starter_P1_SGDtheory

March 13, 2018

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pdb
        import random
        %matplotlib inline
```

1 Part A: one solution

Assuming that I want to find the w that minimizes $\frac{1}{2n}||Xw - y||_2^2$. In this part, X is full rank, and $y \in range(X)$

```
In [2]: X = np.random.normal(scale = 20, size=(100,10))
        print(np.linalg.matrix_rank(X)) # confirm that the matrix is full rank
        # Theoretical optimal solution
        w = np.random.normal(scale = 10, size = (10,1))
        y = X.dot(w)
```

```
10
```

```
In [22]: def sgd(X, y, w_actual, threshold, max_iterations, step_size, gd=False):
             if isinstance(step_size, float):
                 step_size_func = lambda i: step_size
             else:
                 step_size_func = step_size

             # run 10 gradient descent at the same time, for averaging purpose
             # w_guesses stands for the current iterates (for each run)
             w_guesses = [np.zeros((X.shape[1], 1)) for _ in range(10)]
             n = X.shape[0]
             error = []
             it = 0
             above_threshold = True
             previous_w = np.array(w_guesses)

             while it < max_iterations and above_threshold:
                 it += 1
```

```
            curr_error = 0
            for j in range(len(w_guesses)):
                if gd:
                    # Your code, implement the gradient for GD
                    sample_gradient = X.T @ X @ w_guesses[j] - X.T @ y
                    sample_gradient /= len(y)
                else:
                    # Your code, implement the gradient for SGD
                    idx = np.random.randint(len(y))
                    sample_gradient = X[[idx]].T @ X[[idx]] @ w_guesses[j] -  X[[idx]].T @

                # Your code: implement the gradient update
                # learning rate at this step is given by step_size_func(it)
                w_guesses[j] = w_guesses[j] - sample_gradient * step_size_func(it)
                curr_error += np.linalg.norm(w_guesses[j]-w_actual)

            error.append(curr_error/10)

            diff = np.array(previous_w) - np.array(w_guesses)
            diff = np.mean(np.linalg.norm(diff, axis=1))
            above_threshold = (diff > threshold)
            previous_w = np.array(w_guesses)
        return w_guesses, error
```

In [23]:
```
its = 5000
w_guesses, error = sgd(X, y, w, 1e-10, its, 0.0001)
```

In [24]:
```
iterations = [i for i in range(len(error))]
#plt.semilogy(iterations, error, label = "Average error in w")
plt.semilogy(iterations, error, label = "Average error in w")
plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Average Error vs Iterations for SGD with exact sol")
plt.legend()
plt.show()
```


```
In [25]: print("Required iterations: ", len(error))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses])
         print("Final average error: ", average_error)
```

```
Required iterations:  1064
Final average error:  2.009834128221152e-09
```

2 Part B: No solutions, constant step size

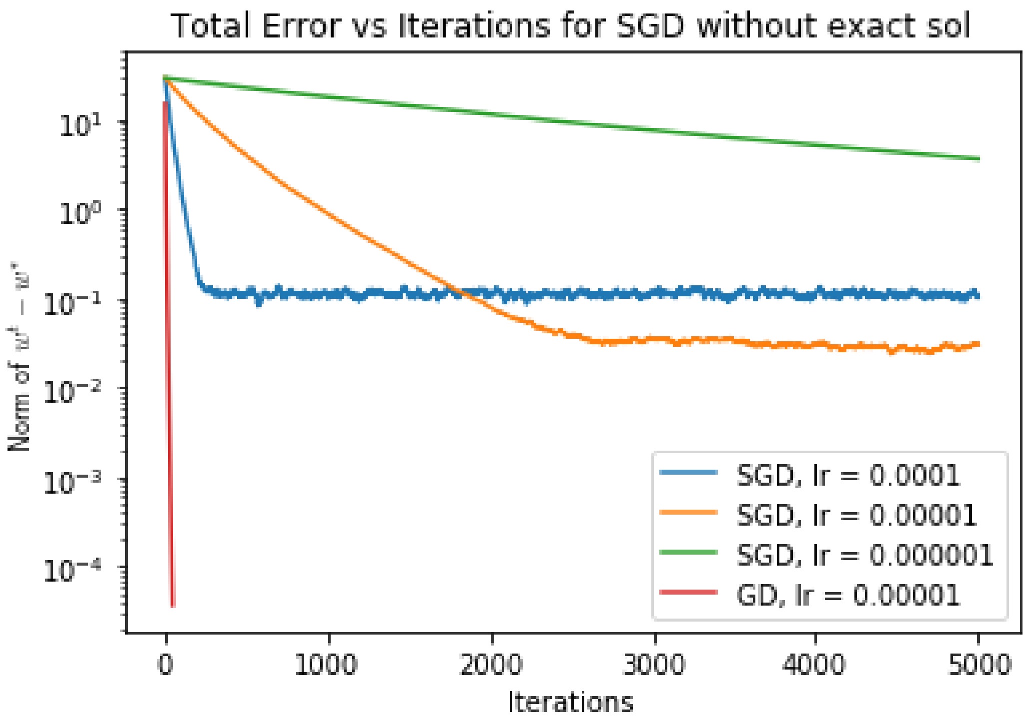
```
In [26]: y2 = y + np.random.normal(scale=5, size = y.shape)
         w=np.linalg.inv(X.T @ X) @ X.T @ y2
```

```
In [27]: its = 5000
         w_guesses2, error2 = sgd(X, y2, w, 1e-5, its, 0.0001)
         w_guesses3, error3 = sgd(X, y2, w, 1e-5, its, 0.00001)
         w_guesses4, error4 = sgd(X, y2, w, 1e-5, its, 0.000001)
```

```
In [28]: w_guess_gd, error_gd = sgd(X, y2, w, 1e-5, its, 0.001, True)
```

```
In [29]: plt.semilogy([i for i in range(len(error2))], error2, label="SGD, lr = 0.0001")
         plt.semilogy([i for i in range(len(error3))], error3, label="SGD, lr = 0.00001")
         plt.semilogy([i for i in range(len(error4))], error4, label="SGD, lr = 0.000001")
         plt.semilogy([i for i in range(len(error_gd))], error_gd, label="GD, lr = 0.00001")
```

```
plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Total Error vs Iterations for SGD without exact sol")
plt.legend()
plt.show()
```




```
In [30]: print("Required iterations, lr = 0.0001: ", len(error2))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses2])
         print("Final average error: ", average_error)

         print("Required iterations, lr = 0.00001: ", len(error3))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses3])
         print("Final average error: ", average_error)

         print("Required iterations, lr = 0.000001: ", len(error4))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses4])
         print("Final average error: ", average_error)

         print("Required iterations, GD: ", len(error_gd))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guess_gd])
         print("Final average error: ", average_error)
```

```
Required iterations, lr = 0.0001:  5000
Final average error:  0.10345803797006706
```

```
Required iterations, lr = 0.00001:  5000
Final average error:  0.03099382986592833
Required iterations, lr = 0.000001:  5000
Final average error:  3.685431873458876
Required iterations, GD:  47
Final average error:  3.7024294265064345e-05
```

3 Part C: No solutions, decreasing step size

In [31]:
```
its = 5000
def step_size(step):
    if step < 500:
        return 1e-4
    if step < 1500:
        return 1e-5
    if step < 3000:
        return 3e-6
    return 1e-6

w_guesses_variable, error_variable = sgd(X, y2, w, 1e-10, its, step_size, False)
```

In [32]:
```
plt.semilogy([i for i in range(len(error_variable))], error_variable, label="Average er
plt.semilogy([i for i in range(len(error2))], error2, label="Average error, lr = 0.0001
plt.semilogy([i for i in range(len(error3))], error3, label="Average error, lr = 0.0000
plt.semilogy([i for i in range(len(error4))], error4, label="Average error, lr = 0.0000

plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Error vs Iterations for SGD with no exact sol")
plt.legend()
plt.show()
```


```
In [33]: print("Required iterations, variable lr: ", len(error_variable))
         average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses_variable])
         print("Average error with decreasing lr:", average_error)
```

```
Required iterations, variable lr:  5000
Average error with decreasing lr: 0.011307058669572923
```