3 、Basic use of PyTorch

The Raspberry Pi motherboard series does not currently support this tutorial.

3.1、About PyTorch

3.1.1、Introduction

PyTorch is an [Open Source](#) [Python](#) machine learning library, based on Torch, using for applications such as natural language processing.

3.1.2、Features

1）、Powerful GPU-accelerated tensor computation

2）、Deep neural networks for automatic derivation systems

3）、Dynamic graph mechanism

3.2、Tensors in PyTorch

3.2.1、Tensor

Tensor is the basic unit of operation in PyTorch. Like Numpy's ndarray, it represents a multi-dimensional matrix. The biggest difference with ndarray is that PyTorch's Tensor can run on the GPU, while numpy's ndarray can only run on the CPU. Running on the GPU greatly speeds up the operation.

3.2.2、Create tensor

1）、There are many ways to create tensors. Different types of tensors can be created by calling APIs of different interfaces.

a = torch.empty(2,2): Create an uninitialized 2*2 tensor.

b = torch.rand(5, 6)：Create a uniformly distributed tensor initialized with each element ranging from 0-1.

c = torch.zeros(5, 5, dtype=torch.long)：Create an initialized all-zero tensor and specify the type of each element as long.

d = c.new_ones(5, 3, dtype=torch.double)：Create a new tensor d based on a known tensor c.

d.size(): Get the shape of tensor d

2) 、Operations between tensors

Operations between tensors are actually operations between matrices. Due to the dynamic graph mechanism, mathematical calculations can be performed directly on tensors, for example,

- Add two tensors:

```
c = torch.zeros(5,3,dtype=torch.long)
d = torch.ones(5,3,dtype=torch.long)
e = c + d
print(e)
```

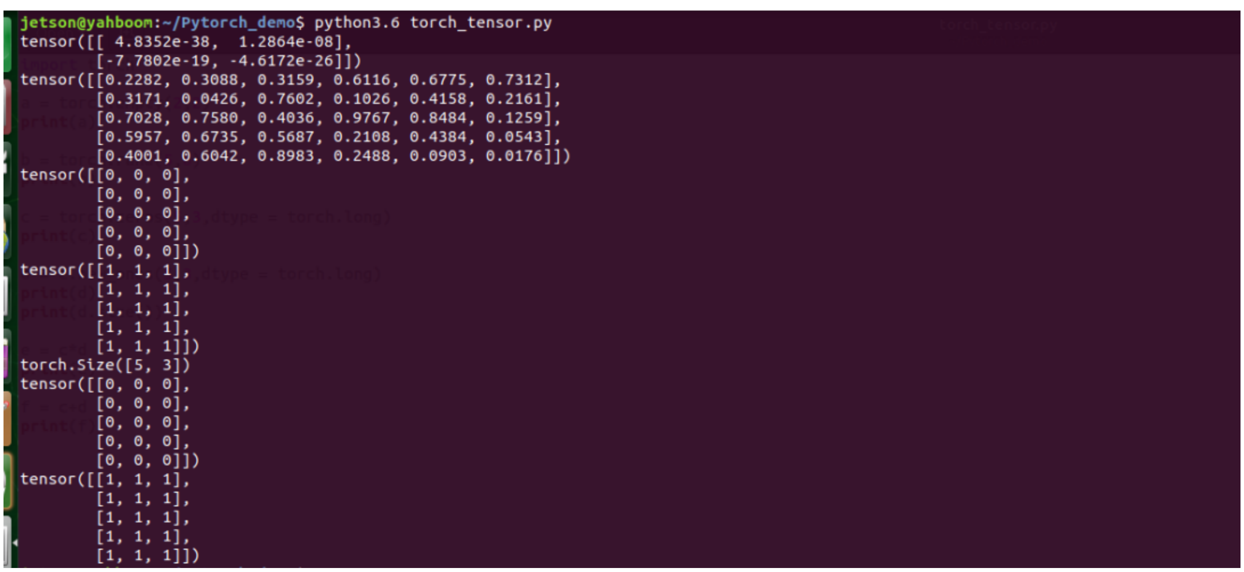
- Multiply two tensors

```
c = torch.zeros(5,3,dtype=torch.long)
d = torch.ones(5,3,dtype=torch.long)
e = c * d
print(e)
```

You can refer to this part of the code: ~/Pytorch_demo/torch_tensor.py

run code,

```
cd ~/Pytorch_demo/
python3.6 torch_tensor.py
```



3.3、Introduction to torchvision package

3.3.1、torchvision is a library specifically used to process images in Pytorch, including four major categories

1）、torchvision.datasets：Load dataset，Pytorch has many data sets such as CIFAR, MNIST, etc. You can use this class to load data sets. The usage is as follows：

```
cifar_train_data = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                download=False, transform=transform)
```

2）、torchvision.models：Load the trained model，Models include the following VCG, ResNet, etc., and their usage is as follows:

```
import torchvision.models as models
resnet18 = models.resnet18()
```

3）、torchvision.transforms：Class for image conversion operations, usage is as follows：

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

4）、torchvision.untils：Arrange pictures into a grid shape, the usage is as follows：

```
torchvision.utils.make_grid(tensor, nrow=8, padding=2, normalize=False, range=None,
scale_each=False, pad_value=0)
```

For more information about the use of torchvision package, please refer to the official website documentation：https://pytorch.org/vision/0.8/datasets.html

3.4、convolutional neural network

3.4.1、Neural Networks

1）、The difference between neural networks and machine learning

Neural networks and machine learning are both used for classification tasks. The difference is that neural networks are more efficient than machine learning, the data are simpler, and fewer parameters are required to perform tasks. The following points are explained.

- Efficiency: The efficiency of neural networks is reflected in the extraction of features. It is different from the characteristics of machine learning. It can train and "correct" itself. We only need to input data, and it will continuously update the characteristics on its own.
- Data simplicity: In the machine learning process, we need to perform some processing on the data before inputting the data, such as normalization, format conversion, etc., but in neural networks, there is no need for too much processing.

- There are fewer parameters to perform tasks: In machine learning, we need to adjust penalty factors, slack variables, etc. to achieve the most suitable effect, but for neural networks, we only need to give a weight w and a bias term b, During the training process, these two values will be continuously revised and adjusted to the optimum to minimize the error of the model.

3.4.2、convolutional neural network

1）、Convolution kernel

The convolution kernel can be understood as a feature extractor, filter (digital signal processing), etc. The neural network has three layers (input layer, hidden layer, output layer). The neurons in each layer can share the convolution kernel, so it will be very convenient to process high-level data. We only need to design the size, number and sliding step of the convolution kernel and let it train by itself.

2）、Three basic layers of convolutional neural networks:：

- convolution layer

  Perform convolution operation, inner product operation of two matrices with the size of convolution kernel, multiply numbers at the same position and then add and sum. The convolution layer close to the input layer sets a small number of convolution kernels, and the later the convolution layer sets, the more convolution kernels it sets.

- Pooling layer

  Through downsampling, the image and parameters are compressed without destroying the quality of the image. There are two pooling methods, MaxPooling (that is, taking the maximum value in the sliding window) and AveragePooling (taking the average of all values in the sliding window).

- Flatten layer&Fully Connected layer

  This layer is mainly about stacking layers. After passing through the pooling layer, the image is compressed and then enters the Flatten layer; the output of the Flatten layer is placed in the Fully Connected layer, and softmax is used to classify it.

3.5、Build LetNet neural network and train data set

3.5.1、Preparation before implementation

1）、environment

The development environment of this project is installed on the jetson development board series, as follows:：

- python 3.6+
- torch 1.8.0
- torchvision 0.9.0

2）、data set

CIFAR-10, 50,000 training images of 32*32 size and 10,000 test images

Note: The data set is saved in the ~/Pytorch_demo/data/cifar-10-batches-py directory.

file name	File function
batches.meta. bet	The file stores the English name of each category and can be viewed using Notepad or other text readers.
data batch l.bin 、 data batch 2.bm 、 …… data batch 5.bin	These five files are training data in the CIFAR-10 data set. Each file stores 10,000 32x32 color images in binary format and the category labels corresponding to these graphics. A total of 50,000 training images
test batch.bin	This file stores the test images and the labels of the test images. 10,000 in total
readme.html	Dataset introduction file

3.5.2、Implementation process

1）、Import related modules

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
```

2）、Load dataset

```
cifar_train_data = torchvision.datasets.CIFAR10(root='./data', train=True,
                                          download=False, transform=transform)
cifar_test_data = torchvision.datasets.CIFAR10(root='./data', train=False,
                                          transform=transform)
```

3）、Encapsulated data set

```
train_data_loader = torch.utils.data.DataLoader(cifar_train_data, batch_size=32,
shuffle=True)
test_data_loader = torch.utils.data.DataLoader(cifar_test_data, batch_size=32,
shuffle=True
```

4）、Build a convolutional neural network

```
class LeNet(nn.Module):
```

```
    #Define the operation operators required by the network, such as convolution, 
fully connected operators, etc.
    def __init__(self):
        super(LeNet, self).__init__()
        #Conv2d parameter meaning: number of input channels, number of output 
channels, kernel size
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.pool = nn.MaxPool2d(2, 2)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 16*5*5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

5) 、Configure the loss function and optimizer for training

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.9)
```

6) 、Start training and testing

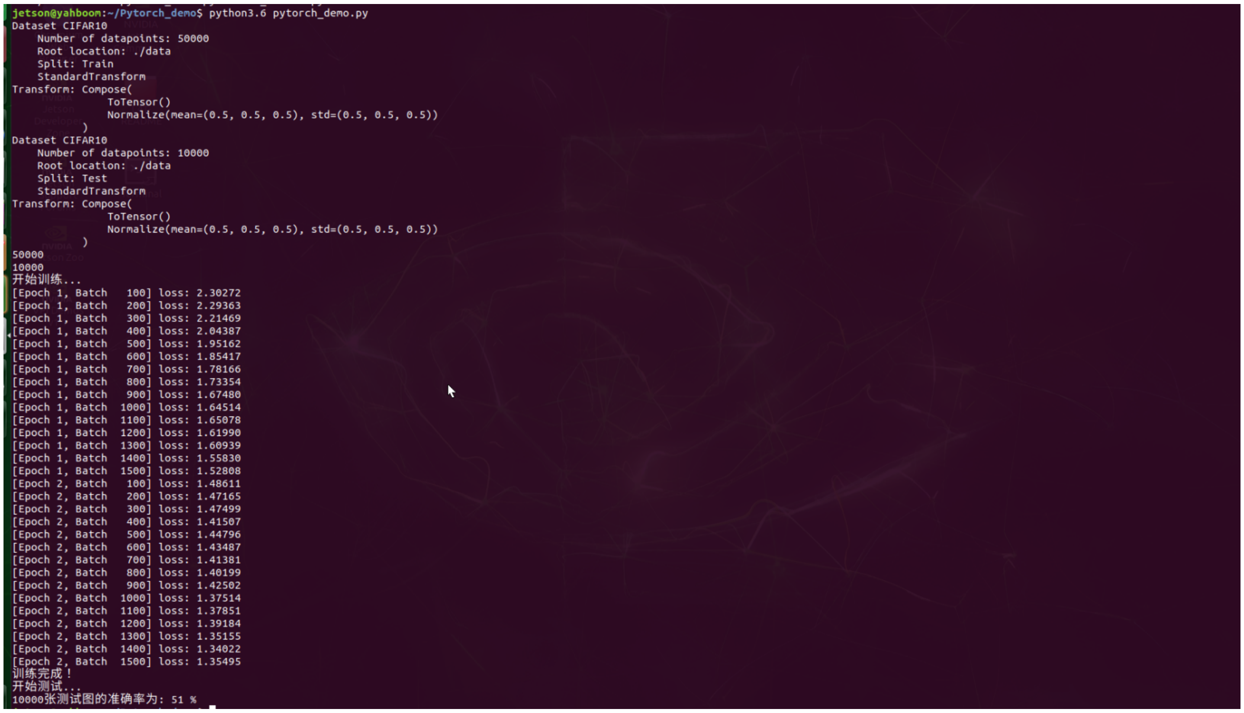
3.5.3、Run program

1) 、Reference code path

```
~/Pytorch_demo/pytorch_demo.py
```

2) 、Run program

```
cd  ~/Pytorch_demo
python3.6 pytorch_demo.py
```



We only trained 2 times here. You can modify the epoch value to modify the training times. The more training times, the higher the accuracy.