CSE 546 HW #4

Sam Kowash

December 5, 2018

Acknowledgments: I collaborated with Tyler Blanton and Michael Ross.

1 Expectation maximization

1. Suppose that we have a set of feature vectors $x_1, \ldots, x_n \in \mathbb{R}^d$, where each vector represents a song, and a sample set $\mathcal{S} \subset \{1, \ldots, n\}$ of listen counts $Y_i \in \mathbb{Z}^+$ for a given user. We assume $Y_i \sim \text{Poisson}(\lambda_i)$, where $\lambda_i \equiv \mathbb{E}[Y_i \mid x_i]$ and we assume a model $\lambda_i = \exp(w^T x_i)$ for some $w \in \mathbb{R}^d$.

   (a) The MLE estimator for this model is

   $$\hat{w} = \arg\max_w \prod_{i \in \mathcal{S}} \frac{\exp(y_i x_i^T w)}{y_i!} \exp\left[-e^{w^T x_i}\right].$$

   There is no closed-form solution for $\hat{w}$, but observe that

   $$\arg\max_w \prod_{i \in \mathcal{S}} \frac{\exp(y_i x_i^T w)}{y_i!} \exp\left[-e^{w^T x_i}\right] = \arg\max_w \exp\left\{-\sum_{i \in \mathcal{S}} \left[e^{x_i^T w} - y_i x_i^T w\right]\right\},$$

   and the exponential is maximized when the negative of its exponent is minimized, so

   $$\hat{w} = \arg\min_w \sum_{i \in \mathcal{S}} \left[e^{x_i^T w} - y_i x_i^T w\right].$$

   Note that the Hessian of each term in the sum is

   $$\nabla_w^2 \hat{w}_i = x_i x_i^T,$$

   which we showed in HW 0 is PSD. The Hessian of the whole objective is then a sum of PSD matrices which is itself PSD, and since a function with PSD Hessian is convex, finding $\hat{w}$ is a convex optimization problem. We can solve for it with a number of methods, including, for example, SGD and Newton's method.

   (b) Suppose we now determine that each song x_i belongs to one of k unknown moods, each of which should have its own weight vector, and want to estimate these clusters from the observed data.

2 Regression with side information

2. We implement kernel regression with the rbf kernel via `cvxpy` on $n = 50$ points y_i drawn from the graph of

   $$f(x) = 10 \sum_{k=1}^{4} \mathbf{1}\{x \geqslant \frac{k}{5}\}$$

   on $[0, 1]$ with Gaussian noise $\epsilon_i \sim \mathcal{N}(0, 1)$ applied to each point x_i and one deliberate outlier $y_{25} = 0$.

Throughout, we use the rbf kernel,

$$k(x_1, x_2) = \exp(-\gamma|x_1 - x_2|^2),$$

where $\gamma > 0$ is a hyperparameter, and the estimator

$$\hat{f}(x) = \sum_{i=1}^{n} \hat{\alpha}_i k(x_i, x), \tag{2.1}$$

where α is the weight vector to optimize.

(a) We first optimize the usual regularized least-squares objective

$$J(\alpha) = \sum_{i=1}^{n} \ell_{\text{LS}} \left(y_i - \sum_{j=1}^{n} K_{ij}\alpha_j \right) + \lambda \alpha^T K \alpha,$$

where $K_{ij} = k(x_i, x_j)$. We used LOOCV over the data points to evaluate randomly-selected hyperparameters (λ, γ), producing the loss contours in Fig. 2.1.

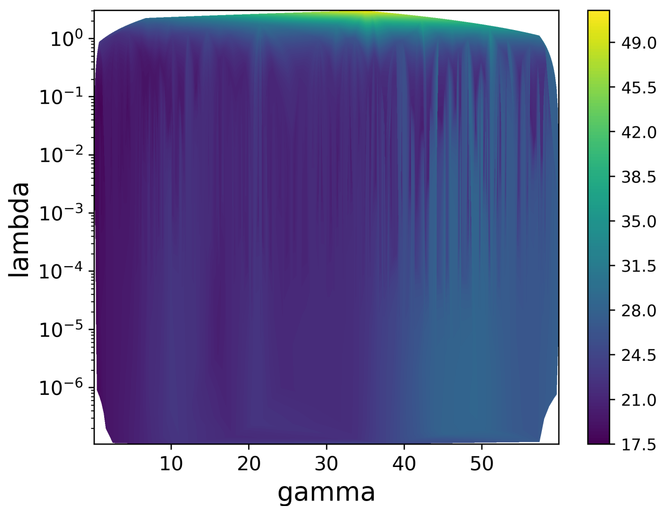

Figure 2.1: LOOCV loss contours for random values of (λ, γ)

From this, we estimated $\gamma = 16$ and $\lambda = 1.5 \times 10^{-5}$ as reasonable hyperparameters (although the loss surface appears very flat in large regions of the parameter space).

(b) We next considered optimizing the Huber loss summed over the data set. A similar LOOCV procedure produced figure

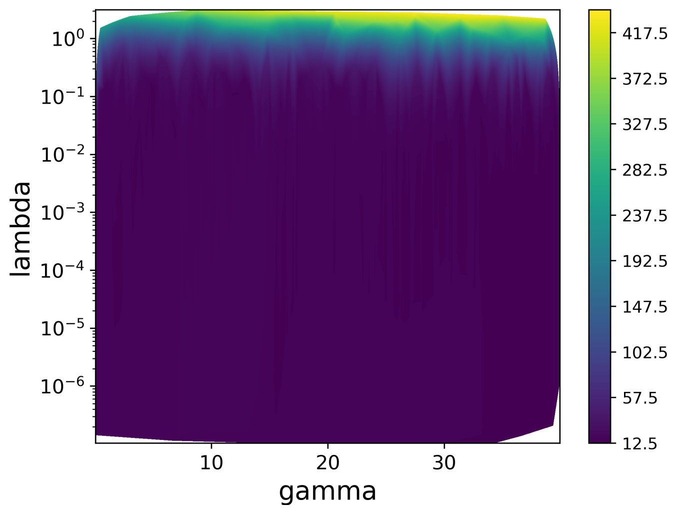

Figure 2.2: LOOCV loss contours for random values of (λ, γ)

This is, if anything, harder to interpret visually, but with a well-scaled set of contours (not shown), we determined $\gamma = 2$ and $\lambda = 10^{-4}$ as appropriate hyperparameter values.

3 Deep learning architectures

3.

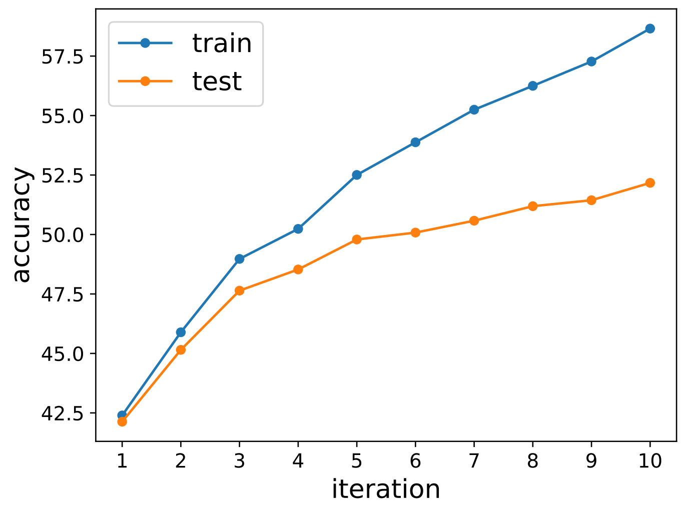

Figure 3.1: Accuracy curves for NN with single fully-connected hidden layer; learning rate of 5×10^{-4}, momentum of 0.5, and $M = 150$

4.

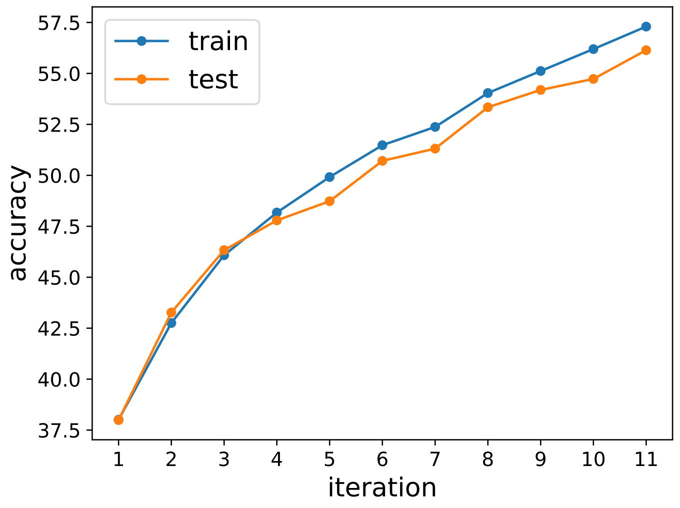

Figure 3.2: Accuracy curves for NN with convolution layer and MaxPool; learning rate of 5×10^{-4}, momentum of 0.7, $M = 100$, $p = 6$, and $N = 9$

5.

```
#!/usr/bin/env python
import numpy as np
import cvxpy as cvx
from datetime import datetime
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

def f(x):
    if type(x) is np.ndarray:
        y = np.zeros_like(x)
        for i in range(4):
            y[np.where(x>=(i+1)/5)] += 1
    else:
        y = 0
        for i in range(4):
            if x >= (i+1)/5:
                y += 1

    return 10*y

def k_scal(x,y,gamma):
    return np.exp(-gamma*(x-y)**2)

def k_matrix(x,gamma):
    n = len(x)
    k = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            k[i,j] = np.exp(-gamma*(x[i]-x[j])**2)
    return k

def predict(x,alpha,gamma): #single scalars at a time
    return lambda y: np.dot(alpha, k_scal(x,y,gamma))

n = 50
x = np.arange(n)/(n-1)
y = f(x) + np.random.randn(n)
y[24] = 0

gams = np.random.uniform(0,60,500)
lambs = np.power(10,np.random.uniform(-7,0.5,500))
params = np.stack((gams,lambs)).T
errs = np.zeros(len(params))



for i in range(len(params)):
    print(f"On␣parameter␣set␣{i+1}")
    for omit in range(n):
        x_loo = np.concatenate((x[:omit],x[omit+1:]))
        y_loo = np.concatenate((y[:omit],y[omit+1:]))
        gamma = params[i,0]
        lambd = params[i,1]

        alpha = cvx.Variable(n-1)

        k_mat = k_matrix(x_loo,gamma)
        objective = cvx.Minimize(cvx.sum_squares(y_loo - k_mat*alpha)
                                 + lambd*cvx.quad_form(alpha,k_mat))

        prob = cvx.Problem(objective)

        prob.solve()

        alpha_sol = alpha.value
        pred = predict(x_loo, alpha_sol, gamma)

        errs[i] += (y[omit] - pred(x[omit]))**2

    errs[i] /= n


ftime = datetime.now().time()
stamp = f"{ftime.hour:02d}_{ftime.minute:02d}_{ftime.second:02d}"
with open(f"../data/2a-{stamp}",'w') as f:
    f.writelines([f"{params[i,0]:12.2f}␣{params[i,1]:12.2e}␣{errs[i]:12.4f}\n"
                  for i in range(len(params))])
```

```python
#!/usr/bin/env python
import numpy as np
import cvxpy as cvx
from datetime import datetime
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

def f(x):
    if type(x) is np.ndarray:
        y = np.zeros_like(x)
        for i in range(4):
            y[np.where(x>=(i+1)/5)] += 1
    else:
        y = 0
        for i in range(4):
            if x >= (i+1)/5:
                y += 1

    return 10*y

def k_scal(x,y,gamma):
    return np.exp(-gamma*(x-y)**2)

def k_matrix(x,gamma):
    n = len(x)
    k = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            k[i,j] = np.exp(-gamma*(x[i]-x[j])**2)
    return k

def predict(x,alpha,gamma): #single scalars at a time
    return lambda y: np.dot(alpha, k_scal(x,y,gamma))

n = 50
x = np.arange(n)/(n-1)
y = f(x) + np.random.randn(n)
y[24] = 0

gams = np.random.uniform(0,40,500)
lambs = np.power(10,np.random.uniform(-7,0.5,500))

params = np.stack((gams,lambs)).T
errs = np.zeros(len(params))



for i in range(len(params)):
    print(f"On␣parameter␣set␣{i+1}")
    for omit in range(n):
        x_loo = np.concatenate((x[:omit],x[omit+1:]))
        y_loo = np.concatenate((y[:omit],y[omit+1:]))
        gamma = params[i,0]
        lambd = params[i,1]

        alpha = cvx.Variable(n-1)

        k_mat = k_matrix(x_loo,gamma)
        objective = cvx.Minimize(cvx.sum(cvx.huber(y_loo - k_mat*alpha))
                                 + lambd*cvx.quad_form(alpha,k_mat))

        prob = cvx.Problem(objective)

        prob.solve()

        alpha_sol = alpha.value
        pred = predict(x_loo, alpha_sol, gamma)

        errs[i] += (y[omit] - pred(x[omit]))**2

    errs[i] /= n


ftime = datetime.now().time()
stamp = f"{ftime.hour:02d}_{ftime.minute:02d}_{ftime.second:02d}"
with open(f"../data/2b-{stamp}",'w') as f:
    f.writelines([f"{params[i,0]:12.2f}␣{params[i,1]:12.2e}␣{errs[i]:12.4f}\n"
                  for i in range(len(params))])
```

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms


def imshow(img):
    img = img/2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)))
    plt.show()

def train(net, trainloader, testloader, calc_acc, criterion, optimizer, rate,
          momentum, max_epochs=10,acc_file=None):
    splits = np.zeros(max_epochs+1)
    splits[0] = time.time()

    for epoch in range(max_epochs):

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data

            optimizer.zero_grad()

            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            if i % 2000 == 1999:
                print(f'[{epoch+1:d},␣{i+1:5d}]␣loss:␣{running_loss/2000:.3f}')

                running_loss = 0.

        splits[epoch+1] = time.time()
        print(f"Finished␣epoch␣{epoch}␣in␣{splits[epoch+1]-splits[epoch]:.0f}")
        print(f"Total␣time:␣{splits[epoch+1]-splits[0]:.0f}")

        if calc_acc:
            train_acc = accuracy(trainloader,net)
            test_acc = accuracy(testloader,net)

            with open(acc_file,'a') as f:
                f.write(f"{train_acc:10.4f}{test_acc:10.4f}\n")

            print(f"After␣epoch␣{epoch+1}:")
            print(f"Training␣accuracy:␣{train_acc:.2f}")
            print(f"Test␣accuracy:␣{test_acc:.2f}")
    print("Done␣training!")

def accuracy(loader,net):
    correct = 0
    total = 0

    with torch.no_grad():
        for data in loader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100*correct/total

################################################################################

class Net(nn.Module):
    def  __init__(self):
        super(Net, self).__init__()
        self.fc = nn.Linear(32*32*3,10)

    def forward(self, x):
        x = x.view(-1, self.num_flat_features(x))
        x = self.fc(x)

        return x

    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

################################################################################
# arg processing and filename init                                             #
################################################################################

n_epochs = int(sys.argv[1])
n_workers = int(sys.argv[2])
rate = float(sys.argv[3])
momentum = float(sys.argv[4])
calc_acc = {'y':True, 'n':False}[sys.argv[5]]

total_file = f"../data/3a_parameter_search"

if calc_acc:
    acc_file = f"../data/3a_r{100000*rate:.0f}_p{100000*momentum:.0f}_accs"
else:
    acc_file = None

################################################################################
# load and normalize data                                                      #
################################################################################

transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5),
                                                     (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='../data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=n_workers)


testset = torchvision.datasets.CIFAR10(root='../data', train=False,
                                        download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                          shuffle=True, num_workers=n_workers)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
           'ship', 'truck')

###############################################################################

net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=rate, momentum=momentum)

train(net, trainloader, testloader, calc_acc,
      criterion, optimizer,
      rate, momentum, n_epochs, acc_file,)

test_acc = accuracy(testloader,net)
with open(total_file,'a') as f:
    f.write(f"{rate:16.4e}{momentum:16.4e}{test_acc:10.4f}\n")
print(f"Final␣test␣acc:␣{test_acc:10.4f}")
```

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import sys
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms


def imshow(img):
    img = img/2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)))
    plt.show()

def train(net, trainloader, testloader, calc_acc, criterion, optimizer, rate,
          momentum, max_epochs=10,acc_file=None):

    for epoch in range(max_epochs):

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data

            optimizer.zero_grad()

            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            if i % 2000 == 1999:
                print(f'[{epoch+1:d},␣{i+1:5d}]␣loss:␣{running_loss/2000:.3f}')

                running_loss = 0.

        if calc_acc:
            train_acc = accuracy(trainloader,net)
            test_acc = accuracy(testloader,net)

            with open(acc_file,'a') as f:
                f.write(f"{train_acc:10.4f}{test_acc:10.4f}\n")

            print(f"After␣epoch␣{epoch+1}:")
            print(f"Training␣accuracy:␣{train_acc:.2f}")
            print(f"Test␣accuracy:␣{test_acc:.2f}")

    print("Done␣training!")

def accuracy(loader,net):
    correct = 0
    total = 0

    with torch.no_grad():
        for data in loader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100*correct/total

################################################################################

class Net(nn.Module):
    def  __init__(self,M=10):
        super(Net, self).__init__()
        self.M = M
        self.fc1 = nn.Linear(32*32*3,self.M)
        self.fc2 = nn.Linear(M,10)

    def forward(self, x):
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x

    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

################################################################################
# arg processing and filename init                                             #
################################################################################

n_epochs = int(sys.argv[1])
n_workers = int(sys.argv[2])
rate = float(sys.argv[3])
momentum = float(sys.argv[4])
calc_acc = {'y':True, 'n':False}[sys.argv[5]]
M = int(sys.argv[6])

total_file = f"../data/3b_parameter_search"

if calc_acc:
    acc_file = f"../data/3b_r{100000*rate:.0f}_p{100000*momentum:.0f}_accs"
else:
    acc_file = None

################################################################################
# load and normalize data                                                      #
################################################################################

transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5),
                                                    (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='../data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=n_workers)


testset = torchvision.datasets.CIFAR10(root='../data', train=False,
                                        download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                          shuffle=True, num_workers=n_workers)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
           'ship', 'truck')

###############################################################################

net = Net(M)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=rate, momentum=momentum)


train(net, trainloader, testloader, calc_acc,
      criterion, optimizer,
      rate, momentum, n_epochs, acc_file,)

test_acc = accuracy(testloader,net)
with open(total_file,'a') as f:
    f.write(f"{rate:16.4e}{momentum:16.4e}{test_acc:10.4f}\n")
print(f"Final␣test␣acc:␣{test_acc:10.4f}")
```

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms


def imshow(img):
    img = img/2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)))
    plt.show()

def train(net, trainloader, testloader, calc_acc, criterion, optimizer,
          max_epochs=10,acc_file=None):

    splits = np.zeros(max_epochs+1)
    splits[0] = time.time()

    for epoch in range(max_epochs):

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data

            optimizer.zero_grad()

            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            if i % 2000 == 1999:
                print(f'[{epoch+1:d},␣{i+1:5d}]␣loss:␣{running_loss/2000:.3f}')

                running_loss = 0.

        splits[epoch+1] = time.time()
        print(f"Finished␣epoch␣{epoch}␣in␣{splits[epoch+1]-splits[epoch]:.0f}")
        print(f"Total␣time:␣{splits[epoch+1]-splits[0]:.0f}")

        if calc_acc:
            train_acc = accuracy(trainloader,net)
            test_acc = accuracy(testloader,net)

            with open(acc_file,'a') as f:
                f.write(f"{train_acc:10.4f}{test_acc:10.4f}\n")

            print(f"After␣epoch␣{epoch+1}:")
            print(f"Training␣accuracy:␣{train_acc:.2f}")
            print(f"Test␣accuracy:␣{test_acc:.2f}")
    print("Done␣training!")

def accuracy(loader,net):
    correct = 0
    total = 0

    with torch.no_grad():
        for data in loader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return 100*correct/total

###############################################################################

class Net(nn.Module):
    def  __init__(self,M=10,p=5,N=14):
        super(Net, self).__init__()
        self.M = M
        self.p = p
        self.N = N
        self.E = int(np.floor((33-self.p)/self.N))

        self.conv = nn.Conv2d(3, self.M, self.p)
        self.pool = nn.MaxPool2d(N)
        self.fc = nn.Linear(self.M*self.E**2, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = x.view(-1,self.num_flat_features(x))
        x = self.fc(x)

        return x

    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

###############################################################################
# arg processing and filename init                                            #
###############################################################################

n_epochs = int(sys.argv[1])
n_workers = int(sys.argv[2])
rate = float(sys.argv[3])
momentum = float(sys.argv[4])
calc_acc = {'y':True, 'n':False}[sys.argv[5]]
M = int(sys.argv[6])
p = int(sys.argv[7])
N = int(sys.argv[8])

total_file = f"../data/3c_parameter_search"
```

```
if calc_acc:
    rate_str = f"{100000*rate:.0f}"
    mom_str = f"{100000*momentum:.0f}"
    acc_file = f"../data/3c_r{rate_str}_p{mom_str}_M{M:d}_N{N:d}_accs"
else:
    acc_file = None

################################################################################
# load and normalize data                                                      #
################################################################################

transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5),
                                                    (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='../data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=n_workers)


testset = torchvision.datasets.CIFAR10(root='../data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=True, num_workers=n_workers)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
           'ship', 'truck')

################################################################################

net = Net(M,p,N)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=rate, momentum=momentum)


train(net, trainloader, testloader, calc_acc,
      criterion, optimizer, n_epochs, acc_file)

test_acc = accuracy(testloader,net)
with open(total_file,'a') as f:
    f.write(f"{rate:16.4e}{momentum:16.4e}{M:d}{p:d}{N:d}{test_acc:10.4f}\n")
print(f"Final␣test␣acc:␣{test_acc:10.4f}")
```

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt




#2a
data = np.genfromtxt('../data/2a-23_14_03')
fig = plt.figure()
ax = fig.add_subplot(111)
cf = ax.tricontourf(data[:,0],data[:,1],data[:,2],150)
plt.colorbar(cf)

plt.xlabel('gamma', size=16)
plt.ylabel('lambda', size=16)

plt.yscale('log')


ax.tick_params(axis='both',labelsize=12)
plt.savefig('../figures/2a_loocv.png',dpi=400,bbox_inches=0)
plt.cla()
plt.clf()











#2b
data = np.genfromtxt('../data/2b-23_35_41')
fig = plt.figure()
ax = fig.add_subplot(111)
cf = ax.tricontourf(data[:,0],data[:,1],data[:,2],200)
plt.colorbar(cf)

plt.xlabel('gamma', size=16)
plt.ylabel('lambda', size=16)

plt.yscale('log')


ax.tick_params(axis='both',labelsize=12)
plt.savefig('../figures/2b_loocv.png',dpi=400,bbox_inches=0)
plt.cla()
plt.clf()
```

```
#2c
```

```
#3a
```

```
#3b
fig = plt.figure()
ax = fig.add_subplot(111)
data = np.genfromtxt('../data/3b_r50_p50000_accs')

ns = np.arange(1,len(data)+1)
ax.plot(ns,data[:,0],'o-',ms=5,label='train')
ax.plot(ns,data[:,1],'o-',ms=5,label='test')
ax.set_xlabel('iteration', size=16)
ax.set_ylabel('accuracy',size=16)
ax.set_xticks(ns)
ax.legend(fontsize=16)
ax.tick_params(axis='both',labelsize=12)
```

```
plt.savefig('../figures/3b_acc.pdf',bbox_inches=0)
plt.cla()
plt.clf()




#3c
fig = plt.figure()
ax = fig.add_subplot(111)
data = np.genfromtxt('../data/3c_r10_p70000_M100_N9_accs')

ns = np.arange(1,len(data)+1)
ax.plot(ns,data[:,0],'o-',ms=5,label='train')
ax.plot(ns,data[:,1],'o-',ms=5,label='test')
ax.set_xlabel('iteration', size=16)
ax.set_ylabel('accuracy',size=16)
ax.set_xticks(ns)
ax.legend(fontsize=16)
ax.tick_params(axis='both',labelsize=12)

plt.savefig('../figures/3c_acc.pdf',bbox_inches=0)
plt.cla()
plt.clf()
```