Claude Shyaka

ID#: 801326243

Homework 6: Fully Connected Neural Nets and Convolution Neural Nets

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from torchvision import datasets
import torch
import torch.nn as nn
from torchvision import transforms

torch.set_printoptions(edgeitems=2, linewidth=75)
torch.manual_seed(123)
```

Out[1]:
```
<torch._C.Generator at 0x28f1d3cf990>
```

In [2]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
```

In [3]:
```
data_path = '../data-unversioned/p1ch7/'
cifar10 = datasets.CIFAR10(data_path, train=True, download=True)
cifar10_val = datasets.CIFAR10(data_path, train=False, download=True)
```

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

In [4]:
```
from torchvision import datasets, transforms
data_path = '../data-unversioned/p1ch7/'

transformed_cifar10 = datasets.CIFAR10(
    data_path, train=True, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465),
                             (0.2470, 0.2435, 0.2616))
    ]))
```

In [5]:
```
transformed_cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465),
                             (0.2470, 0.2435, 0.2616))]))
```

In [6]:
```
img_t, _ = transformed_cifar10[23]
type(img_t)
```

Out[6]:
```
torch.Tensor
```

In [7]:
```
img_t.shape, img_t.dtype
```

Out[7]:
```
(torch.Size([3, 32, 32]), torch.float32)
```

Using Fully Connected Neural Networks

In [8]:
```
import torch.optim as optim
import datetime

def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):

    for epoch in range(1, n_epochs+1):
        for imgs, labels in train_loader:
            batch_size = imgs.shape[0]
            outputs = model(imgs.view(batch_size, -1))
            loss = loss_fn(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if epoch <= 3 or epoch % 10 == 0:
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch, loss))

def val_loop(model, train_loader, val_loader):
    for name, loader in [('train', train_loader), ('val', val_loader)]:

        correct = 0
        total = 0
        with torch.no_grad():
            for imgs, labels in loader:
                batch_size = imgs.shape[0]
                outputs = model(imgs.view(batch_size, -1))
                _, predicted = torch.max(outputs, dim=1)
                total += labels.shape[0]
                correct += int((predicted == labels).sum())

        print("Accuracy {}: {:.2f}".format(name, correct / total))
```

Baseline Fully connected neural net (one hidden layer)

In [9]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)
n_out = 10
model = nn.Sequential(
    nn.Linear(3072, 512),
    nn.Tanh(),
    nn.Linear(512, n_out),
)

optimizer = optim.SGD(model.parameters(), lr=1e-2)
loss_fn = nn.CrossEntropyLoss()

training_loop(
    n_epochs=300,
    optimizer=optimizer,
    model=model,
    loss_fn=loss_fn,
    train_loader=train_loader
)
```

```
2022-12-11 10:58:13.280757 Epoch 1, Training loss 1.8393231630325317
2022-12-11 10:58:30.015504 Epoch 2, Training loss 1.377354621887207
2022-12-11 10:58:43.478048 Epoch 3, Training loss 1.9925981760025024
2022-12-11 10:59:59.939604 Epoch 10, Training loss 1.0380967855453491
2022-12-11 11:01:41.304959 Epoch 20, Training loss 1.15604567527771
2022-12-11 11:03:21.026989 Epoch 30, Training loss 0.7549644112586975
2022-12-11 11:05:01.675494 Epoch 40, Training loss 0.4356018602848053
2022-12-11 11:06:41.957034 Epoch 50, Training loss 0.264767587184906
2022-12-11 11:08:22.161326 Epoch 60, Training loss 0.12443574517965317
2022-12-11 11:10:01.373402 Epoch 70, Training loss 0.09410518407821655
2022-12-11 11:11:42.354460 Epoch 80, Training loss 0.08069346100091934
2022-12-11 11:13:23.798328 Epoch 90, Training loss 0.039708103984594345
2022-12-11 11:15:04.947798 Epoch 100, Training loss 0.05832041800022125
2022-12-11 11:16:45.813132 Epoch 110, Training loss 0.04636174440383911
2022-12-11 11:18:26.164265 Epoch 120, Training loss 0.049561962485313416
2022-12-11 11:20:07.168203 Epoch 130, Training loss 0.028371520340442657
2022-12-11 11:21:47.237991 Epoch 140, Training loss 0.013714182190597057
2022-12-11 11:23:26.646798 Epoch 150, Training loss 0.020319394767284393
2022-12-11 11:25:08.736029 Epoch 160, Training loss 0.011101871728897095
2022-12-11 11:26:49.043395 Epoch 170, Training loss 0.012981481850147247
2022-12-11 11:28:29.849446 Epoch 180, Training loss 0.018580114468932152
2022-12-11 11:30:11.089260 Epoch 190, Training loss 0.014980962499976158
2022-12-11 11:31:50.606455 Epoch 200, Training loss 0.009732008911669254
2022-12-11 11:33:37.320776 Epoch 210, Training loss 0.007163901347666979
2022-12-11 11:35:16.500358 Epoch 220, Training loss 0.009737375192344189
2022-12-11 11:36:56.939044 Epoch 230, Training loss 0.00758917722851038
2022-12-11 11:38:36.459644 Epoch 240, Training loss 0.006323343608528376
2022-12-11 11:40:15.945743 Epoch 250, Training loss 0.00980254914611578
2022-12-11 11:41:55.472943 Epoch 260, Training loss 0.004503951407968998
2022-12-11 11:43:35.803643 Epoch 270, Training loss 0.0066040619276463985
2022-12-11 11:45:15.804862 Epoch 280, Training loss 0.008188340812921524
2022-12-11 11:46:55.771103 Epoch 290, Training loss 0.01005062647163868
2022-12-11 11:48:38.092438 Epoch 300, Training loss 0.004050200339406729
```

In [10]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
val_loop(model, train_loader, val_loader)
```

```
Accuracy train: 1.00
Accuracy val: 0.47
```

In [11]:
```
torch.save(model.state_dict(), data_path + 'all_classes_in_cifar10_one_hidden.pt
# loaded_model = model()
# loaded_model.load_state_dict(torch.load(data_path+'all_classes_in_cifar10_one_
```

Fully connected neural neural (two hidden layers)

In [12]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)
n_out = 10
model = None
model = nn.Sequential(
    nn.Linear(3072, 1024),
    nn.Tanh(),
    nn.Linear(1024, 512),
    nn.Tanh(),
    nn.Linear(512, 128),
    nn.Tanh(),
    nn.Linear(128, 10),
)

optimizer = optim.SGD(model.parameters(), lr=1e-2)
loss_fn = nn.CrossEntropyLoss()

training_loop(
    n_epochs=300,
    optimizer=optimizer,
    model=model,
    loss_fn=loss_fn,
    train_loader=train_loader
)
```

```
2022-12-11 11:49:02.899913 Epoch 1, Training loss 1.6245378255844116
2022-12-11 11:49:19.819966 Epoch 2, Training loss 1.6818033456802368
2022-12-11 11:49:37.004989 Epoch 3, Training loss 1.534187912940979
2022-12-11 11:51:35.551934 Epoch 10, Training loss 1.4621562957763672
2022-12-11 11:54:24.723951 Epoch 20, Training loss 1.0064070224761963
2022-12-11 11:57:13.760355 Epoch 30, Training loss 0.42223429679870605
2022-12-11 12:00:03.094297 Epoch 40, Training loss 0.07197032868862152
2022-12-11 12:02:54.256213 Epoch 50, Training loss 0.1319016069173813
2022-12-11 12:05:44.142279 Epoch 60, Training loss 0.022984590381383896
2022-12-11 12:08:34.084251 Epoch 70, Training loss 0.0033064864110201597
2022-12-11 12:11:24.021114 Epoch 80, Training loss 0.0019372004317119718
2022-12-11 12:14:13.798448 Epoch 90, Training loss 0.0008641885360702872
2022-12-11 12:17:02.126268 Epoch 100, Training loss 0.0011034191120415926
2022-12-11 12:19:52.784060 Epoch 110, Training loss 0.0006099769379943609
2022-12-11 12:22:41.074480 Epoch 120, Training loss 0.0008398221689276397
2022-12-11 12:25:30.723128 Epoch 130, Training loss 0.0008954134536907077
2022-12-11 12:28:21.668476 Epoch 140, Training loss 0.0007451024139299989
2022-12-11 12:31:11.834882 Epoch 150, Training loss 0.0005631508538499475
2022-12-11 12:34:01.583784 Epoch 160, Training loss 0.0005271332920528948
2022-12-11 12:36:50.202954 Epoch 170, Training loss 0.0003790711925830692
2022-12-11 12:39:39.443500 Epoch 180, Training loss 0.00025765557074919343
2022-12-11 12:42:27.638850 Epoch 190, Training loss 0.0004719950957223773
2022-12-11 12:45:16.506408 Epoch 200, Training loss 0.0006778298411518335
2022-12-11 12:48:05.959776 Epoch 210, Training loss 0.00048704916844144464
2022-12-11 12:50:53.919736 Epoch 220, Training loss 0.0003699174558278173
2022-12-11 12:53:47.629453 Epoch 230, Training loss 0.00029789580730721354
2022-12-11 12:56:36.433313 Epoch 240, Training loss 0.0004132403992116451
2022-12-11 12:59:24.512749 Epoch 250, Training loss 0.0003209938295185566
2022-12-11 13:02:12.671812 Epoch 260, Training loss 0.00025040138280019164
2022-12-11 13:05:00.841798 Epoch 270, Training loss 0.0002103303122567013
2022-12-11 13:07:49.223346 Epoch 280, Training loss 0.00028234851197339594
2022-12-11 13:10:37.428494 Epoch 290, Training loss 0.0002949855988845229
2022-12-11 13:13:27.077946 Epoch 300, Training loss 0.00021510093938559294
```

In [13]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
val_loop(model, train_loader, val_loader)
```

```
Accuracy train: 1.00
Accuracy val: 0.47
```

In [14]:
```
torch.save(model.state_dict(), data_path + 'all_classes_in_cifar10_two_hidden_nn
# loaded_model = model()
# loaded_model.load_state_dict(torch.load(data_path+'all_classes_in_cifar10_two_
```

Using Convolution Neural Networks

Using neural net similar to the one lecture notes but with 10 labels

In [15]:
```
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(8 * 8 * 8, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = out.view(-1, 8 * 8 * 8)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [16]:
```
model = None
model = Net()
# model(img.unsqueeze(0))
```

In [17]:
```
device = (torch.device('cuda') if torch.cuda.is_available()
          else torch.device('cpu'))
print(f"Training on device {device}.")
```

```
Training on device cpu.
```

In [18]:
```
import datetime

def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        for imgs, labels in train_loader:
            imgs = imgs.to(device=device)  # <1>
            labels = labels.to(device=device)
            outputs = model(imgs)
            # print(imgs.shape, outputs.shape)
            loss = loss_fn(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_train += loss.item()

        if epoch == 1 or epoch % 10 == 0:
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch,
                loss_train / len(train_loader)))
```

In [19]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)

model = Net().to(device=device)  # <1>
optimizer = optim.SGD(model.parameters(), lr=1e-2)
loss_fn = nn.CrossEntropyLoss()

training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader,
)
```

```
2022-12-11 13:13:52.508301 Epoch 1, Training loss 2.0156423204085407
2022-12-11 13:16:05.405889 Epoch 10, Training loss 1.2216555023437266
2022-12-11 13:18:33.368723 Epoch 20, Training loss 1.0031733327661938
2022-12-11 13:20:59.987900 Epoch 30, Training loss 0.9101834044889416
2022-12-11 13:23:27.816641 Epoch 40, Training loss 0.8509914136451223
2022-12-11 13:25:54.989934 Epoch 50, Training loss 0.8117253520071049
2022-12-11 13:28:21.737352 Epoch 60, Training loss 0.7813487655823798
2022-12-11 13:30:48.293637 Epoch 70, Training loss 0.7542348110386173
2022-12-11 13:33:15.056108 Epoch 80, Training loss 0.7311401380145032
2022-12-11 13:35:41.888631 Epoch 90, Training loss 0.7101649436575678
2022-12-11 13:38:08.861524 Epoch 100, Training loss 0.6938671661764765
2022-12-11 13:40:35.628594 Epoch 110, Training loss 0.6760904482182335
2022-12-11 13:43:03.442203 Epoch 120, Training loss 0.6625848272267509
2022-12-11 13:45:30.827530 Epoch 130, Training loss 0.6487830730579088
2022-12-11 13:47:58.782459 Epoch 140, Training loss 0.6346335206037897
2022-12-11 13:50:25.367372 Epoch 150, Training loss 0.6240607622029531
2022-12-11 13:52:51.995629 Epoch 160, Training loss 0.6145861420942389
2022-12-11 13:55:18.039775 Epoch 170, Training loss 0.6042480942462106
2022-12-11 13:57:44.749598 Epoch 180, Training loss 0.5934932907981336
2022-12-11 14:00:11.158989 Epoch 190, Training loss 0.5860818295603822
2022-12-11 14:02:38.453734 Epoch 200, Training loss 0.5788571949276473
2022-12-11 14:05:04.896224 Epoch 210, Training loss 0.5705193582245761
2022-12-11 14:07:31.662349 Epoch 220, Training loss 0.563242406948753
2022-12-11 14:09:58.151357 Epoch 230, Training loss 0.5561558144438602
2022-12-11 14:12:28.405233 Epoch 240, Training loss 0.5510815901448355
2022-12-11 14:14:56.186957 Epoch 250, Training loss 0.5443046591470918
2022-12-11 14:17:23.324661 Epoch 260, Training loss 0.5399858901262893
2022-12-11 14:19:51.298577 Epoch 270, Training loss 0.5335220971794994
2022-12-11 14:22:18.442372 Epoch 280, Training loss 0.5296325752954654
2022-12-11 14:24:45.854437 Epoch 290, Training loss 0.5260840745266441
2022-12-11 14:27:13.134569 Epoch 300, Training loss 0.5220733637661885
```

In [20]:

```
import collections

train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
all_acc_dict = collections.OrderedDict()

def validate(model, train_loader, val_loader):
    accdict = {}
    for name, loader in [("train", train_loader), ("val", val_loader)]:
        correct = 0
        total = 0

        with torch.no_grad():
            for imgs, labels in loader:
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)
                outputs = model(imgs)
                _, predicted = torch.max(outputs, dim=1) # <1>
                total += labels.shape[0]
                correct += int((predicted == labels).sum())

        print("Accuracy {}: {:.2f}".format(name , correct / total))
        accdict[name] = correct / total
    return accdict

all_acc_dict["baseline"] = validate(model, train_loader, val_loader)
```

```
Accuracy train: 0.82
Accuracy val: 0.61
```

In [21]:
```
torch.save(model.state_dict(), data_path + 'all_classes_in_cifar10_cnn_base.pt')
# loaded_model = Net().to(device=device)
# loaded_model.load_state_dict(torch.load(data_path
#                                         + 'all_classes_in_cifar10_cnn_base.pt'
#                                         map_location=device))
```

In [22]:
```
class Net1(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 16, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(16 * 4 * 4, 128)
        self.fc2 = nn.Linear(128, 10)
        # self.fc3 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = F.max_pool2d(torch.tanh(self.conv3(out)), 2)
        out = out.view(-1, 16 * 4 * 4)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        # out = self.fc3(out)
        return out
```

In [23]:
```
model = None
model = Net1()
# model(img.unsqueeze(0))
```

In [24]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)

model = Net1().to(device=device)
optimizer = optim.SGD(model.parameters(), lr=1e-2)
loss_fn = nn.CrossEntropyLoss()

training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader,
)
```

```
2022-12-11 14:28:09.150179 Epoch 1, Training loss 2.064853612114401
2022-12-11 14:34:37.674423 Epoch 10, Training loss 1.1025405135910835
2022-12-11 14:41:55.699034 Epoch 20, Training loss 0.8562657746207684
2022-12-11 14:49:07.887266 Epoch 30, Training loss 0.7458662279807698
2022-12-11 14:56:17.773746 Epoch 40, Training loss 0.6705487764171322
2022-12-11 15:03:27.162519 Epoch 50, Training loss 0.6126089401333533
2022-12-11 15:10:36.274668 Epoch 60, Training loss 0.5626459585888611
2022-12-11 15:17:46.295135 Epoch 70, Training loss 0.515264455276682
2022-12-11 15:24:57.113478 Epoch 80, Training loss 0.4724246436525184
2022-12-11 15:32:08.335192 Epoch 90, Training loss 0.4301049930169759
2022-12-11 15:39:24.092985 Epoch 100, Training loss 0.3899847258958975
2022-12-11 15:46:38.435604 Epoch 110, Training loss 0.3535082617112438
2022-12-11 15:53:51.594794 Epoch 120, Training loss 0.3165793933279222
2022-12-11 16:01:13.121350 Epoch 130, Training loss 0.2819230659481357
2022-12-11 16:08:35.954663 Epoch 140, Training loss 0.24953677168930583
2022-12-11 16:15:58.499080 Epoch 150, Training loss 0.21859648085349356
2022-12-11 16:40:40.358212 Epoch 160, Training loss 0.18969233474120153
2022-12-11 16:47:55.770435 Epoch 170, Training loss 0.1623769300796873
2022-12-11 16:55:03.812128 Epoch 180, Training loss 0.1382157504939667
2022-12-11 17:02:12.379850 Epoch 190, Training loss 0.11878147475954974
2022-12-11 17:09:20.858388 Epoch 200, Training loss 0.10004898622546278
2022-12-11 17:16:34.165382 Epoch 210, Training loss 0.08254442197006301
2022-12-11 17:23:43.866635 Epoch 220, Training loss 0.0679532005749357
2022-12-11 17:30:53.883176 Epoch 230, Training loss 0.05754269234588384
2022-12-11 17:38:03.447171 Epoch 240, Training loss 0.04721187149434138
2022-12-11 17:45:13.711150 Epoch 250, Training loss 0.039008031913634306
2022-12-11 17:52:23.181251 Epoch 260, Training loss 0.03391805781489786
2022-12-11 17:59:32.453462 Epoch 270, Training loss 0.028365046483800387
2022-12-11 18:06:42.156427 Epoch 280, Training loss 0.024700839759048333
2022-12-11 18:13:51.625063 Epoch 290, Training loss 0.021478109199868138
2022-12-11 18:21:03.999532 Epoch 300, Training loss 0.019198353471391646
```

In [25]:

```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
all_acc_dict = collections.OrderedDict()

def validate(model, train_loader, val_loader):
    accdict = {}
    for name, loader in [("train", train_loader), ("val", val_loader)]:
        correct = 0
        total = 0

        with torch.no_grad():
            for imgs, labels in loader:
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)
                outputs = model(imgs)
                _, predicted = torch.max(outputs, dim=1) # <1>
                total += labels.shape[0]
                correct += int((predicted == labels).sum())

        print("Accuracy {}: {:.2f}".format(name , correct / total))
        accdict[name] = correct / total
    return accdict

all_acc_dict["baseline"] = validate(model, train_loader, val_loader)
```

```
Accuracy train: 1.00
Accuracy val: 0.70
```

In [26]:
```
torch.save(model.state_dict(), data_path + 'all_classes_in_cifar10_cnn_extented.
# loaded_model = Net().to(device=device)
# loaded_model.load_state_dict(torch.load(data_path
#                                         + 'all_classes_in_cifar10_cnn_base.pt'
#
```

In [ ]:
```

```