Homework 1

Monday, January 25, 2021 12:08 AM

Problem 1: Backpropagation (25Pts)

Assume we have defined a loss function l, now for a given layer $\mathbf{y} = f(\mathbf{x}, \mathbf{W})$, write $\frac{\partial l}{\partial \mathbf{x}}$ and $\frac{\partial l}{\partial \mathbf{W}}$ as functions of $\frac{\partial l}{\partial \mathbf{y}}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{W}$ in the following cases:

a) $\mathbf{y} = \mathbf{x}\mathbf{W}, \mathbf{x} \in \mathbb{R}^{1\times n}, \mathbf{W} \in \mathbb{R}^{n\times m}$

$$\boxed{\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial l}{\partial y} W^T} \qquad \boxed{\frac{\partial l}{\partial W} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial W} = x^T \frac{\partial l}{\partial y}}$$

b) $\mathbf{y} = \mathbf{x}\mathbf{W}, \mathbf{x} \in \mathbb{C}^{1\times n}, \mathbf{W} \in \mathbb{C}^{n\times m}$

Pytorch reference on complex derivatives:

$$\frac{\partial l}{\partial x^*} = \left(\frac{\partial l}{\partial y^*}\right)^* \cdot \frac{\partial y}{\partial x^*} + \frac{\partial l}{\partial y^*}\left(\frac{\partial y}{\partial x}\right)^*$$

$X = a + jb \qquad a, b \in \mathbb{R}^n$

$$\frac{\partial y}{\partial x} = \frac{\partial (xW)}{\partial x} = W^T, \quad \frac{\partial y}{\partial x^*} = 0$$

$$\boxed{\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y^*} \cdot W^{*T}} \qquad \boxed{\frac{\partial l}{\partial W} = x^{*T}\frac{\partial l}{\partial y^*}}$$

c) $\mathbf{y} = \|\mathbf{x}\mathbf{W}\|_2^2, \mathbf{x} \in \mathbb{R}^{1\times n}, \mathbf{W} \in \mathbb{R}^{n\times m}$

$$\boxed{\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} 2(xW)W^T} \qquad \boxed{\frac{\partial l}{\partial W} = \frac{\partial l}{\partial y} 2x^T(xW)}$$

d) $\mathbf{y} = \mathbf{x} \odot \mathbf{w}, \mathbf{y}_k = \mathbf{x}_k\mathbf{w}_k$, $\mathbf{x}$ and $\mathbf{w} \in \mathbb{R}^n$

$$\boxed{\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y} \odot w} \qquad \boxed{\frac{\partial l}{\partial W} = \frac{\partial l}{\partial y} \odot x}$$

e) $\mathbf{y} = \mathbf{softmax}(\mathbf{x}), \mathbf{y}_k = \frac{e^{\mathbf{x}_k}}{Z(\mathbf{x})}, Z(\mathbf{x}) = \sum_{i=0}^{n-1} e^{\mathbf{x}_i}, \mathbf{x} \in \mathbb{R}^n$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial y}\frac{\partial y}{\partial x}$$

Split into case when $i = k$ & $i \neq k$

For $i = k$:

$$\frac{\partial Z(x)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{i=0}^{n-1} e^{x_i}\right) = e^{x_i}$$

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k}\frac{\partial y_k}{x_i} = \frac{\partial l}{\partial y_k}\left(\frac{z(x)e^{x_k} - e^{x_k}z'(x)}{(z(x))^2}\right)$$

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k}\frac{\partial y_k}{x_i} = \frac{\partial l}{\partial y_k}\left(\frac{z(x)e^{x_i} - e^{2x_i}}{(z(x))^2}\right)$$

$$= \frac{\partial l}{\partial y_k}\left(\frac{e^{x_i}}{Z(x)}\left[\frac{Z(x) - e^{x_i}}{Z(x)}\right]\right)$$

$$= \frac{\partial l}{\partial y_k}\left(\mathrm{Softmax}(x_i)\left[1 - \mathrm{Softmax}(x_i)\right]\right)$$

For $i \neq k$:

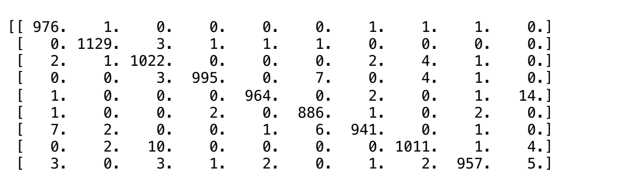
$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k}\frac{\partial y_k}{\partial x_i} = \frac{\partial l}{\partial y_k}\left(\frac{\cancel{z(x)\cdot 0} - e^{x_k}e^{x_i}}{(z(x))^2}\right)$$

$$\frac{\partial l}{\partial x_i} = \frac{\partial l}{\partial y_k}\left(\frac{-e^{x_k}e^{x_i}}{(z(x))^2}\right) = \frac{\partial l}{\partial y_k}\left(-\mathrm{Softmax}(x_k)\,\mathrm{Softmax}(x_i)\right)$$

Read through the sample script you downloaded. Train the model for 2 epochs, and save the weights. This should be done using command line arguments, without modifying the original file. **Include the bash command you used in your HW submission.**

Bash command: python3 main.py --epochs 2 --no-cuda --save-model

Image effects of different noise levels




```
Accuracy for AWGN with a standard deviation of 0

Test set: Average loss: 0.0354, Accuracy: 9872/10000 (99%)

[[ 976.    1.    0.    0.    0.    0.    1.    1.    1.    0.]
 [   0. 1129.    3.    1.    1.    1.    0.    0.    0.    0.]
 [   2.    1. 1022.    0.    0.    0.    2.    4.    1.    0.]
 [   0.    0.    3.  995.    0.    7.    0.    4.    1.    0.]
 [   1.    0.    0.    0.  964.    0.    2.    0.    1.   14.]
 [   1.    0.    0.    2.    0.  886.    1.    0.    2.    0.]
 [   7.    2.    0.    0.    1.    6.  941.    0.    1.    0.]
 [   0.    2.   10.    0.    0.    0.    0. 1011.    1.    4.]
 [   3.    0.    3.    1.    2.    0.    1.    2.  957.    5.]
 [   3.    1.    0.    0.    3.    5.    0.    2.    4.  991.]]

Accuracy for AWGN with a standard deviation of 0.3

Test set: Average loss: 0.2217, Accuracy: 9566/10000 (96%)

[[ 959.    0.    8.    3.    0.    1.    2.    0.    6.    1.]
 [   0. 1049.   63.    1.   11.    2.    0.    5.    4.    0.]
 [   1.    1. 1026.    0.    0.    0.    1.    0.    3.    0.]
 [   0.    0.   15.  960.    1.   20.    0.    3.    9.    2.]
 [   0.    0.    9.    0.  959.    0.    1.    1.    5.    7.]
 [   1.    0.    0.    8.    0.  858.    5.    1.   17.    2.]
 [   4.    0.   10.    1.    3.    7.  914.    0.   19.    0.]
 [   1.    3.   59.    0.    2.    2.    0.  947.    4.   10.]
 [   1.    1.   10.    0.    0.    2.    1.    2.  956.    1.]
 [   2.    1.    2.    2.   21.    7.    1.    6.   29.  938.]]

Accuracy for AWGN with a standard deviation of 0.6

Test set: Average loss: 1.3472, Accuracy: 5098/10000 (51%)

[[443.   0. 282.  14.   8.  29.  16.   0. 178.  10.]
 [  0. 128. 637.  23.  58.  37.   6.  29. 217.   0.]
 [  0.   1. 915.  10.   5.   5.   3.   3.  87.   3.]
 [  0.   1. 163. 524.   5.  63.   1.   5. 242.   6.]
 [  0.   0. 114.  10. 636.  17.   5.   5. 161.  34.]
 [  1.   0.  74.  45.  14. 428.   7.   1. 314.   8.]
 [  2.   2. 165.   2.  38.  43. 468.   3. 232.   3.]
 [  1.   5. 363.  37.  70.  48.   2. 375.  82.  45.]
 [  1.   0.  92.  12.   4.   5.   0.   1. 859.   0.]
 [  1.   0. 106.  22. 159.  28.   4.  20. 347. 322.]]

Accuracy for AWGN with a standard deviation of 1.0

Test set: Average loss: 2.2629, Accuracy: 2164/10000 (22%)

[[ 62.   2. 359.  18.   9.  62.  10.   0. 438.  20.]
 [  1.   5. 525.  54.  30.  41.   3.  10. 464.   2.]
 [  1.   1. 669.  19.  12.  13.   4.   2. 307.   4.]
 [  3.   0. 251. 174.   9.  72.   1.   3. 487.  10.]
 [  0.   0. 253.  18. 169.  42.   9.   3. 472.  16.]
 [  6.   0. 170.  34.  25. 152.  10.   1. 479.  15.]
 [  3.   0. 286.   5.  36.  50. 104.   0. 471.   3.]
 [  3.   0. 453.  57.  62.  75.   3.  55. 292.  28.]
 [  0.   1. 219.  18.   6.  15.   2.   0. 712.   1.]
 [  1.   0. 204.  29.  83.  46.   2.   7. 575.  62.]]
```

Code:

The matplotlib code for showing the images with various AWGN and the computations for the confusion matrix are on main.py. Setting up and loading the model is in test_AWGN.py.

File - /Users/ivanarevalomac/Google Drive/School/Machine Learning/ECE281 Computer Vision/Homeworks/Homework1/image_clas

```python
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR


class AWGN(object):
    def __init__(self, sigma, vmin=0, vmax=1):
        assert isinstance(sigma, (int, float))
        self.sigma = sigma
        self.vmin = vmin
        self.vmax = vmax

    def __call__(self, image):
        noise = torch.from_numpy(np.random.normal(0.0, self.sigma, image.shape).astype(np.float32))
        # Above supports std of 0

        # noise = torch.normal(0.0, self.sigma, image.shape)
        output = image + noise
        torch.clip(output, self.vmin, self.vmax, out=output)

        return output


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
```

Page 1 of 4

File - /Users/ivanarevalomac/Google Drive/School/Machine Learning/ECE281 Computer Vision/Homeworks/Homework1/image_clas

```python
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    confusion_matrix = np.zeros((10,10))
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability

            indices = np.array([target.numpy().squeeze(), pred.numpy().squeeze()]).transpose()
            for i in indices:
                confusion_matrix[i[0],i[1]] += 1

            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


    # suppress: suppress scientific notation
    with np.printoptions(precision=3, suppress=True):
        print(np.array(confusion_matrix))


def main():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
```

Page 2 of 4

File - /Users/ivanarevalomac/Google Drive/School/Machine Learning/ECE281 Computer Vision/Homeworks/Homework1/image_clas

```python
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=False,
                        help='For Saving the current Model')
    parser.add_argument('--test-AWGN', action='store_true', default=False,
                        help='Test AWGN transformation')
    parser.add_argument('--sigma', type=float, default=0.0, metavar='N',
                        help='standard deviation of AWGN')
    args = parser.parse_args()
    use_cuda = not args.no_cuda and torch.cuda.is_available()

    torch.manual_seed(args.seed)

    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}

    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)


    if args.test_AWGN:
        transform = transforms.Compose([transforms.ToTensor()])

        test_dataset  = datasets.MNIST('../data', train=False, transform=transform)

        ##DEBUG STEP
        print(type(test_dataset[0][0]))
        print(test_dataset[0][0].shape)
        print(type(test_dataset[0][0].numpy()))
        print(test_dataset[0][0].numpy().shape)

        # Display effects of AWGN on first image
        fig = plt.figure()
        image = test_dataset[0][0]
        print(image.type())

        for i, sigma in enumerate([0.001, 0.3, 0.6, 1.0]):
            add_noise = AWGN(sigma)
            transformed_sample = add_noise(image)
            # print(transformed_sample)

            ax = plt.subplot(1, 4, i + 1)
            plt.tight_layout()
            plt.axis('off')

            ax.set_title("$\sigma = ${}".format(sigma))
            plt.imshow(transformed_sample.numpy().squeeze(), cmap='gray')

        fig.suptitle("Image effects of different noise levels")
        plt.show()


        #

        # test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)
    else:
```

Page 3 of 4

File - /Users/ivanarevalomac/Google Drive/School/Machine Learning/ECE281 Computer Vision/Homeworks/Homework1/image_clas

```python
        transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,))
            ])


        dataset1 = datasets.MNIST('../data', train=True, download=True,
                           transform=transform)
        dataset2 = datasets.MNIST('../data', train=False,
                           transform=transform)
        train_loader = torch.utils.data.DataLoader(dataset1,**train_kwargs)
        test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

        model = Net().to(device)
        optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

        scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
        for epoch in range(1, args.epochs + 1):
            train(args, model, device, train_loader, optimizer, epoch)
            test(model, device, test_loader)
            scheduler.step()

        if args.save_model:
            torch.save(model.state_dict(), "mnist_cnn.pt")


if __name__ == '__main__':
    main()
```

Page 4 of 4

File - /Users/ivanarevalomac/Google Drive/School/Machine Learning/ECE281 Computer Vision/Homeworks/Homework1/image_clas

```python
from __future__ import print_function
import argparse
import torch
import main
from torchvision import datasets, transforms



def run():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--no-cuda', action='store_true', default=True,
                        help='disables CUDA training')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--test-AWGN', action='store_true', default=False,
                        help='Test AWGN transformation')
    parser.add_argument('--sigma', type=float, default=0.0, metavar='N',
                        help='standard deviation of AWGN')
    args = parser.parse_args()

    use_cuda = not args.no_cuda and torch.cuda.is_available()

    torch.manual_seed(args.seed)

    device = torch.device("cuda" if use_cuda else "cpu")

    test_kwargs = {'batch_size': args.test_batch_size}

    ## import model from main and import weights
    model = main.Net().to(device)
    model.load_state_dict(torch.load("mnist_cnn.pt"))
    # Print model's state_dict
    print("Model's state_dict:")
    for param_tensor in model.state_dict():
        print(param_tensor, "\t", model.state_dict()[param_tensor].size())

    for i, sigma in enumerate([0, 0.3, 0.6, 1.0]):
        transform = transforms.Compose([transforms.ToTensor(), main.AWGN(sigma), transforms.Normalize((0.1307,), (0.3081,))])

        test_dataset = datasets.MNIST('../data', train=False, transform=transform)

        test_loader = torch.utils.data.DataLoader(test_dataset, **test_kwargs)

        print(f"\nAccuracy for AWGN with a standard deviation of {sigma}")
        main.test(model, device, test_loader)


if __name__ == '__main__':
    run()
```

Page 1 of 1