

HW3

October 26, 2018

1 Li Xingxuan 1002189

1.1 Problem 1: Matrix Factorization

In [2]: `import numpy as np`

```
In [3]: def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - np.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = np.dot(P,Q)
        e = 0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - np.dot(P[i,:],Q[:,j]), 2)
                    for k in range(K):
                        e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        if e < 0.001:
            break
    return P, Q.T
```

In [4]: `R = [`

```
[0,1,0],
[1,0,1],
[0,1,2],
]
```

`R = np.array(R)`

```
N = len(R)
M = len(R[0])
```

```

K = 1

P = np.random.rand(N,K)
Q = np.random.rand(M,K)

nP, nQ = matrix_factorization(R, P, Q, K)
print('A')
print(np.dot(nP, nQ.T))
print('U')
print(nP)
print('V')
print(nQ)

A
[[0.78455587 0.76207664 1.21396174]
 [0.75593508 0.7342759  1.16967612]
 [1.20583374 1.17128399 1.86581489]]

U
[[0.90011482]
 [0.86727842]
 [1.38344364]]

V
[[0.87161754]
 [0.84664381]
 [1.34867431]]

```

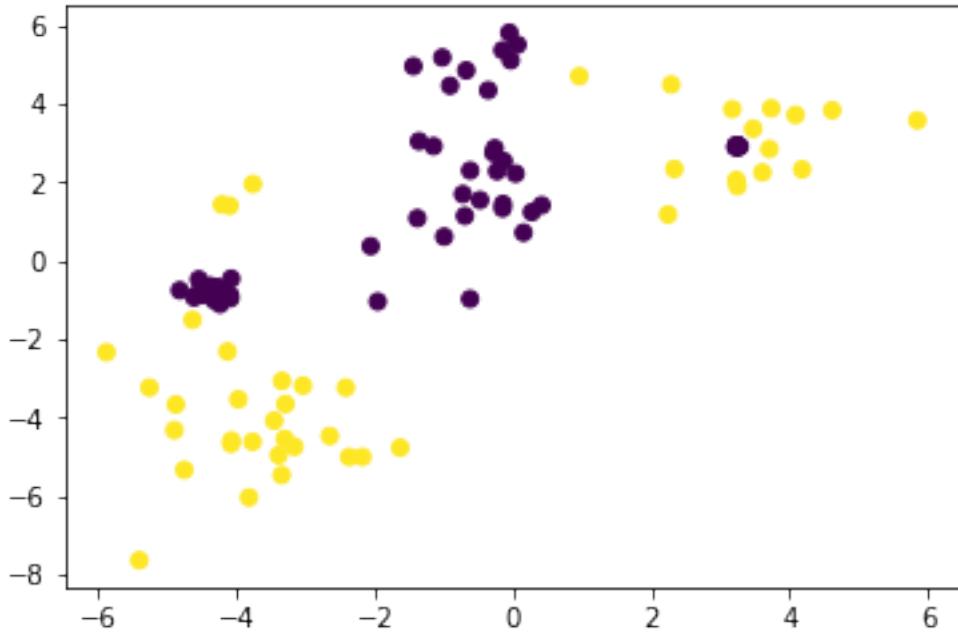
In [82]: # Reference: <http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial>

1.2 Problem 2: Support Vector Machines

```

In [34]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
csv = 'https://www.dropbox.com/s/wt45tvn9ig3o7vu/kernel.csv?dl=1'
data = np.genfromtxt(csv, delimiter=',')
X = data[:, 1:]
Y = data[:, 0]
plt.scatter(X[:,0], X[:,1], c=Y)
plt.show()

```



1.2.1 (a)

```
In [35]: from sklearn.svm import SVC
        clf = SVC(gamma=0.5, kernel='rbf')
        clf.fit(X, Y)

Out[35]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma=0.5, kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

1.2.2 (b)

```
In [75]: def decision(x1, x2, clf):
          x = np.array([x1,x2])
          value = clf.decision_function([x])
          return value

# decision(1,2,clf)
```

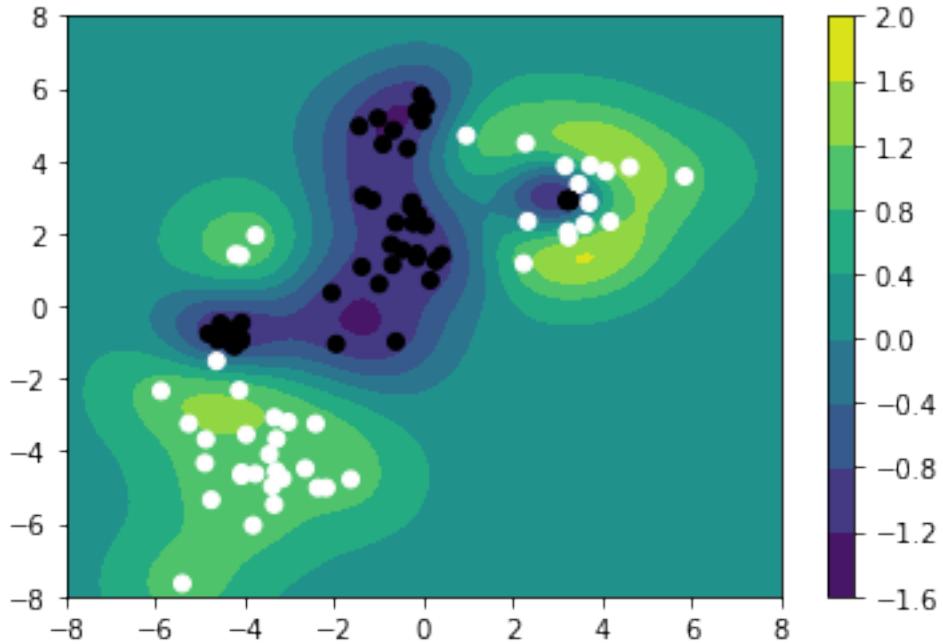
1.2.3 (c)

```
In [37]: vdecision = np.vectorize(decision, excluded=[2])
        x1list = np.linspace(-8.0,8.0,100)
        x2list = np.linspace(-8.0,8.0,100)
        X1, X2 = np.meshgrid(x1list,x2list)
```

```

Z = vdecision(X1,X2,clf)
cp = plt.contourf(X1,X2,Z)
plt.colorbar(cp)
plt.scatter(X[:,0],X[:,1],c=Y,cmap='gray')
plt.show()

```



1.3 Problem 3: Deep Learning

```

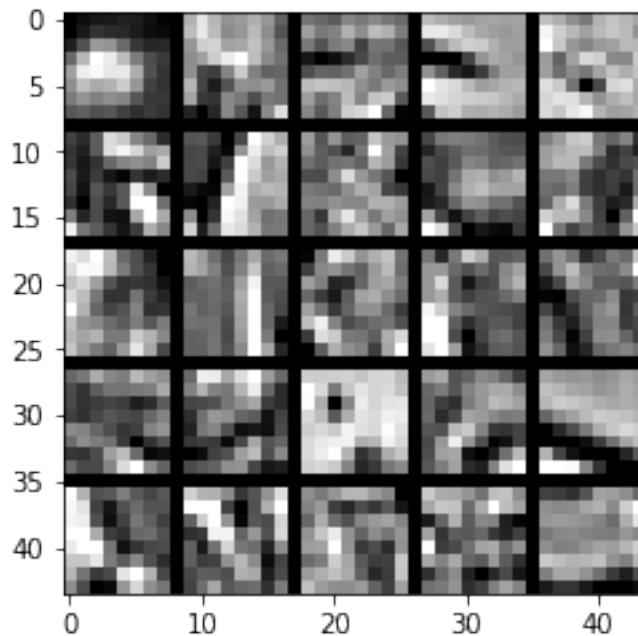
In [57]: %matplotlib inline
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from scipy.optimize import fmin_l_bfgs_b as minimize
from utils import normalize, tile_raster_images, sigmoid
from utils import ravelParameters, unravelParameters
from utils import initializeParameters
from utils import computeNumericalGradient
nV = 8*8
nH = 25
dW = 0.0001
sW = 3
# number of visible units
# number of hidden units
# weight decay term
# sparsity penalty term

```

```

npy = 'images.npy'
X = normalize(np.load(npy))
plt.imshow(tile_raster_images(X=X,
    img_shape=(8,8),tile_shape=(5,5),
    tile_spacing=(1,1)),cmap='gray')
plt.show()

```



1.3.1 (a)

```

In [83]: def sparseAutoencoderCost(theta,nV,nH,dW,sW,X):
    W1,W2,b1,b2 = unravelParameters(theta,nH,nV)
    n = X.shape[0]
    one = np.ones((n,1))

    z2 = np.dot(X,W1) + np.dot(one,(b1.T))
    a2 = sigmoid(z2)
    z3 = np.dot(a2,W2) + np.dot(one,(b2.T))
    a3 = sigmoid(z3)

    eps = a3-X
    loss = (norm(eps)**2)/(2*n)
    decay = (norm(W1)**2 + norm(W2)**2)/2

    # Compute sparsity terms and total cost
    rho = 0.01
    a2mean = np.mean(a2,axis=0).reshape(nH,1)

```

```

k1 = np.sum(rho*np.log(rho/a2mean)+\
            (1-rho)*np.log((1-rho)/(1-a2mean)))
dkl = -rho/a2mean+(1-rho)/(1-a2mean)
cost = loss+dW*decay+sW*k1

d3 = eps*a3*(1-a3)
d2 = (sW*dkl.T+np.dot(d3,W2.T))*a2*(1-a2)
W1grad = np.dot(X.T, d2)/n + dW*W1
W2grad = np.dot(a2.T, d3)/n + dW*W2
b1grad = np.dot(d2.T,one)/n
b2grad = np.dot(d3.T,one)/n

grad = ravelParameters(W1grad,W2grad,b1grad,b2grad)
print(' .',end="")
return cost,grad

```

In [85]: theta = initializeParameters(nH,nV)
cost,grad = sparseAutoencoderCost(theta,nV,nH,dW,sW,X)
print(cost, grad)

```

.49.04830250632397 [ 0.77057418  0.81956205  0.78870559 ... -0.0053743   0.00737526
-0.03224674]

```

1.3.2 (b)

In [66]: print('\nComparing numerical gradient with backdrop gradient')
num_coords = 5
indices = np.random.choice(theta.size,num_coords,replace=False)
numgrad = computeNumericalGradient(lambda t:
sparseAutoencoderCost(t,nV,nH,dW,sW,X)[0],theta,indices)
subnumgrad = numgrad[indices]
subgrad = grad[indices]
diff = norm(subnumgrad-subgrad)/norm(subnumgrad+subgrad)
print('\n',np.array([subnumgrad,subgrad]).T)
print('Relative difference: ', diff)

if diff < 10**(-9):
 print("Pass")
else:
 print("Too large")

```

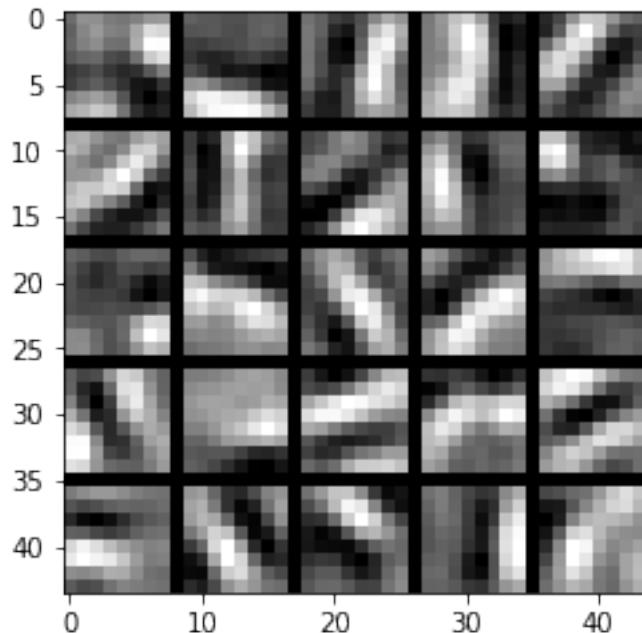
Comparing numerical gradient with backdrop gradient
. . . . .
[[ 0.65180611  0.65180611]
 [ 0.51784787  0.51784787]
 [ 0.53142294  0.53142294]
 [ 0.92930343  0.92930343]
```

```
[-0.01449782 -0.01449782]]  
Relative difference: 1.3491014434412139e-10  
Pass
```

1.3.3 (c)

```
In [69]: print('\nTraining neural network')  
theta = initializeParameters(nH,nV)  
opttheta,cost,messages = minimize(sparseAutoencoderCost,  
        theta,fprime=None,maxiter=400,args=(nV,nH,dW,sW,X))  
W1,W2,b1,b2 = unravelParameters(opttheta,nH,nV)  
plt.imshow(tile_raster_images(X=W1.T,  
        img_shape=(8,8),tile_shape=(5,5),  
        tile_spacing=(1,1)),cmap='gray')  
plt.show()
```

Training neural network



4. Proof of Two Equations

Date _____

No. _____

$$(a) \text{ minimize } \frac{1}{n} \sum_{w \in W} -n(w) \log(\theta_w)$$

subject to $\theta_w \geq 0$ for all $w \in W$, $\sum_{w \in W} \theta_w = 1$

$$\cancel{\nabla L(\theta) = 0} \quad \hat{\theta}_w = \frac{n(w)}{\sum_{w' \in W} n(w')}$$

Yes. If a dictionary is used instead of w , the denominator is different. So it matters.

$$(b) L_n(\mu, b^2) = \frac{d}{2} \log(2\pi b^2) + \frac{1}{2nb^2} \sum_{x \in S} \|x - \mu\|^2$$

$$\frac{\partial L}{\partial \mu} = \frac{1}{2nb^2} \sum_{x \in S} 2(x - \mu) = 0$$

$$\therefore \sum_{x \in S} (x - \mu) = 0$$

$$\therefore n\mu = \sum_{x \in S} x$$

$$\therefore \mu = \frac{1}{n} \sum_{x \in S} x$$

$$\frac{\partial L}{\partial b} = \frac{d}{2b} - \frac{1}{nb^3} \sum_{x \in S} \|x - \mu\|^2 = 0$$

$$\therefore \frac{d}{db} = \frac{\sum_{x \in S} \|x - \mu\|^2}{nb^3}$$

$$\therefore b^2 = \frac{\sum_{x \in S} \|x - \mu\|^2}{nd}$$