a) part of the code

b) mean accuracy (over 10 trials) using GD without stopping criteria: 0.76.

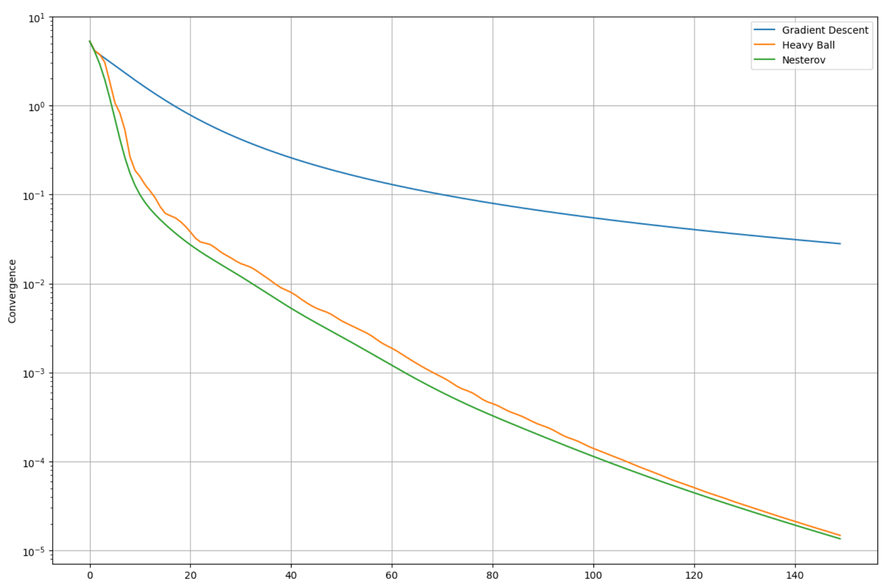
c) mean iterations using GD: 707
mean accuracy using GD: 0.72

d) I used beta = 0.91

1) Heavy Ball: mean iterations using heavy ball: 71.
mean accuracy using heavy ball: 0.74.

2) Nesterov: mean iterations using Nesterov's method: 64.
mean accuracy using Nesterov's method: 0.75.

I would choose Nesterov's method since it can achieve similar accuracies in the least number of iterations as compared to other methods. For the graph below, I used log scale for y-axis as a standard popular plotting method.



Gradient used:

$$\nabla_W = X^T \cdot (P - Y) + \lambda \cdot W$$

Where:

- X is the matrix of training examples.
- P is the matrix of predicted probabilities (the softmax probabilities for each class).
- Y is the matrix of one-hot encoded true class labels.
- W is the matrix of weights.
- λ is the regularization parameter.

hw6_final

November 28, 2023

[ ]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

# loading dataset
data = loadmat('imagenet_data.mat')
X = data['features']
y = data['labels'].flatten()
y = y.astype(int)

# normalizing data
X_mean = np.mean(X, axis=0)
X_norm = X - X_mean
X_norm /= np.linalg.norm(X_norm, axis=1, keepdims=True)
X_norm = np.hstack((X_norm, np.ones((X_norm.shape[0], 1))))

#defining constants
mu = 0.01
lambda_ = 0.1
trials = 10
no_iters = 500
epsilon = 1e-2
beta = 0.91
max_iters = 1000
iters_plot = 150
N = X.shape[0]  # no of data pts
C = len(np.unique(y))  # no of classes
D = X.shape[1] + 1  # no of featues + 1 (bias)
train_size = int(N*0.9)
test_size = N - train_size
```

[ ]:
```
def grad_loss(x_train, y_train, y_enc, w, wx, lambda_):
    ##gradient
    wx_max = np.max(wx, axis=0)
    exp_diff = np.exp(wx - wx_max)
    pred = exp_diff / np.sum(exp_diff, axis=0)
    diff_y = pred - y_enc.T
    grad = np.dot(diff_y, x_train) + lambda_ * w
```

```python
    ##loss
    log_sum = np.log(np.sum(exp_diff, axis=0))
    g_truth = wx[y_train, np.arange(wx.shape[1])]
    loss = (lambda_ / 2) * np.sum(w**2) + np.sum(wx_max - g_truth + log_sum)

    return grad, loss

def predict(w, x):
   return np.argmax(np.dot(x, w.T), axis=1)
```

[ ]:
```python
### gd without stoppping criteria ###

accuracy_ = np.zeros(trials)

for i in range(trials):

    random_ = np.random.permutation(N)
    x_train = X_norm[random_[:train_size], :]
    y_train = y[random_[:train_size]]
    x_test = X_norm[random_[train_size:], :]
    y_test = y[random_[train_size:]]

    w = np.zeros((C, D))
    w_old = np.zeros((C, D))
    y_train_enc = np.eye(C)[y_train]

    for j in range(no_iters):

        w_x = np.dot(w, x_train.T)
        grad_, loss_ = grad_loss(x_train, y_train, y_train_enc, w, w_x, lambda_)

        w_new = w - mu * grad_
        w = w_new

    y_pred = predict(w, x_test)
    acc_tr = np.mean(y_pred == y_test)
    accuracy_[i] = acc_tr


mean_acc = np.mean(accuracy_)
print(f"Mean accuracy using GD without stopping criteria: {mean_acc:.2f}")
```

[ ]:
```python
### gd with stoppping criteria ###

iter_tracker = np.zeros(trials)
accuracy_list = np.zeros(trials)
```

```python
for i in range(trials):

    random_ = np.random.permutation(N)
    x_train = X_norm[random_[:train_size], :]
    y_train = y[random_[:train_size]]
    x_test = X_norm[random_[train_size:], :]
    y_test = y[random_[train_size:]]

    w = np.zeros((C, D))
    w_old = np.zeros((C, D))
    y_train_enc = np.eye(C)[y_train]

    for j in range(max_iters):

        w_x = np.dot(w, x_train.T)
        grad_, loss_ = grad_loss(x_train, y_train, y_train_enc, w, w_x, lambda_)
        exp1 = np.linalg.norm(grad_)**2
        exp2 = 1 + np.abs(loss_)

        if exp1 <= epsilon * exp2:
            break

        w_new = w - mu * grad_
        w = w_new

    y_pred = predict(w, x_test)
    acc_tr = np.mean(y_pred == y_test)
    accuracy_list[i] = acc_tr
    iter_tracker[i] = j


mean_iter = np.mean(iter_tracker)
print(f"Mean iterations using GD: {int(mean_iter)}")

mean_acc = np.mean(accuracy_list)
print(f"Mean accuracy using GD: {mean_acc:.2f}")
```

[ ]:
```python
### heavy ball ###

iter_tracker_hb = np.zeros(trials)
accuracy_list_hb = np.zeros(trials)

for i in range(trials):
    random_ = np.random.permutation(N)
    x_train = X_norm[random_[:train_size], :]
    y_train = y[random_[:train_size]]
```

```
    x_test = X_norm[random_[train_size:], :]
    y_test = y[random_[train_size:]]

    w = np.zeros((C, D))
    w_old = np.zeros((C, D))
    y_train_enc = np.eye(C)[y_train]


    for j in range(max_iters):
        w_x = np.dot(w, x_train.T)
        grad_, loss_ = grad_loss(x_train, y_train, y_train_enc, w, w_x, lambda_)
        exp1 = np.linalg.norm(grad_)**2
        exp2 = 1 + np.abs(loss_)

        if exp1 <= epsilon * exp2:
            break

        w_new = w - mu * grad_ + beta * (w - w_old)
        w_old = w
        w = w_new

    y_pred = predict(w, x_test)
    acc_tr = np.mean(y_pred == y_test)
    accuracy_list_hb[i] = acc_tr
    iter_tracker_hb[i] = j


mean_iter = np.mean(iter_tracker_hb)
print(f"Mean iterations using heavy ball: {int(mean_iter)}")

mean_acc = np.mean(accuracy_list_hb)
print(f"Mean accuracy using heavy ball: {mean_acc:.2f}")
```

[ ]:
```
### Nesterov ###

iter_tracker_nes = np.zeros(trials)
accuracy_list_nes = np.zeros(trials)
for i in range(trials):
    random_ = np.random.permutation(N)
    x_train = X_norm[random_[:train_size], :]
    y_train = y[random_[:train_size]]
    x_test = X_norm[random_[train_size:], :]
    y_test = y[random_[train_size:]]

    w = np.zeros((C, D))
    w_old = np.zeros((C, D))
    y_train_enc = np.eye(C)[y_train]
```

```
    for j in range(max_iters):
        z = w + beta * (w - w_old)
        z_x = np.dot(z, x_train.T)
        grad_z, loss_z = grad_loss(x_train, y_train, y_train_enc, z, z_x,␣
↪lambda_)

        exp1 = np.linalg.norm(grad_z)**2
        exp2 = 1 + np.abs(loss_z)

        if exp1 <= epsilon * exp2:
            break

        w_new = z - mu * grad_z
        w_old = w
        w = w_new

        y_pred = predict(w, x_test)
        acc_tr = np.mean(y_pred == y_test)
        accuracy_list_nes[i] = acc_tr
        iter_tracker_nes[i] = j

mean_iter = np.mean(iter_tracker_nes)
print(f"Mean iterations using Nesterov's method: {int(mean_iter)}")

mean_acc = np.mean(accuracy_list_nes)
print(f"Mean accuracy using Nesterov's method: {mean_acc:.2f}")
```

[ ]:
```
def calc_conv_gd(w_gd, x_train, y_train):
    conv_ = np.zeros(iters_plot)
    y_train_enc = np.eye(C)[y_train - 1]

    for i in range(iters_plot):
        w_x = np.dot(w_gd, x_train.T)
        grad_, loss_ = grad_loss(x_train, y_train,  y_train_enc, w_gd, w_x,␣
↪lambda_)
        exp1 = np.linalg.norm(grad_)**2
        exp2 = 1 + np.abs(loss_)

        conv_[i] = exp1 / exp2
        w_new = w_gd - mu * grad_
        w_gd = w_new

    return conv_
```

```python
def calc_conv_hb(w_hb, x_train, y_train):
    conv_ = np.zeros(iters_plot)
    w_old = np.zeros((C, D))
    y_train_enc = np.eye(C)[y_train - 1]

    for i in range(iters_plot):
        w_x = np.dot(w_hb, x_train.T)
        grad_, loss_ = grad_loss(x_train, y_train,  y_train_enc, w_hb, w_x, 
 ↪lambda_)
        exp1 = np.linalg.norm(grad_)**2
        exp2 = 1 + np.abs(loss_)

        conv_[i] = exp1 / exp2
        w_new = w_hb - mu * grad_ + beta * (w_hb - w_old)
        w_old = w_hb
        w_hb= w_new

    return conv_

def calc_conv_nes(w_nes, x_train, y_train):
    conv_ = np.zeros(iters_plot)
    w_old = np.zeros_like(w_nes)
    y_train_enc = np.eye(C)[y_train - 1]

    for i in range(iters_plot):
        z = w_nes + beta * (w_nes - w_old)
        z_x = np.dot(z, x_train.T)
        grad_z, loss_z = grad_loss(x_train, y_train, y_train_enc, z, z_x, 
 ↪lambda_)
        exp1 = np.linalg.norm(grad_z)**2
        exp2 = 1 + np.abs(loss_z)

        conv_[i] = exp1 / exp2
        w_new = z - mu * grad_z
        w_old = w_nes
        w_nes= w_new

    return conv_
```

[ ]:

```python
w_gd = np.zeros((C,D))
w_hb = np.zeros((C,D))
w_nes = np.zeros((C,D))

random_ = np.random.permutation(N)
x_train = X_norm[random_[:train_size], :]
y_train = y[random_[:train_size]]
```

```
conv_gd = calc_conv_gd(w_gd, x_train, y_train)
conv_hb = calc_conv_hb(w_hb, x_train, y_train)
conv_nest = calc_conv_nes(w_nes, x_train, y_train)

plt.figure(figsize=(15, 10))

plt.yscale('log')
plt.plot(range(iters_plot), conv_gd, label='Gradient Descent')
plt.plot(range(iters_plot), conv_hb, label='Heavy Ball')
plt.plot(range(iters_plot), conv_nest, label='Nesterov')

plt.xlabel('Iterations')
plt.ylabel('Convergence')
plt.legend()
plt.grid(True)
plt.show()
```