Maching Learning Project02

October 1, 2019

[209]:
```
import torch
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
from torch.autograd import Variable
import torchvision
import os

import matplotlib.pyplot as plt
import numpy as np


transform = transforms.Compose([#transforms.Resize((256,256)),
                                transforms.Grayscale(),                 # the␣
 ↪code transforms.Graysclae() is for changing the size [3,100,100] to [1, 100,␣
 ↪100] (notice : [channel, height, width] )
                                transforms.ToTensor(),])


#train_data_path = 'relative path of training data set'
train_data_path = 'C:\\ ␣
 ↪  \\MachineLearningProject\\horse-or-human\\horse-or-human\\train'
trainset = torchvision.datasets.ImageFolder(root=train_data_path,␣
 ↪transform=transform)
# change the valuse of batch_size, num_workers for your program
# if shuffle=True, the data reshuffled at every epoch
trainloader = torch.utils.data.DataLoader(trainset, batch_size=1,␣
 ↪shuffle=False, num_workers=1)


validation_data_path = 'C:\\ ␣
 ↪  \\MachineLearningProject\\horse-or-human\\horse-or-human\\validation'
valset = torchvision.datasets.ImageFolder(root=validation_data_path,␣
 ↪transform=transform)
# change the valuse of batch_size, num_workers for your program
valloader = torch.utils.data.DataLoader(valset, batch_size=1, shuffle=False,␣
 ↪num_workers=1)
```

[375]:
```
train_labels=np.zeros(1027)
test_labels=np.zeros(256)

known_train=np.zeros((10001))
known_test=np.zeros((10001))

train_datas=np.zeros((1027,10001))
test_datas=np.zeros((256,10001))



for epoch in range(1):
    sum=0
    l_rate=0.001
    # load training images of the batch size for every iteration
    for i, data in enumerate(trainloader):

        inputs, labels = data
        train_labels[i]=int(labels)
        reinputs=inputs.reshape(10000)
        reinputs=np.array(reinputs)
        reinputs=np.hstack((reinputs,1))
        train_datas[i]=reinputs

    train_datas=train_datas.T

    for i, data in enumerate(valloader):
        sum+=1
        inputs, labels = data
        test_labels[i]=int(labels)
        reinputs=inputs.reshape(10000)
        reinputs=np.array(reinputs)
        reinputs=np.hstack((reinputs,1))
        test_datas[i]=reinputs

    test_datas=test_datas.T
```

256

[382]:
```
for i in range(10):
    #Vectorizing Logistic Regression for train_set
    L=0
    h=0
```

```
    j=0
    z=np.dot(known_train,train_datas)
    h=1.0/(1.0+np.exp(-z))
    j=-(train_labels*np.log(h)+(1-train_labels)*np.log(1-h))
    j=np.sum(j)
    L=h-train_labels
    L=L.reshape(1027)
    dw=np.dot(train_datas,L)
    dw=dw/1027
    known_train=known_train-l_rate*dw
    print("train=",j)

    #Vectorizing Logistic Regression for test_set
    L_v=0
    h_v=0
    j_v=0
    z_v=np.dot(known_test,test_datas)
    h_v=1.0/(1.0+np.exp(-z_v))
    j_v=-(test_labels*np.log(h_v)+(1-test_labels)*np.log(1-h_v))
    j_v=np.sum(j_v)
    L_v=h_v-test_labels
    L_v=L_v.reshape(256)
    dw_v=np.dot(test_datas,L_v)
    dw_v=dw_v/256
    known_test=known_test-l_rate*dw_v
    print("test=",j_v)
```

```
train= 665.3473385492758
test= 142.05775250688703
train= 663.9319763889662
test= 139.32242964659258
train= 662.5341237968986
test= 136.7042254612495
train= 661.1534397056557
test= 134.19706206002311
train= 659.7895914823127
test= 131.79500380638888
train= 658.4422547005267
test= 129.49255007727518
train= 657.1111129177873
test= 127.28438380764379
train= 655.7958574578502
test= 125.16556337132585
train= 654.4961871983579
test= 123.13135519408294
train= 653.211808363639
```

```
test= 121.17735063866608
```

[ ]:

[69]:
```
classes = ('0', '1')

def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


#
dataiter = iter(trainloader)
images, labels = dataiter.next()
print(images)

imshow(torchvision.utils.make_grid(images))
#  (label)
print(' '.join('%5s' % classes[labels[j]] for j in range(3)))
```

```
tensor([[[[0.5961, 0.5961, 0.5961,  …, 0.9765, 0.9843, 0.9765],
          [0.6000, 0.6000, 0.6000,  …, 0.9569, 0.9882, 0.9765],
          [0.6078, 0.6078, 0.6078,  …, 0.9255, 0.9451, 0.9608],
          …,
          [0.7647, 0.7608, 0.8000,  …, 0.9059, 0.8902, 0.8863],
          [0.7294, 0.7647, 0.7765,  …, 0.8706, 0.8980, 0.9059],
          [0.7490, 0.7725, 0.7451,  …, 0.8941, 0.9020, 0.8745]]],


        [[[0.6000, 0.5961, 0.6000,  …, 0.8353, 0.8745, 0.9059],
          [0.6078, 0.6039, 0.6039,  …, 0.8353, 0.8392, 0.8627],
          [0.6118, 0.6118, 0.6118,  …, 0.8235, 0.8157, 0.8235],
          …,
          [0.7451, 0.7725, 0.7765,  …, 0.8510, 0.8314, 0.7922],
          [0.7608, 0.7686, 0.7647,  …, 0.7922, 0.7843, 0.7647],
          [0.7529, 0.7804, 0.7765,  …, 0.8078, 0.7922, 0.7529]]],


        [[[0.6157, 0.6157, 0.6157,  …, 0.5882, 0.5882, 0.5882],
          [0.6235, 0.6235, 0.6235,  …, 0.5922, 0.5922, 0.5922],
          [0.6275, 0.6275, 0.6275,  …, 0.5961, 0.5961, 0.5961],
          …,
          [0.7412, 0.7412, 0.7412,  …, 0.7451, 0.7294, 0.6902],
          [0.7216, 0.7098, 0.7059,  …, 0.6941, 0.6941, 0.6667],
          [0.7020, 0.7137, 0.7412,  …, 0.6627, 0.6627, 0.6980]]]])
```

0 0 0

[ ]: