.tensor(iris.target, dtype=torch.long)

t = torch.nn.functional.one_hot(torch.tensor(y, dtype=torch.int64), num_classes=3)
t = torch.tensor(t, dtype=torch.float32)


# Normalize the input data
mean = torch.mean(X, dim=0)
std = torch.std(X, dim=0)
X = (X - mean) / std

# Split the dataset into training and validation sets
X_train, X_val, t_train, t_val = train_test_split(X, t, test_size=0.2, random_state=42)

# Create PyTorch Datasets
train_dataset = TensorDataset(X_train, t_train)
val_dataset = TensorDataset(X_val, t_val)

# Define the data loaders
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

```

```
<ipython-input-4-283300a23578>:11: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  t = torch.nn.functional.one_hot(torch.tensor(y, dtype=torch.int64), num_classes=3)
<ipython-input-4-283300a23578>:12: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  t = torch.tensor(t, dtype=torch.float32)
```

```
print(X.shape)
print(t.shape)
```

```
torch.Size([150, 2])
torch.Size([150, 3])
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
X.shape
y.shape
```

```
torch.Size([150])
```

Exercise 2.3.4 Revisiting Iris data (HW)

We use train_loader to train the model, and use test_loader to calculate the test accuracy. Use the following code to train the model, and calculate the test accuracy. In this case, you need to define your own loss function to calculate cross-entropy function from q_hat and tt.

```
forward=torch.nn.Linear(2,3,bias=True)

def loss_ftn(t,q):
  -torch.mean(torch.sum(t*torch.log(q),axis=1))

history=[]
n_train=len(train_dataset)
epochs=100

optimizer=torch.optim.SGD(forward.parameters(),lr=0.1)

for epoch in range(epochs):
  LOSS=0
  for xx,tt in train_loader:
    b=forward(xx)
    q=torch.softmax(b,axis=1)
    loss=loss_ftn(t,q)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    LOSS+=loss.item()*len(xx)
    history.append(LOSS/n_train)


```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-16-6ed6d699683d> in <cell line: 12>()
     15     b=forward(xx)
     16     q=torch.softmax(b,axis=1)
---> 17     loss=loss_ftn(t,q)
     18     loss.backward()
     19     optimizer.step()

<ipython-input-16-6ed6d699683d> in loss_ftn(t, q)
      2
      3 def loss_ftn(t,q):
----> 4   -torch.mean(torch.sum(t*torch.log(q),axis=1))
      5
      6 history=[]

RuntimeError: The size of tensor a (150) must match the size of tensor b (16) at non-singleton dimension 0
```

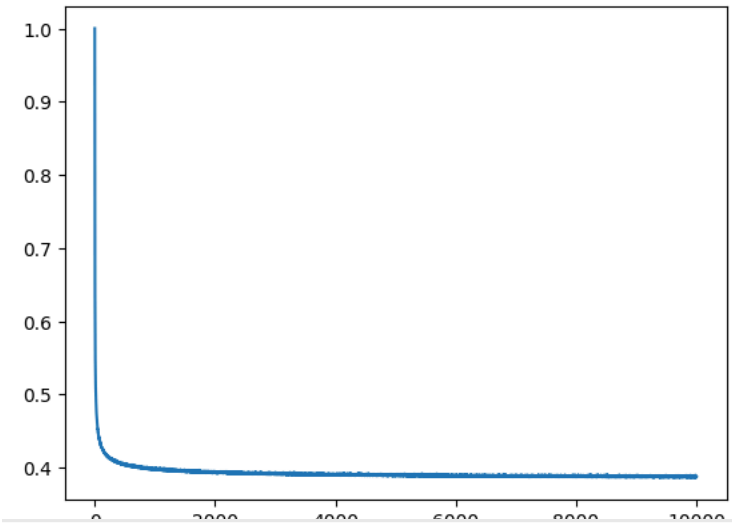
다음 단계: 오류 설명

```

forward = torch.nn.Linear(2, 3, bias=True)
def loss_ftn(t,q):
  return -torch.mean(torch.sum(t*torch.log(q), axis=1))

epochs = 10000
optimizer = torch.optim.SGD(forward.parameters(), lr=0.1)
history=[]
n_train = len(train_loader.dataset)
for epoch in range(epochs):
  LOSS = 0
  for xx, t in train_loader:
    b = forward(xx)
```

```
    q = torch.softmax(b, axis=1)
    loss = loss_ftn(t, q)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    LOSS +=loss.item()*len(xx)
  history.append(LOSS/n_train)
```

```
plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x79f2d3957eb0>]
```




```
def accuracy(phat,t):
  yhat=torch.argmax(phat,axis=1).to(torch.float32)
  y=torch.argmax(t,axis=1).to(torch.float32)
  accuracy=torch.mean((y.reshape([-1])==yhat).to(torch.float32))
  return accuracy
```

```
phat=torch.softmax(forward(X_val),axis=1)
accuracy=accuracy(phat,t_val)
accuracy
```

```
tensor(0.9000)
```

```
#accuracy 구하기
Correct=0
for xx, t in test_loader:
  b=forward(xx)
  q=torch.softmax(b,axis=1)
  yhat=torch.argmax(q,axis=1)
  y=torch.argmax(t,axis=1)
  correct=torch.sum((yhat==y)*1.0).item()
  Correct+=correct
Correct/len(test_loader.dataset)

#답은 같이 나옴

X[:2,:2]. #시험, 데이터에서 예측하기 : 확률이 나온다. X는 정규화 하기 전,
#[-1.0,1.1] 어떤 종류일까? 시험문제를 낼거야 홋
```

```
0.9
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
#먼저 정규화 시키기

xx=torch.tensor([[5.0,4.1]],dtype=torch.float32)
xx_std=(xx-mean)/std
xx_std

#q만들기
b=forward(xx_std)
q=torch.softmax(b,axis=1) #100%확률로 0번째 종류
q


```

```
tensor([[1.0000e+00, 1.8087e-10, 3.4207e-11]], grad_fn=<SoftmaxBackward0>)
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
def accuracy_ftn(phat,t):
  yhat=torch.argmax(phat,axis=1).to(torch.float32)
  y=torch.argmax(t,axis=1).to(torch.float32)
  accuracy=torch.mean((y.reshape([-1])==yhat).to(torch.float))
  return accuracy
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
#2  Exercise 2.3.5 (Fashion Mnist data)
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
from torchvision import datasets, transforms
```

```
# transform to normalize the data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])

# Download and load the training data
trainset = datasets.FashionMNIST('./data', download=True, train=True, transform=transfor
train_loader = DataLoader(trainset, batch_size=64, shuffle=True)

# Download and load the test data
validationset = datasets.FashionMNIST('./data', download=True, train=False, transform=tr
test_loader = DataLoader(validationset, batch_size=64, shuffle=True)


```

```
숨겨진 출력 표시
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

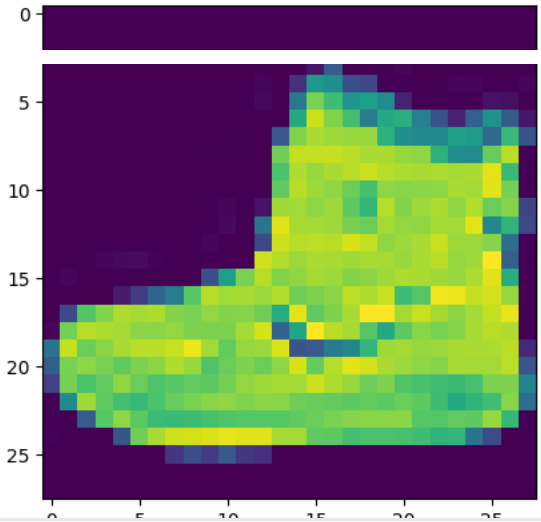
```
xx, yy = trainset[0]
```

```
x,y=validationset[0]
```

```
plt.imshow(xx.reshape(28,28))
my_name = { 0: "T-shirt/top", 1: "Trouser",
            2: "Pullover", 3: "Dress", 4: "Coat", 5: "Sandal", 6: "Shirt", 7: "Sneaker
```

```
print(my_name[yy])


```

```
Ankle boot
```



Do the multinomial regression so that the regression machine can determine the label from the images. Calculate the test accuracy. In the test dataset, identify at least 5 cases where your predictions are wrong, and explain why possibly they were mis-classified.

```
xx.shape
```

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

```
import torch.nn.functional as F
```

Do the multinomial regression so that the regression machine can determine the label from the images.

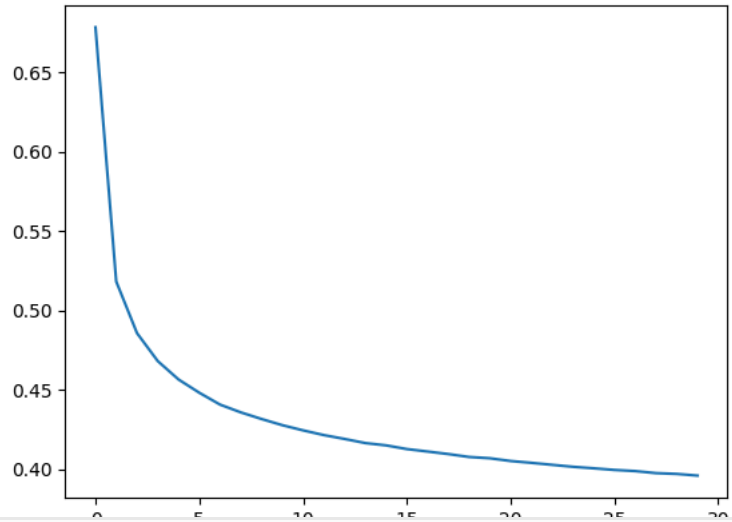
Calculate the test accuracy.

In the test dataset, identify at least 5 cases where your predictions are wrong, and explain why possibly they were mis-classified.

```
#multinomial regression
```

```
z_model=nn.Linear(784,10,bias=True)

def loss_fn(qhat,t):
  loss = F.nll_loss(torch.log(qhat), t)
  return loss

optimizer=torch.optim.SGD(z_model.parameters(),lr=0.01)
history=[]
epochs=30


for epoch in range(epochs):
  epoch_loss=0.0
  for xx, tt in train_loader:
    xx = xx.view(xx.size(0), -1)
    zhat=z_model(xx)
    qhat=torch.softmax(zhat,axis=1)
    loss=loss_fn(qhat,tt)
    loss.backward()
    epoch_loss+=loss.item()*len(xx)
    optimizer.step()
    optimizer.zero_grad()
  history.append(epoch_loss/len(trainset))
plt.plot(history)

```

```
[<matplotlib.lines.Line2D at 0x78cd04734610>]
```




```
def accuracy_ftn(phat,t):
  yhat = torch.argmax(phat, axis=1).to(torch.float32)
  y = torch.tensor(t).long() if isinstance(t, int) else t.long()
  #y=torch.argmax(t,axis=1).to(torch.float32)
  accuracy=torch.mean((y.reshape([-1])==yhat).to(torch.float32))
  return accuracy
```

```
phat=torch.softmax(z_model(x.view(1, -1)),axis=1) # Flatten the input tensor x before pa
accuracy=accuracy_ftn(phat,y).to(torch.float32)
accuracy


```

```
tensor(1.)
```