COMPSCI 371 Homework 9

Group Members: Mayur Sekhar, Jai Kasera, Rithvik Neti

Problem 0 (3 points)

Part 1: Correlation and Convolution

Problem 1.1 (Exam Style)

$$\begin{aligned}
\texttt{conv}(x, h, \texttt{'full'}) &= [6, 17, 8, 5, 18, 22, 4] \\
\texttt{conv}(x, h, \texttt{'valid'}) &= [8, 5, 18] \\
\texttt{conv}(x, h, \texttt{'same'}) &= [17, 8, 5, 18, 22] \\
\\
\texttt{corr}(x, h, \texttt{'full'}) &= [3, 16, 11, 4, 14, 24, 8] \\
\texttt{corr}(x, h, \texttt{'valid'}) &= [11, 4, 14] \\
\texttt{corr}(x, h, \texttt{'same'}) &= [16, 11, 4, 14, 24]
\end{aligned}$$

Problem 1.2 (Exam Style)

$$\begin{aligned}
\texttt{conv}(a, b, \texttt{'valid'}) &= \begin{bmatrix} 15 & 16 \\ 31 & 18 \end{bmatrix} \\
\texttt{corr}(a, b, \texttt{'valid'}) &= \begin{bmatrix} 16 & 19 \\ 36 & 28 \end{bmatrix}
\end{aligned}$$

Part 2: Gradients of a Convolution

Problem 2.1 (Exam Style)

$$C_f = \begin{bmatrix} h_1 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & h_3 & h_2 \\ 0 & 0 & 0 & 0 & h_3 \end{bmatrix}$$

Problem 2.2 (Exam Style)

$$C_v = \begin{bmatrix} h_3 & h_2 & h_1 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 \end{bmatrix}$$

$$C_s = \begin{bmatrix} h_2 & h_1 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & h_3 & h_2 \end{bmatrix}$$

Problem 2.3 (Exam Style)

$$X_f = \begin{bmatrix} x_1 & 0 & 0 \\ x_2 & x_1 & 0 \\ x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ x_5 & x_4 & x_3 \\ 0 & x_5 & x_4 \\ 0 & 0 & x_5 \end{bmatrix}$$

Problem 2.4 (Exam Style)

$$J_x = \frac{\partial y}{\partial x} = C_f = \begin{bmatrix} h_1 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & h_3 & h_2 \\ 0 & 0 & 0 & 0 & h_3 \end{bmatrix}$$

$$J_h = \frac{\partial y}{\partial h} = X_f = \begin{bmatrix} x_1 & 0 & 0 \\ x_2 & x_1 & 0 \\ x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ x_5 & x_4 & x_3 \\ 0 & x_5 & x_4 \\ 0 & 0 & x_5 \end{bmatrix}$$

Problem 2.5 (Exam Style)

Given that $g_y = \frac{\partial l}{\partial y}$:

$$g_x = \frac{\partial l}{\partial x} = C_f^\intercal g_y = \begin{bmatrix} h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & h_1 & h_2 & h_3 & 0 & 0 & 0 \\ 0 & 0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & h_1 & h_2 & h_3 & 0 \\ 0 & 0 & 0 & 0 & h_1 & h_2 & h_3 \end{bmatrix} \cdot \frac{\partial l}{\partial y}$$

$$g_h = \frac{\partial l}{\partial h} = X_f^\intercal g_y = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & 0 & 0 \\ 0 & x_1 & x_2 & x_3 & x_4 & x_5 & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix} \cdot \frac{\partial l}{\partial y}$$

Problem 2.6 (Exam Style)

The operation $g_x = C_f^\intercal g_y$ corresponds to the full correlation of g_y with h (kernel not flipped).

- Type: Full Correlation
- Operands: g_y and h

- $g_x = \texttt{conv}(g, h, \texttt{'valid'})$

The operation $g_h = X_f^\top g_y$ corresponds to the full correlation of g_y with x (kernel not flipped).

- Type: Full Correlation
- Operands: g_y and x
- $g_h = \texttt{conv}(g, x, \texttt{'valid'})$

Part 3: A Neural Network Classifier

In [1]:
```
from types import SimpleNamespace
import torch
import torch.nn as nn
import torch.nn.functional as fct
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from math import prod
%matplotlib inline
```

In [2]:
```
def cifar_10_loaders(data_root='./data', batch_size=4):
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    train_set = torchvision.datasets.CIFAR10(
        root=data_root, train=True, download=True, transform=transform
    )
    test_set = torchvision.datasets.CIFAR10(
        root=data_root, train=False, download=True, transform=transform
    )
    class_names = (
        'plane', 'car', 'bird', 'cat', 'deer',
        'dog', 'frog', 'horse', 'ship', 'truck'
    )
    loader = torch.utils.data.DataLoader(
        train_set, batch_size=1, shuffle=False, num_workers=2
    )
    iterator = iter(loader)
    images, _ = next(iterator)
    shape = np.array(images.size()[1:])
    train_loader = torch.utils.data.DataLoader(
        train_set, batch_size=batch_size, shuffle=False, num_workers=2
```

```
    )
    test_loader = torch.utils.data.DataLoader(
        test_set, batch_size=batch_size, shuffle=False, num_workers=2
    )
    return SimpleNamespace(
        train=train_loader, test=test_loader, classes=class_names, input_sha
    )
```

In [3]:
```
loaders = cifar_10_loaders()
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./dat
a/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:06<00:00, 25848220.34it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

In [4]:
```
def show_sample(loader, classes):
    iterator = iter(loader)
    images, labels = next(iterator)
    images = images / 2 + 0.5     # un-normalize
    array = images.numpy()
    n = array.shape[0]
    plt.figure(figsize=(7, 2), tight_layout=True)
    for k, a in enumerate(array):
        plt.subplot(1, n, k + 1)
        plt.imshow(np.transpose(a, (1, 2, 0)))
        plt.xticks([])
        plt.yticks([])
        plt.xlabel(classes[labels[k]], fontsize=16)
    plt.show()
```

In [13]:
```
class SimpleNet(nn.Module):
    def __init__(self, input_shape, convolutions=False):
        super().__init__()
        self.convolutions = convolutions
        image_pixels = np.copy(input_shape[1:])
        if self.convolutions:
            self.conv1 = nn.Conv2d(3, 6, 5)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(6, 16, 5)
            for j in range(2):
                image_pixels -= 4
                image_pixels //= 2
            m = 16 * prod(list(image_pixels))
            self.fc1 = nn.Linear(m, 120)
        else:
            self.convolutions = False
            input_size = prod(list(input_shape))
            self.fc1 = nn.Linear(input_size, 120)
```

```
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        if self.convolutions:
            x = self.pool(fct.relu(self.conv1(x)))
            x = self.pool(fct.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = fct.relu(self.fc1(x))
        x = fct.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Problem 3.1 (Exam Style)

Overall, the purpose of the for loop is to iterate through the two convolution-pooling layers in the network. The loop essentially applies the dimension transformations of the convolution and pooling layers to the image. This is done by the two lines explained below.

image_pixels -= 4: this line is subtracting 4 from the image size, specifically 4 from both the height and the width. For each convolutional layer, the kernel size is 5x5 which reduces 4 pixels, 2 from each side.

image_pixels //= 2: this is line is dividing both the height and width of the image by 2, because each pooling layer is using 2x2 nonoverlapping windows to reduce the number of dimensions.

Problem 3.2

Calculations for convolutions = False:

First Fully Connected Layer:

Input size = $3 * 32 * 32 = 3072$. Thus, the number of parameters = $3072 * 120 + 120 = 368760$

Second Fully Connected Layer:

$84 * 120 + 84 = 850$

Third Fully Connected Layer:

$84 * 120 + 84 = 10164$

Calculations for convolutions = True:

First Fully Connected Layer:

We start with an image size of 32x32. After the first convolution-pooling layer, this becomes 32x32 --> (32-4) x (32-4) --> 28x28 --> 28//2 x 28//2 --> 14 x 14. After the second convolution pooling layer, this becomes 14x14 --> (14-4)x(14-4) --> 10x10 --> (10//2)x(10//2) --> 5x5. Thus, our number of parameters = $(5 * 5 * 16 * 120) + 120 = 48120$

Second Fully Connected Layer:

$84 * 120 + 120 = 10164$

Third Fully Connected Layer:

$84 * 10 + 10 = 850$

First Convolutional Layer:

3 input channels, 6 output channels, kernel size is 5x5. This gives us $(3 * 6 * 5 * 5) + 6 = 456$

Second Convolutional Layer:

6 input channels, 16 output channels, kernel size is 5x5. This gives us $(6 * 16 * 5 * 5) + 16 = 2416$

Total parameters

For convolutions = False, summing up the values we get $368760 + 850 + 10164 = 379,774$

For convolutions = True, summing up the values we get $48120 + 10164 + 850 + 456 + 2416 = 62,006$

These 2 numbers corroborate the numbers given in the problem statement.

Problem 3.3

In [14]:
```
def train(
    model, loader, epochs,
    start_epoch=0, learning_rate=0.001,
    momentum=0.9, print_interval=2000
):
```

```
    loss_function = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(
        model.parameters(), lr=learning_rate, momentum=momentum
    )
    fmt = '[{:2d}, {:5d}] risk over last {} mini-batches: {:.3f}'
    for epoch in range(epochs):
        running_risk = 0.
        for i, batch in enumerate(loader, 0):
            images, labels = batch
            optimizer.zero_grad()

            predictions = model(images)
            risk = loss_function(predictions, labels)
            risk.backward()
            optimizer.step()

            running_risk += risk.item()
            if i % print_interval == print_interval - 1:
                block_risk = running_risk / print_interval
                ep = epoch + start_epoch
                print(fmt.format(ep + 1, i + 1, print_interval, block_risk))
                running_risk = 0.
```

In [15]:

```
def evaluate(model, loader, set_name):
    correct, total = 0, 0
    with torch.no_grad():
        for batch in loader:
            images, labels = batch
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print('{} accuracy {:.2f} percent'.format(set_name, accuracy))
    return accuracy
```

In [16]:

```
def experiment(data_loaders, convolutions, epochs):
    model = SimpleNet(data_loaders.input_shape, convolutions=convolutions)
    print('training')
    train(model, data_loaders.train, epochs=epochs)
    print('evaluating')
    training_accuracy = evaluate(model, data_loaders.train, 'training')
    test_accuracy = evaluate(model, data_loaders.test, 'test')
    return training_accuracy, test_accuracy
```

In [17]:

```
wo_train_acc, wo_test_acc = experiment(loaders, False, 1)
```

```
training
[ 1,  2000] risk over last 2000 mini-batches: 1.910
[ 1,  4000] risk over last 2000 mini-batches: 1.735
[ 1,  6000] risk over last 2000 mini-batches: 1.649
[ 1,  8000] risk over last 2000 mini-batches: 1.607
[ 1, 10000] risk over last 2000 mini-batches: 1.592
[ 1, 12000] risk over last 2000 mini-batches: 1.571
evaluating
training accuracy 47.25 percent
test accuracy 46.18 percent
```

The training accuracy of the model with convolution layers was 47.69%, and the test accuracy was 46.86%. For the model without convolution layers, the training accuracy was 47.25% and the test accuracy was 46.18%. Because of this, the model with convolutions gives better test accuracy by 0.68%, and generalizes better since its test accuracy is higher and its training accuracy is higher as well.

Problem 3.4 (Exam Style)

Yes, the model shows an overall tendency to overfit because as the epochs become larger, the training accuracy tends to increase while the validation accuracy tends to decrease. This means that the model is overfitting to the training data as we run more epochs and this overfitting is causing the validation accuracy to go down.

For best test-time performance, I would stop after 10 epochs because the validation accuracy is at its highest, of around 62%.

Problem 3.5 (Exam Style)

The statistical accuracy of this predictor is 10%, if it ignores its input and predicts a class drawn uniformly at random from all 10 possible classes.

The predictors in the previous problems all have a test/validation accuracy that is greater than 10%, so yes, they do perform better than the blind predictor.