In [35]:
```
import numpy as np
import scipy.special
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_covtype
from sklearn import linear_model
import  sklearn
import torch
import torch.optim as optim

class LogisticRegression:
    def __init__(self):
        pass

    def fit(self, X, y, lr=0.1, momentum=0, niter=100):
        '''
        Train a multiclass logistic regression model on the given training set.

        Parameters
        ----------
        X: training examples, represented as an input array of shape (n_sample,
           n_features).
        y: labels of training examples, represented as an array of shape
           (n_sample,) containing the classes for the input examples
        lr: learning rate for gradient descent
        niter: number of gradient descent updates
        momentum: the momentum constant (see assignment task sheet for an explanation)

        Returns
        -------
        self: fitted model
        '''
        self.classes_ = np.unique(y)
        self.class2int = dict((c, i) for i, c in enumerate(self.classes_))
        y = np.array([self.class2int[c] for c in y])

        n_features = X.shape[1]
        self.n_classes = len(self.classes_)
        # print(self.n_classes)
        n_classes = len(self.classes_)

        self.intercept_ = np.zeros(n_classes)
```

```
        self.coef_ = np.zeros((n_classes, n_features))

        # Implement your gradient descent training code here; uncomment the code below to do "random training"
        self.intercept_ = np.random.randn(*self.intercept_.shape)
        self.coef_ = np.random.randn(*self.coef_.shape)
        return  self

    def predict_proba(self, X):
        '''
        Predict the class distributions for given input examples.

        Parameters
        ----------
        X: input examples, represented as an input array of shape (n_sample,
           n_features).

        Returns
        -------
        y: predicted class distributions, represented as an array of shape (n_sample,
           n_classes)
        '''


        # replace pass with your code
                # replace pass with your code
        X=sklearn.preprocessing.normalize(X) # normalize X to prevent overflow
        eOY_prime = np.exp(self.coef_.dot(X.T)).T # X^T * W or o
        eoY_prime_max = eOY_prime.max(0) # max of o
        for i in range(len(eOY_prime)):
            # print(sumY_prime[i].shape)
            eOY_prime[i] = eOY_prime[i] - eoY_prime_max  #subtract all o_y by max(O)
        # max_oi =
        # print("sumY_prime.sum()",eOY_prime.sum())
        sumeoY_prime = eOY_prime.sum()  #calculate sum of e^{o_y}
        output = np.zeros(shape=(len(X),self.n_classes))
        for i in range(len(output)):
            o_i = np.dot(self.coef_,X[i].T)
            output[i] = np.exp(o_i)/sumeoY_prime  #e^{o_y} / sum(e^{o_y})
        return output

    def predict(self, X):
        '''
        Predict the classes for given input examples.

        Parameters
```

```
        ----------
        X: input examples, represented as an input array of shape (n_sample,
           n_features).

        Returns
        -------
        y: predicted class labels, represented as an array of shape (n_sample,)
        '''
        X_softemaxed_prob = self.predict_proba(X)
        return  X_softemaxed_prob.argmax(1)
        # replace pass with your code
X, y = fetch_covtype(return_X_y=True)
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3, random_state=42)
cls = LogisticRegression()
cls.fit(X,y)
cls.predict(X)
```

Out[35]:

```
array([3, 3, 3, ..., 3, 3, 3], dtype=int64)
```

In [35]:

```
```

4(a) pass

4(b)

see above code predict_proba() and predict() , and the result of of cls.predict(X)

4(C)

See below code .

In [57]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_covtype
from  sklearn.metrics  import  log_loss
import torch
from tqdm import  tqdm
```

```
class log_loss(torch.nn.Module):
    def __init__(self):
        super(log_loss,self).__init__()
    def forward(self,y,y_pred):
        return -(y*torch.log(y_pred)+(1-y)*torch.log(1-y_pred)).mean()


class LogisticRegression:
    def __init__(self):
        pass

    def fit(self, X, y, X_ts,y_ts,lr=0.2, momentum=0, niter=100):
        print("Training param is LR {}, niter {} , momentum {} ,number of traning example {}".format(lr,niter,momentum,len(X)))
        n_features = X.shape[1]
        self.classes_ = np.unique(y)
        n_classes = len(self.classes_)
        y_onehot = (y-1).long()
        y_onehot = torch.eye(7,requires_grad=True)[y_onehot]
        self.classes_ = np.unique(y)
        self.class2int = dict((c, i) for i, c in enumerate(self.classes_))
        self.intercept_ = torch.autograd.Variable(torch.rand(n_classes),requires_grad=True)
        self.coef_ = torch.autograd.Variable(torch.rand((n_classes,n_features)),requires_grad=True)
        y_1 = y-1
        loss_fu = log_loss()
        pbar = tqdm(range(niter))
        early_stop_e = 50
        record_epoch_for_early_stop = 0
        best_valid_acc = 0
        for i in pbar:
            loss=- loss_fu( y_onehot,self.predict_proba(X))
            if self.coef_.grad is not None:
                self.coef_.grad.zero_() # important: reset the stored gradient to 0
            if self.intercept_.grad is not None:
                self.intercept_.grad.zero_()
            loss.backward(retain_graph=True)
            self.coef_.data.add_(lr*self.coef_.grad.data)
            output = self.predict(X)
            test_output = self.predict(X_ts)
            test_acc = (test_output==torch.tensor(y_ts)).sum()/len(y_ts)
            if test_acc > best_valid_acc:
                best_valid_acc = test_acc
                record_epoch_for_early_stop=0
            else:
                record_epoch_for_early_stop +=1
```

```
                if record_epoch_for_early_stop > early_stop_e:
                    print("Early stopping because valid acc has not been improved for {} epoch".format(early_stop_e))
                    print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(ou
                    break
            if i % 25 ==0:
                print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(output
            pbar.set_description("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.ite
        return self


    def predict_proba(self, X):
        X=sklearn.preprocessing.normalize(X) # normalize X to prevent overflow
        X = torch.tensor(X,requires_grad=True).T.float()
        eOY_prime = X.T @ self.coef_.T # X^T * W or o
        eOY_prime = eOY_prime - torch.max(eOY_prime)
        output = torch.softmax(eOY_prime,1)
        return output


    def predict(self, X):
        X_softemaxed_prob = self.predict_proba(X)
        return  X_softemaxed_prob.argmax(1)
```

4(d)

To get the best loss , the simplest way is to extend nber of iterations. Another way is increase lr to get fast converge . So , I did two experiments : (1) extend niter to 1000 (2) increase lr from 0.1 to 0.2 The minimum loss i got is -0.2961 , and traning accuracy is 0.3660 and test acc is 0.3640:

In [58]:
```
X, y = fetch_covtype(return_X_y=True)
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.7, random_state=42)
cls = LogisticRegression()
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.1)
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =1000,lr=0.1)
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.2)
```

```
  0%|          | 0/500 [00:00<?, ?it/s]
Training param is LR 0.1, niter 500 , momentum 0 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4021  training acc is  0.2003  test acc is  0.1991:   0%|          | 1/500 [00:00<03:43,  2.24it/s]
Epoch 0 trainig loss is  -0.4021  training acc is  0.2003  test acc is  0.1991
Epoch 25 trainig loss is  -0.3886  training acc is  0.3116  test acc is  0.3096:   5%|▌         | 26/500 [00:12<03:44,  2.11it/
s]
```

```
Epoch 25 trainig loss is  -0.3886  training acc is  0.3116  test acc is  0.3096
Epoch 50 trainig loss is  -0.3764  training acc is  0.3554  test acc is  0.3536:  10%|█          | 51/500 [00:23<03:24,  2.20it/s]
Epoch 50 trainig loss is  -0.3764  training acc is  0.3554  test acc is  0.3536
Epoch 75 trainig loss is  -0.3656  training acc is  0.3660  test acc is  0.3640:  15%|█▌         | 76/500 [00:35<03:17,  2.15it/s]
Epoch 75 trainig loss is  -0.3656  training acc is  0.3660  test acc is  0.3640
Epoch 100 trainig loss is  -0.3560  training acc is  0.3660  test acc is  0.3640:  20%|██         | 101/500 [00:47<03:09,  2.11it/s]
Epoch 100 trainig loss is  -0.3560  training acc is  0.3660  test acc is  0.3640
Epoch 125 trainig loss is  -0.3476  training acc is  0.3660  test acc is  0.3640:  25%|██▌        | 126/500 [00:59<02:55,  2.14it/s]
Epoch 125 trainig loss is  -0.3476  training acc is  0.3660  test acc is  0.3640
Epoch 125 trainig loss is  -0.3476  training acc is  0.3660  test acc is  0.3640:  25%|██▌        | 126/500 [00:59<02:56,  2.12it/s]
  0%|          | 0/1000 [00:00<?, ?it/s]
Early stopping because valid acc has not been improved for 50 epoch
Epoch 126 trainig loss is  -0.3473  training acc is  0.3660  test acc is  0.3640
Training param is LR 0.1, niter 1000 , momentum 0 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4115  training acc is  0.0093  test acc is  0.0096:   0%|          | 1/1000 [00:00<08:46,  1.90it/s]
Epoch 0 trainig loss is  -0.4115  training acc is  0.0093  test acc is  0.0096
Epoch 25 trainig loss is  -0.3974  training acc is  0.0152  test acc is  0.0158:   3%|▎         | 26/1000 [00:12<07:33,  2.15it/s]
Epoch 25 trainig loss is  -0.3974  training acc is  0.0152  test acc is  0.0158
Epoch 50 trainig loss is  -0.3845  training acc is  0.0828  test acc is  0.0833:   5%|▌         | 51/1000 [00:23<07:32,  2.10it/s]
Epoch 50 trainig loss is  -0.3845  training acc is  0.0828  test acc is  0.0833
Epoch 75 trainig loss is  -0.3729  training acc is  0.1266  test acc is  0.1268:   8%|▊         | 76/1000 [00:35<07:36,  2.02it/s]
Epoch 75 trainig loss is  -0.3729  training acc is  0.1266  test acc is  0.1268
Epoch 100 trainig loss is  -0.3626  training acc is  0.1718  test acc is  0.1724:  10%|█         | 101/1000 [00:47<06:44,  2.22it/s]
Epoch 100 trainig loss is  -0.3626  training acc is  0.1718  test acc is  0.1724
Epoch 125 trainig loss is  -0.3534  training acc is  0.2164  test acc is  0.2160:  13%|█▎        | 126/1000 [00:59<06:47,  2.14it/s]
Epoch 125 trainig loss is  -0.3534  training acc is  0.2164  test acc is  0.2160
Epoch 150 trainig loss is  -0.3453  training acc is  0.2526  test acc is  0.2520:  15%|█▌        | 151/1000 [01:11<06:49,  2.07it/s]
Epoch 150 trainig loss is  -0.3453  training acc is  0.2526  test acc is  0.2520
```

```
Epoch 175 trainig loss is  -0.3381  training acc is  0.2869  test acc is  0.2855:  18%|██        | 176/1000 [01:23<06:21,  2.16it/s]
Epoch 175 trainig loss is  -0.3381  training acc is  0.2869  test acc is  0.2855
Epoch 200 trainig loss is  -0.3319  training acc is  0.3127  test acc is  0.3115:  20%|██        | 201/1000 [01:34<06:21,  2.09it/s]
Epoch 200 trainig loss is  -0.3319  training acc is  0.3127  test acc is  0.3115
Epoch 225 trainig loss is  -0.3265  training acc is  0.3296  test acc is  0.3283:  23%|██▎       | 226/1000 [01:46<05:44,  2.25it/s]
Epoch 225 trainig loss is  -0.3265  training acc is  0.3296  test acc is  0.3283
Epoch 250 trainig loss is  -0.3218  training acc is  0.3477  test acc is  0.3462:  25%|██▌       | 251/1000 [01:58<05:55,  2.11it/s]
Epoch 250 trainig loss is  -0.3218  training acc is  0.3477  test acc is  0.3462
Epoch 275 trainig loss is  -0.3177  training acc is  0.3569  test acc is  0.3553:  28%|██▊       | 276/1000 [02:09<05:40,  2.13it/s]
Epoch 275 trainig loss is  -0.3177  training acc is  0.3569  test acc is  0.3553
Epoch 300 trainig loss is  -0.3141  training acc is  0.3614  test acc is  0.3596:  30%|███       | 301/1000 [02:21<05:18,  2.19it/s]
Epoch 300 trainig loss is  -0.3141  training acc is  0.3614  test acc is  0.3596
Epoch 325 trainig loss is  -0.3110  training acc is  0.3638  test acc is  0.3618:  33%|███▎      | 326/1000 [02:32<05:15,  2.14it/s]
Epoch 325 trainig loss is  -0.3110  training acc is  0.3638  test acc is  0.3618
Epoch 350 trainig loss is  -0.3083  training acc is  0.3652  test acc is  0.3631:  35%|███▌      | 351/1000 [02:44<05:06,  2.12it/s]
Epoch 350 trainig loss is  -0.3083  training acc is  0.3652  test acc is  0.3631
Epoch 375 trainig loss is  -0.3059  training acc is  0.3657  test acc is  0.3636:  38%|███▊      | 376/1000 [02:56<04:48,  2.16it/s]
Epoch 375 trainig loss is  -0.3059  training acc is  0.3657  test acc is  0.3636
Epoch 400 trainig loss is  -0.3038  training acc is  0.3659  test acc is  0.3639:  40%|████      | 401/1000 [03:08<04:53,  2.04it/s]
Epoch 400 trainig loss is  -0.3038  training acc is  0.3659  test acc is  0.3639
Epoch 425 trainig loss is  -0.3019  training acc is  0.3660  test acc is  0.3639:  43%|████▎     | 426/1000 [03:19<04:33,  2.10it/s]
Epoch 425 trainig loss is  -0.3019  training acc is  0.3660  test acc is  0.3639
Epoch 450 trainig loss is  -0.3003  training acc is  0.3660  test acc is  0.3640:  45%|████▌     | 451/1000 [03:31<04:15,  2.15it/s]
Epoch 450 trainig loss is  -0.3003  training acc is  0.3660  test acc is  0.3640
Epoch 475 trainig loss is  -0.2988  training acc is  0.3660  test acc is  0.3640:  48%|████▊     | 476/1000 [03:42<03:59,  2.19it/s]
Epoch 475 trainig loss is  -0.2988  training acc is  0.3660  test acc is  0.3640
```

```
Epoch 500 trainig loss is  -0.2975  training acc is  0.3660  test acc is  0.3640:  50%|█████     | 501/1000 [03:54<03:55,
2.12it/s]
Epoch 500 trainig loss is  -0.2975  training acc is  0.3660  test acc is  0.3640
Epoch 525 trainig loss is  -0.2964  training acc is  0.3660  test acc is  0.3640:  53%|█████▎    | 526/1000 [04:06<03:38,
2.17it/s]
Epoch 525 trainig loss is  -0.2964  training acc is  0.3660  test acc is  0.3640
Epoch 532 trainig loss is  -0.2961  training acc is  0.3660  test acc is  0.3640:  53%|█████▎    | 533/1000 [04:09<03:38,
2.13it/s]
  0%|          | 0/500 [00:00<?, ?it/s]
Early stopping because valid acc has not been improved for 50 epoch
Epoch 533 trainig loss is  -0.2960  training acc is  0.3660  test acc is  0.3640
Training param is LR 0.2, niter 500 , momentum 0 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4033  training acc is  0.0603  test acc is  0.0620:   0%|          | 1/500 [00:00<03:44,  2.23it/s]
Epoch 0 trainig loss is  -0.4033  training acc is  0.0603  test acc is  0.0620
Epoch 25 trainig loss is  -0.3759  training acc is  0.3391  test acc is  0.3367:   5%|▌         | 26/500 [00:12<03:43,  2.12it/
s]
Epoch 25 trainig loss is  -0.3759  training acc is  0.3391  test acc is  0.3367
Epoch 50 trainig loss is  -0.3544  training acc is  0.3503  test acc is  0.3480:  10%|█         | 51/500 [00:23<03:37,  2.06it/
s]
Epoch 50 trainig loss is  -0.3544  training acc is  0.3503  test acc is  0.3480
Epoch 75 trainig loss is  -0.3381  training acc is  0.3569  test acc is  0.3544:  15%|█▌        | 76/500 [00:35<03:12,  2.20i
t/s]
Epoch 75 trainig loss is  -0.3381  training acc is  0.3569  test acc is  0.3544
Epoch 100 trainig loss is  -0.3259  training acc is  0.3602  test acc is  0.3578:  20%|██        | 101/500 [00:46<03:04,  2.17
it/s]
Epoch 100 trainig loss is  -0.3259  training acc is  0.3602  test acc is  0.3578
Epoch 125 trainig loss is  -0.3167  training acc is  0.3617  test acc is  0.3595:  25%|██▌       | 126/500 [00:58<02:53,  2.1
5it/s]
Epoch 125 trainig loss is  -0.3167  training acc is  0.3617  test acc is  0.3595
Epoch 150 trainig loss is  -0.3098  training acc is  0.3624  test acc is  0.3602:  30%|███       | 151/500 [01:10<02:40,  2.1
8it/s]
Epoch 150 trainig loss is  -0.3098  training acc is  0.3624  test acc is  0.3602
Epoch 175 trainig loss is  -0.3045  training acc is  0.3626  test acc is  0.3603:  35%|███▌      | 176/500 [01:21<02:31,  2.
14it/s]
Epoch 175 trainig loss is  -0.3045  training acc is  0.3626  test acc is  0.3603
Epoch 200 trainig loss is  -0.3004  training acc is  0.3623  test acc is  0.3600:  40%|████      | 201/500 [01:33<02:14,  2.
22it/s]
Epoch 200 trainig loss is  -0.3004  training acc is  0.3623  test acc is  0.3600
```

```
Epoch 219 trainig loss is  -0.2979  training acc is  0.3618  test acc is  0.3596:  44%|████▍     | 220/500 [01:42<02:10, 2.15it/s]
Early stopping because valid acc has not been improved for 50 epoch
Epoch 220 trainig loss is  -0.2978  training acc is  0.3618  test acc is  0.3595
```

Out[58]:

```
<__main__.LogisticRegression at 0x20f65400580>
```

4(e)

4(e):

given: $W_1 = 0$, $W_2 = W_1 - \eta g_1$,

$$W_{t+1} = W_t - \eta g_t + \beta(W_t - W_{t-1}) \quad ①$$

WTS:

$$W_{t+1} = W_t - \eta(g_t + \beta g_{t-1} + \dots + \beta^{t-1} g_1), \text{ for any } t \geq 3$$

base case: $t=3$

$$W_4 = W_3 - \eta g_3 + \beta(W_3 - W_2) \quad [①]$$

$$W_3 = W_2 - \eta g_2 + \beta(W_2 - W_1)$$

$$= W_2 - \eta g_2 + \beta(\cancel{W_1} - \eta g_1 - \cancel{W_1})$$

$$= W_2 - \eta g_2 - \beta \eta g_1$$

$$W_3 = W_2 - \eta(g_2 + \beta g_1) \quad ②$$

from ②

$$W_4 = W_3 - \eta g_3 + \beta(W_2 - \eta(g_2 + \beta g_1) - W_2)$$

$$= W_3 - \eta g_3 + \beta(-\eta(g_2 + \beta g_1))$$

$$= W_3 - \eta g_3 - \beta \eta g_2 - \beta^2 \eta g_1$$

$$= W_3 - \eta(g_3 + \beta g_2 + \beta^2 g_1)$$. So base case holds □

Induction step:

Assume $n=t$ holds:

$W_n = W_{n-1} - \eta(g_{n-1} + \beta g_{n-2} + \dots + \beta^{n-1-1\,(=n-2)} g_1)$ by induction hypothesis [IH]

$W_n = W_{n-1} - \eta(g_{n-1}+\beta g_{n-2} + \dots + g_1)$ by induction hypothesis[IH]

WTS: $n=t+1$ holds

$W_{n+1} = W_n - \eta(g_{n-1} + \beta g_{n-2} + \dots + \beta^{n-1} g_1)$,

... continue

$W_{n+1} = W_n - \eta g_n + \beta(W_n - W_{n-1})$ from given

$= W_n - \eta g_n + \beta(\cancel{W_{n-1}} - \eta(g_{n-1} + \beta g_{n-2} + \dots + \beta^{n-2} g_1) - \cancel{W_{n-1}})$ from IH

$= W_n - \eta g_n - \beta\eta(g_{n-1} + \beta g_{n-2} + \dots + \beta^{n-2} g_1)$

$= W_n - \eta g_n - \eta(\beta g_{n-1} + \beta^2 g_{n-2} + \dots + \beta^{n-1} g_1)$

$= W_n - \eta(g_n + \beta g_{n-1} + \dots + \beta^{n-1} g_1)$

So $n=t+1$ holds

So proof of induction is complete

So $W_{t+1} = W_t - \eta(g_t + \beta g_{t-1} + \dots + \beta^{t-1} g_1)$, for any $t \geq 3$, $t \in \mathbb{Z}^+$ $\square$

4(f)

In [73]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_covtype
from  sklearn.metrics  import  log_loss
import torch
from tqdm import  tqdm
from torch.optim import  SGD

class log_loss(torch.nn.Module):
    def __init__(self):
        super(log_loss,self).__init__()
    def forward(self,y,y_pred):
        return -(y*torch.log(y_pred)+(1-y)*torch.log(1-y_pred)).mean()


class LogisticRegression:
    def __init__(self):
        pass

    def fit(self, X, y, X_ts,y_ts,lr=0.2, momentum=0, niter=100):
        print("Training param is LR {}, niter {} , momentum {} ,number of traning example {}".format(lr,niter,momentum,len(X)))
        n_features = X.shape[1]
        self.classes_ = np.unique(y)
        n_classes = len(self.classes_)
        y_onehot = (y-1).long()
        y_onehot = torch.eye(7,requires_grad=True)[y_onehot]
        self.classes_ = np.unique(y)
        self.class2int = dict((c, i) for i, c in enumerate(self.classes_))
        self.intercept_ = torch.autograd.Variable(torch.rand(n_classes),requires_grad=True)
        self.coef_ = torch.autograd.Variable(torch.rand((n_classes,n_features)),requires_grad=True)
        y_1 = y-1
        loss_fu = log_loss()
        pbar = tqdm(range(niter))
        early_stop_e = 50
        record_epoch_for_early_stop = 0
        best_valid_acc = 0
        optimizer = torch.optim.SGD([self.coef_], lr=lr, momentum=momentum)
        for i in range(niter):
            optimizer.zero_grad()

            loss=- loss_fu( y_onehot,self.predict_proba(X))
            if self.coef_.grad is not None:
                self.coef_.grad.zero_() # important: reset the stored gradient to 0
            if self.intercept_.grad is not None:
```

```
                self.intercept_.grad.zero_()
            optimizer.step()
            loss.backward(retain_graph=True)
            self.coef_.data.add_(lr*self.coef_.grad.data)
            output = self.predict(X)
            test_output = self.predict(X_ts)
            test_acc = (test_output==torch.tensor(y_ts)).sum()/len(y_ts)
            if test_acc > best_valid_acc:
                best_valid_acc = test_acc
                record_epoch_for_early_stop=0
            else:
                record_epoch_for_early_stop +=1
                if record_epoch_for_early_stop > early_stop_e:
                    print("Early stopping because valid acc has not been improved for {} epoch".format(early_stop_e))
                    print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(ou
                    break
            if i % 25 ==0:
                print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(output
            # pbar.set_description("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.i
        return self


    def predict_proba(self, X):
        X=sklearn.preprocessing.normalize(X) # normalize X to prevent overflow
        X = torch.tensor(X).T.float()
        coef_trans = self.coef_.T
        coef_trans.requies_grad=True
        eOY_prime = X.T @ coef_trans # X^T * W or o
        eOY_prime_subtracted_max = eOY_prime - torch.max(eOY_prime)
        output = torch.softmax(eOY_prime_subtracted_max,1)
        return output


    def predict(self, X):
        X_softemaxed_prob = self.predict_proba(X)
        return  X_softemaxed_prob.argmax(1)
```

4(f)

I import SGD optimizer paramter from pytorch and given our coeffients to the optimizer. I did three experiments with same leanring rate but differnet momentum. The best log loss I got is loss is -0.3445 training accuracy is 0.3660 test accuracy is 0.3640

In [75]:
```
X, y = fetch_covtype(return_X_y=True)
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.7, random_state=42)
cls = LogisticRegression()
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.1,momentum=0.9)
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.1,momentum=0.95)
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.2,momentum=0.99)
```

```
  0%|          | 0/500 [00:00<?, ?it/s]
Training param is LR 0.1, niter 500 , momentum 0.9 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4393  training acc is  0.0714  test acc is  0.0711
Epoch 25 trainig loss is  -0.4231  training acc is  0.1961  test acc is  0.1965
Epoch 50 trainig loss is  -0.4083  training acc is  0.3490  test acc is  0.3469
Epoch 75 trainig loss is  -0.3949  training acc is  0.3634  test acc is  0.3615
Epoch 100 trainig loss is  -0.3828  training acc is  0.3660  test acc is  0.3640
Epoch 125 trainig loss is  -0.3721  training acc is  0.3660  test acc is  0.3640

  0%|          | 0/500 [00:00<?, ?it/s]
Early stopping because valid acc has not been improved for 50 epoch
Epoch 148 trainig loss is  -0.3632  training acc is  0.3660  test acc is  0.3640
Training param is LR 0.1, niter 500 , momentum 0.95 ,number of traning example 174303
Epoch 0 trainig loss is  -0.3898  training acc is  0.3660  test acc is  0.3640
Epoch 25 trainig loss is  -0.3779  training acc is  0.3660  test acc is  0.3640
Epoch 50 trainig loss is  -0.3673  training acc is  0.3660  test acc is  0.3640

  0%|          | 0/500 [00:00<?, ?it/s]
Early stopping because valid acc has not been improved for 50 epoch
Epoch 51 trainig loss is  -0.3669  training acc is  0.3660  test acc is  0.3640
Training param is LR 0.2, niter 500 , momentum 0.99 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4143  training acc is  0.0596  test acc is  0.0613
Epoch 25 trainig loss is  -0.3859  training acc is  0.3582  test acc is  0.3560
Epoch 50 trainig loss is  -0.3637  training acc is  0.3660  test acc is  0.3640
Epoch 75 trainig loss is  -0.3468  training acc is  0.3660  test acc is  0.3640
Early stopping because valid acc has not been improved for 50 epoch
Epoch 79 trainig loss is  -0.3445  training acc is  0.3660  test acc is  0.3640
```

Out[75]:
```
<__main__.LogisticRegression at 0x20f684080a0>
```

4(G)

In [105…

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_covtype
from  sklearn.metrics  import  log_loss
import torch
from tqdm import  tqdm
from torch.optim import  SGD
from  sklearn.preprocessing import  StandardScaler
class log_loss(torch.nn.Module):
    def __init__(self):
        super(log_loss,self).__init__()
    def forward(self,y,y_pred):
        return -(y*torch.log(y_pred)+(1-y)*torch.log(1-y_pred)).mean()


class LogisticRegression:
    def __init__(self):
        pass

    def fit(self, X,  y,  X_ts,y_ts,lr=0.2, momentum=0, niter=100,norm=True):
        print("Training param is LR {}, niter {} , momentum {} ,number of traning example {}".format(lr,niter,momentum,len(X)))
        n_features = X.shape[1]
        self.classes_ = np.unique(y)
        n_classes = len(self.classes_)
        y_onehot =  (y-1).long()
        y_onehot =  torch.eye(7,requires_grad=True)[y_onehot]
        self.classes_ = np.unique(y)
        self.class2int = dict((c,  i) for i,  c in enumerate(self.classes_))
        self.intercept_ = torch.autograd.Variable(torch.zeros(n_classes)+0.01,requires_grad=True)
        self.coef_ = torch.autograd.Variable(torch.zeros((n_classes,n_features))+0.01,requires_grad=True)
        y_1 = y-1
        loss_fu = log_loss()
        # pbar = tqdm(range(niter))
        early_stop_e = 100
        record_epoch_for_early_stop = 0
        best_valid_acc = 0
        # optimizer = torch.optim.SGD([self.coef_],  lr=lr,  momentum=momentum)
        if norm:
            scaler = StandardScaler()
            scaler.fit(X)
            X= scaler.transform(X)
            X_ts= scaler.transform(X_ts)

        for i in range(niter):
```

```
            # optimizer.zero_grad()

            loss=- loss_fu( y_onehot,self.predict_proba(X))
            if self.coef_.grad is not None:
                self.coef_.grad.zero_() # important: reset the stored gradient to 0
            if self.intercept_.grad is not None:
                self.intercept_.grad.zero_()
            # optimizer.step()
            loss.backward(retain_graph=True)
            self.coef_.data.add_(-lr*self.coef_.grad.data)
            output = self.predict(X)
            test_output = self.predict(X_ts)
            test_acc = (test_output==torch.tensor(y_ts)).sum()/len(y_ts)
            if test_acc > best_valid_acc:
                best_valid_acc = test_acc
                record_epoch_for_early_stop=0
            else:
                record_epoch_for_early_stop +=1
                if record_epoch_for_early_stop > early_stop_e:
                    print("Early stopping because valid acc has not been improved for {} epoch".format(early_stop_e))
                    print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(ou
                    break
            if i % 25 ==0:
                print("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.item(),(output
            # pbar.set_description("Epoch {} trainig loss is  {:.4f}  training acc is  {:.4f}  test acc is  {:.4f}".format(i,loss.i
        return self


    def predict_proba(self, X):
        X = torch.tensor(X).T.float()
        coef_trans = self.coef_.T
        coef_trans.requies_grad=True
        eOY_prime = X.T @ coef_trans # X^T * W or o
        eOY_prime_subtracted_max = eOY_prime - torch.max(eOY_prime)
        output = torch.softmax(eOY_prime_subtracted_max,1)
        return output


    def predict(self, X):
        X_softemaxed_prob = self.predict_proba(X)
        return  X_softemaxed_prob.argmax(1)
```

In [108…

```
X, y = fetch_covtype(return_X_y=True)
```

```
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.7, random_state=42)
cls = LogisticRegression()
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.02,momentum=0.9,norm=False)
cls.fit(X_tr,torch.tensor(y_tr),X_ts,y_ts,niter =500,lr=0.02,momentum=0.9)
```

```
Training param is LR 0.02, niter 500 , momentum 0.9 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4101  training acc is  0.0603  test acc is  0.0620
Epoch 25 trainig loss is  nan  training acc is  0.0000  test acc is  0.0000
Epoch 50 trainig loss is  nan  training acc is  0.0000  test acc is  0.0000
Epoch 75 trainig loss is  nan  training acc is  0.0000  test acc is  0.0000
Epoch 100 trainig loss is  nan  training acc is  0.0000  test acc is  0.0000
Early stopping because valid acc has not been improved for 100 epoch
Epoch 101 trainig loss is  nan  training acc is  0.0000  test acc is  0.0000
Training param is LR 0.02, niter 500 , momentum 0.9 ,number of traning example 174303
Epoch 0 trainig loss is  -0.4101  training acc is  0.3515  test acc is  0.3482
Epoch 25 trainig loss is  -0.4145  training acc is  0.3516  test acc is  0.3481
Epoch 50 trainig loss is  -0.4192  training acc is  0.3515  test acc is  0.3482
Epoch 75 trainig loss is  -0.4240  training acc is  0.3514  test acc is  0.3482
Epoch 100 trainig loss is  -0.4290  training acc is  0.3514  test acc is  0.3481
Early stopping because valid acc has not been improved for 100 epoch
Epoch 109 trainig loss is  -0.4309  training acc is  0.3512  test acc is  0.3481
```

Out[108]:
```
<__main__.LogisticRegression at 0x20ff479d1f0>
```

4(G) To avoid the difference brought by weights random inistialization , I initialize all weights to be 0.01 .And do two comparison between two experiments , one without standard normalization and one with standard normalization .It seems that standard normalization would help model to avoid gradient vanish and help model more faster to converge

In [ ]:

In [ ]: