ensemble_pytorch_answer

December 20, 2021

1 BTVN: Training Neural Networks (Tiếp)

Trong phần này các bạn sẽ làm quen với kỹ thuật model ensemble để tăng độ chính xác khi suy diễn

[ ]:

```
!nvidia-smi
from google.colab import drive
drive.mount('/content/drive')

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import glob
import cv2
import torch.nn.functional as F
from torch.autograd import Variable
import os

import torchvision
import torchvision.transforms as transforms

from torch.nn import CrossEntropyLoss, Dropout, Softmax, Linear, Conv2d,␣
 ↪LayerNorm
import matplotlib.pyplot as plt
from torchsummary import summary
```

```
Thu Sep  9 09:01:58 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.63.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P8    29W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Mounted at /content/drive
```

Tải dữ liệu và cài đặt một kiến trúc mạng nơ-ron đơn giản theo mô tả phía dưới

[ ]:

```python
def load_data(data_dir="./data"):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    trainset = torchvision.datasets.CIFAR10(
        root=data_dir, train=True, download=True, transform=transform)

    testset = torchvision.datasets.CIFAR10(
        root=data_dir, train=False, download=True, transform=transform)

    return trainset, testset


class Net(nn.Module):
    def __init__(self, l1=120, l2=84):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, l1)
        self.fc2 = nn.Linear(l1, l2)
        self.fc3 = nn.Linear(l2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(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
```

```
model = Net()
if torch.cuda.is_available():
    model.cuda()
summary(model, (3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
            Conv2d-3           [-1, 16, 10, 10]           2,416
         MaxPool2d-4             [-1, 16, 5, 5]               0
            Linear-5                  [-1, 120]          48,120
            Linear-6                   [-1, 84]          10,164
            Linear-7                   [-1, 10]             850
================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.06
Params size (MB): 0.24
Estimated Total Size (MB): 0.31
----------------------------------------------------------------
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:718: UserWarning:
Named tensors and all their associated APIs are an experimental feature and
subject to change. Please do not use them for anything important until they are
released as stable. (Triggered internally at
/pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation,
ceil_mode)
```

Hàm đánh giá độ chính xác trên tập test

[ ]:

```
def test_accuracy(net, device="cpu"):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total
```

Hàm huấn luyện mô hình

[ ]:

```
def train(net, criterion, optimizer, save_path, device="cpu"):
    T_cur = 0
    for epoch in range(1, epochs+1):  # loop over the dataset multiple times
        running_loss = 0.0
        epoch_steps = 0
        T_cur += 1

        # warm-up
        if epoch <= warm_epoch:
            optimizer.param_groups[0]['lr'] = (1.0 * epoch) / warm_epoch  *␣
 ↪init_lr
        else:
            # cosine annealing lr
            optimizer.param_groups[0]['lr'] = last_lr + (init_lr - last_lr) *␣
 ↪(1 + np.cos(T_cur * np.pi / T_max)) / 2

        for i, data in enumerate(trainloader, 0):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            epoch_steps += 1
            if i + 1 == len(trainloader):
                print("[Epoch %d] loss: %.3f" % (epoch, running_loss /␣
 ↪epoch_steps))
                running_loss = 0.0

    print("Finished Training")
    print("Test accuracy:", test_accuracy(net, device))
    torch.save(net.state_dict(), save_path)
```

Thiết lập các tham số và hai kiến trúc mạng khác nhau

[ ]:

```python
epochs = 10
warm_epoch = 5
init_lr = 1e-2
last_lr = 1e-4
T_max = epochs

configs = [{'l1': 64, 'l2': 32}, {'l1': 128, 'l2': 64}]

trainset, testset = load_data('./data')
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=128,
    shuffle=True,
)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=4, shuffle=False, num_workers=2)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data/cifar-10-python.tar.gz
```

```
  0%|          | 0/170498071 [00:00<?, ?it/s]
```

```
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Huấn luyện hai mạng mô tả trong configs

[ ]:

```python
os.makedirs('./snapshot', exist_ok=True)

for i, cfg in enumerate(configs):
    print(cfg)
    net = Net(cfg['l1'], cfg['l2'])
    device = "cpu"
    if torch.cuda.is_available():
        device = "cuda:0"
        if torch.cuda.device_count() > 1:
            net = nn.DataParallel(net)
    net.to(device)

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

    save_path = f'./snapshot/model{i}.pth'
    train(net, criterion, optimizer, save_path, device)
```

```
{'l1': 64, 'l2': 32}
[Epoch 1] loss: 2.293
[Epoch 2] loss: 1.969
[Epoch 3] loss: 1.645
[Epoch 4] loss: 1.464
```

```
[Epoch 5] loss: 1.346
[Epoch 6] loss: 1.205
[Epoch 7] loss: 1.150
[Epoch 8] loss: 1.112
[Epoch 9] loss: 1.093
[Epoch 10] loss: 1.085
Finished Training
Test accuracy: 0.5958
{'l1': 128, 'l2': 64}
[Epoch 1] loss: 2.303
[Epoch 2] loss: 2.249
[Epoch 3] loss: 1.819
[Epoch 4] loss: 1.558
[Epoch 5] loss: 1.394
[Epoch 6] loss: 1.224
[Epoch 7] loss: 1.164
[Epoch 8] loss: 1.123
[Epoch 9] loss: 1.100
[Epoch 10] loss: 1.091
Finished Training
Test accuracy: 0.5889
```

Kết hợp kết quả hai mạng (ensemble)

[ ]:

```python
from tqdm import tqdm

def test_ensemble(device="cuda:0"):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in tqdm(testloader):
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            final_outputs = torch.zeros((4, 10)).to(device)
            for i, cfg in enumerate(configs):
                net = Net(cfg['l1'], cfg['l2'])
                net.to(device)
                net.load_state_dict(torch.load(f'./snapshot/model{i}.pth'))
                outputs = net(images)
                final_outputs = final_outputs.add(outputs)

            final_outputs.div(len(configs))
            _, predicted = torch.max(final_outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return correct / total
```

```
[ ]: from tqdm import tqdm

     def test_ensemble(device="cuda:0"):
         correct = 0
         total = 0
         with torch.no_grad():
             for data in tqdm(testloader):
                 images, labels = data
                 images, labels = images.to(device), labels.to(device)
                 final_outputs = torch.zeros((4, 10)).to(device)
                 for i, cfg in enumerate(configs):
                     net = Net(cfg['l1'], cfg['l2'])
                     net.to(device)
                     net.load_state_dict(torch.load(f'./snapshot/model{i}.pth'))
                     outputs = net(images)
                     final_outputs = final_outputs.add(outputs)

                 final_outputs.div(len(configs))
                 _, predicted = torch.max(final_outputs.data, 1)
                 total += labels.size(0)
                 correct += (predicted == labels).sum().item()

         return correct / total
```

```
[ ]: test_ensemble()
```

```
100%|      | 2500/2500 [00:37<00:00, 67.16it/s]
```

```
[ ]: 0.6165
```

```
[ ]:
```