IFT 6390

Fundamentals of machine learning

Professor: Ioannis Mitliagkas

Homework 3

Ying Xiao

Yan Ai

Jinfang Luo

2018.11.05

1. (1)

$$\frac{1}{2}\left(\tanh\left(\frac{1}{2}x\right)+1\right)$$
$$=\frac{1}{2}\left(\frac{\exp(\frac{1}{2}x)-\exp(-\frac{1}{2}x)}{\exp(\frac{1}{2}x)+\exp(-\frac{1}{2}x)}+1\right)$$
$$=\frac{1}{2}\left(\frac{\exp(\frac{1}{2}x)-\exp(-\frac{1}{2}x)+\exp(\frac{1}{2}x)+\exp(-\frac{1}{2}x)}{\exp(\frac{1}{2}x)+\exp(-\frac{1}{2}x)}\right)$$
$$=\frac{1}{2}\left(\frac{2\exp(\frac{1}{2}x)}{\exp(\frac{1}{2}x)+\exp(-\frac{1}{2}x)}\right)$$
$$=\frac{\exp(\frac{1}{2}x)\exp(-\frac{1}{2}x)}{\left(\exp(\frac{1}{2}x)+\exp(-\frac{1}{2}x)\right)\exp(-\frac{1}{2}x)}$$
$$=\frac{1}{1+\exp(-x)}$$
$$=sigmoid(x)$$

(2)

$$ln\ \ sigmoid(x)$$
$$=ln\frac{1}{1+\exp(-x)}$$
$$=ln1-ln(1+\exp(-x))$$
$$=-ln(1+\exp(-x))$$
$$=-softplus(-x)$$

(3)

$$sigmoid'(x)=\frac{d\ \ sigmoid(x)}{dx}$$
$$=\frac{d}{dx}\left(\frac{1}{1+\exp(-x)}\right)$$
$$=-\frac{-\exp(-x)}{(1+\exp(-x))^2}$$
$$=\frac{1}{1+\exp(-x)}\frac{1+\exp(-x)-1}{1+\exp(-x)}$$
$$=sigmoid(x)(1-sigmoid(x))$$

(4)

$$tanh'(x)$$
$$=\frac{d}{dx}\left(\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}\right)$$
$$=\frac{(\exp(x)+\exp(-x))(\exp(x)+\exp(-x))-(\exp(x)-\exp(-x))(\exp(x)-\exp(-x))}{(\exp(x)+\exp(-x))^2}$$
$$=\frac{(\exp(x)+\exp(-x))^2-(\exp(x)-\exp(-x))^2}{(\exp(x)+\exp(-x))^2}$$
$$=1-\frac{(\exp(x)-\exp(-x))^2}{(\exp(x)+\exp(-x))^2}$$
$$=1-\tanh^2(x)$$

(5)

$$sign(x)=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

(6)

$$abs(x)=|x|$$
$$abs'(x)=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

(7)

$$rect(x)=\begin{cases}x, & x\geq 0\\ 0, & x<0\end{cases}$$
$$rect'(x)=\begin{cases}1, & x>0\\ 0, & x\leq 0\end{cases}$$

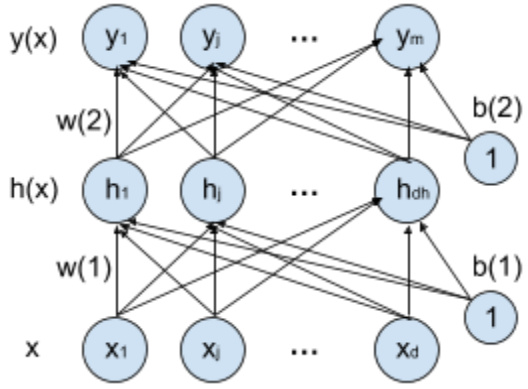
(8)

$$||x||_2^2=\Sigma_i x_i^2$$
$$\frac{\partial||x||_2^2}{\partial x_i}=2x_i$$

(9)

$$||x||_1=\Sigma_i |x_i|$$
$$\frac{\partial||x||_1}{\partial x_i}=\begin{cases}1, & x_i\geq 0\\ -1, & x_i<0\end{cases}$$

2.



（1）

(2)

The dimension of $W^{(2)}$ *is* $m \times d_h$, *and the dimension of* $b^{(2)}$ *is* m.
$O^a(h^s)=W^{(2)}h^s+b^{(2)}$ *where* $O_k^a(h^s)=W_{k\bullet}^{(2)}h^s+b_k^{(2)}$
and $0 \le k \le m$, $W_{k\bullet}^{(2)}$ *is the* k^{th} *row of* $W^{(2)}$, $b_k^{(2)}$ *is the* k^{th} *element of* $b^{(2)}$.

(3)

$$softmax(x_i)=\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad so \quad O_k^s=softmax(O_k^a)=\frac{e^{O_k^a}}{\sum_{i=1}^{m} e^{O_i^a}}$$

$$\because e^x>0, \quad \sum_x e^x>0, \quad \therefore O_k^s>0$$

$$and \quad \sum_{i=1}^{m} O_i^s = \sum_{i=1}^{m} \frac{e^{O_i^a}}{\sum_{j=1}^{m} e^{O_j^a}} = \frac{\sum_{i=1}^{m} e^{O_i^a}}{\sum_{j=1}^{m} e^{O_j^a}} = 1$$

This is very important, because of these two situation of O_k^s, *we can use it as probability.*

(4)

$$O_y^s = softmax(O_y^a) = \frac{e^{O_y^a}}{\sum_{i=1}^{m} e^{O_i^a}}$$

$$so \quad L(x,y) = -\log O_y^s$$

$$= -\log(softmax(O_y^a)) = -\log \frac{e^{O_y^a}}{\sum_{i=1}^{m} e^{O_i^a}}$$

$$= \log \sum_{i=1}^{m} e^{O_i^a} - O_y^a$$

(5)

$$\hat{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i)$$

The set θ of parameters include: $W^{(1)}$, $b^{(1)}$, $W^{(2)}$, $b^{(2)}$.

There is no scalar parameter here.

We use $\frac{\partial}{\partial \theta} \hat{R}(\theta) = 0$ *to find the optimal values for these parameters.*

(6)

The (batch) gradient descent equation for this problem is

$$Loop: \quad \theta \leftarrow \theta - \eta \frac{\partial \hat{R}(\theta)}{\partial \theta}$$

$$where \quad \frac{\partial \hat{R}(\theta)}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial}{\partial \theta} L(x^i, y^i)$$

(7)

$$\frac{\partial L}{\partial O^a} = \frac{\partial}{\partial O^a}(-logO_y^s) = \frac{\partial}{\partial O^a}(-log(softmax(O_y^a)))$$

$when\ \ k \neq y$

$$\frac{\partial L}{\partial O_k^a} = \frac{\partial}{\partial O_k^a}(-log(softmax(O_y^a))) = \frac{\partial}{\partial O_k^a}(log\sum_{i=1}^{m} e^{O_i^a} - O_y^a) = \frac{\partial}{\partial O_k^a} log\sum_{i=1}^{m} e^{O_i^a}$$

$$= \frac{\frac{\partial}{\partial O_k^a}\sum_{i=1}^{m} e^{O_i^a}}{\sum_{i=1}^{m} e^{O_i^a}} = \frac{e^{O_k^a}}{\sum_{i=1}^{m} e^{O_i^a}} = O_k^s$$

$when\ \ k = y$

$$\frac{\partial L}{\partial O_y^a} = \frac{\partial}{\partial O_y^a}(-log(softmax(O_y^a))) = \frac{\partial}{\partial O_y^a}(log\sum_{i=1}^{m} e^{O_i^a} - O_y^a) = \frac{\partial}{\partial O_y^a} log\sum_{i=1}^{m} e^{O_i^a} - 1$$

$$= \frac{\frac{\partial}{\partial O_y^a}\sum_{i=1}^{m} e^{O_i^a}}{\sum_{i=1}^{m} e^{O_i^a}} - 1 = \frac{e^{O_y^a}}{\sum_{i=1}^{m} e^{O_i^a}} - 1 = O_y^s - 1$$

$$\therefore \frac{\partial L}{\partial O^a} = O^s - onehot_m(y)$$

(8)

```
def softmax(x):
    return np.exp(x)/np.sum(np.exp(x))
def onehot(m, y):
    mat=[0]*m
    mat[y]=1
    return mat.T
grad_oa=softmax(oa)-onehot[m,y]
```

(9)

$$\frac{\partial L}{\partial W_{kj}^{(2)}} = \frac{\partial L}{\partial O_k^a}\frac{\partial O_k^a}{\partial W_{kj}^{(2)}} = \frac{\partial L}{\partial O_k^a}\cdot\frac{\partial}{\partial W_{kj}^{(2)}}(W_{k\bullet}^{(2)}h^s + b_k^{(2)}) = \frac{\partial L}{\partial O_k^a}\cdot h_j^s$$

$$\frac{\partial L}{\partial b_k^{(2)}} = \frac{\partial L}{\partial O_k^a}\frac{\partial O_k^a}{\partial b_k^{(2)}} = \frac{\partial L}{\partial O_k^a}\cdot\frac{\partial}{\partial b_k^{(2)}}(W_{k\bullet}^{(2)}h^s + b_k^{(2)}) = \frac{\partial L}{\partial O_k^a}$$

(10)

$$\frac{\partial L}{\partial W_{kj}^{(2)}} = \frac{\partial L}{\partial O_k^a} \cdot h_j^s$$

$$\therefore \frac{\partial L}{\partial W^{(2)}} = \begin{bmatrix} \frac{\partial L}{\partial O_1^a} \cdot h_1^s & \frac{\partial L}{\partial O_1^a} \cdot h_2^s & \cdots & \frac{\partial L}{\partial O_1^a} \cdot h_{d_h}^s \\ \frac{\partial L}{\partial O_2^a} \cdot h_1^s & \frac{\partial L}{\partial O_2^a} \cdot h_2^s & \cdots & \frac{\partial L}{\partial O_2^a} \cdot h_{d_h}^s \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial L}{\partial O_m^a} \cdot h_1^s & \frac{\partial L}{\partial O_m^a} \cdot h_2^s & \cdots & \frac{\partial L}{\partial O_m^a} \cdot h_{d_h}^s \end{bmatrix} = \frac{\partial L}{\partial O^a} \cdot (h^s)^T$$

$$\frac{\partial L}{\partial b_k^{(2)}} = \frac{\partial L}{\partial O_k^a}$$

$$\therefore \frac{\partial L}{\partial b^{(2)}} = \begin{bmatrix} \frac{\partial L}{\partial O_1^a} & \frac{\partial L}{\partial O_2^a} & \cdots & \frac{\partial L}{\partial O_m^a} \end{bmatrix} = \frac{\partial L}{\partial O^a}$$

grad_b2 = grad_oa

grad_w2 = grad_oa*hs.T

(11)

$$\frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{m} \frac{\partial L}{\partial o_k^a} \frac{\partial o_k^a}{\partial h_j^s}$$

$$where \quad \frac{\partial o_k^a}{\partial h_j^s} = \frac{\partial}{\partial h_j^s}(W_{k\bullet}^{(2)} h^s + b_k^{(2)}) = \frac{\partial}{\partial h_j^s}(W_{kj}^{(2)} h_j^s) = W_{kj}^{(2)}$$

$$\therefore \frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{m} (\frac{\partial L}{\partial o_k^a} \cdot W_{kj}^{(2)})$$

(12)

$$\frac{\partial L}{\partial h_j^s}=\sum_{k=1}^{m}\left(\frac{\partial L}{\partial o_k^a}\cdot W_{kj}^{(2)}\right)$$

$$\therefore \frac{\partial L}{\partial h^s}=\begin{bmatrix}\frac{\partial L}{\partial h_1^s} & \frac{\partial L}{\partial h_2^s} & \cdots & \frac{\partial L}{\partial h_{d_h}^s}\end{bmatrix}^T$$

$$=\begin{bmatrix}\sum_{k=1}^{m}\left(\frac{\partial L}{\partial o_k^a}\cdot W_{k1}^{(2)}\right) & \sum_{k=1}^{m}\left(\frac{\partial L}{\partial o_k^a}\cdot W_{k2}^{(2)}\right) & \cdots & \sum_{k=1}^{m}\left(\frac{\partial L}{\partial o_k^a}\cdot W_{kd_h}^{(2)}\right)\end{bmatrix}^T$$

$$=\left(\left(\frac{\partial L}{\partial o^a}\right)^T W^{(2)}\right)^T=W^{(2)T}\frac{\partial L}{\partial o^a}$$

grad_hs=W2.T*grad_oa

(13)

$$\frac{\partial L}{\partial h_j^a}=\frac{\partial L}{\partial h_j^s}\frac{\partial h_j^s}{\partial h_j^a}=\frac{\partial L}{\partial h_j^s}\frac{\partial}{\partial h_j^a}(rect(h_j^a))$$

$$where\ \ rect(h_j^a)=\begin{cases}h_j^a, & h_j^a\geq 0\\ 0, & h_j^a<0\end{cases}$$

$$\therefore \frac{\partial}{\partial h_j^a}(rect(h_j^a))=\begin{cases}1, & h_j^a>0\\ 0, & h_j^a\leq 0\end{cases}$$

$$\therefore \frac{\partial L}{\partial h_j^a}=\frac{\partial L}{\partial h_j^s}\frac{\partial h_j^s}{\partial h_j^a}=\begin{cases}\frac{\partial L}{\partial h_j^s}, & h_j^a>0\\ 0, & h_j^a\leq 0\end{cases}$$

(14)

$$\frac{\partial L}{\partial h_j^a}=\frac{\partial L}{\partial h_j^s}\frac{\partial h_j^s}{\partial h_j^a}=\begin{cases}\frac{\partial L}{\partial h_j^s}, & h_j^a>0\\ 0, & h_j^a\leq 0\end{cases}$$

```
grad_ha=[]
        grad_ha=[]
        for i in range(dh):
        if (ha[i,0] <= 0) : grad_ha.append(0)
        else: grad_ha.append(grad_hs[i,0])
```

(15)

$$\frac{\partial L}{\partial W_{kj}^{(1)}}=\frac{\partial L}{\partial h_j^a}\frac{\partial h_j^a}{\partial W_{jk}^{(1)}}=\frac{\partial L}{\partial h_j^a}\frac{\partial}{\partial W_{jk}^{(1)}}(W_{j\bullet}^{(1)}x+b_j^{(1)})=\frac{\partial L}{\partial h_j^a}\cdot x_k$$

$$\frac{\partial L}{\partial b_j^{(1)}}=\frac{\partial L}{\partial h_j^a}\frac{\partial h_j^a}{\partial b_j^{(1)}}=\frac{\partial L}{\partial h_j^a}\frac{\partial}{\partial b_j^{(1)}}(W_{j\bullet}^{(1)}x+b_j^{(1)})=\frac{\partial L}{\partial h_j^a}$$

(16)

$$\frac{\partial L}{\partial W_{jk}^{(1)}}=\frac{\partial L}{\partial h_j^a}\cdot x_k$$

$$\frac{\partial L}{\partial W^{(1)}}=\begin{bmatrix}\frac{\partial L}{\partial h_1^a}\cdot x_1 & \frac{\partial L}{\partial h_1^a}\cdot x_2 & \cdots & \frac{\partial L}{\partial h_1^a}\cdot x_d\\ \frac{\partial L}{\partial h_2^a}\cdot x_1 & \frac{\partial L}{\partial h_2^a}\cdot x_2 & \cdots & \frac{\partial L}{\partial h_2^a}\cdot x_d\\ \vdots & \vdots & \cdots & \vdots\\ \frac{\partial L}{\partial h_{h_d}^a}\cdot x_1 & \frac{\partial L}{\partial h_{h_d}^a}\cdot x_2 & \cdots & \frac{\partial L}{\partial h_{h_d}^a}\cdot x_d\end{bmatrix}=\frac{\partial L}{\partial h^a}\cdot x^T$$

$$\frac{\partial L}{\partial b^{(1)}}=\frac{\partial L}{\partial h^a}$$

```
grad_w1=grad_ha*x.T
grad_b1=grad_ha
```

(17)

$$\frac{\partial L}{\partial x_i} = \sum_{k=1}^{d_h} \frac{\partial L}{\partial h_k^a} \frac{\partial h_k^a}{\partial x_i} = \sum_{k=1}^{d_h} \frac{\partial L}{\partial h_k^a} \frac{\partial}{\partial x_i} (W_k^{(1)} x + b^{(1)}) = \sum_{k=1}^{d_h} \frac{\partial L}{\partial h_k^a} W_{ki}^{(1)}$$

$$\frac{\partial L}{\partial x} = W^{(1)T} \frac{\partial L}{\partial h^a}$$

(18)

If we will now consider a regularized emprical risk, $\tilde{R} = \hat{R} + L(\theta)$,then the gradient of the parameters will change from $\theta \leftarrow \theta - \eta \frac{\partial \hat{R}}{\partial \theta}$ to $\theta \leftarrow \theta - \eta \frac{\partial \tilde{R}}{\partial \theta}$. That derivation will be affected if the equation of variable contain W:

$$\frac{\partial \tilde{R}}{\partial W_{kj}^{(2)}} = \frac{\partial \hat{R}}{\partial W_{kj}^{(2)}} + 2\lambda_{22} W_{kj}^{(2)} \pm \lambda_{22} \quad (if \quad W_{kj}^{(2)} \geq 0 \quad then \quad +, \quad else \quad -)$$

$$\frac{\partial \tilde{R}}{\partial W_{kj}^{(1)}} = \frac{\partial \hat{R}}{\partial W_{kj}^{(1)}} + 2\lambda_{12} W_{kj}^{(1)} \pm \lambda_{11} \quad (if \quad W_{kj}^{(1)} \geq 0 \quad then \quad +, \quad else \quad -)$$

3. PRACTICAL PART (50 pts) :

1. As a beginning, start with an implementation that computes the gradients for a single example, and check that the gradient is correct using the finite difference method described above.

onehot, rect, softmax

```
def onehot(m,y): # y can not be a number
    result = []
    y = y.T
    for i in range(len(y)):
        result.append(np.eye(m)[int(y[i])])
    result = np.matrix(result).T
    return result

def rect(x): # x must be a matrix
    result=[]
    for i in range(np.shape(x)[1]):
        temp=[]
        for j in range(np.shape(x)[0]):
            temp.append(max(0,x[j,i]))
        result.append(temp)
    result = np.matrix(result).T
    return result

def softmax(x): # x must be a matrix
    result = []
    m,n = np.shape(x)
    for i in range(n):
        temp = []
        sum = np.sum(np.exp(x[:,i]))
#         print(sum)
        for j in range(m):
            temp.append(np.exp(x[j,i])/sum)
        result.append(temp)
    result = np.matrix(result).T
    return result
```

w and b

```
#initial the parameters

dh = 2
d = 2
m = 2

w1 = np.matrix((np.random.uniform(-1/d**0.5,1/d**0.5,d*dh)).reshape(dh,d))
print('w1:\n',w1)
b1 = np.matrix([0]*dh).T
w2 = np.matrix((np.random.uniform(-1/dh*0.5,1/dh*0.5,dh*m)).reshape(m,dh))
print('w2:\n',w2)
b2 = np.matrix([0]*m).T
```

fprop

```
#calculate x,ha,hs,oa,os

def fprop(x,y,w1,b1,w2,b2):
#     mean = np.mean(x)
#     s = np.std(x)
#     x = (x-mean)/s
    ha = w1*x+b1
#     print(w1,'\n',x[:,0])
#     print('ha:\n',np.shape(ha))
    hs = rect(ha)
#     print('hs:\n',np.shape(hs))
    oa = w2*hs+b2
#     print('oa:\n',np.shape(oa))
    os = softmax(oa)
#     print('os:\n',np.shape(os))
    L = -np.log(os[int(y)])
    return(ha,hs,oa,os,L)
```

bprop

```
def bprop(m,x,y,w1,b1,w2,b2):
    ha = fprop(x,y,w1,b1,w2,b2)[0]
    hs = fprop(x,y,w1,b1,w2,b2)[1]
    os = fprop(x,y,w1,b1,w2,b2)[3]
    grad_oa = os-onehot(m,y)
    grad_w2 = grad_oa*hs.T
    grad_b2 = grad_oa
    grad_hs = w2.T*grad_oa
    grad_ha=[]
    for i in range(dh):
        if (ha[i,0] <= 0) : grad_ha.append(0)
        else: grad_ha.append(grad_hs[i,0])
    grad_ha = np.matrix(grad_ha).T
    grad_w1 = grad_ha*x.T
    grad_b1 = grad_ha
    return(grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1, grad_b1)
```

2. Display the gradients for both methods (direct computation and finite difference) for a small network (e.g. d = 2 and dh = 2) with random weights and for a single example.

```
# verify for w2[0,0]
# dL/d(w2[0,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[1][0,0]
# (fprop(x[:,0],y[0,0],w1,b1,w2',b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b22))/epsilon where w2'=w2+[[epsilon,0],[0,0]]
epsilon = (10)**(-5)
w2_new = np.zeros([2,2])
w2_new[0,0] = epsilon
w2_new += w2
diff_gw2 = (fprop(x[:,0],y[0,0],w1,b1,w2_new,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw2 == gw2[0,0])
```

```
[[ True]]
```

```
# verify for w2[0,1]
# dL/d(w2[0,1])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[1][0,1]
# (fprop(x[:,0],y[0,0],w1,b1,w2',b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w2'=w2+[[0,epsilon],[0,0]]
w2_new = np.zeros([2,2])
w2_new[0,1] = epsilon
w2_new += w2
diff_gw2 = (fprop(x[:,0],y[0,0],w1,b1,w2_new,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw2/gw2[0,1] )
```

```
[[1.00000064]]
```

```
# verify for w2[1,0]
# dL/d(w2[1,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[1][1,0]
# (fprop(x[:,0],y[0,0],w1,b1,w2',b2)-fprop(x,w1,b1,w2,b2))/epsilon where w2'=w2+[[0,0],[epsilon,0]]
w2_new = np.zeros([2,2])
w2_new[1,0] = epsilon
w2_new += w2
diff_gw2 = (fprop(x[:,0],y[0,0],w1,b1,w2_new,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw2 == gw2[1,0] )
```

```
[[ True]]
```

```
# verify for w2[1,1]
# dL/d(w2[1,1])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[1][1,1]
# (fprop(x[:,0],y[0,0],w1,b1,w2',b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w2'=w2+[[0,0],[0,epsilon]]
w2_new = np.zeros([2,2])
w2_new[1,1] = epsilon
w2_new += w2
diff_gw2 = (fprop(x[:,0],y[0,0],w1,b1,w2_new,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw2 / gw2[1,1] )
```

```
[[0.99999937]]
```

```
# verify for b2[0,0]
# dL/d(b2[0,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[2][0,0]
# (fprop(x[:,0],y[0,0],w1,b1,w2,b2')-fprop(x,w1,b1,w2,b2))/epsilon where b2'=b2+[epsilon,0]
b2_new = np.zeros([1,2]).T
b2_new[0,0] = epsilon
b2_new += b2
diff_b2 = (fprop(x[:,0],y[0,0],w1,b1,w2,b2_new)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_b2 / gb2[0,0] )
```

```
[[1.00000244]]
```

```
# verify for b2[1,0]
# dL/d(b2[1,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[2][1,0]
# (fprop(x[:,0],y[0,0],w1,b1,w2,b2')-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where b2'=b2+[0,epsilon]
b2_new = np.zeros([1,2]).T
b2_new[1,0] = epsilon
b2_new += b2
diff_b2 = (fprop(x[:,0],y[0,0],w1,b1,w2,b2_new)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_b2 / gb2[1,0] )
```

```
[[0.99999756]]
```

```
# verify for w1[0,0]
# dL/d(w2[0,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[3][0,0]
# (fprop(x[:,0],y[0,0],w1',b1,w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w1'=w1+[[epsilon,0],[0,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[0,0] = epsilon
w1_new += w1
diff_gw1 = (fprop(x[:,0],y[0,0],w1_new,b1,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw1 == gw1[0,0] )
```

```
[[ True]]
```

```
# verify for w1[0,1]
# dL/d(w1[0,1])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[3][0,1]
# (fprop(x[:,0],y[0,0],w1',b1,w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w1'=w1+[[0,epsilon],[0,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[0,1] = epsilon
w1_new += w1
diff_gw1 = (fprop(x[:,0],y[0,0],w1_new,b1,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw1 == gw1[0,1] )
```

```
[[ True]]
```

```
# verify for w1[1,0]
# dL/d(w1[1,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[3][1,0]
# (fprop(x[:,0],y[0,0],w1',b1,w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w1'=w1+[[0,0],[epsilon,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[1,0] = epsilon
w1_new += w1
diff_gw1 = (fprop(x[:,0],y[0,0],w1_new,b1,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw1 / gw1[1,0] )
```

```
[[1.00000034]]
```

```
# verify for w1[1,1]
# dL/d(w1[1,1])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[3][1,1]
# (fprop(x[:,0],y[0,0],w1',b1,w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where w1'=w1+[[0,0],[0,epsilon]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[1,1] = epsilon
w1_new += w1
diff_gw1 = (fprop(x[:,0],y[0,0],w1_new,b1,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_gw1 / gw1[1,1] )
```

```
[[1.00000015]]
```

```
# verify for b1[0,0]
# dL/d(b1[0,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[4][0,0]
# (fprop(x[:,0],y[0,0],w1,b1',w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where b1'=b1+[epsilon,0]
b1_new = np.zeros([1,2]).T
b1_new[0,0] = epsilon
b1_new += b1
diff_b1 = (fprop(x[:,0],y[0,0],w1,b1_new,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_b1 == gb1[0,0])
```

```
[[ True]]
```

```
# verify for b1[1,0]
# dL/d(b1[0,0])=bprop(m,x[:,0],np.matrix(y[0,0]),os)[4][1,0]
# (fprop(x[:,0],y[0,0],w1,b1',w2,b2)-fprop(x[:,0],y[0,0],w1,b1,w2,b2))/epsilon where b1'=b1+[0,epsilon]
b1_new = np.zeros([1,2]).T
b1_new[1,0] = epsilon
b1_new += b1
diff_b1 = (fprop(x[:,0],y[0,0],w1,b1_new,w2,b2)[-1]-fprop(x[:,0],y[0,0],w1,b1,w2,b2)[-1])/epsilon
print (diff_b1 / gb1[1,0])
```

```
[[1.00000047]]
```

3. Add a hyperparameter for the minibatch size K to allow compute the gradients on a minibatch of K examples (in a matrix), by looping over the K examples (this is a small addition to your previous code).

```
def fprop_k_loop(x,y,w1,b1,w2,b2,k):
    sum_L = 0
    for i in k:
        sum_L += fprop(x[:,i],y[0,i],w1,b1,w2,b2)[-1]
    return np.linalg.det(sum_L/len(k))
```

```
def bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh):
    grad_oa = np.zeros([m,1])
    grad_w2 = np.zeros([m,dh])
    grad_b2 = np.zeros([m,1])
    grad_hs = np.zeros([dh,1])
    grad_ha = np.zeros([dh,1])
    grad_w1 = np.zeros([dh,d])
    grad_b1 = np.zeros([dh,1])
    for i in k :
        [grad_oai, grad_w2i, grad_b2i, grad_hsi, grad_hai, grad_w1i, grad_b1i]
        = bprop(m,x[:,i],np.matrix(y[0,i]),w1,b1,w2,b2)
        grad_oa += grad_oai
        grad_w2 += grad_w2i
        grad_b2 += grad_b2i
        grad_hs += grad_hsi
        grad_ha += grad_hai
        grad_w1 += grad_w1i
        grad_b1 += grad_b1i
    grad_oa /= len(k)
    grad_w2 /= len(k)
    grad_b2 /= len(k)
    grad_hs /= len(k)
    grad_ha /= len(k)
    grad_w1 /= len(k)
    grad_b1 /= len(k)
    return (grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1,grad_b1)
```

4. Display the gradients for both methods (direct computation and finite difference) for a small network (e.g. d = 2 and dh = 2) with random weights and for a minibatch with 10 examples (you can use examples from both classes from the two circles dataset).

```
# verify for each parameter on a minibatch k = [0,1,2,...,10]
k=np.arange(10)
grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1,grad_b1 = bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh)
print('grad_oa:\n',grad_oa,'\ngrad_w2:\n',grad_w2,'\ngrad_b2:\n',grad_b2,'\ngrad_w1:\n',grad_w1,'\ngrad_b1:\n',grad_b1)
```

```
# verify for w2[0,0]
# dL/d(w2[0,0])= grad_w2[0,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2',b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w2'=w2+[[epsilon,0][0,0]]
epsilon = (10)**(-5)
w2_new = np.zeros([2,2])
w2_new[0,0] = epsilon
w2_new += w2
diff_gw2 = (fprop_k_loop(x,y,w1,b1,w2_new,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw2 / grad_w2[0,0])
```

```
0.9999985538216867
```

```
# verify for w2[0,1]
# dL/d(w2[0,1])= grad_w2[0,1] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2',b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w2'=w2+[[0,epsilon],[0,0]]
epsilon = (10)**(-5)
w2_new = np.zeros([2,2])
w2_new[0,1] = epsilon
w2_new += w2
diff_gw2 = (fprop_k_loop(x,y,w1,b1,w2_new,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw2 / grad_w2[0,1])
```

```
0.9999981057717863
```

```
# verify for w2[1,0]
# dL/d(w2[1,0])= grad_w2[1,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2',b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w2'=w2+[[0,0],[epsilon,0]
epsilon = (10)**(-5)
w2_new = np.zeros([2,2])
w2_new[1,0] = epsilon
w2_new += w2
diff_gw2 = (fprop_k_loop(x,y,w1,b1,w2_new,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw2 / grad_w2[1,0])
```

```
1.0000014461139695
```

```
# verify for w2[1,1]
# dL/d(w2[1,1])= grad_w2[0,1] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2',b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w2'=w2+[[0,0],[0,epsilon]]
epsilon = (10)**(-5)
w2_new = np.zeros([2,2])
w2_new[1,1] = epsilon
w2_new += w2
diff_gw2 = (fprop_k_loop(x,y,w1,b1,w2_new,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw2 / grad_w2[1,1])
```

```
1.000001894280663
```

```
# verify for b2[0,0]
# dL/d(b2[0,0])= grad_b2[0,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2,b2',k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where b2'=b2+[[epsilon],[0]]
epsilon = (10)**(-5)
b2_new = np.zeros([2,1])
b2_new[0,0] = epsilon
b2_new += b2
diff_b2 = (fprop_k_loop(x,y,w1,b1,w2,b2_new,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_b2 / grad_b2[0,0])
```

```
0.9999936697076376
```

```
# verify for b2[1,0]
# dL/d(b2[1,0])= grad_b2[1,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2,b2',k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where b2'=b2+[[0],[epsilon]]
epsilon = (10)**(-5)
b2_new = np.zeros([2,1])
b2_new[1,0] = epsilon
b2_new += b2
diff_b2 = (fprop_k_loop(x,y,w1,b1,w2,b2_new,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_b2 / grad_b2[1,0])
```

```
1.0000063303590017
```

```
# verify for w1[0,0]
# dL/d(w1[0,0])= grad_w1[0,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1',b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w1'=w1+[[epsilon,0],[0,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[0,0] = epsilon
w1_new += w1
diff_gw1 = (fprop_k_loop(x,y,w1_new,b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw1 / grad_w1[0,0])
```

```
0.9999997059476512
```

```
# verify for w1[0,1]
# dL/d(w1[0,1])= grad_w1[0,1] we have got it in part3.3
# (fprop_k_loop(x,y,w1',b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w1'=w1+[[0,epsilon],[0,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[0,1] = epsilon
w1_new += w1
diff_gw1 = (fprop_k_loop(x,y,w1_new,b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw1 / grad_w1[0,1])
```

```
0.9999916654829426
```

```
# verify for w1[1,0]
# dL/d(w1[1,0])= grad_w1[1,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1',b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w1'=w1+[[0,0][epsilon,0]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[1,0] = epsilon
w1_new += w1
diff_gw1 = (fprop_k_loop(x,y,w1_new,b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw1 / grad_w1[1,0])
```

```
0.9999993291806428
```

```
# verify for w2[1,1]
# dL/d(w2[1,1])= grad_w2[1,1] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1,w2',b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where w2'=w2+[[0,0][0,epsilon]]
epsilon = (10)**(-5)
w1_new = np.zeros([2,2])
w1_new[1,1] = epsilon
w1_new += w1
diff_gw1 = (fprop_k_loop(x,y,w1_new,b1,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_gw1 / grad_w1[1,1])
```

```
0.9999992191666146
```

```
# verify for b1[0,0]
# dL/d(b1[0,0])= grad_b1[0,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1',w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where b1'=b1+[[epsilon],[0]]
epsilon = (10)**(-5)
b1_new = np.zeros([2,1])
b1_new[0,0] = epsilon
b1_new += b1
diff_b1 = (fprop_k_loop(x,y,w1,b1_new,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_b1 / grad_b1[0,0])
```

```
1.0000003333662255
```

```
# verify for b1[1,0]
# dL/d(b1[1,0])= grad_b1[1,0] we have got it in part3.3
# (fprop_k_loop(x,y,w1,b1',w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon where b1'=b1+[[0],[epsilon]]
epsilon = (10)**(-5)
b1_new = np.zeros([2,1])
b1_new[1,0] = epsilon
b1_new += b1
diff_b1 = (fprop_k_loop(x,y,w1,b1_new,w2,b2,k)-fprop_k_loop(x,y,w1,b1,w2,b2,k))/epsilon
print (diff_b1 / grad_b1[1,0])
```

```
0.9999991512086401
```

5. Train your neural network using gradient descent on the two circles dataset. Plot the decision regions for several different values of the hyperparameters (weight decay, number of hidden units, early stopping) so as to illustrate their effect on the capacity of the model.

```
n = np.shape(x)[1]
inds = np.arange(n)
np.random.shuffle(inds)
train_inds = inds[:800]
test_inds = inds[800:]
x_train = x[:,train_inds]
y_train = y[0,train_inds]
x_test = x[:, test_inds]
y_test = y[0, test_inds]
```

```
def L_theta(lamda,w1,w2,dh,m,d):
    if len(lamda) != 4 : return "Lamda is error!"
    else:
        w1=w1.reshape(1,dh*d)
        w11 = np.sum(abs(w1))
        w12 = np.linalg.det(w1*w1.T)
        w2=w2.reshape(1,dh*m)
        w21 = np.sum(abs(w2))
        w22 = np.linalg.det(w2*w2.T)
    W = [w11,w12,w21,w22]
    return np.linalg.det(lamda*np.matrix(W).T)

def train(x,y,w1,b1,w2,b2,num,lamda,dh,d,m,eta):
    m, n = np.shape(x)
    sum_loss = 0
    Rmin = 10000
    R = 0
    itera = 0
    while(itera < 100):
        for i in range(int(n/num)):
            k = np.arange(i*num,(i+1)*num)
            sum_loss = fprop_k_loop(x,y,w1,b1,w2,b2,k)
        R = (1/(n/num))*sum_loss+L_theta(lamda,w1,w2,dh,m,d)
        if R < Rmin:
            Rmin = R
            w1min = w1
            w2min = w2
            b1min = b1
            b2min = b2
            grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1,grad_b1 = bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh)
            w1 -= eta*grad_w1
            b1 -= eta*grad_b1
            w2 -= eta*grad_w2
            b2 -= eta*grad_b2
            itera += 1
        else:
            break
    if itera == 10000 :
        print(' not finish yet...')
    return (Rmin, w1min, w2min, b1min, b2min)
```

```
print(Rmin, w1min, w2min, b1min, b2min)
```

```
1.0611640237028122 [[-0.24489704 -0.09070214]
 [ 0.2109871  -0.1885247 ]] [[-0.03957763 -0.01495402]
 [ 0.0115575  -0.08545502]] [[-1.75631058e-07]
 [ 3.46704925e-05]] [[ 0.00049939]
 [-0.00049939]]
```

6. As a second step, copy your existing implementation to modify it to a new implementation that will use matrix calculus (instead of a loop) on batches of size K to improve efficiency. Take the matrix expressions in numpy derived in the first part, and adapt them for a minibatch of size K. Show in your report what you have modified (describe the former and new expressions with the shapes of each matrices).

```
def prol_k(x,k): #x is a matrix of 1 row, then prolong x to a matrix of k row, each row equal to x
    result=[]
    for i in range(k):
        temp=[]
        for j in range(np.shape(x)[1]):
            temp.append(x[0,j])
        result.append(temp)
    result=np.matrix(result)
    return result
```

```
prol_k(hs.T,10) # hs.T  1*2 -->> 10*2
print(np.shape(b1.T)[1])
print('b1:',b1)
b1_m = prol_k(b1.T,10).T # b d*1 -->> d*k
b1_m.T
```

```
def fprop_k_m(x,y,w1,b1,w2,b2,k,i): # add k and i: k is the size of minibatch, and i is the index of starting point.
    b1_m = prol_k(b1.T,k).T # change b1 from dh*1 to dh*k
    ha = w1*x[:,i:i+k]+b1 # x from a vectot of d*1 change to a matrix of d*k, so b1 must change to dh*k to match
    hs = rect(ha)
    b2 = prol_k(b2.T,k).T #change b2 from m*1 to m*k
    oa = w2*hs+b2 # hs from a vectot of dh*1 change to a matrix of dh*k, so b2 must change to m*k to match
    os = softmax(oa)
    lable = []  # lable and temp are used for choosing the value of os to calculate loss
    temp = []
    for j in range(k):
        lable.append(int(y[0,i+j]))
        temp.append(j)
    L = -np.log(os[lable,temp])
    L_sum = np.sum(L)/n

    return(ha, hs, oa, os, L, L_sum)
```

the shapes of each matrices

```
# Here, b1 is dh * k, w1 is dh * d, b2 is m * k, w2 is m * dh, ha and hs are dh * k, oa and os are m * k
```

```
def bprop_k_m(m,x,y,w1,b1,w2,b2,k,i): # add k and i: k is the size of minibatch, and i is the index of starting point.
    ha = fprop_k_m(x,y,w1,b1,w2,b2,k,i)[0]
    hs = fprop_k_m(x,y,w1,b1,w2,b2,k,i)[1]
    os = fprop_k_m(x,y,w1,b1,w2,b2,k,i)[3]
    grad_oa = os-onehot(m,y[0,i:i+k])
    grad_w2 = grad_oa*hs.T
    grad_b2 = grad_oa
    grad_hs = w2.T*grad_oa
    grad_ha = []
    for j in range(k):
        temp = []
        for m in range(dh):
            if (ha[m,j] <= 0) : temp.append(0)
            else:
                temp.append(grad_hs[m,j])
        grad_ha.append(temp)
    grad_ha = np.matrix(grad_ha).T
    grad_w1 = grad_ha*x[:,i:i+k].T
    grad_b1 = grad_ha
    return(np.sum(grad_oa,axis=1)/k, grad_w2, np.sum(grad_b2,axis=1)/k, np.sum(grad_hs,axis=1)/k,
           np.sum(grad_ha,axis=1)/k, grad_w1, np.sum(grad_b1,axis=1)/k)
```

the shapes of each matrices

```
# Here, grad_b1 is dh * 1, grad_w1 is dh * d, grad_b2 is m * 1, grad_w2 is m * dh,
# grad_ha and grad_hs are dh * 1, grad_oa and grad_os are m * 1
```

7. Compare both implementations (with a loop and with matrix calculus) to check that they both give the same values for the gradients on the parameters, first for K = 1, then for K = 10. Display the gradients for both methods.

K = 1

```
# with a loop
k=np.arange(1)
grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1,grad_b1
= bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh)
print('grad_oa:\n',grad_oa,'\ngrad_w2:\n',grad_w2,'\ngrad_b2:\n',grad_b2,
      '\ngrad_w1:\n',grad_w1,'\ngrad_b1:\n',grad_b1)
```

```
grad_oa:
 [[ 0.50191452]
 [-0.50191452]]
grad_w2:
 [[ 0.          0.04740948]
 [ 0.         -0.04740948]]
grad_b2:
 [[ 0.50191452]
 [-0.50191452]]
grad_w1:
 [[0.         0.        ]
 [0.02594793 0.01131677]]
grad_b1:
 [[0.        ]
 [0.03538547]]
```

```
# with matrix calculus
grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1, grad_b1
= bprop_k_m(m,x,y,w1,b1,w2,b2,1,0)
print('grad_oa:\n',grad_oa,'\ngrad_w2:\n',grad_w2,'\ngrad_b2:\n',grad_b2,
      '\ngrad_w1:\n',grad_w1,'\ngrad_b1:\n',grad_b1)
```

```
grad_oa:
 [[ 0.50191452]
 [-0.50191452]]
grad_w2:
 [[ 0.          0.04740948]
 [-0.         -0.04740948]]
grad_b2:
 [[ 0.50191452]
 [-0.50191452]]
grad_w1:
 [[0.         0.        ]
 [0.02594793 0.01131677]]
grad_b1:
 [[0.        ]
 [0.03538547]]
```

K = 10

```
# with a loop
k=np.arange(10)
grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1,grad_b1
= bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh)
print('grad_oa:\n',grad_oa,'\ngrad_w2:\n',grad_w2,'\ngrad_b2:\n',grad_b2,
      '\ngrad_w1:\n',grad_w1,'\ngrad_b1:\n',grad_b1)
```

```
grad_oa:
 [[-0.1993741]
 [ 0.1993741]]
grad_w2:
 [[-0.03218852 -0.02421717]
 [ 0.03218852  0.02421717]]
grad_b2:
 [[-0.1993741]
 [ 0.1993741]]
grad_w1:
 [[-6.70019602e-03 -5.63076924e-05]
 [-9.86571300e-03 -1.98683298e-03]]
grad_b1:
 [[ 0.00770831]
 [-0.0104821 ]]
```

```
# with matrix calculus
grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1, grad_b1
= bprop_k_m(m,x,y,w1,b1,w2,b2,10,0)
print('grad_oa:\n',grad_oa,'\ngrad_w2:\n',grad_w2,'\ngrad_b2:\n',grad_b2,
      '\ngrad_w1:\n',grad_w1,'\ngrad_b1:\n',grad_b1)
```

```
grad_oa:
 [[-0.1993741]
 [ 0.1993741]]
grad_w2:
 [[-0.32188522 -0.24217167]
 [ 0.32188522  0.24217167]]
grad_b2:
 [[-0.1993741]
 [ 0.1993741]]
grad_w1:
 [[-0.06700196 -0.00056308]
 [-0.09865713 -0.01986833]]
grad_b1:
 [[ 0.00770831]
 [-0.0104821 ]]
```

8. Time how long takes an epoch on fashion MNIST (1 epoch = 1 full traversal through the whole training set) for K = 100 for both versions (loop over a minibatch and matrix calculus).

```
import utils.mnist_reader as mnist_reader
from utils import mnist_reader
X_train, Y_train = mnist_reader.load_mnist('data/fashion', kind='train')
X_test, Y_test = mnist_reader.load_mnist('data/fashion', kind='t10k')
```

```
m = 10
dh = 50
train_data = np.matrix(XM_train).T
label = np.matrix(YM_train)
d = np.shape(train_data)[0]
w1 = np.matrix((np.random.uniform(-1/d**0.5,1/d**0.5,d*dh)).reshape(dh,d))
print('w1:\n',w1)
b1 = np.matrix([0]*dh).T
w2 = np.matrix((np.random.uniform(-1/dh*0.5,1/dh*0.5,dh*m)).reshape(m,dh))
print('w2:\n',w2)
b2 = np.matrix([0]*m).T
```

```
import time
def mnist_loop_all(x, y, w1, b1, w2, b2, dh, m):
    start = time.time()
    num = int(np.shape(XM_train)[1]/100)
    for i in range(num):
        k = np.arange(i*100, (i+1)*100)
        fprop_k_loop(x,y,w1,b1,w2,b2,k)
        goa_m, gw2_m, gb2_m, ghs_m, gha_m, gw1_m,gb1_m = bprop_k_loop(m,x,y,w1,b1,w2,b2,k,dh)
    end = time.time()
    running_time = end-start
    return running_time
```

```
print('The time for looping all the data by minibatch:',
      mnist_loop_all(train_data, label, w1, b1, w2, b2, dh, m))
```

```
The time for loopint all the data by minibatch: 0.6707990169525146
```

```
def mnist_matrix_all(x, y, w1, b1, w2, b2, dh, m):
    start = time.time()
    num = int(np.shape(XM_train)[1]/100)
    for i in range(num):
        fprop_k_m(x,y,w1,b1,w2,b2,100,i*100)
        bprop_k_m(m,x,y,w1,b1,w2,b2,100,i*100)
    end = time.time()
    running_time = end-start
    return running_time
```

```
print('The time for using matrix to calculate all the data by minibatch:',
      mnist_loop_all(train_data, label, w1, b1, w2, b2, dh, m))
```

```
The time for using matrix to calculate all the data by minibatch: 0.6026718616485596
```

So, the matrix calculate is fast then looping.

9. Adapt your code to compute the error (proportion of misclassified examples) on the training set as well as the total loss on the training set during each epoch of the training procedure, and at the end of each epoch, it computes the error and average loss on the validation set and the test set. Display the 6 corresponding figures (error and average loss on train/valid/test), and write them in a log file.

```
data = np.loadtxt(open('cercles.txt','r'))
n = np.shape(data)[0]
inds = np.arange(n)
np.random.shuffle(inds)
train_inds = inds[:700]
valid_inds = inds[700:900]
test_inds = inds[900:1100]
x_train = np.matrix(data[train_inds,:-1]).T
y_train = np.matrix(data[train_inds,-1])
x_valid = np.matrix(data[valid_inds,:-1]).T
y_valid = np.matrix(data[valid_inds,-1])
x_test = np.matrix(data[test_inds,:-1]).T
y_test = np.matrix(data[test_inds,-1])
```

```
# here we choose matrix calculate, because it's faster than looping
# modifier fprop to add error
def fprop_error(x,y,w1,b1,w2,b2,k,i): # add k and i: k is the size of minibatch, and i is the index of starting point.
    b1_m = prol_k(b1.T,k).T # change b1 from dh*1 to dh*k
    ha = w1*x[:,i:i+k]+b1 # x from a vectot of d*1 change to a matrix of d*k, so b1 must change to dh*k to match
    hs = rect(ha)
    b2 = prol_k(b2.T,k).T #change b2 from m*1 to m*k
    oa = w2*hs+b2 # hs from a vectot of dh*1 change to a matrix of dh*k, so b2 must change to m*k to match
    os = softmax(oa)
    err = 0
    for j in range(k):
        if (os[0,j] >= os[1,j]):
            pre_y = 0
        else:
            pre_y = 1
        if pre_y != y[0,i+j]:
            err += 1
    lable = []  # lable and temp are used for choosing the value of os to calculate loss
    temp = []
    for j in range(k):
        lable.append(int(y[0,i+j]))
        temp.append(j)
    L = -np.log(os[lable,temp])
    L_sum = np.sum(L)/k
    return(ha, hs, oa, os, L, L_sum,err)
```

```
def train(x,y,w1,b1,w2,b2,k,lamda,dh,d,m,eta,epsilon): # k is size of minibatch, eta is hyperparameter
    m, n = np.shape(x)
    sum_loss = 0
    err_sum = 0
    Rmin = 10000
    R = 0
    itera = 0
    err_prop = 0
    grad_b2 = 1
    num = int(n/k)
    while((itera < 1000 )& (np.sum(grad_b2) > epsilon)):
        for i in range(num):
            ha, hs, oa, os, L, loss, err = fprop_error(x,y,w1,b1,w2,b2,k,i*k)
            sum_loss += loss*k
            err_sum += err
        R = (1/n)*sum_loss+L_theta(lamda,w1,w2,dh,m,d)
        if R < Rmin:
            Rmin = R
            w1min = w1
            w2min = w2
            b1min = b1
            b2min = b2
            sum_loss_result = sum_loss
            err_result = err_sum
            for i in range(num):
                grad_oa, grad_w2, grad_b2, grad_hs, grad_ha, grad_w1, grad_b1 = bprop_k_m(m,x,y,w1,b1,w2,b2,k,i*k)
                w1 -= eta*grad_w1
                b1 -= eta*grad_b1
                w2 -= eta*grad_w2
                b2 -= eta*grad_b2
            itera += 1
        else:
            break
    if itera == 1000 :
        print(' not finish yet...')
    return (Rmin, w1min, w2min, b1min, b2min, sum_loss_result/n, err_result/n)
```

```
Rmin, w1min, w2min, b1min, b2min, loss_avg, err_prop = train(x_train,y_train,w1,b1,w2,b2,100,[0.01,0.01,0.01,0.01],dh,d
```

```
print(loss_avg)
print(err_prop)
```

```
0.694757623537982
0.5042857142857143
```

```
Rmin, w1min, w2min, b1min, b2min, loss_avg, err_prop = train(x_valid,y_valid,w1,b1,w2,b2,100,[0.01,0.01,0.01,0.01],dh,d
print(loss_avg)
print(err_prop)
```

```
0.6945979647817484
0.53
```

```
Rmin, w1min, w2min, b1min, b2min, loss_avg, err_prop = train(x_test,y_test,w1,b1,w2,b2,100,[0.01,0.01,0.01,0.01],dh,d,m
print(loss_avg)
print(err_prop)
```

```
0.6934102248402416
0.48
```