HW03

October 2, 2018

0.1 Object Recognition in Image

Renjie Wei Train a deep convolution network on a GPU with PyTorch for the CIFAR10 dataset. The convolution network uses (A) dropout, (B) trained with RMSprop or ADAM, and (C) data augmentation. For 10% extra credit, compare dropout test accuracy (i) using the heuristic prediction rule and (ii) Monte Carlo simulation. For full credit, the model should achieve 80-90% Test Accuracy. Submit via Compass (1) the code and (2) a paragraph (in a PDF document) which reports the results and briefly describes the model architecture. ###### (A)The accuracy is 85.58 % ###### (B)Using dropout/ data augmentation/ trained with ADAM optimizer ###### (C)The convolution network is as follows:

PytorchDeepConv
(0): Conv2d(3, 64, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(0): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
(1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(2): ReLU()
(3): Dropout(p=0.4)
(0): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(0): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
(1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(2): ReLU()
(3): Dropout(p=0.4)
(0): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(2, 2))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
(1): ReLU()
(2): Dropout(p=0.4)
(0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
(1): ReLU()
(2): Dropout(p=0.4)
(0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(3): Dropout(p=0.4)

(0): Linear(in_features=1024, out_features=500, bias=True)
(1): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(3): Dropout(p=0.5)
(4): Linear(in_features=500, out_features=10, bias=True)

In [ ]:
```
import numpy as np
import torch
import torchvision
import torch.nn as nn
from torch.autograd import Variable
import torchvision.transforms as transforms
import time
#import h5py
```

In [ ]:
```
# Hyper parameters
num_epochs = 60
hidden_sizes = 500
input_channels = 3
num_classes = 10
batch_size = 100
learning_rate = 0.001
```

In [ ]:
```
torch.cuda.is_available()
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

In [ ]:
```
# Download and construct CIFAR-10 dataset
transform = transforms.Compose(
    [transforms.RandomHorizontalFlip(0.5),
     transforms.RandomAffine(degrees=15, translate=(0.1,0.1)),
     transforms.RandomResizedCrop(size=32, scale=(0.8, 1.0)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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

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



# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
```

```
        test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                                  batch_size=batch_size,
                                                  shuffle=False)
        #classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tr
```

```
In [ ]: class PytorchDeepConv(nn.Module):
            def __init__(self, input_channels, num_classes=10):
                super(PytorchDeepConv, self).__init__()
                #Layer 1
                self.layer1 = nn.Sequential(
                    nn.Conv2d(in_channels=input_channels, out_channels=64, kernel_size=4, stri
                    nn.BatchNorm2d(64),
                    nn.ReLU())

                # Layer 2
                self.layer2 = nn.Sequential(
                    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=4, stride=1, padding
                    nn.MaxPool2d(kernel_size=2, stride=2),
                    nn.ReLU(),
                    nn.Dropout(0.4))

                # Layer 3
                self.layer3 = nn.Sequential(
                    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=4, stride=1, padding
                    nn.BatchNorm2d(64),
                    nn.ReLU())

                # Layer 4
                self.layer4 = nn.Sequential(
                    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=4, stride=1, padding
                    nn.MaxPool2d(kernel_size=2, stride=2),
                    nn.ReLU(),
                    nn.Dropout(0.4))

                # Layer 5
                self.layer5 = nn.Sequential(
                    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=4, stride=1, padding
                    nn.BatchNorm2d(64),
                    nn.ReLU())
                # Layer 6
                self.layer6 = nn.Sequential(
                    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding
                    nn.ReLU(),
                    nn.Dropout(0.4))

                # Layer 7
                self.layer7 = nn.Sequential(
```

```
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding
            nn.ReLU(),
            nn.Dropout(0.4))

        #Layer 8
        self.layer8 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Dropout(0.4))

        # Fully Connected

        self.layer9 = nn.Sequential(
            nn.Linear(64*4*4, 500),
            nn.BatchNorm1d(500),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(500, num_classes))


    def forward(self, x):

        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = self.layer6(out)
        out = self.layer7(out)
        out = self.layer8(out)
        out = out.view(out.size(0), -1)

        # Linear function (readout)
        out = self.layer9(out)

        return out
```

```
In [ ]: model = PytorchDeepConv(input_channels, num_classes).to(device)
        #print(model)
```

```
In [ ]: #Define loss and optimizer
        # Use Adam as the optimizer
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
In [ ]: iter = 0
        accuracies = []
```

```python
for epoch in range(num_epochs):
    if(epoch > 6):
        for group in optimizer.param_groups:
            for p in group['params']:
                state = optimizer.state[p]
                if(state['step'] >= 1024):
                    state['step'] = 1000

    for i, (images, labels) in enumerate(train_loader):

        images, labels = images.to(device), labels.to(device)

        # Clear gradients w.r.t parameters
        optimizer.zero_grad()

        # Forward pass to get output/logits
        outputs = model(images)

        # Calculate Loss: Softmax --> cross entropy loss
        loss = criterion(outputs, labels)

        # Getting gradients w.r.t paramters
        loss.backward()

        # Updating parameters
        optimizer.step()

        iter += 1

        if iter % 500 == 0:
            # Calculate Accuracy
            correct = 0
            total = 0
            # Iterate through test dataset
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)

                # Forward pass only to get logits/output
                outputs = model(images)

                # Get predictions from the maximum value
                _, predicted = torch.max(outputs.data, 1)

                # Total number of labels
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
```

```
accuracy = 100 * correct / total
accuracies.append(accuracy)

# Print Loss
print('Iteration: {}. Loss: {}. Accuracy: {}'.format(iter, loss.data[0], ac
```