Problem 1: Perceptron Learning (30 pts)

1. Standard subgradient descent with the step size Gamma= 1 for each iteration.

Code:

```
perceptron_data = importdata('perceptron.data',',');

X = perceptron_data(:,1:end-1);
Y = perceptron_data(:,end);

w = zeros(1,size(X,2));
b = 0;

w_first3 = zeros(3,size(X,2));
b_first3 = zeros(3,1);

grad_w = ones(size(w));
grad_b = ones(size(b));

p_loss = Inf;
iter = 0;
max_iter = 1000;

% step size
gamma = 1;
loss_history = [];

while iter < max_iter

    pred = (X * w.') + b ;

    loss_each = -1 * ( Y .* pred);
    incorrect = loss_each >= 0;

    p_loss = sum(loss_each .* incorrect) ;
    loss_history = [loss_history ,p_loss] ;

    grad_w = -1 * (sum((incorrect .* Y) .* X));
    grad_b = -1 *(sum(incorrect .* Y));

    % check if all our gradients are zero, stop if they are
    if (grad_b == 0) && (all(grad_w==0))
        break
    end

    iter = iter + 1;

    w = w - gamma * grad_w;
    b = b - gamma * grad_b;

    % to note the first 3 values
    if iter <= 3
        w_first3(iter,:) = w;
        b_first3(iter,:) = b;
    end
```

```
end

plot(loss_history(2:end));
w_first3
b_first3
iter
w
b
```

Results:

Standard Gradient Descent ; Step size = 1

First 3 iterations weight vectors =

```
  1.0e+03 *
   1.2790    0.4601   -0.1086   -1.6723
   1.3073    0.4327   -0.0276   -1.5238
   1.2552    0.4255    0.0188   -1.4347
```

First 3 iterations bias values =

```
  -354
  -493
  -625
```

Total number of iterations = 46

Final weights
w = [685.7993 , 243.8995, 8.2420, -797.6251]

Final bias
b = -1485

2. Stochastic subgradient descent where exactly one component of the sum is chosen to approximate the gradient at each iteration. Instead of picking a random component at each iteration, you should iterate through the data set starting with the first element, then the second, and so on until the Mth element, at which point you should start back at the beginning again.

Again, use the step size t = 1.

Code:

```
perceptron_data = importdata('perceptron.data',',');

X = perceptron_data(:,1:end-1);
Y = perceptron_data(:,end);

w = zeros(1,size(X,2));
b = 0;

w_first3 = zeros(3,size(X,2));
b_first3 = zeros(3,1);

grad_w = ones(size(w));
grad_b = ones(size(b));

p_loss = Inf;
iter = 0;
max_iter = Inf;

gamma = 1;
loss_history = [];
idx = 1;

while iter < max_iter

    pred = (X * w.') + b ;

    loss_each = -1 * ( Y .* pred);
    incorrect = loss_each >= 0;

    if sum(incorrect) == 0
        break
    end

    p_loss = sum(loss_each .* incorrect) ;
    loss_history = [loss_history ,p_loss] ;

    if idx > size(X,1)
        idx = mod(idx,size(X,1)) + 1
    end

    % go to the next incorrect sample
    if incorrect(idx) == 0
        offset = find(incorrect(idx:end),1);
        if isempty(offset)
            idx = 1;
            offset = find(incorrect(idx:end),1);
        end
        idx = idx + offset - 1;
    end

    grad_w = -1 * (Y(idx) * incorrect(idx) * X(idx,:));
    grad_b = -1 * (Y(idx) * incorrect(idx));

    % check if all our gradients are zero, stop if they are
    % For the sake of uniformity, we use this instead in assignment

    w = w - gamma * grad_w;
    b = b - gamma * grad_b;

    iter = iter + 1;
    idx = idx + 1;

    if iter <= 3
        w_first3(iter,:) = w;
        b_first3(iter,:) = b;
    end

end
```

Results:

Stochastic Gradient Descent, Stepsize = 1

First 3 iterations weight vectors =

```
   4.6175    2.4697    1.9677   -1.8134
   3.4532    0.1694    2.6280   -4.6471
   0.4561    4.9290   -2.2589   -4.6515
```

First 3 iterations bias values =

```
   -1
   -2
   -3
```

Total number of iterations = 8694

Final weights
w = [149.2771, 52.5335, 1.6717, -172.8919]

Final Bias
b = -322

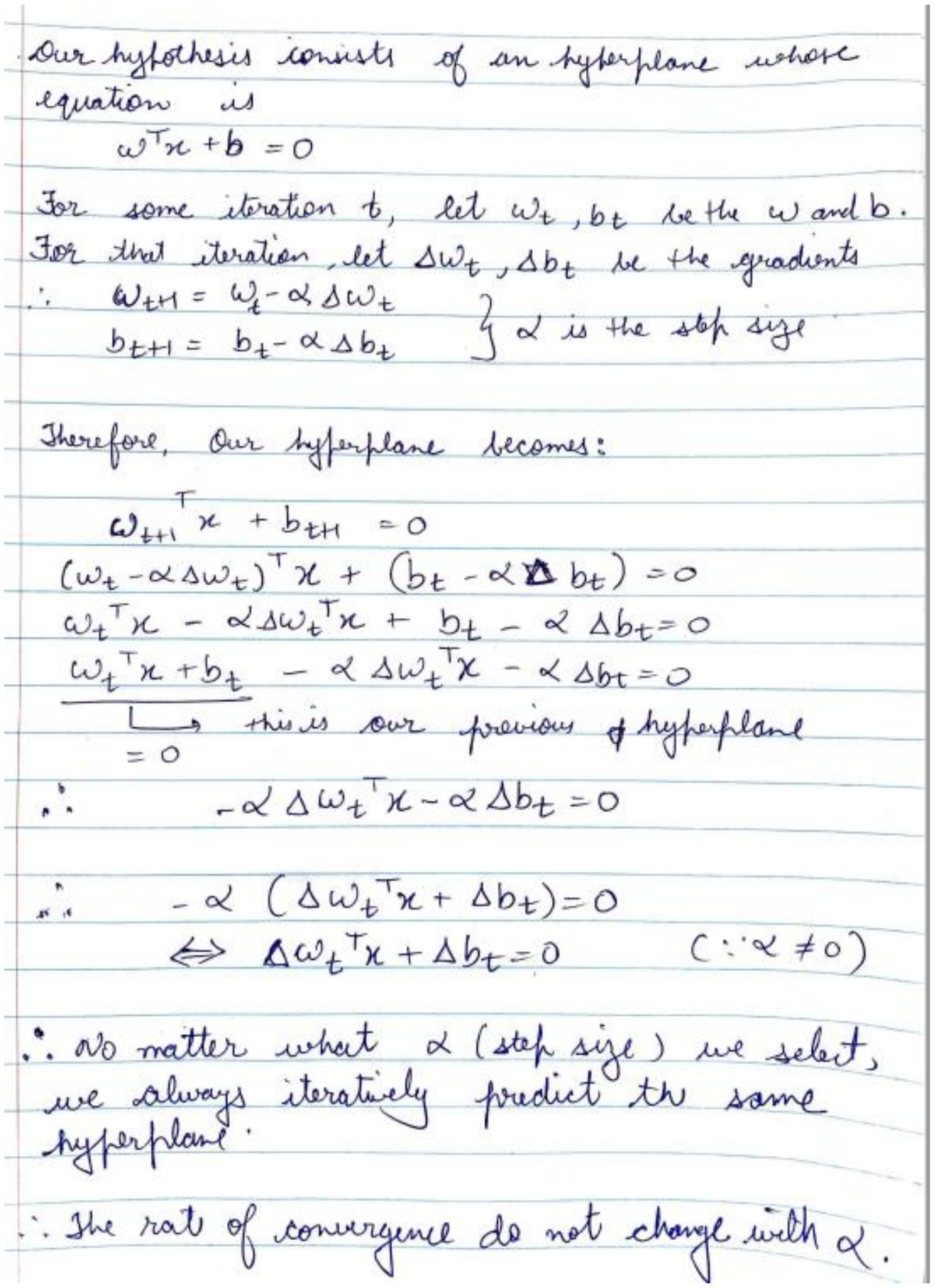
3. How does the rate of convergence change as you change the step size? Provide some example step sizes to back up your statements.

Answer:

For the standard as well as stochastic gradient descent, for the perceptron algorithm, The code takes the same number of iterations to converge with step sizes 0.001, 1 and 1000.

The rate of convergence doesn't change with the step size because our hypothesis is scale invariant in w and b.

i.e.



Our hypothesis consists of an hyperplane whose equation is

$$w^T x + b = 0$$

For some iteration t, let w_t, b_t be the w and b.
For that iteration, let Δw_t, Δb_t be the gradients

$$\therefore \left.\begin{aligned} w_{t+1} &= w_t - \alpha \Delta w_t \\ b_{t+1} &= b_t - \alpha \Delta b_t \end{aligned}\right\} \alpha \text{ is the step size}$$

Therefore, Our hyperplane becomes:

$$w_{t+1}^T x + b_{t+1} = 0$$

$$(w_t - \alpha \Delta w_t)^T x + (b_t - \alpha \Delta b_t) = 0$$

$$w_t^T x - \alpha \Delta w_t^T x + b_t - \alpha \Delta b_t = 0$$

$$\underline{w_t^T x + b_t} - \alpha \Delta w_t^T x - \alpha \Delta b_t = 0$$

$\hookrightarrow$ this is our previous ~~of~~ hyperplane

$= 0$

$$\therefore \quad -\alpha \Delta w_t^T x - \alpha \Delta b_t = 0$$

$$\therefore \quad -\alpha (\Delta w_t^T x + \Delta b_t) = 0$$

$$\Leftrightarrow \Delta w_t^T x + \Delta b_t = 0 \qquad (\because \alpha \neq 0)$$

$\therefore$ No matter what α (step size) we select, we always iteratively predict the same hyperplane.

$\therefore$ The rate of convergence do not change with α.

4. What is the smallest, in terms of number of data points, two-dimensional data set containing both class labels on which the algorithm, with step size one, fails to converge? Use this example to explain why the method may fail to converge more generally.

Answer:

The smallest number of data points, in a 2-D data set containing both class labels on which the algorithm fails to converge is 2.

i.e.

if for some input x1,x2 we get both +ve and -ve label, the algorithm will fail to converge.
(1,1) -> 1
(1,1) -> -1

In general, If we contain an ambiguous dataset where in we get both the labels for same inputs, the algorithm fails to converge.

Also, if no linear separator is possible for the data in our feature space, the algorithm fails to converge.

If we don't have ambiguous data, that if no input has both the labels then the minimum number of points that wouldn't converge are **3 co-linear points with an opposite label in between.**