Nirbhay Sharma (B19CSE114)

Optimization for machine learning

Note: Outputs of all the codes are provided at the end of the code

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:
    first_term = (x[:,:-1] * (np.dot(x,weights) - y)).sum()/ x.shape[0]
    second_term = (np.dot(x,weights) - y).sum() / x.shape[0]
    return np.array([[first_term],[second_term]],dtype=np.float64)

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
w_0 = np.array([[0],[0]], dtype=np.float64)

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
x = np.hstack([x,np.ones(x.shape)])
w, iterations = find_w(w_0, x, y, beta1, beta2, epsilon, r)


# print('\n')
print(w)

print(f"total_iterations: {iterations}")

print(calculate_f(x, w, y))

"""
[[3.67217187]
 [0.0716981 ]]
total_iterations: 3
2.0027539536886514
"""
```

Que-2

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:
    first_term = (x[:,:-1] * (np.dot(x,weights) - y)).sum()/ x.shape[0]
    second_term = (np.dot(x,weights) - y).sum() / x.shape[0]
    return np.array([[first_term],[second_term]],dtype=np.float64)

def return_dataset(x:np.array, y:np.array, batch_size:int):
    total_batches = int(x.shape[0] / batch_size)
    remaining_points = x.shape[0] - total_batches * batch_size
    for batch in range(total_batches):
        yield (x[batch:batch+batch_size],y[batch:batch+batch_size])
    if remaining_points != 0:
        yield (x[-remaining_points:],y[-remaining_points:])

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 = 21
w_0 = np.array([[0],[0]], dtype=np.float64)

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
x = np.hstack([x,np.ones(x.shape)])
data_loader = return_dataset(x, y, batch_size)

iteration = 0
while calculate_f(x, w_0, y) > 2.003:
    for idx, (dx,dy) in enumerate(data_loader):
        cur_w = find_w(w_0, dx, dy, iteration)
        iteration += 1
        if calculate_f(x, cur_w, y) < 2.003:
            w_0 = cur_w
            break;
        if calculate_f(x, cur_w, y) < calculate_f(x, w_0, y):
            w_0 = cur_w
print(w_0)
print(f"total_iterations: {iteration}")

print(calculate_f(x, w_0, y))
```

```
"""
[[3.69306165]
 [0.07112588]]
total_iterations: 54
2.002996584560328
"""
```

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,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,dtype=np.float64)])
    common_term = (np.dot(new_x,weights) - y) / x.shape[0]
    first_term = (x ** 2 * common_term).sum()
    second_term = (x * common_term).sum()
    third_term = (common_term).sum()
    return np.array([[first_term],[second_term],
[third_term]],dtype=np.float64)

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
w_0 = np.array([[0],[0],[0]], dtype=np.float64)

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
w, iterations = find_w(w_0, x, y, beta1, beta2, epsilon, r)


# print('\n')
print(w)

print(f"total_iterations: {iterations}")

print(calculate_f(x, w, y))

"""
[[-0.00485504]
 [ 3.67176844]
 [ 0.07355137]]
total_iterations: 82
2.0027581447644307
"""
```

Que-4

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:
    """
    x.shape = (N,1)
    w.shape = (3,1)
    """
    x = np.hstack([x**2, x, np.ones(x.shape,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,dtype=np.float64)])
    common_term = (np.dot(new_x,weights) - y) / x.shape[0]
    first_term = (x ** 2 * common_term).sum()
    second_term = (x * common_term).sum()
    third_term = (common_term).sum()
    return np.array([[first_term],[second_term],
[third_term]],dtype=np.float64)

def return_dataset(x:np.array, y:np.array, batch_size:int):
    total_batches = int(x.shape[0] / batch_size)
    remaining_points = x.shape[0] - total_batches * batch_size
    for batch in range(total_batches):
        yield (x[batch:batch+batch_size],y[batch:batch+batch_size])
    if remaining_points != 0:
        yield (x[-remaining_points:],y[-remaining_points:])

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
w_0 = np.array([[0],[0],[0]], dtype=np.float64)

data = pd.read_csv(sys.argv[1])
data = np.array(data)
x = data[:,:-1]
y = data[:,-1:]
data_loader = return_dataset(x, y, batch_size)

iteration = 0
while calculate_f(x, w_0, y) > 2.003:
    for idx, (dx,dy) in enumerate(data_loader):
        cur_w = find_w(w_0, dx, dy, iteration)
        iteration += 1
        if calculate_f(x, cur_w, y) < 2.003:
            w_0 = cur_w
            break;
        if calculate_f(x, cur_w, y) < calculate_f(x, w_0, y):
            w_0 = cur_w
    if iteration == 20000:
        break
    data_loader = return_dataset(x, y, batch_size)
print(w_0)
print(f"total_iterations: {iteration}")

print(calculate_f(x, w_0, y))

"""
[[-0.03801777]
 [ 3.67846218]
 [ 0.15390869]]
total_iterations: 20000
2.005129382213879
"""
```