Logistic Regression on LOAD DIGITS dataset from scratch

importing libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import np_utils
import sklearn
```

```
Using TensorFlow backend.
```

Loading Dataset

In [2]:
```
from sklearn.datasets import load_digits
digits=load_digits()
```

In [3]:
```
Y=digits['target']
images=digits['images']
```

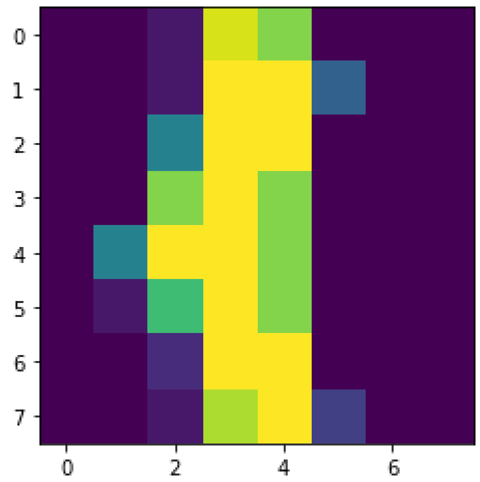
One hot Encoding

In [4]:
```
Y = np_utils.to_categorical(Y,10)
```

In [5]:
```
images=images/16
```

In [6]:
```
plt.imshow(images[99])
Y[99]
```

Out[6]:
```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```



Flattening

In [7]:
```
images_new=images.reshape(images.shape[0],-1)
print(images_new.shape,Y.shape)
```

```
(1797, 64) (1797, 10)
```

Train Test Split

In [8]:
```
train_X,test_X,train_Y,test_Y=train_test_split(images_new,Y,test_size=0.3)
```

In [9]:
```
train_X=train_X.T
test_X=test_X.T
train_Y=train_Y.T
test_Y=test_Y.T
```

Randomly intializing weights and bais

In [15]:
```
W=np.random.randn(train_X.shape[0],train_Y.shape[0])
b=np.zeros((train_Y.shape[0],1))
```

In [16]:

```
print(W.shape)
print(b.shape)
```

```
(64, 10)
(10, 1)
```

LOGISTIC REGRESSION SGD with momentum

In [17]:

```
v1 = 0
v2 = 0
mu = 0.99
learning_rate = 1
```

In [20]:

```
for i in range(2000):
    m=train_X.shape[1]
    train_A=1/(1+np.exp(-(np.dot(W.T,train_X)+b)))

    # column normalization
    for j in range(train_X.shape[1]):
        train_A[:,j]=train_A[:,j]/np.sum(train_A[:,j])
    cost1=1/m*(- np.sum(train_Y* np.log(train_A+1e-6)))
    print('iteration=',i,'-----train logloss---------',cost1)
    #cost=sklearn.metrics.log_loss(train_Y,A)

    test_A=1/(1+np.exp(-(np.dot(W.T,test_X)+b)))
    for j in range(test_X.shape[1]):
        test_A[:,j]=test_A[:,j]/np.sum(test_A[:,j])
    cost2=1/m*(- np.sum(test_Y* np.log(test_A+1e-6)))
    print('iteration=',i,'-----test logloss---------',cost2)

    dW=(1/m)*np.dot(train_X,(train_A-train_Y).T)
    db=(1/m)*np.sum(train_A-train_Y)

    v1 = mu * v1 - learning_rate * dW # integrate velocity
    W += v1 # integrate position

    v2 = mu * v2 - learning_rate * db # integrate velocity
    b += v2 # integrate position
```

```
iteration= 1987 -----test logloss--------- 0.624485425594043
iteration= 1988 -----train logloss--------- 1.498059368270154
iteration= 1988 -----test logloss--------- 0.6244854156208702
iteration= 1989 -----train logloss--------- 1.4980593665411939
iteration= 1989 -----test logloss--------- 0.6244854056606044
iteration= 1990 -----train logloss--------- 1.4980593648084026
iteration= 1990 -----test logloss--------- 0.6244853957132129
iteration= 1991 -----train logloss--------- 1.4980593630715806
iteration= 1991 -----test logloss--------- 0.6244853857786635
iteration= 1992 -----train logloss--------- 1.498059361330566
iteration= 1992 -----test logloss--------- 0.6244853758569244
iteration= 1993 -----train logloss--------- 1.4980593595852219
iteration= 1993 -----test logloss--------- 0.6244853659479646
iteration= 1994 -----train logloss--------- 1.4980593578354406
iteration= 1994 -----test logloss--------- 0.6244853560517526
iteration= 1995 -----train logloss--------- 1.4980593560811661
iteration= 1995 -----test logloss--------- 0.6244853461682581
iteration= 1996 -----train logloss--------- 1.4980593543223497
iteration= 1996 -----test logloss--------- 0.6244853362974512
```

In [21]:

```
#checking accuracy
count=0
for i in range(test_Y.shape[1]):
    if(np.argmax(test_Y[:,i])==np.argmax(test_A[:,i])):
        count+=1
accuracy=count/test_Y.shape[1]*100
print(accuracy)
```

```
25.555555555555554
```

LOGISTIC REGRESSION WITH ADAM

In [22]:

```
eps  = 1e-8
beta1 = 0.9
beta2 = 0.999

learning_rate = 0.001
m1 = 0
v1 = 0

m2 = 0
v2 = 0
```

In [23]:
```
W=np.random.randn(train_X.shape[0],train_Y.shape[0])
b=np.zeros((train_Y.shape[0],1))
```

In [25]:
```
for i in range(2000):
    m=train_X.shape[1]
    train_A=1/(1+np.exp(-(np.dot(W.T,train_X)+b)))

    # column normalization
    for j in range(train_X.shape[1]):
        train_A[:,j]=train_A[:,j]/np.sum(train_A[:,j])
    cost1=1/m*(- np.sum(train_Y* np.log(train_A+1e-6)))
    print('iteration=',i,'-----train logloss---------',cost1)
    #cost=sklearn.metrics.log_loss(train_Y,A)

    test_A=1/(1+np.exp(-(np.dot(W.T,test_X)+b)))
    for j in range(test_X.shape[1]):
        test_A[:,j]=test_A[:,j]/np.sum(test_A[:,j])
    cost2=1/m*(- np.sum(test_Y* np.log(test_A+1e-6)))
    print('iteration=',i,'-----test logloss---------',cost2)

    dW=(1/m)*np.dot(train_X,(train_A-train_Y).T)
    db=(1/m)*np.sum(train_A-train_Y)

    m1 = beta1*m1 + (1-beta1)*dW
    mt1 = m1 / (1-beta1**(i+1))
    v1= beta2*v1 + (1-beta2)*(dW**2)
    vt1 = v1 / (1-beta2**(i+1))
    W += - learning_rate * mt1 / (np.sqrt(vt1) + eps)

    m2 = beta1*m2 + (1-beta1)*db
    mt2 = m2 / (1-beta1**(i+1))
    v2= beta2*v2 + (1-beta2)*(db**2)
    vt2 = v2 / (1-beta2**(i+1))
    b += - learning_rate * mt2 / (np.sqrt(vt2) + eps)
```

```
iteration= 1986 -----test logloss--------- 0.5718389543856418
iteration= 1987 -----train logloss--------- 1.3628685843306143
iteration= 1987 -----test logloss--------- 0.5718513251534759
iteration= 1988 -----train logloss--------- 1.3628896401197883
iteration= 1988 -----test logloss--------- 0.5718637379920413
iteration= 1989 -----train logloss--------- 1.3629107961635811
iteration= 1989 -----test logloss--------- 0.5718761928396787
iteration= 1990 -----train logloss--------- 1.3629320523204518
iteration= 1990 -----test logloss--------- 0.5718886896347841
iteration= 1991 -----train logloss--------- 1.3629534084490234
iteration= 1991 -----test logloss--------- 0.5719012283158214
iteration= 1992 -----train logloss--------- 1.3629748644081605
iteration= 1992 -----test logloss--------- 0.5719138088213548
iteration= 1993 -----train logloss--------- 1.3629964200568494
iteration= 1993 -----test logloss--------- 0.5719264310899979
iteration= 1994 -----train logloss--------- 1.3630180752542294
iteration= 1994 -----test logloss--------- 0.5719390950604262
iteration= 1995 -----train logloss--------- 1.363039829859652
iteration= 1995 -----test logloss--------- 0.571951800671404
iteration= 1996 -----train logloss--------- 1.363061683732624
```

In [26]:
```
#checking accuracy
count=0
for i in range(test_Y.shape[1]):
    if(np.argmax(test_Y[:,i])==np.argmax(test_A[:,i])):
        count+=1
accuracy=count/test_Y.shape[1]*100
print(accuracy)
```

```
88.70370370370371
```

REFERENCES :

- http://cs231n.github.io/neural-networks-3/ (http://cs231n.github.io/neural-networks-3/)
- https://stats.stackexchange.com/questions/219241/gradient-for-logistic-loss-function (https://stats.stackexchange.com/questions/219241/gradient-for-logistic-loss-function)

OBSERVATION :

- Accuracy of SGD with momentum on test dataset is 25.5 %.
- Accuracy of ADAM on test dataset is 88.7 %.