Logistic Regression

In [ ]:
```
import sklearn
import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
iris=load_iris()
X=iris['data']
y=iris['target']
```

In [ ]:
```
y.shape
```

Out[ ]:
```
(150,)
```

In [ ]:
```
### Preprocess the Data
scaler=StandardScaler()
X=scaler.fit_transform(X)
X=torch.tensor(X,dtype=torch.float32)
y=torch.tensor(y,dtype=torch.float32)
```

In [ ]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=32)
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
torch.Size([120, 4])
```

In [ ]:
```
class LogisticRegression(nn.Module):
  def __init__(self,n_features,h1):
    super(LogisticRegression,self).__init__()
    self.layer1=nn.Linear(n_features,h1)
    self.layer2=nn.Linear(h1,3)
    self.softmax=nn.Softmax(dim=1)

  def forward(self,x):
    out=F.relu(self.layer1(x))
    out=F.relu(self.layer2(out))
    out=self.softmax(out)
    return out
```

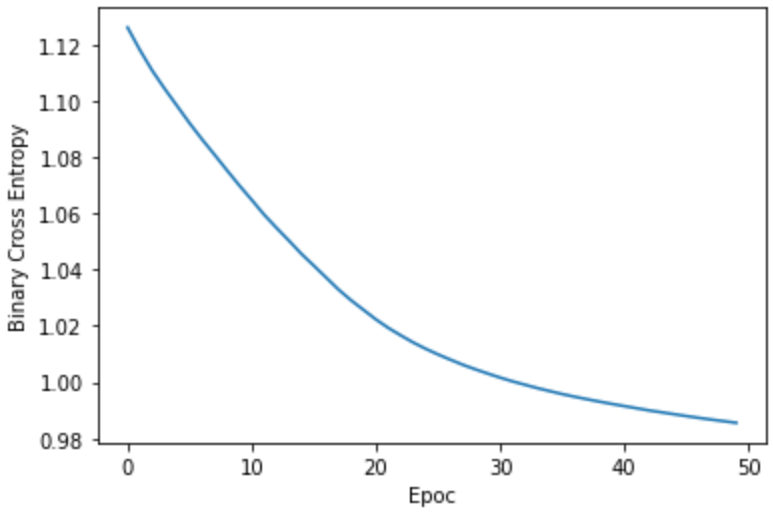
In [ ]:
```
def train(model,X,y,epocs=50,optimizer=None,loss_fn=None,bsz=20,seed=20):
  torch.manual_seed(seed)
  cost=[]
  for epoch in range(epocs):
    suffle_idx=torch.randperm(y.size(0))
    batches =torch.split(suffle_idx,bsz)
    for batch_idx in batches:
      ###1. Compute the output
      yhat=model(X[batch_idx])

      ###2. Compute the error

      loss=loss_fn(yhat,y[batch_idx].long())

      ###3. Compute the Gradient
      optimizer.zero_grad()
      loss.backward()

      ###4. Update the Parameters
      optimizer.step()

      ### Log the loss
    with torch.no_grad():
      yhat=model(X)
      curr_loss=loss_fn(yhat,y.long())
      print('Epoc:%3d ' % epoch,end="")
      print('BCE %.2f' % curr_loss)
      cost.append(curr_loss)
  return cost
```

In [ ]:
```
model=LogisticRegression(n_features=X_train.size(1),h1=20)
optimizer=torch.optim.Adam(model.parameters(),lr=.001)
loss_fn=nn.CrossEntropyLoss()
```

In [ ]:
```
cost =train(model,X_train,y_train,optimizer=optimizer,loss_fn=loss_fn)
```

```
Epoc:  0 BCE 1.13
Epoc:  1 BCE 1.12
Epoc:  2 BCE 1.11
Epoc:  3 BCE 1.10
Epoc:  4 BCE 1.10
Epoc:  5 BCE 1.09
Epoc:  6 BCE 1.09
Epoc:  7 BCE 1.08
Epoc:  8 BCE 1.08
Epoc:  9 BCE 1.07
Epoc: 10 BCE 1.06
Epoc: 11 BCE 1.06
Epoc: 12 BCE 1.05
Epoc: 13 BCE 1.05
Epoc: 14 BCE 1.05
Epoc: 15 BCE 1.04
Epoc: 16 BCE 1.04
Epoc: 17 BCE 1.03
Epoc: 18 BCE 1.03
Epoc: 19 BCE 1.03
Epoc: 20 BCE 1.02
Epoc: 21 BCE 1.02
Epoc: 22 BCE 1.02
Epoc: 23 BCE 1.01
Epoc: 24 BCE 1.01
Epoc: 25 BCE 1.01
Epoc: 26 BCE 1.01
Epoc: 27 BCE 1.01
Epoc: 28 BCE 1.00
Epoc: 29 BCE 1.00
Epoc: 30 BCE 1.00
Epoc: 31 BCE 1.00
Epoc: 32 BCE 1.00
Epoc: 33 BCE 1.00
Epoc: 34 BCE 1.00
Epoc: 35 BCE 1.00
Epoc: 36 BCE 0.99
Epoc: 37 BCE 0.99
Epoc: 38 BCE 0.99
Epoc: 39 BCE 0.99
Epoc: 40 BCE 0.99
Epoc: 41 BCE 0.99
Epoc: 42 BCE 0.99
Epoc: 43 BCE 0.99
Epoc: 44 BCE 0.99
Epoc: 45 BCE 0.99
Epoc: 46 BCE 0.99
Epoc: 47 BCE 0.99
Epoc: 48 BCE 0.99
Epoc: 49 BCE 0.99
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(range(len(cost)),cost)
plt.ylabel('Binary Cross Entropy')
plt.xlabel('Epoc')
plt.show()
```



In [ ]:
```
from sklearn.metrics import accuracy_score
```

In [ ]:
```
with torch.no_grad():
  train_pred=model(X_train)
  #train_pred
  train_acc=(torch.argmax(train_pred,dim=1)==y_train).float().mean()
  print('Training Accuracy: %.2f' %train_acc)
  test_pred=model(X_test)
  test_acc=(torch.argmax(test_pred,dim=1)==y_test).float().mean()
  print('Testing Accuracy: %.2f' %test_acc)
```

```
Training Accuracy: 0.65
Testing Accuracy: 0.70
```

Multinomial Logistic Regression

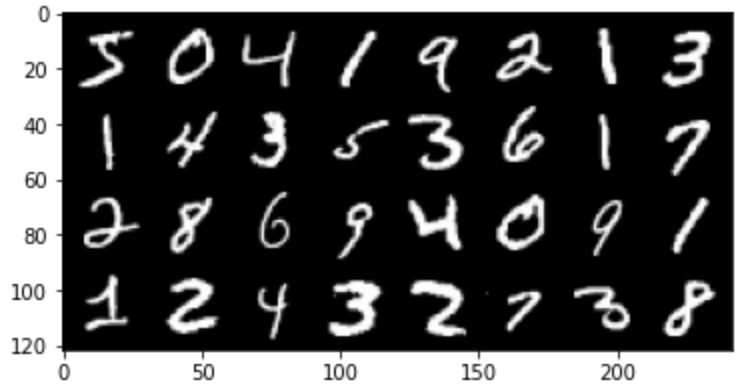
In [ ]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [ ]:
```
from matplotlib.transforms import Transform
transform=transforms.ToTensor()

### Data Loading
traindata=torchvision.datasets.MNIST(root='./data',train=True,download=True,transform=tr
ansform)
trainloader=torch.utils.data.DataLoader(traindata,batch_size=32,shuffle=False)

testdata=torchvision.datasets.MNIST(root='./data',train=False,download=True,transform=tr
ansform)
testloader=torch.utils.data.DataLoader(testdata,batch_size=32,shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIS
T/raw/train-images-idx3-ubyte.gz


Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIS
T/raw/train-labels-idx1-ubyte.gz


Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/
raw/t10k-images-idx3-ubyte.gz


Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/
raw/t10k-labels-idx1-ubyte.gz


Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
def imshow(img):
  img=img.numpy()
  img=np.transpose(img,(1,2,0))
  plt.imshow(img)
  plt.show()
```

In [ ]:
```
dataiter=iter(trainloader)
images,labels=dataiter.next()
imshow(torchvision.utils.make_grid(images))
print(labels)
```




```
tensor([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9, 4, 0, 9, 1,
        1, 2, 4, 3, 2, 7, 3, 8])
```

In [ ]:
```
images.view(images.size(0),-1).shape
```

Out[ ]:
```
torch.Size([32, 784])
```

In [ ]:
```
print(images[0].shape)
```

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

In [ ]:
```
import torch.nn as nn
import torch.nn.functional as F
class LogisticNet(nn.Module):
  def __init__(self):
    super().__init__()
    self.layer1=nn.Linear(1*28*28,100)
    self.layer2=nn.Linear(100,30)
    self.layer3=nn.Linear(30,10)

  def forward(self,x):
    x=F.relu(self.layer1(x))
    x=F.relu(self.layer2(x))
    x=self.layer3(x)
    return x



model=LogisticNet()
```

In [ ]:
```
model
```

Out[ ]:
```
LogisticNet(
  (layer1): Linear(in_features=784, out_features=100, bias=True)
  (layer2): Linear(in_features=100, out_features=30, bias=True)
  (layer3): Linear(in_features=30, out_features=10, bias=True)
)
```

In [ ]:
```
import torch.optim as optim
```

In [ ]:

```
def train_mnst(model,train_loader,test_loader,EPOCHS=100,lr=.001,seed=32):
  torch.manual_seed(seed)
  device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
  cost=[]
  model.to(device)
  optimizer=optim.SGD(model.parameters(),lr=lr)
  loss_function=nn.CrossEntropyLoss()
  for epoch in range(EPOCHS):

    curr_loss=0.0
    currect=0.0
    t_currect=0.0
    for i,data in enumerate(train_loader,0):


      ## Get a Batch of Data
      input,target=data[0].to(device),data[1].to(device)
      ###1. Compute the output

      ###Flatten the input
      input=input.view(input.size(0),-1)
      yhat=model(input)

      ###2. Compute the loss
      loss=loss_function(yhat,target)


      ###3. Compute the gradient

      optimizer.zero_grad()
      loss.backward()

      ###4. Update the parameters

      optimizer.step()
      ### Log the record
      with torch.no_grad():
        yhat=model(input)
        err=loss_function(yhat,target)
        curr_loss+=err
        b_currect=((torch.argmax(yhat,dim=1)==target).sum()).float()

        currect+=b_currect
    ### Calculate the accuracy
    with torch.no_grad():
      for data in test_loader:
          ims,lbs=data[0].to(device),data[1].to(device)
          pred=model(ims.view(ims.size(0),-1))
          t_currect+=(torch.argmax(pred,dim=1)==lbs).sum()

      print('Current Epoch: %3d' %(epoch+1),end="")
      print(' | Loss: %5.2f' %curr_loss,end="")
      accuracy=currect.sum()/len(traindata)*100
      test_accuracy=t_currect.sum()/len(testdata)*100
      #print(currect)
      print(' | Accuracy: %.2f' %accuracy,end="")
      print(' | Test Accuracy: %.2f' %test_accuracy )
  return cost
```

In [ ]:
```
cost=train_mnst(model,trainloader,testloader,EPOCHS=30)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-9ab78b465309> in <module>()
----> 1 cost=train_mnst(model,trainloader,testloader,EPOCHS=30)

NameError: name 'model' is not defined
```

In [ ]:
```
len(traindata)
```

Out[ ]:
```
60000
```

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
prob=model(images.view(32,-1).to(device))
(torch.argmax(prob,dim=1).to('cpu')==labels).sum().float()/32*100
```

Out[ ]:
```
tensor(93.7500)
```