CSE 326 **Due on Feb 12**

Problem Set I Gustavo Grinsteins CSE 326

1 Written Exercises

Answer the following questions in 25-100 words each. Full scores will not be given if missing detailed steps for all computations.

1. (15%) Suppose we are trying to find the minimum of the function $f(x) = 3x^2 - 2x + 1$, for univarite (scalar) x. First, find the minimum of this function by using calculus. Second, verify that this function is convex (and therefore the point you found in the first step is a *global* minimum). Finally, perform (by hand – show your work) three steps of gradient descent with step size $\epsilon = 0.1$ and the initial point $x_0 = 1$. How close does it get to the true solution?

   (**Gustavo's Solution:**)

   a) Finding the function's minimum

   To find the minimum of this function, the critical point is found by performing the derivative of the given function and setting it equal to zero and solving for x:

   $$\frac{df(x)}{dx} = 6,\ 6x - 2 = 0, \quad x = 1/3 \quad \text{(critical point)},$$

   in order to check if this critical point is a local maximum or minimum we can take the second derivative of the function:

   $$\frac{d^2 f(x)}{dx^2} = 6,$$

   since $\frac{d^2 f(x)}{dx^2} \geq 0$ for all x in the interval of $[-\infty, +\infty]$. In addition, by the concavity theorem we can confirm that the function given is concave upward making our critical point a global minimum.

   b) By-hand iteration (n = 3) of gradient decent algorithm with step size $\epsilon = 0.1$ and the initial point $x_0 = 1$.:

   Algorithm for gradient descent:

   $\epsilon = 0.1$;

   $x[1] = 1$;

   $n = 3$;

   for i = 1; i $\geq$ n; i++

   $x_{i+1} = x_i - \epsilon \bigtriangledown f(x_i)$;

   end

First iteration (n=1): $x = 1 - (0.1)(6(1) - 2) = 0.6$

Second iteration (n=2): $x = 0.6 - (0.1)(6(0.6) - 2) = 0.44$

Third iteration (n=3): $x = 0.44 - (0.1)(6(0.44) - 2) = 0.376$

Comparing our result with our actual value (using three significant figures):
$\%Error = \left|\frac{0.333-0.376}{0.333}\right| (100) = 12.9$

Considering that the iteration of the algorithm only happened three times, an percent error of %12.9 is decent for this function's minimum,

2. (15%) 0/1 loss is hard to optimize using gradient descent because it is poorly behaved (discontinuous, non-differentiable, etc.). A common "smooth" version of 0/1 loss is the *sigmoid* loss, which makes use of our favorite function, the sigmoid: $\sigma(z) = 1/(1 + \exp[-z])$. In particular, we define our loss function $l(y, f(x)) = -\log(\sigma(y \cdot f(x)))$. First, verify that this is a reasonable loss function: when $f(x)$ is correct, the loss is low, and when $f(x)$ is incorrect, the loss is high. You may assume that $f(x)$ produces *real values*, where positive values mean class $+1$ and negative values mean class -1.

(**Gustavo's Solution:**)

To understand the behavior of the given loss function, the loss function is expanded in the following way:

$l(y, f(x)) = -\log(\sigma(y \cdot f(x))) = -\log\left(\frac{1}{1+\exp[-(y \cdot f(x))]}\right)$

To make this equation more appealing to sight, lets make $[y \cdot f(x)] = a$ and reduce the notation of $l(y, f(x))$ to l; then we have:

$l = -\log\left(\frac{1}{1+\exp[-a]}\right)$

Using logarithmic rules we get:

$l = -[\log(1) - \log(1 + \exp[-a])]$

Finally, the equation is reduced to:

$l = \log(1 + \frac{1}{\exp[a]})$

Now, lets analyze what the loss function is doing as we change the value of a:

If a is a positive value, this means that our label variable y and our activation function $f(x)$ share the same sign (positive or negative). In other words, $f(x)$ is correct. For this case the loss function should return a small value. To check this lets do the following:

$\lim_{a\to\infty} \log(1 + \frac{1}{\exp[a]}) = \log(1 + \frac{1}{\infty}) = \log(1 + 0) = 0$

This confirms that our loss is low when $f(x)$ is correct.

If a is a negative value, this means that our label variable y and our activation function $f(x)$ share opposite sign. In other words, $f(x)$ is incorrect. For this case the loss function should return a large value. To check this lets do the following:

$\lim_{a\to\infty} \log(1 + \frac{1}{\exp[-a]}) = \log(1+\infty) = \log(\infty) = \infty$

This confirms that our loss is high when $f(x)$ is incorrect.

3. (15%) Consider optimizing a linear function under sigmoid loss, so that we have

$$f(\mathbf{X},\ \mathbf{y},\ \mathbf{w},\ \mathbf{b}) = \sum_n \sigma(y_n(\mathbf{w}^T\mathbf{x}_n + b)).$$

Compute the gradient of this function with respect to $\mathbf{w}$ and with respect to b so that we might construct a gradient descent algorithm.

(**Gustavo's Solution:**)

It is known that:

$\nabla f(X, y, w, b) =< \frac{\partial f}{\partial X}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial w}, \frac{\partial f}{\partial b} >$

The problem is just asking for $\frac{\partial f}{\partial w}$ and $\frac{\partial f}{\partial b}$. These are calculated as follows:

First, lets expand the function given:

$f(X, y, w, b) = \sum_n \sigma(y_n(\mathbf{w}^T\mathbf{x}_n + b)) = \sum_n \left(\frac{1}{1+\exp[-y_n(\mathbf{w}^T\mathbf{x}_n+b)]}\right)$

Now lets find the desired partial derivatives using the chain rule several times:

$\frac{\partial f}{\partial w} = \sum_n (-1)(1 + \exp[-y_n(\mathbf{w}^T\mathbf{x}_n + b)])^{-2} \exp[-y_n(\mathbf{w}^T\mathbf{x}_n + b)](-y_n\mathbf{x}_n)$

reducing we get:

$\frac{\partial f}{\partial w} = \sum_n \frac{\exp[-y_n(\mathbf{w}^T\mathbf{x}_n+b)](y_n\mathbf{x}_n)}{(1+\exp[-y_n(\mathbf{w}^T\mathbf{x}_n+b)])^2}$

Following the same procedure we calculate the other partial derivative:

$\frac{\partial f}{\partial b} = \sum_n (-1)(1 + \exp[-y_n(\mathbf{w}^T\mathbf{x}_n + b)])^{-2} \exp[-y_n(\mathbf{w}^T\mathbf{x}_n + b)](-y_n)$

Finally we get:

$\frac{\partial f}{\partial w} = \sum_n \frac{\exp[-y_n(\mathbf{w}^T\mathbf{x}_n+b)](y_n)}{(1+\exp[-y_n(\mathbf{w}^T\mathbf{x}_n+b)])^2}$

4. (10%) Consider the perceptron update learning rule for binary classification (so the loss is 0/1 loss). What effect does the parameter η have on the learning process?

   (**Gustavo's Solution:**)

   The perceptron is a supervised learning algorithm for binary classifiers. In essence, this algorithm takes an input vector which is multiplied by a weight value then passed through an activation function to finally give a classification value. If the input data provided is linearly separable, the perceptron convergence theorem guarantees to find a solution in a finite amount of time.

   In the algorithm, the weight value of the perceptron is updated after obtaining the prediction results. The update happens as described by the following formula:

   $$w^{new} = w^{old} + \eta \sum_{n} (d - o) x_i$$

   In this formula η $(0 < \eta < 1)$ is the learning rate, d is the desired output, o is the output from the algorithm, and x is a specific value from the input vector. From this formula we can see that the η scales the weight vector w but it does not affects the result from the algorithm. The value attributed to η will depend on the behavior we want the learning to have. If we want the algorithm to have a fast adaptation to a changing input vector, then a large η value will be selected. If stable weight estimates are wanted by averaging past input values, then an small η is needed.

2 Programming Task

Students shall submit your own codes and results in one folder. All experimental results are required to be reported in a single PDF file with the above written exercises.

CSE 326 (45%)

- Implement linear regression by using gradient descent algorithm. This takes the initial weight vector w_0 (e.g., all zero vector), the input data X and Y, a gradient descent step size ϵ (hand-tuning parameter), and a number of iterations to run. It produces the estimated final weight vector w.
- Report a detailed algorithm description with gradient term calculation.
- Plot the final result of the estimated regression line, and your best convergence graph of the algorithm estimation.

The initial code with data is provided in 'LinearRegression.m' file.

(**Gustavo's Solution:**)

The code for this question was created by using the gradient descent algorithm and the energy function reviewed in class. The parameters that are optimized by this algorithm are the weight w

and bias θ values of the linear regression formula $y = wx + \theta$. With these optimized parameters we are able to find the line of best fit for the data provided.

Derivation of the equations used in the code:

The energy function reviewed in class is the following:

$$E(w, \theta, y, x) = Min_{w,\theta} \sqrt{\sum_n (y_n - wx_n + \theta)^2}$$

For this problem, since the values minimized are w and θ and we are looking for a graph that shows the convergence of the parameters, the gradient descent algorithm will take the following form:

for i = 1; i $\geq$ n; i++

$w_{i+1} = w_i - \epsilon \frac{\partial E}{\partial w}$;

$\theta_{i+1} = \theta_i - \epsilon \frac{\partial E}{\partial \theta}$;

$E_i = \sum_n (y_n - w_{i+1} x_n + \theta_{i+1})^2$

end

Where $\frac{\partial E}{\partial w} = \sum_n (y_n - (wx_n + \theta))(-x_i)$ and $\frac{\partial E}{\partial \theta} = \sum_n (y_n - (wx_n + \theta))$.

Notice that to simplify the math the square root term in the energy function and the constant that results from the partial derivative were omitted since the focus is optimizing the functions.

Description of the algorithm:

The logic of the algorithm is to keep updating the parameters we want to optimize by iterating through the negative information of the gradient functions until we reach a local minimum. The resulting converged values of w and θ are used to plot the best fit regression line. One thing to note is that if the step size ϵ is too big, the parameters will not converge. This means that the ϵ value caused the algorithm to step through the local minimum and headed to an asymptote in the function.

The E_i term inside of the gradient descent loop will allow the code to generate a plot that will graphically show the convergence of w and θ.

Final results and Output plots(next page):

Using the linear model $y = wx + \theta$ and the gradient descent algorithm, the converged value of the weight is $w = 1.37 * 10^{-4}$ and for the bias $\theta = -5.12 * 10^{-11}$ using a step size of $\epsilon = 1.0 * 10^{-16}$.

Note that on figure 2 the algorithm only takes about 10 iterations to converge the values.

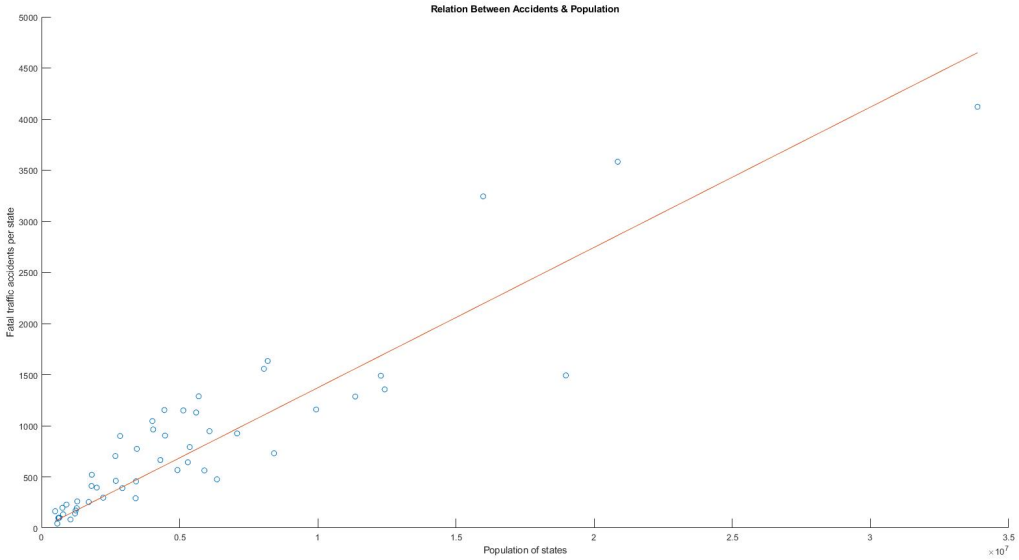

Figure 1: Estimated regression line

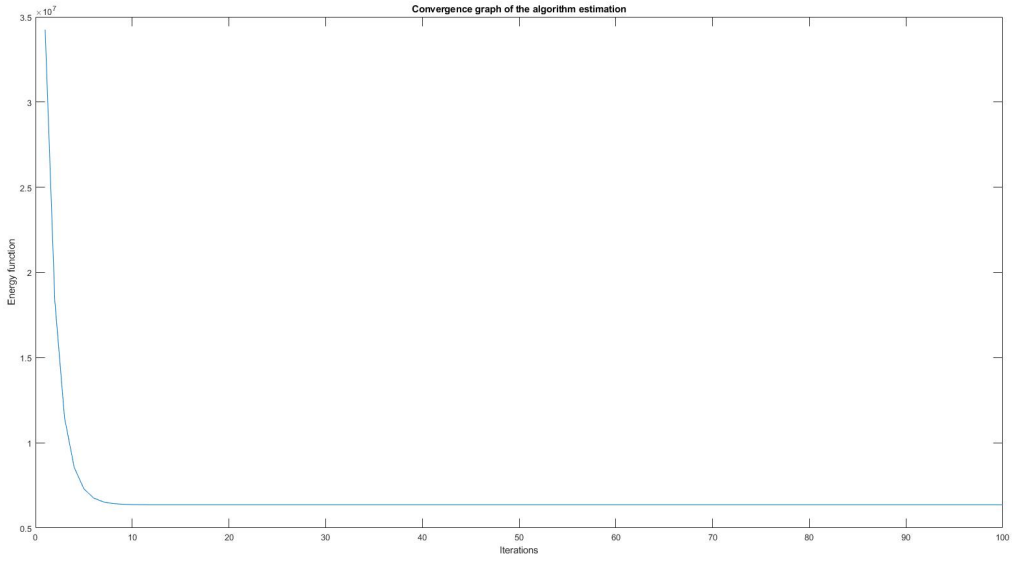

Figure 2: Best convergence graph