TensorBoard 可视化

tensorboard 是 tensorflow 中非常好用的可视化工具，那么我们能不能在 pytorch 中也使用这个工具呢？当然可以，在下面这个 github 中已经有人为我们准备好了这个工具，下面我们来讲一下

首先需要安装 tensorflow 和 tensorboardX，使用

`pip install tensorflow`

`pip install tensorboradX`

就可以了

当然这个 tensorboard 只是一个简化版，所以一些功能，比如画计算图等是不支持的，但是可以支持画出 loss 曲线，下面举个例子

```
import numpy as np
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import Variable
from torchvision.datasets import CIFAR10
from utils import resnet
from torchvision import transforms as tfs
from datetime import datetime
from tensorboardX import SummaryWriter
```

```
# 使用数据增强
def train_tf(x):
    im_aug = tfs.Compose([
        tfs.Resize(120),
        tfs.RandomHorizontalFlip(),
        tfs.RandomCrop(96),
        tfs.ColorJitter(brightness=0.5, contrast=0.5, hue=0.5),
        tfs.ToTensor(),
        tfs.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])
    x = im_aug(x)
    return x

def test_tf(x):
    im_aug = tfs.Compose([
        tfs.Resize(96),
```

```
        tfs.ToTensor(),
        tfs.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])
    x = im_aug(x)
    return x

train_set = CIFAR10('./data', train=True, transform=train_tf)
train_data = torch.utils.data.DataLoader(train_set, batch_size=256, shuffle=True,
num_workers=4)
valid_set = CIFAR10('./data', train=False, transform=test_tf)
valid_data = torch.utils.data.DataLoader(valid_set, batch_size=256, shuffle=False,
num_workers=4)

net = resnet(3, 10)
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()
```

```
writer = SummaryWriter()

def get_acc(output, label):
    total = output.shape[0]
    _, pred_label = output.max(1)
    num_correct = (pred_label == label).sum().data[0]
    return num_correct / total

if torch.cuda.is_available():
    net = net.cuda()
prev_time = datetime.now()
for epoch in range(30):
    train_loss = 0
    train_acc = 0
    net = net.train()
    for im, label in train_data:
        if torch.cuda.is_available():
            im = Variable(im.cuda())  # (bs, 3, h, w)
            label = Variable(label.cuda())  # (bs, h, w)
        else:
            im = Variable(im)
            label = Variable(label)
        # forward
        output = net(im)
        loss = criterion(output, label)
        # backward
        optimizer.zero_grad()
        loss.backward()
```

```
        optimizer.step()

        train_loss += loss.data[0]
        train_acc += get_acc(output, label)
    cur_time = datetime.now()
    h, remainder = divmod((cur_time - prev_time).seconds, 3600)
    m, s = divmod(remainder, 60)
    time_str = "Time %02d:%02d:%02d" % (h, m, s)
    valid_loss = 0
    valid_acc = 0
    net = net.eval()
    for im, label in valid_data:
        if torch.cuda.is_available():
            im = Variable(im.cuda(), volatile=True)
            label = Variable(label.cuda(), volatile=True)
        else:
            im = Variable(im, volatile=True)
            label = Variable(label, volatile=True)
        output = net(im)
        loss = criterion(output, label)
        valid_loss += loss.data[0]
        valid_acc += get_acc(output, label)
    epoch_str = (
                "Epoch %d. Train Loss: %f, Train Acc: %f, Valid Loss: %f, Valid Acc:
%f, "
                % (epoch, train_loss / len(train_data),
                   train_acc / len(train_data), valid_loss / len(valid_data),
                   valid_acc / len(valid_data)))
    prev_time = cur_time
    # ================ 使用 tensorboard ===============
    writer.add_scalars('Loss', {'train': train_loss / len(train_data),
                                'valid': valid_loss / len(valid_data)}, epoch)
    writer.add_scalars('Acc', {'train': train_acc / len(train_data),
                               'valid': valid_acc / len(valid_data)}, epoch)
    # ================================================
    print(epoch_str + time_str)
```

```
Epoch 0. Train Loss: 1.877906, Train Acc: 0.315410, Valid Loss: 2.198587, Valid Acc:
0.293164, Time 00:00:26
Epoch 1. Train Loss: 1.398501, Train Acc: 0.498657, Valid Loss: 1.877540, Valid Acc:
0.400098, Time 00:00:27
Epoch 2. Train Loss: 1.141419, Train Acc: 0.597628, Valid Loss: 1.872355, Valid Acc:
0.446777, Time 00:00:27
Epoch 3. Train Loss: 0.980048, Train Acc: 0.658367, Valid Loss: 1.672951, Valid Acc:
0.475391, Time 00:00:27
```

```
Epoch 4. Train Loss: 0.871448, Train Acc: 0.695073, Valid Loss: 1.263234, Valid Acc:
0.578613, Time 00:00:28
Epoch 5. Train Loss: 0.794649, Train Acc: 0.723992, Valid Loss: 2.142715, Valid Acc:
0.466699, Time 00:00:27
Epoch 6. Train Loss: 0.736611, Train Acc: 0.741554, Valid Loss: 1.701331, Valid Acc:
0.500391, Time 00:00:27
Epoch 7. Train Loss: 0.695095, Train Acc: 0.756816, Valid Loss: 1.385478, Valid Acc:
0.597656, Time 00:00:28
Epoch 8. Train Loss: 0.652659, Train Acc: 0.773796, Valid Loss: 1.029726, Valid Acc:
0.676465, Time 00:00:27
Epoch 9. Train Loss: 0.623829, Train Acc: 0.784144, Valid Loss: 0.933388, Valid Acc:
0.682520, Time 00:00:27
Epoch 10. Train Loss: 0.581615, Train Acc: 0.798792, Valid Loss: 1.291557, Valid Acc:
0.635938, Time 00:00:27
Epoch 11. Train Loss: 0.559358, Train Acc: 0.805708, Valid Loss: 1.430408, Valid Acc:
0.586426, Time 00:00:28
Epoch 12. Train Loss: 0.534197, Train Acc: 0.816853, Valid Loss: 0.960802, Valid Acc:
0.704785, Time 00:00:27
Epoch 13. Train Loss: 0.512111, Train Acc: 0.822389, Valid Loss: 0.923353, Valid Acc:
0.716602, Time 00:00:27
Epoch 14. Train Loss: 0.494577, Train Acc: 0.828225, Valid Loss: 1.023517, Valid Acc:
0.687207, Time 00:00:27
Epoch 15. Train Loss: 0.473396, Train Acc: 0.835212, Valid Loss: 0.842679, Valid Acc:
0.727930, Time 00:00:27
Epoch 16. Train Loss: 0.459708, Train Acc: 0.840290, Valid Loss: 0.826854, Valid Acc:
0.726953, Time 00:00:28
Epoch 17. Train Loss: 0.433836, Train Acc: 0.847931, Valid Loss: 0.730658, Valid Acc:
0.764258, Time 00:00:27
Epoch 18. Train Loss: 0.422375, Train Acc: 0.854401, Valid Loss: 0.677953, Valid Acc:
0.778125, Time 00:00:27
Epoch 19. Train Loss: 0.410208, Train Acc: 0.857370, Valid Loss: 0.787286, Valid Acc:
0.754102, Time 00:00:27
Epoch 20. Train Loss: 0.395556, Train Acc: 0.862923, Valid Loss: 0.859754, Valid Acc:
0.738965, Time 00:00:27
Epoch 21. Train Loss: 0.382050, Train Acc: 0.866554, Valid Loss: 1.266704, Valid Acc:
0.651660, Time 00:00:27
Epoch 22. Train Loss: 0.368614, Train Acc: 0.871213, Valid Loss: 0.912465, Valid Acc:
0.738672, Time 00:00:27
Epoch 23. Train Loss: 0.358302, Train Acc: 0.873964, Valid Loss: 0.963238, Valid Acc:
0.706055, Time 00:00:27
Epoch 24. Train Loss: 0.347568, Train Acc: 0.879620, Valid Loss: 0.777171, Valid Acc:
0.751855, Time 00:00:27
Epoch 25. Train Loss: 0.339247, Train Acc: 0.882215, Valid Loss: 0.707863, Valid Acc:
0.777734, Time 00:00:27
Epoch 26. Train Loss: 0.329292, Train Acc: 0.885830, Valid Loss: 0.682976, Valid Acc:
0.790527, Time 00:00:27
```

```
Epoch 27. Train Loss: 0.313049, Train Acc: 0.890761, Valid Loss: 0.665912, Valid Acc:
0.795410, Time 00:00:27
Epoch 28. Train Loss: 0.305482, Train Acc: 0.891944, Valid Loss: 0.880263, Valid Acc:
0.743848, Time 00:00:27
Epoch 29. Train Loss: 0.301507, Train Acc: 0.895289, Valid Loss: 1.062325, Valid Acc:
0.708398, Time 00:00:27
```

训练完成之后，目录中会出现一个文件夹叫 runs，在终端运行

`tensorboard --logdir runs`

我们就能够在网页端打开 tensorboard 了