**DL Sample code 3**

from numpy import vstack

from pandas import read\_csv

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score

from torch.utils.data import Dataset

from torch.utils.data import DataLoader

from torch.utils.data import random\_split

from torch import Tensor

from torch.nn import Linear

from torch.nn import ReLU

from torch.nn import Sigmoid

from torch.nn import Module

from torch.optim import SGD

from torch.nn import BCELoss

from torch.nn.init import kaiming\_uniform\_

from torch.nn.init import xavier\_uniform\_

# dataset definition

class CSVDataset(Dataset):

# load the dataset

def \_\_init\_\_(self, path):

# load the csv file as a dataframe

df = read\_csv(path, header=None)

# store the inputs and outputs

self.X = df.values[:, :-1]

self.y = df.values[:, -1]

# ensure input data is floats

self.X = self.X.astype('float32')

# label encode target and ensure the values are floats

self.y = LabelEncoder().fit\_transform(self.y)

self.y = self.y.astype('float32')

self.y = self.y.reshape((len(self.y), 1))

# number of rows in the dataset

def \_\_len\_\_(self):

return len(self.X)

# get a row at an index

def \_\_getitem\_\_(self, idx):

return [self.X[idx], self.y[idx]]

# get indexes for train and test rows

def get\_splits(self, n\_test=0.33):

# determine sizes

test\_size = round(n\_test \* len(self.X))

train\_size = len(self.X) - test\_size

# calculate the split

return random\_split(self, [train\_size, test\_size])

# model definition

class MLP(Module):

# define model elements

def \_\_init\_\_(self, n\_inputs):

super(MLP, self).\_\_init\_\_()

# input to first hidden layer

self.hidden1 = Linear(n\_inputs, 10)

kaiming\_uniform\_(self.hidden1.weight, nonlinearity='relu')

self.act1 = ReLU()

# second hidden layer

self.hidden2 = Linear(10, 8)

kaiming\_uniform\_(self.hidden2.weight, nonlinearity='relu')

self.act2 = ReLU()

# third hidden layer and output

self.hidden3 = Linear(8, 1)

xavier\_uniform\_(self.hidden3.weight)

self.act3 = Sigmoid()

# forward propagate input

def forward(self, X):

# input to first hidden layer

X = self.hidden1(X)

X = self.act1(X)

# second hidden layer

X = self.hidden2(X)

X = self.act2(X)

# third hidden layer and output

X = self.hidden3(X)

X = self.act3(X)

return X

# prepare the dataset

def prepare\_data(path):

# load the dataset

dataset = CSVDataset(path)

# calculate split

train, test = dataset.get\_splits()

# prepare data loaders

train\_dl = DataLoader(train, batch\_size=32, shuffle=True)

test\_dl = DataLoader(test, batch\_size=1024, shuffle=False)

return train\_dl, test\_dl

# train the model

def train\_model(train\_dl, model):

# define the optimization

criterion = BCELoss()

optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)

# enumerate epochs

for epoch in range(100):

# enumerate mini batches

for i, (inputs, targets) in enumerate(train\_dl):

# clear the gradients

optimizer.zero\_grad()

# compute the model output

yhat = model(inputs)

# calculate loss

loss = criterion(yhat, targets)

# credit assignment

loss.backward()

# update model weights

optimizer.step()

# evaluate the model

def evaluate\_model(test\_dl, model):

predictions, actuals = list(), list()

for i, (inputs, targets) in enumerate(test\_dl):

# evaluate the model on the test set

yhat = model(inputs)

# retrieve numpy array

yhat = yhat.detach().numpy()

actual = targets.numpy()

actual = actual.reshape((len(actual), 1))

# round to class values

yhat = yhat.round()

# store

predictions.append(yhat)

actuals.append(actual)

predictions, actuals = vstack(predictions), vstack(actuals)

# calculate accuracy

acc = accuracy\_score(actuals, predictions)

return acc

# make a class prediction for one row of data

def predict(row, model):

# convert row to data

row = Tensor([row])

# make prediction

yhat = model(row)

# retrieve numpy array

yhat = yhat.detach().numpy()

return yhat

# prepare the data

path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/ionosphere.csv'

train\_dl, test\_dl = prepare\_data(path)

print(len(train\_dl.dataset), len(test\_dl.dataset))

# define the network

model = MLP(34)

# train the model

train\_model(train\_dl, model)

# evaluate the model

acc = evaluate\_model(test\_dl, model)

print('Accuracy: %.3f' % acc)

# make a single prediction (expect class=1)

row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38223,0.84356,-0.38542,0.58212,-0.32192,0.56971,-0.29674,0.36946,-0.47357,0.56811,-0.51171,0.41078,-0.46168,0.21266,-0.34090,0.42267,-0.54487,0.18641,-0.45300]

yhat = predict(row, model)

print('Predicted: %.3f (class=%d)' % (yhat, yhat.round()))