GradientDescent

November 26, 2024

1 Data Generation

[1]:

```
%matplotlib inline
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

dim_theta = 10
data_num = 1000
scale = .1

theta_true = np.ones((dim_theta,1))
print('True theta:', theta_true.reshape(-1))

A = np.random.uniform(low=-1.0, high=1.0, size=(data_num,dim_theta))
y_data = A @ theta_true + np.random.normal(loc=0.0, scale=scale,␣
 ↪size=(data_num, 1))

A_test = np.random.uniform(low=-1.0, high=1.0, size=(50, dim_theta))
y_test = A_test @ theta_true + np.random.normal(loc=0.0, scale=scale, size=(50,␣
 ↪1))
```

```
True theta: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

2 Solving for the exact mean squared loss (solving Ax = b)

[85]:

```
'''
Using numpy's lstsq() to get the closed-form least squares solution
'''
theta_pred = la.lstsq(A, y_data)[0]

print('Empirical theta: ', theta_pred.reshape(-1))
```

```
Empirical theta:  [1.01117113 0.99879624 0.99707909 1.01475117 1.00365197
0.98530922
 1.00109998 0.99715709 0.99246624 0.99988042]
```

3 SGD Variants Noisy Function

[11]:

```
batch_size = 1
max_iter = 1000
lr = 0.001
theta_init = np.random.random((10,1)) * 0.1
```

[15]:

```
def noisy_val_grad(theta_hat, data_, label_, deg_=2.):
    gradient = np.zeros_like(theta_hat)
    loss = 0

    for i in range(data_.shape[0]):
        x_ = data_[i, :].reshape(-1,1)
        y_ = label_[i, 0]
        err = np.sum(x_ * theta_hat) - y_

        grad = deg_ * np.abs(err) ** (deg_ - 1) * np.sign(err) * x_
        l = np.abs(err) ** deg_

        loss += l / data_.shape[0]
        gradient += grad / data_.shape[0]

    return loss, gradient
```

4 Running SGD Variants

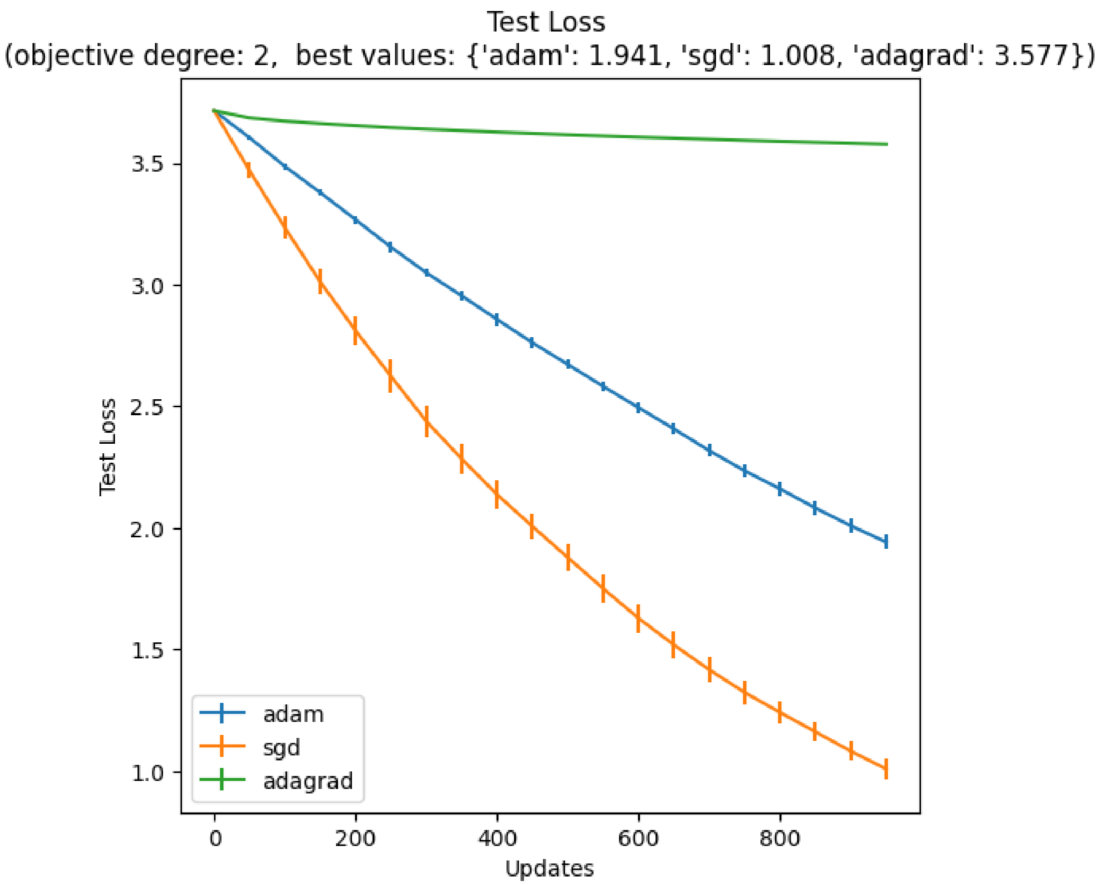
[95]:

```
#@title Parameters
deg_ = 2 #@param {type: "number"}
num_rep = 15 #@param {type: "integer"}
max_iter = 1000 #@param {type: "integer"}
fig, ax = plt.subplots(figsize=(6, 6))
best_vals = {}
test_exp_interval = 50 #@param {type: "integer"}
grad_artificial_normal_noise_scale = 0. #@param {type: "number"}

# dictionary for keeping track of parameters
params = {}

for method_idx, method in enumerate(['adam', 'sgd', 'adagrad']):
    test_loss_mat = []
    train_loss_mat = []

    for replicate in range(num_rep):
        if replicate % 20 == 0:
            print(method, replicate)
        if method == 'adam':
            beta_1 = 0.9
            beta_2 = 0.999
            m = np.zeros((dim_theta, 1))
            v = np.zeros((dim_theta, 1))
            epsilon = 1e-8

        if method == 'adagrad':
            epsilon = 1e-8
            squared_sum = np.zeros((dim_theta, 1))

        theta_hat = theta_init.copy()
        test_loss_list = []
        train_loss_list = []

        for t in range(max_iter):
            idx = np.random.choice(data_num, batch_size) # Split data
            train_loss, gradient = noisy_val_grad(theta_hat, A[idx,:],␣
↪y_data[idx,:], deg_=deg_)
            artificial_grad_noise = np.random.randn(10, 1) *␣
↪grad_artificial_normal_noise_scale + np.sign(np.random.random((10, 1)) - 0.
↪5) * 0.
            gradient = gradient + artificial_grad_noise
            train_loss_list.append(train_loss)

            if t % test_exp_interval == 0:
                test_loss, _ = noisy_val_grad(theta_hat, A_test[:,:], y_test[:,:
↪], deg_=deg_)
                test_loss_list.append(test_loss)

            if method == 'adam':
                m = beta_1 * m + (1-beta_1) * gradient
                v = beta_2 * v + (1-beta_2) * (gradient * gradient)
                m_hat = m / (1 - beta_1 ** (t+1))
                v_hat = v / (1 - beta_2 ** (t+1))
                theta_hat = theta_hat - lr * (m_hat/(np.sqrt(v_hat)+epsilon))

            elif method == 'adagrad':
                squared_sum = squared_sum + (gradient * gradient)
                theta_hat = theta_hat - lr * gradient * (1/np.
↪sqrt(squared_sum+epsilon))

            elif method == 'sgd':
                theta_hat = theta_hat - lr * gradient

        test_loss_mat.append(test_loss_list)
        train_loss_mat.append(train_loss_list)
```

```
    print(method, 'done')
    x_axis = np.arange(max_iter)[::test_exp_interval]

    test_loss_np = np.array(test_loss_mat)

    '''
    Hints:
    1. Use test_loss_np in np.mean() with axis = 0
    '''
    test_loss_mean = np.mean(test_loss_np, axis=0)

    '''
    Hints:
    1. Use test_loss_np in np.std() with axis = 0
    2. Divide by np.sqrt() using num_rep as a parameter
    '''
    test_loss_se = np.std(test_loss_np, axis=0) / np.sqrt(num_rep)

    plt.errorbar(x_axis, test_loss_mean, yerr=2.5*test_loss_se, label=method)
    best_vals[method] = min(test_loss_mean)

    # this only gets the parameters for one replicate (the last one) but it's␣
 ↪okay because we only need one to answer q3.
    params[method] = theta_hat

best_vals = { k: int(v * 1000) / 1000. for k,v in best_vals.items() } # A weird␣
 ↪way to round numbers
plt.title(f'Test Loss \n(objective degree: {deg_},  best values: {best_vals})')
plt.ylabel('Test Loss')
plt.legend()
plt.xlabel('Updates')
```

```
adam 0
adam done
sgd 0
sgd done
adagrad 0
adagrad done
```

```
[95]: Text(0.5, 0, 'Updates')
```



```
[94]: print(params["adam"].reshape(-1))
```

```
[0.35388391 0.27269951 0.31367533 0.36023661 0.28976016 0.36528544
 0.33467022 0.36753296 0.29897261 0.30564255]
```

```
[ ]:
```