HW1-Part2_nm3191-ksw352

February 19, 2019

0.1 CIFAR-10 CNN using Pytorch

0.2 Submitted by :-

0.3 Namit Mohale - nm3191

0.4 Karanpreet Singh Wadhwa - ksw352

```
In [0]: import matplotlib.pyplot as plt
        import numpy as np
        import sys
        import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        import torchvision
        from torchvision.transforms import transforms
        from torch.autograd import Variable
        import torch.optim as optim
        %matplotlib inline
```

```
In [3]: 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', train=True,
                                                download=True, transform=transform)
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=50,
                                                  shuffle=True, num_workers=2)

        testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                               download=True, transform=transform)
        testloader = torch.utils.data.DataLoader(testset, batch_size=1000,
                                                 shuffle=False, num_workers=2)

        classes = ('plane', 'car', 'bird', 'cat',
                   'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
In [0]: class CifarCNN(nn.Module):
            def __init__(self, dim1=3, dim2=64, kernel_size1=5, kernel_size2=3, dim3=128, dim4=
                super(CifarCNN, self).__init__()

                self.dim4 = dim4
                self.kernel_size = kernel_size2

                self.conv1 = nn.Conv2d(dim1, dim2, kernel_size1, padding = 2, bias = True)
                self.conv2 = nn.Conv2d(dim2, dim3, kernel_size2, padding = 1, bias = True)
                self.conv3 = nn.Conv2d(dim3, dim4, kernel_size2, padding = 1, bias = True)
                self.pool = nn.MaxPool2d(d_pool, d_pool)
                self.dropout = nn.Dropout2d()
                self.fc1 = nn.Linear(dim4 * 8 * 8, dim5)
                self.fc2 = nn.Linear(dim5, dim6)
                self.fc3 = nn.Linear(dim6, n_classes)

            def forward(self, x):
                #forward pass
                #x is the input
                x = self.pool(F.relu(self.conv1(x)))
                x = self.pool(F.relu(self.conv2(x)))
                x = F.leaky_relu(self.dropout(self.conv3(x)))
                x = x.view(-1, self.dim4 * 8 * 8)
                x = F.relu(self.fc1(x))
                #x = F.relu(self.dropout(self.fc2(x)))
                x = F.dropout(F.relu(self.fc2(x)), training=self.training)
                x = self.fc3(x)
                # x = F.softmax(self.fc3(x), dim=0)
                return x

        net = CifarCNN()
```

The CNN model : 1. Convolutes 32x32x3 sample with 64 5x5 filter to 32x32x64 2. Applies ReLu 3. Takes a 2x2 maxpool and make it 16x16x64 sample 4. Convolutes 16x16x64 sample with 128 3x3 filter to 16x16x128 5. Applies ReLu 6. Takes a 2x2 maxpool and make it 8x8x128 sample 7. Convolutes 8x8x128 sample with 256 3x3 filter to 8x8x256 8. Applies dropout 9. Applies leaky-ReLu activation 10. Flattens the sample into a 9216 long layer 11. Applies fully connections with the first hidden layer with 1024 nodes followed by a ReLu 12. Another fully connected hidden layer with 512 nodes followed by a ReLu 13. Maps this to final output layer with 10 nodes

0.5 The model has been programmed on Google Colab and runs on GPU only

```
In [5]: criterion = nn.CrossEntropyLoss()

        #optimizer
        optimizer = optim.Adam(net.parameters(), lr=0.0001)

        net.cuda()

        for epoch in range(25): # loop over the dataset multiple times
            running_loss = 0.0
            total = 0
            correct = 0
            for i, data in enumerate(trainloader, 0):
                # get the inputs
                inputs, labels = data
                inputs, labels = inputs.cuda(), labels.cuda()

                # 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()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                predicted = predicted.to("cpu")
                labels = labels.to("cpu")
                correct += (predicted == labels).sum().item()

                # predictions = outputs.data.max(1)[1]

                # accuracy = np.sum(predictions.cpu().numpy()==labels.cpu().numpy())/50*100
                if i % 500 == 499:    # print every 6000 mini-batches
                    print('[%d, %5d] loss: %.3f' %
                          (epoch + 1, i + 1, running_loss / 500))
                    running_loss = 0.0

            print('Accuracy : ', 100 * correct / total)
```

```
[1,   500] loss: 1.774
[1,  1000] loss: 1.459
Accuracy :  41.104
[2,   500] loss: 1.319
[2,  1000] loss: 1.211
Accuracy :  54.934
[3,   500] loss: 1.105
[3,  1000] loss: 1.061
```

```
Accuracy :  61.572
[4,   500] loss: 0.968
[4,  1000] loss: 0.944
Accuracy :  66.21
[5,   500] loss: 0.864
[5,  1000] loss: 0.845
Accuracy :  69.984
[6,   500] loss: 0.782
[6,  1000] loss: 0.763
Accuracy :  72.906
[7,   500] loss: 0.697
[7,  1000] loss: 0.699
Accuracy :  75.592
[8,   500] loss: 0.629
[8,  1000] loss: 0.628
Accuracy :  78.204
[9,   500] loss: 0.550
[9,  1000] loss: 0.555
Accuracy :  80.76
[10,   500] loss: 0.483
[10,  1000] loss: 0.493
Accuracy :  82.924
[11,   500] loss: 0.420
[11,  1000] loss: 0.423
Accuracy :  85.428
[12,   500] loss: 0.355
[12,  1000] loss: 0.361
Accuracy :  87.45
[13,   500] loss: 0.306
[13,  1000] loss: 0.305
Accuracy :  89.538
[14,   500] loss: 0.246
[14,  1000] loss: 0.258
Accuracy :  91.332
[15,   500] loss: 0.202
[15,  1000] loss: 0.211
Accuracy :  93.014
[16,   500] loss: 0.165
[16,  1000] loss: 0.177
Accuracy :  94.276
[17,   500] loss: 0.137
[17,  1000] loss: 0.153
Accuracy :  95.032
[18,   500] loss: 0.119
[18,  1000] loss: 0.127
Accuracy :  95.862
[19,   500] loss: 0.101
[19,  1000] loss: 0.112
```

```
Accuracy :  96.364
[20,   500] loss: 0.089
[20,  1000] loss: 0.093
Accuracy :  97.0
[21,   500] loss: 0.077
[21,  1000] loss: 0.086
Accuracy :  97.324
[22,   500] loss: 0.073
[22,  1000] loss: 0.077
Accuracy :  97.46
[23,   500] loss: 0.066
[23,  1000] loss: 0.069
Accuracy :  97.722
[24,   500] loss: 0.058
[24,  1000] loss: 0.067
Accuracy :  97.954
[25,   500] loss: 0.053
[25,  1000] loss: 0.058
Accuracy :  98.128
```

Accuracy on Training set - 98.128%

```
In [8]: correct_test = 0
        total_test = 0
        final_test_output = []

        net.cuda()
        with torch.no_grad():
            for data in testloader:
                images, labels = data
                images, labels = images.cuda(), labels.cuda()
                outputs = net(images)
                _, predicted_test = torch.max(outputs.data, 1)
                total_test += labels.size(0)
                predicted_test = predicted_test.to("cpu")
                labels = labels.to("cpu")
                correct_test += (predicted_test == labels).sum().item()
                for item in predicted:
                    final_test_output.append(item.item())

        print('Accuracy of the network on the 10000 test images: %d %%' % (
            100 * correct_test / total_test))
```

```
Accuracy of the network on the 10000 test images: 74 %
```

Lines below are just to save the file to system from google colab

```
In [0]: final_test_np = np.asarray(final_test_output)
        filename = "HW1_Part2_nm3191_ksw352"
        np.save(filename, final_test_np)
```

```
In [15]: !dir
```

```
data  HW1_Part2_nm3191_ksw352.npy  sample_data
```

```
In [0]: from google.colab import files
        files.download('HW1_Part2_nm3191_ksw352.npy')
```

The network model described above, as we believe, is the perfect combination of complexity, accuracy and time spent. The network was getting better accuracy on the test set with a more complex network but was taking a longer time and other system resources. We believe that it is not advisable to to have a high complexity and longer time taken for training such a model on CPU. Hence we decided to settle for a little low accuracy but a model that would run efficiently even on a CPU.

Hyper-parameter tuning began with changing the optimization function. Adam Optimizer was used to check the result on accuracy. Other than this, neurons were increased upto 2048 in one hidden layer but this only turned the training slow without having a substantial impact on the error rate. Various settings of learning rate and momentum were used and then we decided on one which was giving the maximum accuracy score. Finally, a dropout regularization function was used to avoid overfitting and it yielded better results as accuracy went up to mid 70s. The accuracy also tended to improve when we were training the network for more than 25 epochs.

```
In [ ]:
```