Name: Stanly Gomes
Student ID: 801118166

Homework 6

Problem1

December 10, 2022

[46]:
```
# Library import code from book
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Personal imports
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
import time

from torchvision import datasets
from torchvision import transforms
```

[ ]:
```
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
```

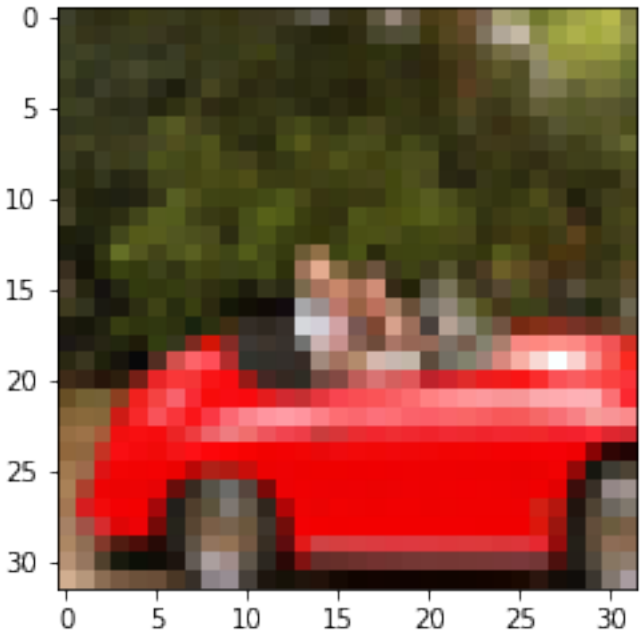
[ ]:
```
# Code from dlwpt repository
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
fig = plt.figure(figsize=(8,3))
num_classes = 10
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    ax.set_title(class_names[i])
    img = next(img for img, label in cifar10 if label == i)
    plt.imshow(img)
plt.show()
```

[ ]:
```
img, label = cifar10[99]
img, label, class_names[label]
```

[ ]:
```
(<PIL.Image.Image image mode=RGB size=32x32 at 0x2669040BC70>, 1, 'automobile')
```

[ ]:
```
plt.imshow(img)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x26691220fa0>
```



[ ]:
```
to_tensor = transforms.ToTensor()
img_t = to_tensor(img)
img_t.shape
```

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

[ ]:
```
img_t.cuda()
```

[ ]:
```
tensor([[[0.2431, 0.1961, 0.1804,  …, 0.6549, 0.7176, 0.5373],
         [0.2471, 0.2157, 0.2039,  …, 0.6392, 0.6706, 0.5686],
         [0.2275, 0.2510, 0.2196,  …, 0.6000, 0.5882, 0.4824],
         …,
         [0.6745, 0.5608, 0.5098,  …, 0.3686, 0.5529, 0.5451],
         [0.7176, 0.5882, 0.3137,  …, 0.3176, 0.5294, 0.5608],
         [0.8196, 0.7137, 0.5451,  …, 0.2314, 0.5098, 0.6627]],

        [[0.2510, 0.1961, 0.1725,  …, 0.6745, 0.7216, 0.5333],
         [0.2549, 0.2078, 0.1961,  …, 0.6627, 0.6824, 0.5725],
         [0.2431, 0.2588, 0.2353,  …, 0.6078, 0.6039, 0.5020],
         …,
         [0.5294, 0.4314, 0.2196,  …, 0.2941, 0.4235, 0.4118],
         [0.5725, 0.4627, 0.2510,  …, 0.2824, 0.4627, 0.4902],
         [0.6824, 0.5922, 0.4275,  …, 0.2118, 0.4667, 0.6118]],

        [[0.1725, 0.1020, 0.0745,  …, 0.2706, 0.2980, 0.2824],
         [0.1451, 0.1020, 0.1059,  …, 0.2392, 0.2941, 0.3020],
         [0.1412, 0.1451, 0.1451,  …, 0.2431, 0.2510, 0.2235],
         …,
         [0.3882, 0.3294, 0.1647,  …, 0.2196, 0.3373, 0.3176],
         [0.4588, 0.3725, 0.1725,  …, 0.2353, 0.3843, 0.4314],
         [0.5647, 0.4824, 0.3255,  …, 0.1843, 0.4353, 0.6275]]],
       device='cuda:0')
```

[ ]:
```
# Convert dataset to tensors
tensor_cifar10 = datasets.CIFAR10(data_path, train=True, download=False,
                                  transform=transforms.ToTensor())
```

[ ]:
```
# Normalizing Data
imgs = torch.stack([img_t for img_t, _ in tensor_cifar10], dim=3)
print(imgs.view(3, -1).mean(dim=1))
print(imgs.view(3, -1).std(dim=1))
```

```
tensor([0.4914, 0.4822, 0.4465])
tensor([0.2470, 0.2435, 0.2616])
```

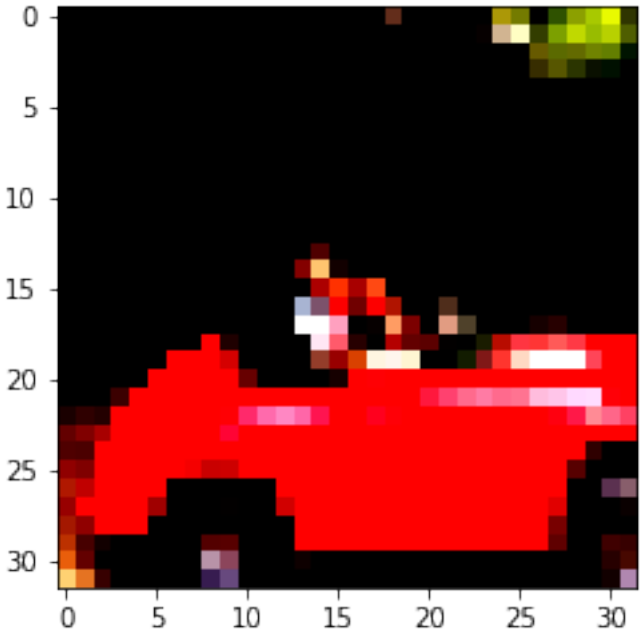
[40]:
```
transforms.Normalize((0.4915, 0.4823, 0.4468), (0.2470, 0.2435, 0.2616))
transformed_cifar10 = datasets.CIFAR10(
    data_path, train=True, download=False,
```

```
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
transformed_cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
transformed_cifar10
```

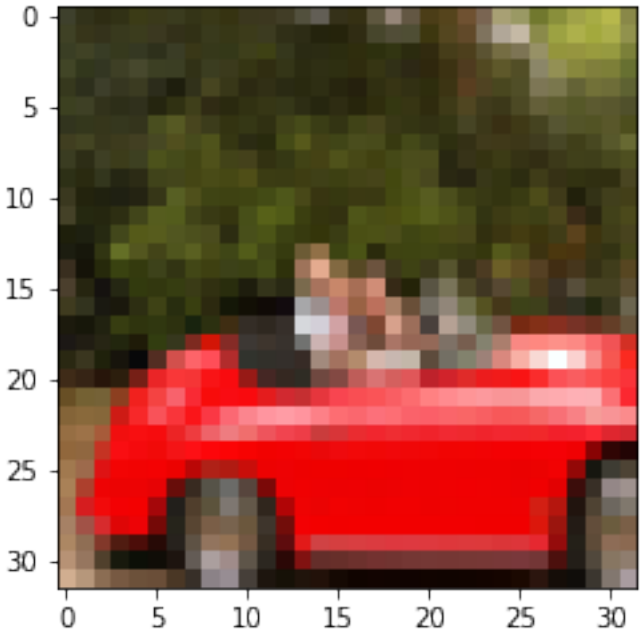
[40]:
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ../data-unversioned/p1ch7/
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.4915, 0.4823, 0.4468), std=(0.247, 0.2435,
0.2616))
           )
```

[41]:
```
img_t, _ = transformed_cifar10[99]
plt.imshow(img_t.permute(1, 2, 0))
plt.show()
img, _ = cifar10[99]
plt.imshow(img)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



[41]: <matplotlib.image.AxesImage at 0x2662b9d1b50>



1 Problem 1A: Create NN with one hidden layer

[42]:
```
def softmax(x):
    return torch.exp(x) / torch.exp(x).sum()
```

[59]:
```
n_out = 10
model = nn.Sequential(
            nn.Linear(3072, 512),
            nn.ReLU(),
            nn.Linear(512, n_out),
            nn.LogSoftmax(dim=1)
        )
```

[60]:
```
if torch.cuda.is_available():
    # Create a device object for the GPU
    device = torch.device('cuda')
else:
    # Create a device object for the CPU
    device = torch.device('cpu')
device
```

[60]:
```
device(type='cuda')
```

[68]:
```
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)
model = model.to(device)
learning_rate = 1e-2
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
loss_fn = nn.NLLLoss()
n_epochs = 300
# torch.cuda.empty_cache()
timer_s = time.time()

count_loss = []

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

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

    count_loss.append(float(loss))
```

```
    if epoch %15 == 0 or epoch == 300:
        print("Epoch: %d, Loss: %f" % (epoch, float(loss)))
timer_e = time.time()
train_time = timer_e - timer_s
print("Training Time:", train_time, "seconds")
```

```
Epoch: 0, Loss: 0.445650
Epoch: 15, Loss: 0.546036
Epoch: 30, Loss: 0.164014
Epoch: 45, Loss: 0.090564
Epoch: 60, Loss: 0.103511
Epoch: 75, Loss: 0.055877
Epoch: 90, Loss: 0.011263
Epoch: 105, Loss: 0.010230
Epoch: 120, Loss: 0.013145
Epoch: 135, Loss: 0.007493
Epoch: 150, Loss: 0.007798
Epoch: 165, Loss: 0.008456
Epoch: 180, Loss: 0.003039
Epoch: 195, Loss: 0.004563
Epoch: 210, Loss: 0.003655
Epoch: 225, Loss: 0.003615
Epoch: 240, Loss: 0.005348
Epoch: 255, Loss: 0.004412
Epoch: 270, Loss: 0.002757
Epoch: 285, Loss: 0.003547
Training Time: 3476.1903138160706 seconds
```

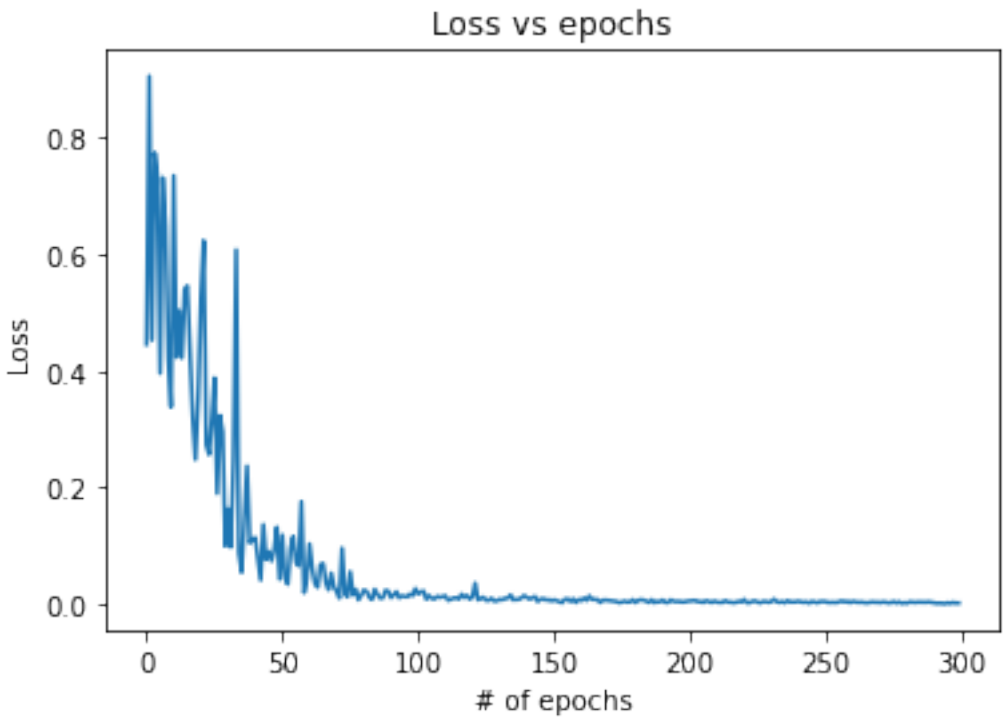
[71]:
```
# Training Accuracy
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
train_correct = 0
train_total = 0

with torch.no_grad():
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model(imgs.view(imgs.shape[0], -1))
        _, predicted = torch.max(outputs, dim=1)
        train_total += labels.shape[0]
        train_correct += int((predicted == labels).sum())

print("Training Accuracy: %f" % (train_correct / train_total))

# Validation Accuracy
```

```
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
val_correct = 0
val_total = 0

val_loss = []

with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model(imgs.view(imgs.shape[0], -1))
        _, predicted = torch.max(outputs, dim=1)
        val_total += labels.shape[0]
        val_correct += int((predicted == labels).sum())

print("Validation Accuracy: %f" % (val_correct / val_total))
```

```
Training Accuracy: 1.000000
Validation Accuracy: 0.538500
```

[79]:
```
# Save Model
torch.save(model.state_dict(), 'CIFAR10_1Layer.pt')
```

[73]:
```
# Plot Loss over epochs
plt.figure(1)
plt.plot(range(n_epochs), count_loss)
plt.title("Loss vs epochs")
plt.xlabel('# of epochs')
plt.ylabel('Loss')
plt.show()
```



2 1b: Add two more hidden layers to NN

[88]:
```
model_3 = nn.Sequential(
     nn.Linear(3072, 1024),
     nn.ReLU(),
     nn.Linear(1024, 512),
     nn.ReLU(),
     nn.Linear(512, 128),
     nn.ReLU(),
     nn.Linear(128, 10))
model_3
```

[88]:
```
Sequential(
  (0): Linear(in_features=3072, out_features=1024, bias=True)
  (1): ReLU()
  (2): Linear(in_features=1024, out_features=512, bias=True)
  (3): ReLU()
  (4): Linear(in_features=512, out_features=128, bias=True)
  (5): ReLU()
  (6): Linear(in_features=128, out_features=10, bias=True)
)
```

[86]:
```
transforms.Normalize((0.4915, 0.4823, 0.4468), (0.2470, 0.2435, 0.2616))
transformed_cifar10 = datasets.CIFAR10(
    data_path, train=True, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
transformed_cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
```

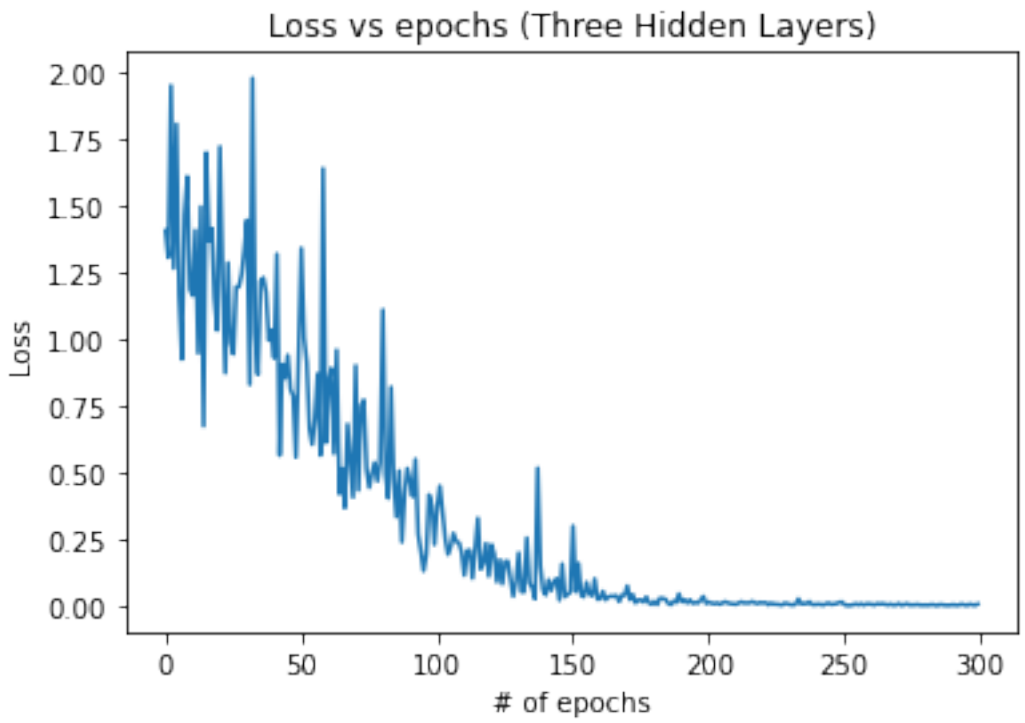
[90]:
```
train_loader_3 = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                             shuffle=True)
model_3 = model_3.to(device)
learning_rate_3 = 1e-3
optimizer_3 = optim.SGD(model_3.parameters(), lr=learning_rate_3)
loss_fn_3 = nn.CrossEntropyLoss()
n_epochs = 300

timer_s = time.time()
count_loss_3 = []
for epoch in range(n_epochs):
    for imgs, labels in train_loader_3:
        imgs = imgs.to(device)
        labels = labels.to(device)
        outputs = model_3(imgs.view(imgs.shape[0], -1))
        loss = loss_fn_3(outputs, labels)

        optimizer_3.zero_grad()
        loss.backward()
        optimizer_3.step()

    count_loss_3.append(float(loss))
    if epoch %15 == 0 or epoch == 299:
        print("Epoch: %d, Loss: %f" % (epoch, float(loss)))
timer_e = time.time()
train_time = timer_e - timer_s
print("Training Time:", train_time, "seconds")
```

```
Epoch: 0, Loss: 1.405433
Epoch: 15, Loss: 1.697608
Epoch: 30, Loss: 1.445171
Epoch: 45, Loss: 0.939078
```

```
Epoch: 60, Loss: 0.838340
Epoch: 75, Loss: 0.443933
Epoch: 90, Loss: 0.469855
Epoch: 105, Loss: 0.223419
Epoch: 120, Loss: 0.229977
Epoch: 135, Loss: 0.068372
Epoch: 150, Loss: 0.299775
Epoch: 165, Loss: 0.036444
Epoch: 180, Loss: 0.018165
Epoch: 195, Loss: 0.013542
Epoch: 210, Loss: 0.006593
Epoch: 225, Loss: 0.007379
Epoch: 240, Loss: 0.007242
Epoch: 255, Loss: 0.004605
Epoch: 270, Loss: 0.009399
Epoch: 285, Loss: 0.005825
Epoch: 299, Loss: 0.007568
Training Time: 3242.2549340724945 seconds
```

[92]:
```python
# Save Model
torch.save(model_3.state_dict(), 'CIFAR10_3Layer.pt')
```

[95]:
```python
# Plot Loss over epochs
plt.figure(1)
plt.plot(range(n_epochs), count_loss_3)
plt.title("Loss vs epochs (Three Hidden Layers)")
plt.xlabel('# of epochs')
plt.ylabel('Loss')
plt.show()
```



[99]:
```
# Training Accuracy
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
train_correct = 0
train_total = 0

with torch.no_grad():
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model_3(imgs.view(imgs.shape[0], -1))
        _, predicted = torch.max(outputs, dim=1)
        train_total += labels.shape[0]
        train_correct += int((predicted == labels).sum())

print("Training Accuracy: %f" % (train_correct / train_total))

# Validation Accuracy
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                           shuffle=False)
val_correct = 0
val_total = 0
```

```
val_loss_3 = []

with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model_3(imgs.view(imgs.shape[0], -1))
        loss = loss_fn_3(outputs, labels)
        _, predicted = torch.max(outputs, dim=1)
        val_total += labels.shape[0]
        val_correct += int((predicted == labels).sum())

print("Validation Accuracy: %f" % (val_correct / val_total))
```

```
Training Accuracy: 1.000000
Validation Accuracy: 0.523600
```

[ ]:

Problem2

December 10, 2022

[48]:
```
# Library import code from book
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Personal imports
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
import time
import datetime

from torchvision import datasets
from torchvision import transforms
import torch.nn.functional as F
```

[49]:
```
if torch.cuda.is_available():
    # Create a device object for the GPU
    device = torch.device('cuda')
else:
    # Create a device object for the CPU
    device = torch.device('cpu')
device
```

[49]:
```
device(type='cuda')
```

[50]:
```
data_path = '../data-unversioned/p1ch7/'
cifar10 = datasets.CIFAR10(data_path, train=True, download=False)
cifar10_val = datasets.CIFAR10(data_path, train=False, download=False)
```

[51]:
```
transformed_cifar10 = datasets.CIFAR10(
    data_path, train=True, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
```

```
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
transformed_cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4915, 0.4823, 0.4468),
                             (0.2470, 0.2435, 0.2616))
    ]))
```

1 Problem 2A: CNN with two hidden layers

Since MaxPool and Activation Function do not have parameters that need to be trained, the functional version of these elements will be used instead (where the outputs from each node layer will be passed into these functions).

[52]:
```
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
```

[56]:
```
def training_loop(n_epochs, optimizer, model, loss_fn, train_loader, val_loader,
                  train_loss, val_loss):
    timer_s = time.time()
    for epoch in range(0, n_epochs):
        loss_train = 0.0
        loss_val = 0.0
        for imgs, labels in train_loader:
            imgs = imgs.to(device)
            labels = labels.to(device)
            outputs = model(imgs)
            t_loss = loss_fn(outputs, labels)

            optimizer.zero_grad()
            torch.cuda.empty_cache()
```

```
            t_loss.backward()
            optimizer.step()

            loss_train += t_loss.item()

        for imgs, labels in val_loader:
            imgs = imgs.to(device)
            labels = labels.to(device)
            outputs = model(imgs)
            v_loss = loss_fn(outputs, labels)

            loss_val += v_loss.item()

        train_loss.append(loss_train/len(train_loader))
        val_loss.append(loss_val/len(val_loader))

        if epoch % 30 == 0 or epoch == 299:
            print('Epoch {}, Training loss {}, Validation loss {}'.format(
                epoch, loss_train / len(train_loader),
                loss_val/len(val_loader)))
    timer_e = time.time()
    train_time = timer_e - timer_s
    print("Training Time:", train_time, "seconds")
```

[57]:
```
torch.cuda.empty_cache()
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                           shuffle=True)
model = Net()
model = model.to(device)

learning_rate = 1e-3
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()
# optimizer = optimizer.to(device)
loss_fn = loss_fn.to(device)

train_loss = []
val_loss = []
```

[58]:
```
training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader,
```

```
    val_loader = val_loader,
    train_loss = train_loss,
    val_loss = val_loss
)
```

```
Epoch 0, Training loss 2.2831570303348627, Validation loss 2.243882756324331
Epoch 30, Training loss 1.5458097459410158, Validation loss 1.531758727541395
Epoch 60, Training loss 1.3636351297883427, Validation loss 1.3613317848011186
Epoch 90, Training loss 1.2277369410790446, Validation loss 1.2398202043430062
Epoch 120, Training loss 1.1006188426938508, Validation loss 1.124895600376615
Epoch 150, Training loss 1.0229382665870745, Validation loss 1.067679418500062
Epoch 180, Training loss 0.9690379347185345, Validation loss 1.01651333623631
Epoch 210, Training loss 0.9313014038383504, Validation loss 1.029131023367499
Epoch 240, Training loss 0.9020522409082984, Validation loss 0.9790690886746546
Epoch 270, Training loss 0.8769501789146678, Validation loss 0.9634382793098498
Epoch 299, Training loss 0.857290778280524, Validation loss 0.9641073258819094
Training Time: 5524.452919244766 seconds
```

[59]:
```
torch.save(model.state_dict(), 'CIFAR10_1LayerCNN.pt')
```

[61]:
```
torch.cuda.empty_cache()
```

[ ]:
```
# Plot Training and Validation Loss over epochs
plt.figure(1)
plt.plot(range(n_epochs), train_loss, 'g')
plt.plot(range(n_epochs), val_loss, 'b')
plt.title("Loss vs epochs")
plt.xlabel('# of epochs')
plt.ylabel('Loss')
plt.show()
```

[64]:
```
# Training Accuracy
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
train_correct = 0
train_total = 0

with torch.no_grad():
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model(imgs)
        _, predicted = torch.max(outputs, dim=1)
        train_total += labels.shape[0]
        train_correct += int((predicted == labels).sum())

print("Training Accuracy: %f" % (train_correct / train_total))
```

```
# Validation Accuracy
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
val_correct = 0
val_total = 0

with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model(imgs)
        _, predicted = torch.max(outputs, dim=1)
        val_total += labels.shape[0]
        val_correct += int((predicted == labels).sum())

print("Validation Accuracy: %f" % (val_correct / val_total))
```

```
Training Accuracy: 0.696000
Validation Accuracy: 0.662900
```

2 Problem 2B: CNN with an additional layer

[80]:
```
class Net_3conv(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 4 * 4, 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 = F.max_pool2d(torch.tanh(self.conv3(out)), 2)
        out = out.view(-1, 64 * 4 * 4)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

[81]:
```
torch.cuda.empty_cache()
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=True)
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=True)
```

```
model_3conv = Net_3conv().to(device)

learning_rate = 1e-3
optimizer = optim.SGD(model_3conv.parameters(), lr=learning_rate)
loss_fn_3 = nn.CrossEntropyLoss()
# optimizer = optimizer.to(device)
loss_fn_3 = loss_fn_3.to(device)

train_loss_3conv = []
val_loss_3conv = []
```

[82]:
```
training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model_3conv,
    loss_fn = loss_fn_3,
    train_loader = train_loader,
    val_loader = val_loader,
    train_loss = train_loss_3conv,
    val_loss = val_loss_3conv
)
```

```
Epoch 0, Training loss 2.2983914491770516, Validation loss 2.284728853565872
Epoch 30, Training loss 1.5436920279737019, Validation loss 1.5278783171040238
Epoch 60, Training loss 1.2742418855657358, Validation loss 1.2689268831994123
Epoch 90, Training loss 1.098003488291255, Validation loss 1.1130929923361275
Epoch 120, Training loss 0.9730362661201936, Validation loss 1.004539734618679
Epoch 150, Training loss 0.8726362977796198, Validation loss 0.9264594536678047
Epoch 180, Training loss 0.792600735679002, Validation loss 0.8676711761268081
Epoch 210, Training loss 0.7288371145039263, Validation loss 0.8315076888746517
Epoch 240, Training loss 0.6752930858251079, Validation loss 0.8096057506883221
Epoch 270, Training loss 0.6288914327578776, Validation loss 0.7833951705959952
Epoch 299, Training loss 0.5890144555617476, Validation loss 0.7769650934608119
Training Time: 5670.2540946006775 seconds
```

[83]:
```
torch.save(model_3conv.state_dict(), 'CIFAR10_3LayerCNN.pt')
```

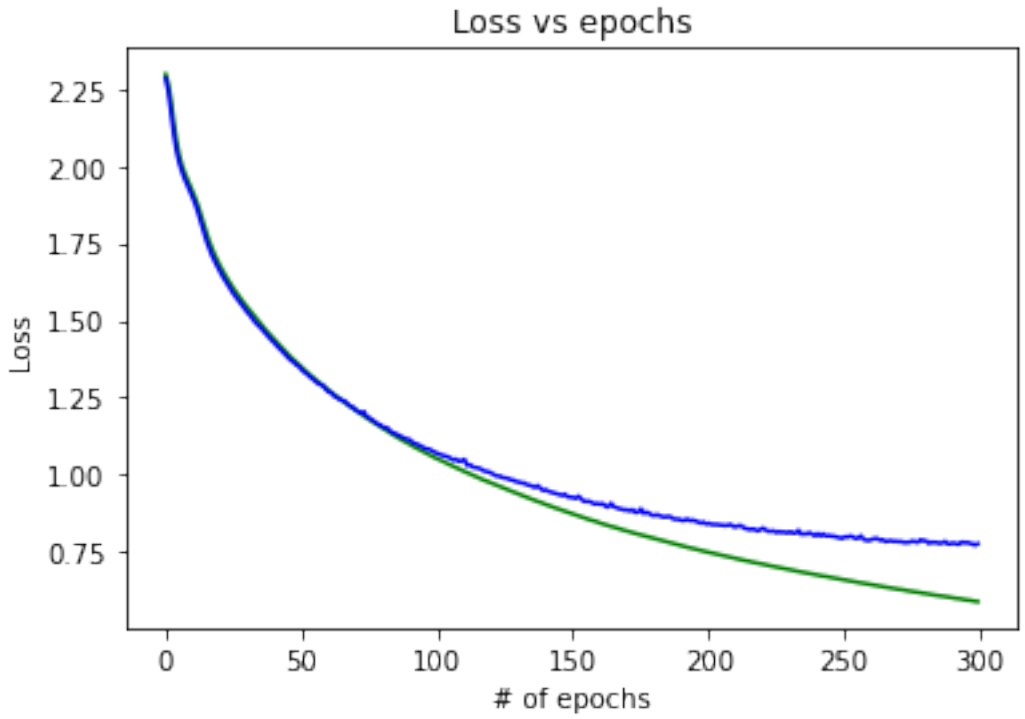
[84]:
```
torch.cuda.empty_cache()
```

[86]:
```
# Training Accuracy
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64,
                                           shuffle=False)
train_correct = 0
train_total = 0

with torch.no_grad():
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
```

```
        labels = labels.to(device)

        outputs = model_3conv(imgs)
        _, predicted = torch.max(outputs, dim=1)
        train_total += labels.shape[0]
        train_correct += int((predicted == labels).sum())

print("Training Accuracy: %f" % (train_correct / train_total))

# Validation Accuracy
val_loader = torch.utils.data.DataLoader(transformed_cifar10_val, batch_size=64,
                                         shuffle=False)
val_correct = 0
val_total = 0

with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)

        outputs = model_3conv(imgs)
        _, predicted = torch.max(outputs, dim=1)
        val_total += labels.shape[0]
        val_correct += int((predicted == labels).sum())

print("Validation Accuracy: %f" % (val_correct / val_total))
```

```
Training Accuracy: 0.800320
Validation Accuracy: 0.730200
```

[87]:
```
# Plot Training and Validation Loss over epochs
plt.figure(1)
plt.plot(range(n_epochs), train_loss_3conv, 'g')
plt.plot(range(n_epochs), val_loss_3conv, 'b')
plt.title("Loss vs epochs")
plt.xlabel('# of epochs')
plt.ylabel('Loss')
plt.show()
```


Loss vs epochs
2.25
2.00
1.75
1.50
1.25
1.00
0.75
Loss
0
50
100
150
200
250
300
# of epochs

[ ]: