20171062_assignment4

April 13, 2020

Meher Shashwat Nigam 20171062

[ ]:
```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from tqdm import notebook
import math
```

[ ]:
```
CIFAR100_TRAIN_MEAN = (0.5070751592371323, 0.48654887331495095, 0.
 ↪4409178433670343)
CIFAR100_TRAIN_STD = (0.2673342858792401, 0.2564384629170883, 0.
 ↪27615047132568404)
EPOCH = 40
LR=0.001

#data preprocessing
workers = 2
batch_size = 128
shuffle = True

classes=('apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee',␣
 ↪'beetle',
    'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
    'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
    'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',
    'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster',
    'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion',
    'lizard', 'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
    'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear',
    'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
    'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
    'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',
    'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
```

```
    'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',
    'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',
    'worm')
```

[ ]:
```
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print(device)
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
def get_training_dataloader(mean, std, batch_size=16, num_workers=2,␣
 ↪shuffle=True):
    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(15),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ])
    cifar100_training = torchvision.datasets.CIFAR100(root='./data',␣
 ↪train=True, download=True, transform=transform_train)
    cifar100_training_loader = torch.utils.data.DataLoader(cifar100_training,␣
 ↪shuffle=shuffle, num_workers=num_workers, batch_size=batch_size)

    return cifar100_training_loader

def get_test_dataloader(mean, std, batch_size=16, num_workers=2, shuffle=True):
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ])
    cifar100_test = torchvision.datasets.CIFAR100(root='./data', train=False,␣
 ↪download=True, transform=transform_test)
    cifar100_test_loader = torch.utils.data.DataLoader(cifar100_test,␣
 ↪shuffle=shuffle, num_workers=num_workers, batch_size=batch_size)

    return cifar100_test_loader


cifar100_training_loader = get_training_dataloader(
    CIFAR100_TRAIN_MEAN,
    CIFAR100_TRAIN_STD,
    num_workers=workers,
    batch_size=batch_size,
    shuffle=shuffle
)
```

```
cifar100_test_loader = get_test_dataloader(
    CIFAR100_TRAIN_MEAN,
    CIFAR100_TRAIN_STD,
    num_workers=workers,
    batch_size=batch_size,
    shuffle=shuffle
)
```

[ ]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # Convolution Layers
        self.conv1 = nn.Conv2d(3, 64, 3)
        self.conv2 = nn.Conv2d(64, 256, 3)
        self.conv3 = nn.Conv2d(256, 512, 5)
        self.conv4 = nn.Conv2d(512, 1024, 1)

        # Batch Norm Layers
        self.batchnorm1 = nn.BatchNorm2d(64)
        self.batchnorm2 = nn.BatchNorm2d(256)
        self.batchnorm3 = nn.BatchNorm2d(512)
        self.batchnorm4 = nn.BatchNorm2d(1024)

        # Pooling Layers
        self.maxpool = nn.MaxPool2d(2, 2)
        self.avgpool = nn.AvgPool2d(2, 2)

        # FC Layers
        self.fc1 = nn.Linear(1024 * 5 * 5, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, 100)

    def forward(self, x):
        x = F.relu(self.batchnorm1(self.conv1(x)))
        x = self.avgpool(F.relu(self.batchnorm2(self.conv2(x))))
        x = self.avgpool(F.relu(self.batchnorm3(self.conv3(x))))
        x = F.relu(self.batchnorm4(self.conv4(x)))
        x = x.view(-1, 1024 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net = Net()
net.to(device)
```

[ ]:
```
def acc_calc(net):
    correct = 0
    total = 0
    net.eval()
    with torch.no_grad():
        for data in cifar100_test_loader:
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = net(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    acc = 100 * correct / total
    print('Accuracy of the network: %d' % (
        acc))
    return acc
```

[ ]:
```
loss_function = nn.CrossEntropyLoss() # best results with SGD 59 % after 40␣
 ↪epochs
# loss_function = nn.MultiMarginLoss() # with SGD gave 46% after 35 epochs
# loss_function = nn.SmoothL1Loss() # with SGD gave 57%


optimizer = optim.SGD(net.parameters(), lr=LR, momentum=0.9, weight_decay=5e-4)␣
 ↪# 61 after 88 epochs
# optimizer = optim.Adam(net.parameters(),lr=LR,weight_decay=5e-4) # best␣
 ↪accuracy of 58% on test set
# optimizer = optim.Adagrad(net.parameters(),lr=LR,weight_decay=5e-4)


iter_per_epoch = len(cifar100_training_loader)
nbatches = math.floor(50000/batch_size)
best_acc = 0

for epoch in range(1,EPOCH):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in notebook.tqdm(enumerate(cifar100_training_loader, 0)):

        net.train()
        # get the inputs
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
```

```
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)

        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('[Epoch %d] loss: %.3f' %(epoch, running_loss / nbatches))
    running_loss = 0.0

    if epoch>20:
        acc = acc_calc(net)

        if acc>best_acc:
          best_acc = acc
          torch.save(net.state_dict(), './drive/My Drive/colab_models/
↪sgd_mml_latest.pth'.format(epoch))
          print("checkpoint saved")

print('Finished Training')
```

1 Report

1.0.1 Made a 4 layer CNN architecture ending with 3 layer fully connected layer for object classification over CIFAR-100 dataset.

General structure of (conv->batchnorm->activation->pooling)

1.1 Parameters:

- Preprocessed training data with :
  - Random horizontal flip that flips the training images with 50% probability.
  - Random tilt(15 degrees) that rotates the images anywhere between (+15 to -15 degrees)
  - The above two increase the diversity of data available for training models(data augmentation).
  - Normalize the layers of every image according to the mean and standard deviation of the training set.
- Used ReLU `activation function` in the final model, experimented with `sigmoid, tanh`

- `Batch normalization` to reduce sensitivity to initial learning rates.
- `Loss functions used`, and accuracies over test set.

Tested over SGD optimizer, batch size = 128, Learning rate = 0.001, momentum=0.9, weight_decay=5e-4

`Loss function`	`Accuracy`	`Epochs`
CrossEntropyLoss	61	88
MultiMarginLoss	53	37
SmoothL1Loss	57	39

- `Optimizers used` and the accuracies over the test set:

  Tested over CrossEntropyLoss

`Optimizer`	`Accuracy`	`Epochs`	`Other parameters`
SGD	61	88	Learning rate = 0.001, momentum=0.9, weight_decay=5e-4
Adam	58	40	Learning rate = 0.001,weight_decay=5e-4
Adagrad	56	35	Learning rate = 0.001,weight_decay=5e-4

- `Batch size` was chosen to be 128 for all experiments, as this size gave appropriate training speed. Lower batch sizes take longer training time, given there are 50000 samples to train in every epoch.
- Best models were saved to show later.