CIFAR10 classification using Neural Networks

```
%matplotlib inline
```

Load CIFAR10 Dataset using torchvision

```
import torch
import torchvision
import torch.nn as nn
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import gc
```

```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
normalize = transforms.Normalize(
        mean=[0.4914, 0.4822, 0.4465],
        std=[0.2023, 0.1994, 0.2010],
    )
transform = transforms.Compose([
          transforms.Resize((224,224)),
          transforms.ToTensor(),
          normalize,
  ])
# Classes of CIFAR10 with label number for easy look up
classes = {0:'plane', 1:'car', 2:'bird', 3:'cat', 4:'deer', 5:'dog', 6:'frog', 7:'horse', 8:'ship', 9:'truck'}

# define batch size
batch_size = 128

# Train set
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,shuffle=True, num_workers=2,pin_memory=True)

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

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

```
# look at train set
print(trainset)
# print out tensor shape and corresponding label of one image.
random_image = trainset[5][0]
random_image_label = trainset[5][1]
print(random_image.shape,random_image_label)
```

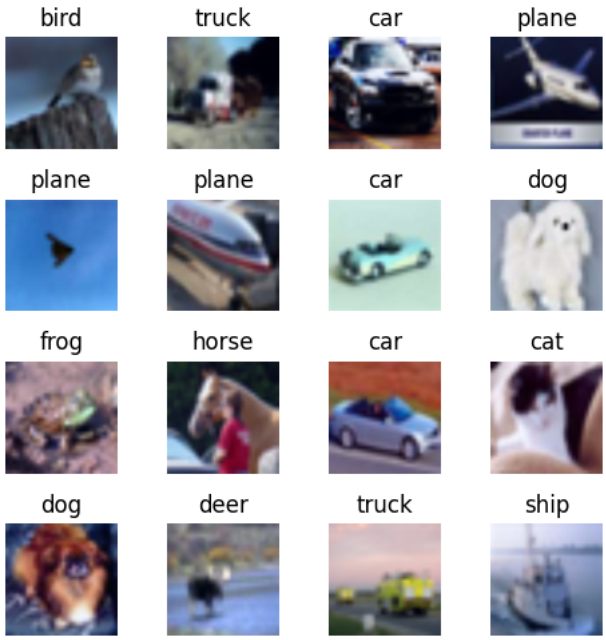
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, antialias=warn)
               ToTensor()
               Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.201])
           )
torch.Size([3, 224, 224]) 1
```

```
# Print the Image using Matplotlib
plt.imshow(np.transpose(random_image, (1, 2, 0)))
print("The label of the image is:", classes[random_image_label])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RG
The label of the image is: car
```

Visualize some images of a batch

```
batch, labels = next(iter(trainloader))

plt.figure(figsize=(5, 5))
for i in range(16):
    plt.subplot(4, 4, i+1)
    # De-normalize
    t = batch[i]
    min, max = torch.min(t), torch.max(t)
    im = np.transpose((t-min)/(max-min), (1, 2, 0)).numpy()
    plt.imshow(im)
    plt.title(classes[labels.numpy()[i]])
    plt.axis("off")
plt.tight_layout()
plt.show()
```



Costruct ResNet18 Architecture

From: https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/

```
# Define residual block
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride = 1, downsample = None):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
                        nn.Conv2d(in_channels, out_channels, kernel_size = 3, stride = stride, padding = 1),
                        nn.BatchNorm2d(out_channels),
                        nn.ReLU())
        self.conv2 = nn.Sequential(
                        nn.Conv2d(out_channels, out_channels, kernel_size = 3, stride = 1, padding = 1),
                        nn.BatchNorm2d(out_channels))
        self.downsample = downsample
```

```
        self.relu = nn.ReLU()
        self.out_channels = out_channels

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

```
# ResNet18
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes = 10):
        super(ResNet, self).__init__()
        self.inplanes = 64
        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 64, kernel_size = 7, stride = 2, padding = 3),
                        nn.BatchNorm2d(64),
                        nn.ReLU())
        self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)
        self.layer0 = self._make_layer(block, 64, layers[0], stride = 1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride = 2)
        self.layer2 = self._make_layer(block, 256, layers[2], stride = 2)
        self.layer3 = self._make_layer(block, 512, layers[3], stride = 2)
        self.avgpool = nn.AvgPool2d(7, stride=1)
        self.fc = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:

            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

Training

```
num_classes = 10
num_epochs = 30
learning_rate = 0.01

print('torch.cuda.is_available()', torch.cuda.is_available())
model = ResNet(ResidualBlock, [2, 2, 2, 2]).to(device)

# cross entropy Loss and SGD optimizer with momentum
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay = 0.001, momentum = 0.9)

# Train the model
total_step = len(trainloader)
```

```
torch.cuda.is_available() True
```

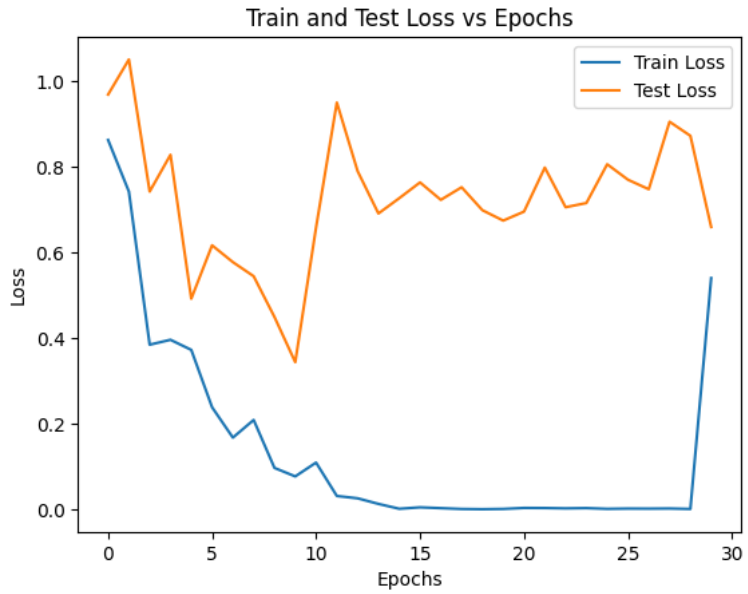
```
train_loss = []
test_loss = []
accuracy = []
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(trainloader):
        # Move tensors to the configured device
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print ('Epoch [{}/{}], Loss: {:.4f}' .format(epoch+1, num_epochs, loss.item()))
    train_loss.append(loss.item())


    # Validation
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in testloader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            del images, labels, outputs

        print('Accuracy of the network on the {} validation images: {} %'.format(50000, 100 * correct / total))
        test_loss.append(loss.item())
        accuracy.append(100 * correct / total)
```

```
Epoch [2/30], Loss: 0.7420
Accuracy of the network on the 50000 validation images: 74.87 %
Epoch [3/30], Loss: 0.3851
Accuracy of the network on the 50000 validation images: 79.76 %
Epoch [4/30], Loss: 0.3965
Accuracy of the network on the 50000 validation images: 81.39 %
Epoch [5/30], Loss: 0.3730
Accuracy of the network on the 50000 validation images: 81.64 %
Epoch [6/30], Loss: 0.2393
Accuracy of the network on the 50000 validation images: 82.17 %
Epoch [7/30], Loss: 0.1686
Accuracy of the network on the 50000 validation images: 81.57 %
Epoch [8/30], Loss: 0.2097
Accuracy of the network on the 50000 validation images: 82.41 %
Epoch [9/30], Loss: 0.0980
Accuracy of the network on the 50000 validation images: 82.67 %
Epoch [10/30], Loss: 0.0781
Accuracy of the network on the 50000 validation images: 82.61 %
Epoch [11/30], Loss: 0.1104
Accuracy of the network on the 50000 validation images: 83.74 %
Epoch [12/30], Loss: 0.0325
Accuracy of the network on the 50000 validation images: 83.94 %
Epoch [13/30], Loss: 0.0270
Accuracy of the network on the 50000 validation images: 83.43 %
Epoch [14/30], Loss: 0.0141
Accuracy of the network on the 50000 validation images: 84.9 %
Epoch [15/30], Loss: 0.0027
Accuracy of the network on the 50000 validation images: 86.21 %
Epoch [16/30], Loss: 0.0060
Accuracy of the network on the 50000 validation images: 86.67 %
Epoch [17/30], Loss: 0.0041
Accuracy of the network on the 50000 validation images: 86.85 %
Epoch [18/30], Loss: 0.0022
Accuracy of the network on the 50000 validation images: 86.81 %
Epoch [19/30], Loss: 0.0017
Accuracy of the network on the 50000 validation images: 86.93 %
Epoch [20/30], Loss: 0.0023
Accuracy of the network on the 50000 validation images: 86.95 %
Epoch [21/30], Loss: 0.0046
Accuracy of the network on the 50000 validation images: 87.07 %
Epoch [22/30], Loss: 0.0044
Accuracy of the network on the 50000 validation images: 86.94 %
Epoch [23/30], Loss: 0.0036
```

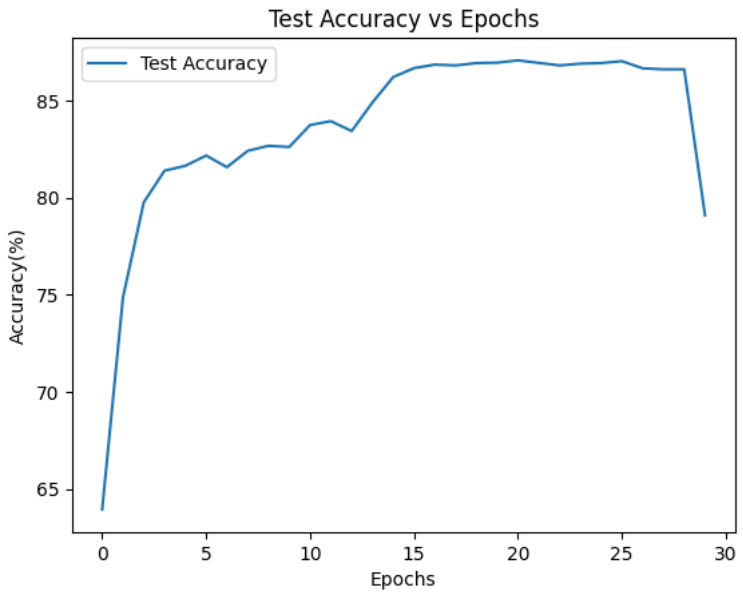
```
Accuracy of the network on the 50000 validation images: 86.9 %
Epoch [25/30], Loss: 0.0025
Accuracy of the network on the 50000 validation images: 86.93 %
Epoch [26/30], Loss: 0.0032
Accuracy of the network on the 50000 validation images: 87.03 %
Epoch [27/30], Loss: 0.0030
Accuracy of the network on the 50000 validation images: 86.66 %
Epoch [28/30], Loss: 0.0033
Accuracy of the network on the 50000 validation images: 86.61 %
Epoch [29/30], Loss: 0.0023
Accuracy of the network on the 50000 validation images: 86.61 %
Epoch [30/30], Loss: 0.5404
Accuracy of the network on the 50000 validation images: 79.09 %
```

Plot Losses and Accuracy

```
plt.plot(train_loss, label='Train Loss')
plt.plot(test_loss, label='Test Loss')
plt.title("Train and Test Loss vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.plot(accuracy, label='Test Accuracy')
plt.title("Test Accuracy vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy(%)")
plt.legend()
plt.show()
```



Advanced Data Analysis
Homework Week 6

Aswin Vijay

June 5, 2023

Homework 6

Delta in the last layer of for the j^th unit in multi-class classification is,

$$\delta_j^{(L)} = \frac{\partial J_n}{\partial u_j^{(L)}}$$

$$J_n(w) = -\sum_{k=1}^{K} y_{nk} \log f_k(x_n; w)$$

$$f_k = z_k^{(L)} = \frac{exp(u_k^{(L)})}{\sum_{j=1}^{K} exp(u_j^{(L)})}$$

For K output clsses and n^{th} sample. Then J_n becomes,

$$J_n(w) = -\sum_{k=1}^{K} y_{nk} \log \frac{exp(u_k^{(L)})}{\sum_{j=1}^{K} exp(u_j^{(L)})}$$

$$J_n(w) = -\sum_{k=1}^{K} y_{nk} \log f_k$$

$$\frac{\partial J_n}{\partial u_j^{(L)}} = \frac{\partial J_n}{\partial f_k} \cdot \frac{\partial f_k}{\partial u_j^{(L)}}$$

$$\frac{\partial J_n}{\partial u_j^{(L)}} = -\sum_{k=1}^{K} \frac{y_{nk}}{f_k} \cdot \frac{\partial f_k}{\partial u_j^{(L)}}$$

We need to calculate $\frac{\partial f_k}{\partial u_j^{(L)}} = \frac{\partial z_k^{(L)}}{\partial u_j^{(L)}}$. Let us compute the following derivative,

$$\frac{\partial}{\partial u_j} \frac{exp(u_k)}{\sum_{i=1}^{K} exp(u_i)}$$

Case 1: when $k = j$

$$\frac{\partial}{\partial u_j} \frac{exp(u_j)}{\sum_{i=1}^{K} exp(u_i)} = \frac{\sum_{i=1}^{K} exp(u_i)^2 - exp(u_j)^2}{\sum_{i=1}^{K} exp(u_i)^2} = f_j(1 - f_j)$$

Case 2: when $k \neq j$

$$\frac{\partial}{\partial u_j} \frac{exp(u_k)}{\sum_{i=1}^{K} exp(u_i)} = \frac{-exp(u_j)exp(u_k)}{\sum_{i=1}^{K} exp(u_i)^2} = f_k f_j$$

Using the above we have,

$$\frac{\partial J_n}{\partial u_j^{(L)}} = -\sum_{k=1}^{K} \frac{y_{nk}}{f_k} \cdot \frac{\partial f_k}{\partial u_j^{(L)}} = -\sum_{k=1}^{K} \cdot y_n k(1 - f_j)$$

Since $y_{nk} = 1$ only when $k = j$, we can further simlify as,

$$\frac{\partial J_n}{\partial u_j^{(L)}} = f_j - y_{nj}$$

Thus proven.