Nirbhay Sharma (B19CSE114)

Optimization for Machine Learning

Que-1

Code

```
import numpy as np
import copy, sys
import pandas as pd

roll_no_last = 4

def calculate_f(x:np.array, weights:np.array, y:np.array) -> int:
    return ((np.dot(x,weights) - y)**2).sum() / (2 * x.shape[0])

def grad_f(x:np.array, weights:np.array, y:np.array) -> np.array:
    common_term = np.dot(x,weights) - y # N x 1, x: N x 5
    gradient = np.dot(x.T, common_term) # 5 x 1
    gradient = gradient / x.shape[0]
    return gradient

def check_armijo_wolf_cond(w_k: np.array, d_k: np.array, x:np.array,
y:np.array, alpha:float, beta1:float, beta2:float) -> bool:
    xk_alp_dk = w_k + alpha * d_k;
    f_grad = grad_f(x, w_k, y);
    f_grad_alph = grad_f(x, xk_alp_dk, y)

    armijo_left = calculate_f(x, xk_alp_dk, y);
    armijo_right = calculate_f(x, w_k, y) + alpha * beta1
*np.dot(f_grad.T,d_k);

    wolf_left = np.dot(f_grad_alph.T, d_k)
    wolf_right = beta2 * np.dot(f_grad.T,d_k);

    return (armijo_left <= armijo_right) and (wolf_left >= wolf_right)


def steepest_direction(w_k:np.array, d_k:np.array, x:np.array, y:np.array,
beta1: float, beta2: float, r:float) -> float:
    alpha = 1;
    while not check_armijo_wolf_cond(w_k, d_k, x, y, alpha, beta1, beta2):
        alpha = alpha * r
    return alpha;

def find_w(w_0:np.array, x:np.array, y:np.array, beta1:float, beta2:float,
epsilon:float, r:float) -> np.array:
    w_k = w_0
    d_k = -grad_f(x, w_k, y)
```

```
    norm_grad = np.linalg.norm(-d_k)

    iterations = 0
    while norm_grad > epsilon:
        alpha = steepest_direction(w_k, d_k, x, y, beta1, beta2, r);
        w_k = w_k + alpha * d_k
        d_k = -grad_f(x, w_k, y)
        norm_grad = np.linalg.norm(-d_k)
        iterations += 1
        if iterations == 2000:
            break;
    return w_k, iterations


beta1 = 1e-4;
beta2 = 0.9;
epsilon = 0.01
r = 0.5

data = pd.read_excel(sys.argv[1])
data = np.array(data)
w_0 = np.array([[0] for _ in range(data.shape[1]-1)], dtype=np.float64)
x = data[:,:-1]
y = data[:,-1:]
print(x.shape, y.shape)
w, iterations = find_w(w_0, x, y, beta1, beta2, epsilon, r)


# print('\n')
print("weight values")
print(w)

print(f"total_iterations: {iterations}")
# print(f"final objective function value: {calculate_f(x,w,y)}")

R = 14
r = 4

new_input = np.array([[R + r/10],[100 + R + 2*r/10],[R-1 + r/10],[54.3],[r
+ R/100]]).reshape(1,-1)
print(new_input)
print("output for new input")
print(np.dot(new_input,w))

"""
(205, 5) (205, 1)
weight values
[[-107.39959718]
 [ 470.7610122 ]
 [-272.51752862]
 [-745.75614296]
 [  15.32276041]]
total_iterations: 2000
[[ 14.4  114.8   13.4   54.3    4.14]]
```

```
output for new input
[[8413.95278305]]
"""
```

Que-2

Code

```
import numpy as np
import copy, sys, random
import pandas as pd

roll_no_last = 4

class dataset:
    def __init__(self, x:np.array, y:np.array, batch_size:int):
        self.x = x
        self.y = y
        self.bs = batch_size
        self.batch_index = 0
        self.no_batch = x.shape[0] // batch_size + (0 if x.shape[0] %
batch_size == 0 else 1)
        print(f"Total number of batches: {self.no_batch}")

    def __next__(self):
        cur_dataset = (self.x[self.batch_index * self.bs: (self.batch_index
+ 1) * self.bs], self.y[self.batch_index * self.bs: (self.batch_index + 1)
* self.bs])
        self.batch_index = (self.batch_index + 1) % self.no_batch
        return cur_dataset

def calculate_f(x:np.array, weights:np.array, y:np.array) -> int:
    return ((np.dot(x,weights) - y)**2).sum() / (2 * x.shape[0])

def grad_f(x:np.array, weights:np.array, y:np.array) -> np.array:
    common_term = np.dot(x,weights) - y # N x 1, x: N x 5
    gradient = np.dot(x.T, common_term) # 5 x 1
    gradient = gradient / x.shape[0]
    return gradient

def find_w(w_k:np.array, x:np.array, y:np.array, iteration:int) ->
np.array:
    d_k = -grad_f(x, w_k, y)
    alpha = 1 / (iteration + 1)
    w_k = w_k + alpha * d_k
    return w_k

beta1 = 1e-4;
beta2 = 0.9;
epsilon = 0.001
r = 0.5
```

```
batch_size = 10

data = pd.read_excel(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
x = x.astype(np.float64)
y = y.astype(np.float64)
x = x/1e2
y = y/1e3
w_0 = np.array([[0] for _ in range(data.shape[1]-1)], dtype=np.float64)
data_loader = list(zip(x,y))

iteration = 0
while True:
    random_samples = random.sample(data_loader, batch_size)
    dx,dy = zip(*random_samples)
    dx = np.vstack(dx)
    dy = np.vstack(dy)
    cur_w = find_w(w_0, dx, dy, iteration)
    iteration += 1
    if calculate_f(x, cur_w, y) < calculate_f(x, w_0, y):
        w_0 = cur_w
    if iteration == 100:
        break
print("weight values")
print(w_0)
print(f"total_iterations: {iteration}")

R = 14
r = 4
new_input = np.array([[R + r/10],[100 + R + 2*r/10],[R-1 + r/10],[54.3],[r
+ R/100]]).reshape(1,-1)
print(new_input)
new_input = new_input / 1e2
print('output on sample input')
print(np.dot(new_input,w_0) * 1e3)

"""
weight values
[[2.82781254]
 [5.12214926]
 [1.70893591]
 [1.32076932]
 [0.09105886]]
total_iterations: 100
[[ 14.4  114.8   13.4   54.3    4.14]]
output on sample input
[[7237.37734854]]
"""
```

Que-3

Code

```
import numpy as np
import copy, sys
import pandas as pd

def calculate_f(x:np.array, weights:np.array, y:np.array) -> int:
    """
    x.shape = (N,1)
    w.shape = (3,1)
    """
    x = np.hstack([x**2, x, np.ones((x.shape[0],1),dtype=np.float64)])
    return ((np.dot(x,weights) - y)**2).sum() / (2 * x.shape[0])

def grad_f(x:np.array, weights:np.array, y:np.array) -> np.array:
    new_x = np.hstack([x**2, x, np.ones((x.shape[0],1),dtype=np.float64)])
    common_term = np.dot(new_x,weights) - y
    grads = np.dot(new_x.T, common_term)
    grads = grads / y.shape[0]
    return grads

def check_armijo_wolf_cond(w_k: np.array, d_k: np.array, x:np.array,
y:np.array, alpha:float, beta1:float, beta2:float) -> bool:
    xk_alp_dk = w_k + alpha * d_k;
    f_grad = grad_f(x, w_k, y);
    f_grad_alph = grad_f(x, xk_alp_dk, y)

    armijo_left = calculate_f(x, xk_alp_dk, y);
    armijo_right = calculate_f(x, w_k, y) + alpha * beta1
*np.dot(f_grad.T,d_k);

    wolf_left = np.dot(f_grad_alph.T, d_k)
    wolf_right = beta2 * np.dot(f_grad.T,d_k);

    return (armijo_left <= armijo_right) and (wolf_left >= wolf_right)


def steepest_direction(w_k:np.array, d_k:np.array, x:np.array, y:np.array,
beta1: float, beta2: float, r:float) -> float:
    alpha = 1;
    while not check_armijo_wolf_cond(w_k, d_k, x, y, alpha, beta1, beta2):
        alpha = alpha * r
    return alpha;

def find_w(w_0:np.array, x:np.array, y:np.array, beta1:float, beta2:float,
epsilon:float, r:float) -> np.array:
    w_k = w_0
    d_k = -grad_f(x, w_k, y)
    norm_grad = np.linalg.norm(-d_k)

    iterations = 0
    while norm_grad > epsilon:
        alpha = steepest_direction(w_k, d_k, x, y, beta1, beta2, r);
```

```
        w_k = w_k + alpha * d_k
        d_k = -grad_f(x, w_k, y)
        norm_grad = np.linalg.norm(-d_k)
        iterations += 1
        if iterations == 1000:
            break;
    return w_k, iterations


beta1 = 1e-4;
beta2 = 0.9;
epsilon = 0.01
r = 0.5

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
x = x / 1e2
y = y / 1e2
w_0 = np.array([[0] for _ in range(2*x.shape[1] + 1)], dtype=np.float64)
w, iterations = find_w(w_0, x, y, beta1, beta2, epsilon, r)

# print('\n')
print("weight values")
print(w)

print(f"total_iterations: {iterations}")

print(f"objective function value: {calculate_f(x, w, y)}")


"""
weight values
[[ 1.43123329e-02]
 [ 1.06933490e-02]
 [ 2.74550002e-03]
 [-1.53774443e-03]
 [ 7.43829578e-05]
 [ 5.01807844e-04]
 [ 1.43165190e-03]
 [ 6.07989622e-03]
 [ 3.80186701e-03]
 [ 6.51654657e-03]
 [ 2.29182350e-03]
 [ 1.45989026e-04]
 [ 5.89108817e-04]
 [ 3.54756960e-03]
 [ 2.92014144e-03]
 [ 5.07208941e-03]
 [ 7.27987673e-04]]
total_iterations: 1000
objective function value: 0.009841334196678
"""
```

Que-4

Code

```
import numpy as np
import copy, sys, random
import pandas as pd

roll_no_last = 4

def calculate_f(x:np.array, weights:np.array, y:np.array) -> int:
    """
    x.shape = (N,1)
    w.shape = (3,1)
    """
    x = np.hstack([x**2, x, np.ones((x.shape[0],1),dtype=np.float64)])
    return ((np.dot(x,weights) - y)**2).sum() / (2 * x.shape[0])

def grad_f(x:np.array, weights:np.array, y:np.array) -> np.array:
    new_x = np.hstack([x**2, x, np.ones((x.shape[0],1),dtype=np.float64)])
    common_term = np.dot(new_x,weights) - y
    grads = np.dot(new_x.T, common_term)
    grads = grads / y.shape[0]
    return grads

def find_w(w_k:np.array, x:np.array, y:np.array, iteration:int) ->
np.array:
    d_k = -grad_f(x, w_k, y)
    alpha = 1 / (iteration + 1)
    w_k = w_k + alpha * d_k
    return w_k

batch_size = 20

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
x = x / 1e3
y = y / 1e6
w_0 = np.array([[0] for _ in range(2*x.shape[1] + 1)], dtype=np.float64)
data_loader = list(zip(x,y))

iteration = 0
while True:
    data = random.sample(data_loader, batch_size)
    dx,dy = zip(*data)
    dx = np.vstack(dx)
    dy = np.vstack(dy)
    cur_w = find_w(w_0, dx, dy, iteration)
    iteration += 1
```

```
    if calculate_f(x, cur_w, y) < calculate_f(x, w_0, y):
        w_0 = cur_w
    if iteration == 100:
        break
print("weight values")
print(w_0)
print(f"total_iterations: {iteration}")

print(f"objective function value {calculate_f(x, w_0, y)}")

"""
weight values
[[2.09461615e-06]
 [2.14663145e-07]
 [1.99289825e-08]
 [2.09493519e-07]
 [2.35191377e-09]
 [6.86626558e-06]
 [5.93245427e-06]
 [4.05981082e-07]
 [4.94062729e-06]
 [1.38726089e-06]
 [2.45857463e-07]
 [1.41617772e-06]
 [1.70268829e-07]
 [8.20296830e-06]
 [7.53024647e-06]
 [2.21742784e-06]
 [9.68303089e-06]]
total_iterations: 100
objective function value 1.4292393913900596e-10
"""
```

To run the code:

Use the following commands on terminal

```
python que1.py car_price.xlsx
python que2.py car_price.xlsx
python que3.py Concrete_Data_Yeh.csv
python que4.py Concrete_Data_Yeh.csv
```