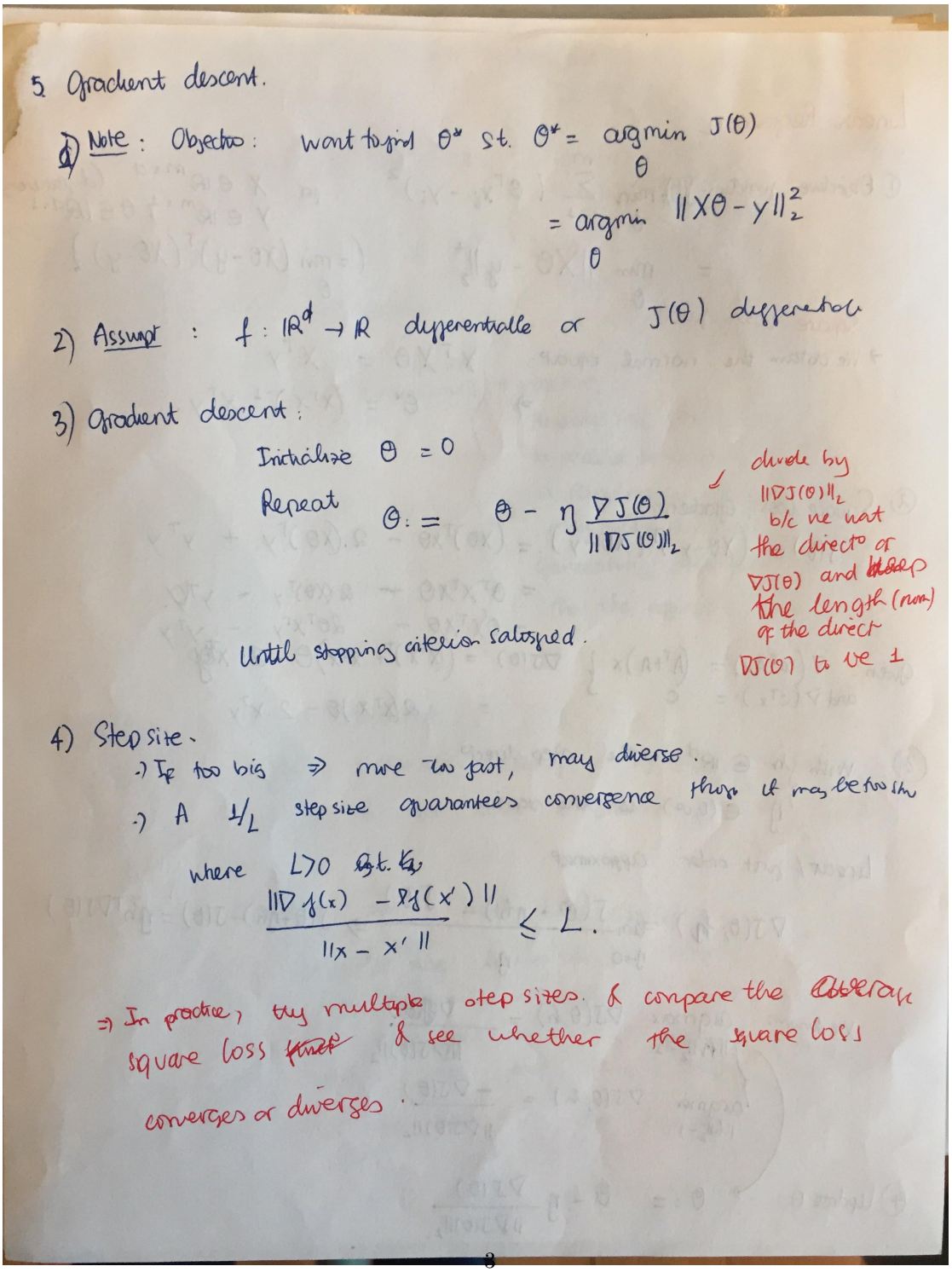
Machine Learning and Computational Statistics
Gradient Descent

Nhung Le

Note: This document consists of concepts and exercises related to Gradient Descent.

1 Concepts



5 Gradient descent.

1) Note: Objective: want to find θ^* s.t. $\theta^* = \underset{\theta}{\operatorname{argmin}}\ J(\theta)$

$= \underset{\theta}{\operatorname{argmin}}\ \|X\theta - y\|_2^2$

2) Assump: $f: \mathbb{R}^d \to \mathbb{R}$ differentiable or $J(\theta)$ differentiable

3) Gradient descent:

Initialize $\theta = 0$

Repeat $\theta := \theta - \eta \frac{\nabla J(\theta)}{\|\nabla J(\theta)\|_2}$

divide by $\|\nabla J(\theta)\|_2$ b/c we want the direction of $\nabla J(\theta)$ and keep the length (num) of the direction $\nabla J(\theta)$ to be 1

Until stopping criterion satisfied.

4) Step size.

- If too big ⇒ move too fast, may diverge.
- A $1/L$ step size guarantees convergence though it may be too slow

where $L > 0$ s.t. $\frac{\|\nabla f(x) - \nabla f(x')\|}{\|x - x'\|} \leq L$.

⇒ In practice, try multiple step sizes & compare the different square loss & see whether the square loss converges or diverges.

6. Batch gradient descent VS. Stochastic gradient descent. (SGD)

Batch Gradient Descent.

- Compute the gradient using the whole dataset
- When to use.
  1) Convex, funct
  2) Relatively smooth err manifold

⇒ move somewhat directly toward an optimum solution (local/global).

SGD.

- Compute the gradient using a simple sample.
- When to use:
  1) Error manifolds that have a lot of local/maxima / minima.

→ faster.

Recommend using minibatch SGD to reduce computational demand why a randomly sampled minibatch may reflect the true data generating distribut better than the actual full batch.

7. Stochastic gradient descent.

For $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} f_i(\theta)$, rather than taking $-\nabla J(\theta)$ as our step direct, we take $-\nabla f_i(\theta)$ for some i chosen randomly from $\{1, 2, \ldots, m\}$.

This approximat may be poor but UNBIASED!

- eat $f_i(\theta)$ would be the loss on the ith example.
- the data pairs are randomly shuffled, & we sweep thru the whole trains one by one & perform an updat for each train example individually
- Each pas thru the data = one epoch.

Eg. $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x_i - y_i)^2$

$= \frac{1}{m} \sum_{i=1}^{m} f_i(\theta)$.

For eg: num epoch = 3.

num features = 2 $\Rightarrow \theta = [\theta_1 \quad \theta_2]$

num instance (m) = 4.

Epoch 0: shuffle i (i = num instance).

i_list = [0, 1, 2, 3].

-) for each i, we cal θ[epoch 0][i]. e.g. for i = 1,

using the fmcl: $\theta := \theta - \eta \frac{\nabla f_i(\theta)}{\|\nabla f_i(\theta)\|_2^2}$

(θ: the cur theta; η: stepsize; the last theta)

θ[epoch 0][1] = $[\theta_{0,1,0} \quad \theta_{0,1,1}]$

(epoch, index, the jth entry of θ.)

-) After this, we get θ[epoch 0]

θ[epoch 0] = $[\theta_{0,3,0} \quad \theta_{0,3,1}]$

b/c we get the last index = 3

-) the index i helps calculat $\nabla f_i(\theta)$.

Epoch 1. shuffle i; i_list = [0, 2, 1, 3].

-) for i = 0.

$\theta_{1,0,0} = \theta_{0,3} - \eta \frac{\nabla f_0(\theta)}{\|\nabla f_0(\theta)\|_2^2}$.

b/c: { i = 0 → use i = 0 to calculat $\nabla f_i(\theta)$

the last theta = $\theta_{0,3}$ (epoch 0, index 3 b/c the last index i in epoch 0 is 3.)

2 Linear Regression and Gradient Descent

2.1 Feature Normalization

When feature values differ greatly, we can get much slower rates of convergence of gradient-based algorithms. Furthermore, when we start using regularization (introduced in a later problem), features with larger values are treated as "more important", which is not usually what you want. One common approach to feature normalization is perform an affine transformation (i.e. shift and rescale) on each feature so that all feature values in the training set are in $[0, 1]$. Each feature gets its own transformation. We then apply the same transformations to each feature on the test[1] set. It's important that the transformation is "learned" on the training set, and then applied to the test set. It is possible that some transformed test set values will lie outside the $[0, 1]$ interval.

2.2 Gradient Descent Setup

In linear regression, we consider the hypothesis space of linear functions $h_\theta : \mathbf{R}^d \to \mathbf{R}$, where

$$h_\theta(x) = \theta^T x,$$

for $\theta, x \in \mathbf{R}^d$, and we choose θ that minimizes the following "average square loss" objective function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right)^2,$$

where $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbf{R}^d \times \mathbf{R}$ is our training data.

While this formulation of linear regression is very convenient, it's more standard to use a hypothesis space of "affine" functions:

$$h_{\theta,b}(x) = \theta^T x + b,$$

which allows a "bias" or nonzero intercept term. The standard way to achieve this, while still maintaining the convenience of the first representation, is to add an extra dimension to x that is always a fixed value, such as 1. You should convince yourself that this is equivalent. We'll assume this representation, and thus we'll actually take $\theta, x \in \mathbf{R}^{d+1}$.

1. Let $X \in \mathbf{R}^{m \times (d+1)}$ be the **design matrix**, where the i'th row of X is x_i. Let $y = (y_1, \ldots, y_m)^T \in \mathbf{R}^{m \times 1}$ be the "response". The objective function $J(\theta)$:
   $J(\theta) = \frac{1}{m}(X\theta - y)^T \cdot (X\theta - y)$
2. The gradient of J:
   $\nabla_\theta J(\theta) = \frac{2}{m} X^T (X\theta - y)$

[1] Throughout this assignment we refer to the "test" set. It may be more appropriate to call this set the "validation" set, as it will be a set of data on which we compare the performance of multiple models. Typically a test set is only used once, to assess the performance of the model that performed best on the validation set.

3. In our search for a θ that minimizes J, suppose we take a step from θ to $\theta + \eta h$, where $h \in \mathbf{R}^{d+1}$ is the "step direction" (recall, this is not necessarily a unit vector) and $\eta \in (0, \infty)$ is the "step size" (note that this is not the actual length of the step, which is $\eta\|h\|$). An approximate expression for the change in objective function value $J(\theta + \eta h) - J(\theta)$.

   $J(\theta + \eta h) - J(\theta) \approx \nabla_\theta J(\theta)^T(\theta + \eta h - \theta) = \eta \nabla_\theta J(\theta)^T h$

4. The expression for updating θ in the gradient descent algorithm. Let η be the step size.

   $\theta_{n+1} = \theta_n - \eta \nabla J(\theta_n)$

5. Compute $J(\theta)$ for a given θ. [2]

6. compute $\nabla_\theta J(\theta)$ [3]

2.3 Gradient Checker

For many optimization problems, coding up the gradient correctly can be tricky. Luckily, there is a nice way to numerically check the gradient calculation. If $J : \mathbf{R}^d \to \mathbf{R}$ is differentiable, then for any vector $h \in \mathbf{R}^d$, the directional derivative of J at θ in the direction h is given by[4]

$$\lim_{\varepsilon \to 0} \frac{J(\theta + \varepsilon h) - J(\theta - \varepsilon h)}{2\epsilon}.$$

We can approximate this directional derivative by choosing a small value of $\varepsilon > 0$ and evaluating the quotient above. We can get an approximation to the gradient by approximating the directional derivatives in each coordinate direction and putting them together into a vector. In other words, take $h = (1, 0, 0, \ldots, 0)$ to get the first component of the gradient. Then take $h = (0, 1, 0, \ldots, 0)$ to get the second component. And so on. See http://ufldl.stanford.edu/wiki/index.php/Gradient_checking_and_advanced_optimization for details.

2.4 Batch Gradient Descent[5]

1. Batch Gradient Descent [6]

2. Step size: The step size affects whether and how fast gradient descent converges (i.e., if step size is too large, gradient descent may not converge[7]. As shown below, when the step size is large (e.g., 0.5) the loss oscilates around a lot but does tend to become smaller over time. When the step size is smaller (e.g., 0.05, 0.01), the loss converges faster, especially when we get closer to the optimal point [8]

[2] Refer to function `compute_square_loss(X, y, theta)` Q2.2 HW1 2019

[3] Refer to `compute_square_loss_gradient` Q2.2 HW1 2019

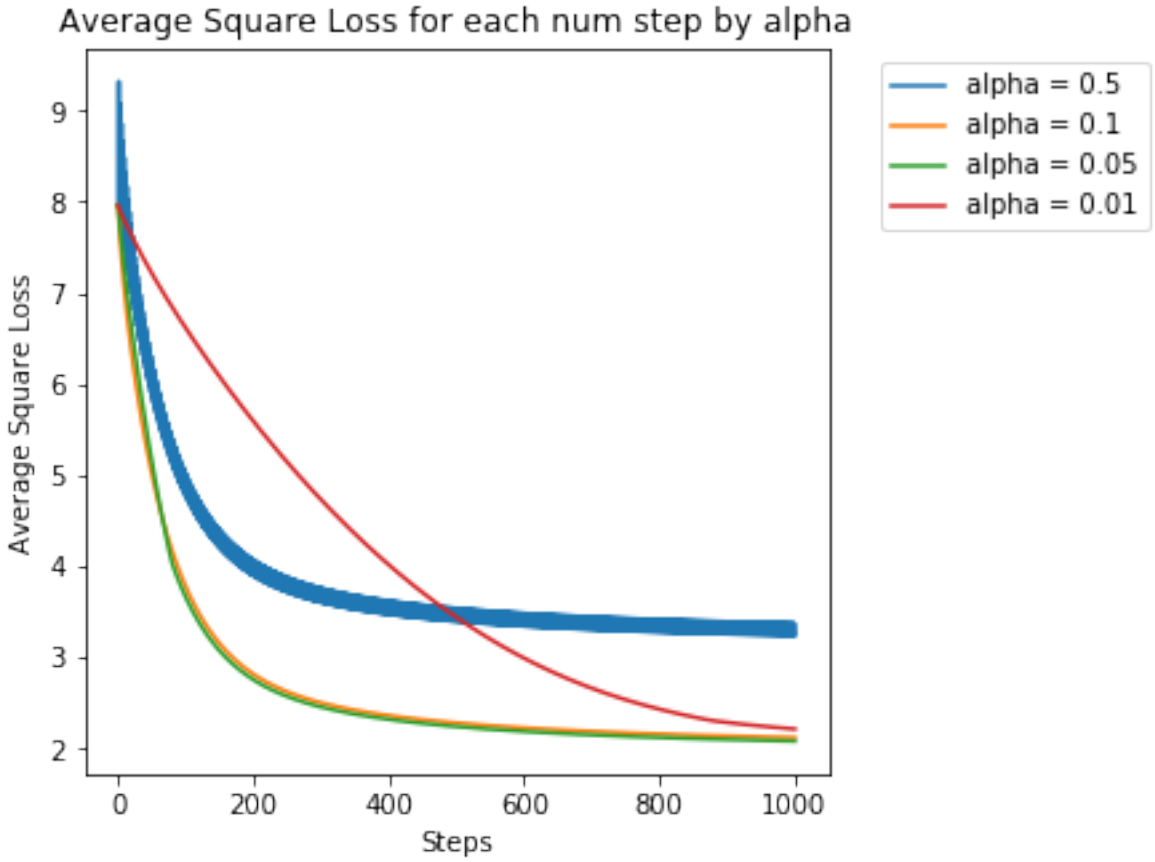
[4] Of course, it is also given by the more standard definition of directional derivative, $\lim_{\varepsilon \to 0} \frac{1}{\varepsilon}[J(\theta + \varepsilon h) - J(\theta)]$. The form given gives a better approximation to the derivative when we are using small (but not infinitesimally small) ε.

[5] Sometimes people say "batch gradient descent" or "full batch gradient descent" to mean gradient descent, defined as we discussed in class. They do this to distinguish it from stochastic gradient descent and minibatch gradient descent, which they probably use as their default.

[6] Refer to Q2.4 HW1-2019

[7] For the mathematically inclined, there is a theorem that if the objective function is convex and differentiable, and the gradient of the objective is Lipschitz continuous with constant $L > 0$, then gradient descent converges for fixed steps of size $1/L$ or smaller. See https://www.cs.cmu.edu/~ggordon/10725-F12/scribes/10725_Lecture5.pdf, Theorem 5.1.

[8] Refer to HW1-2019



3 Ridge Regression (i.e. Linear Regression with ℓ_2 regularization) and Gradient Descent

When we have a large number of features compared to instances, regularization can help control overfitting. Ridge regression is linear regression with ℓ_2 regularization. The regularization term is sometimes called a penalty term. The objective function for ridge regression is

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta,$$

where λ is the regularization parameter, which controls the degree of regularization. Note that the bias parameter is being regularized as well. We will address that below.

1. The gradient of $J(\theta)$
   $\nabla_{theta}J(\theta) = \frac{2}{m}X^T(X\theta - y) + 2\lambda\theta$
2. Updating θ in the gradient descent algorithm

$\theta_{n+1} = \theta_n - \eta[\frac{2}{m}X^T(X\theta - y) + 2\lambda\theta]$

3. Compute regularized square loss gradient [9]

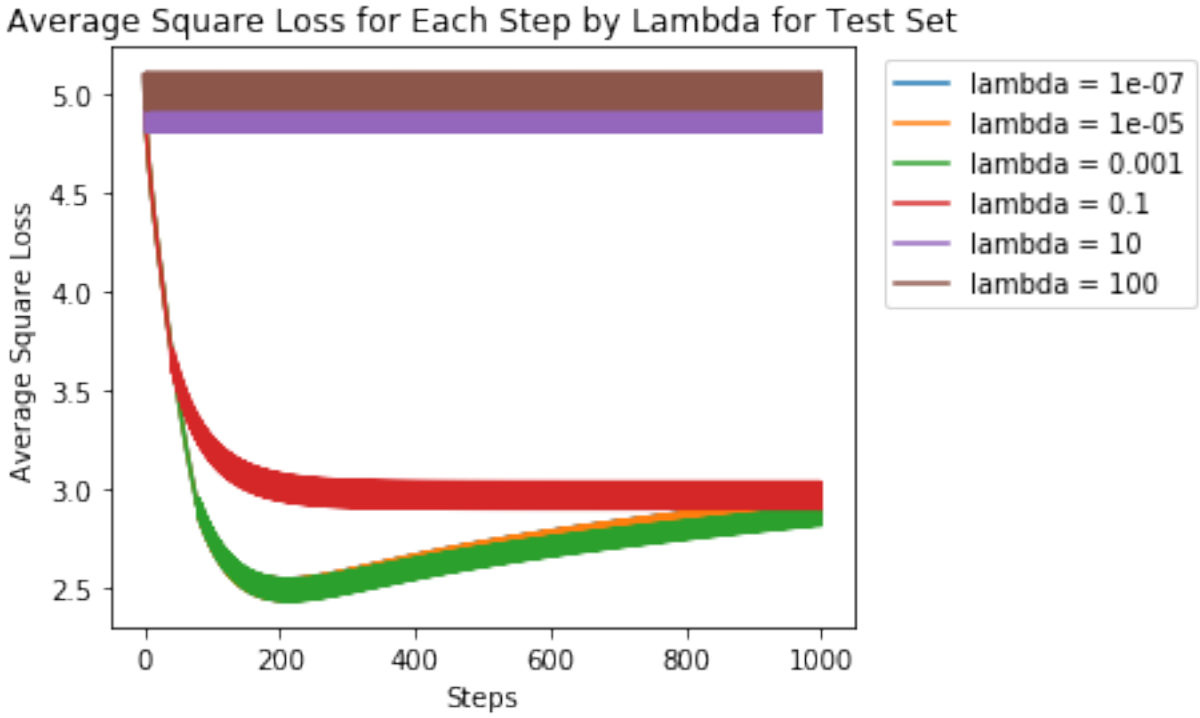
4. Compute gradient descent [10]

5. For regression problems, we may prefer to leave the bias term unregularized. One approach is to change $J(\theta)$ so that the bias is separated out from the other parameters and left unregularized. Another approach that can achieve approximately the same thing is to use a very large number B, rather than 1, for the extra bias dimension. When B is really large, it means the corresponding coefficient is small, leading to smaller effect of the intercept on regularization. Thus, the bias term is un-regularized.

   Suppose we have an affine function $f(x) = \theta_0 B + \theta^T x$. where we have separated out B from the vector x. If we increase B, the corresponding coefficient θ_0 will have to decrease by the same factor to end up with the same function f. A smaller coefficient θ_0 incurs less regularization penalty. As $B \to \infty, B_0 \to 0$ and the regularization effect approaches 0

6. Now fix $B = 1$, find the θ^*_λ that minimizes $J(\theta)$ over a range of λ. The goal is to find λ that gives the minimum average square loss on the test set. It's hard to predict what λ that will be, so you should start your search very broadly, looking over several orders of magnitude. For example, $\lambda \in \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 1, 10, 100\}$. Once you find a range that works better, keep zooming in. You may want to have $\log(\lambda)$ on the x-axis rather than λ. [If you like, you may use sklearn to help with the hyperparameter search.]

[9] Refer to `_regularized_square_loss_gradient` function from HW1-2019
[10] Refer to `regularized_grad_descent.` from HW1 - 2019



4 Stochastic Gradient Descent

When the training data set is very large, evaluating the gradient of the objective function can take a long time, since it requires looking at each training example to take a single gradient step. When the objective function takes the form of an average of many values, such as

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} f_i(\theta)$$

(as it does in the empirical risk), stochastic gradient descent (SGD) can be very effective. In SGD, rather than taking $-\nabla J(\theta)$ as our step direction, we take $-\nabla f_i(\theta)$ for some i chosen uniformly at random from $\{1, \ldots, m\}$. The approximation is poor, but we will show it is unbiased.
In machine learning applications, each $f_i(\theta)$ would be the loss on the ith example (and of course we'd typically write n instead of m, for the number of training points). In practical implementations for ML, the data points are **randomly shuffled**, and then we sweep through the whole training set one by one, and perform an update for each training example individually. One pass through the data is called an **epoch**. Note that each epoch of SGD touches as much data as a single step of batch gradient descent. You can use the same ordering for each epoch, though optionally you could investigate whether reshuffling after each epoch affects the convergence speed.

1. The objective function
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x_i) - y_i\right)^2 + \lambda\theta^T\theta$$
can be written in the form $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} f_i(\theta)$ with $f_i(\theta)$ to be:
$f_i(\theta) = (h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta$

2. We can easily show that the stochastic gradient $\nabla f_i(\theta)$, for i chosen uniformly at random from $\{1, \ldots, m\}$, is an **unbiased estimator** of $\nabla J(\theta)$. In other words, $\mathbb{E}\left[\nabla f_i(\theta)\right] = \nabla J(\theta)$ for any θ.

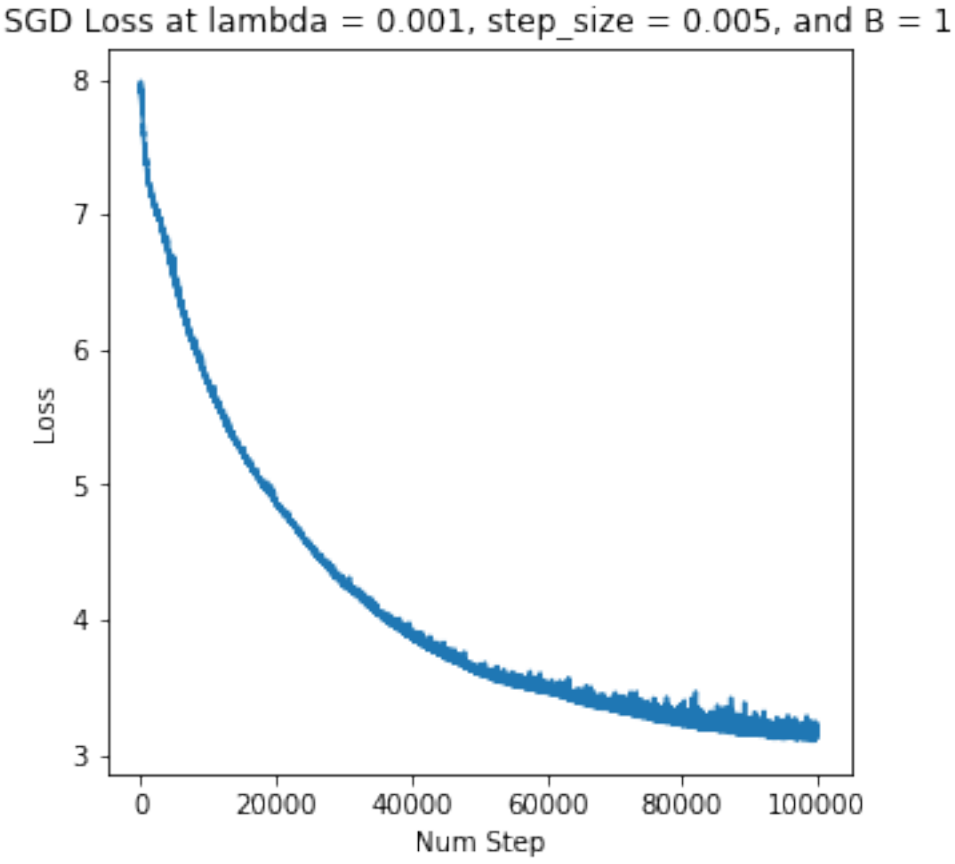
3. The update rule for θ in SGD for the ridge regression objective function.
$\nabla f_i(\theta) = 2(h_\theta x_i - y_i)x_i + 2\lambda x_i = 2(\theta_i^T x_i - y_i)x_i + 2\lambda x_i$
$\theta_{i+1} = \theta_i - \eta\nabla f_i(\theta) = \theta_i - \eta[2(\theta_i^T x_i - y_i)x_i + 2\lambda x_i]$

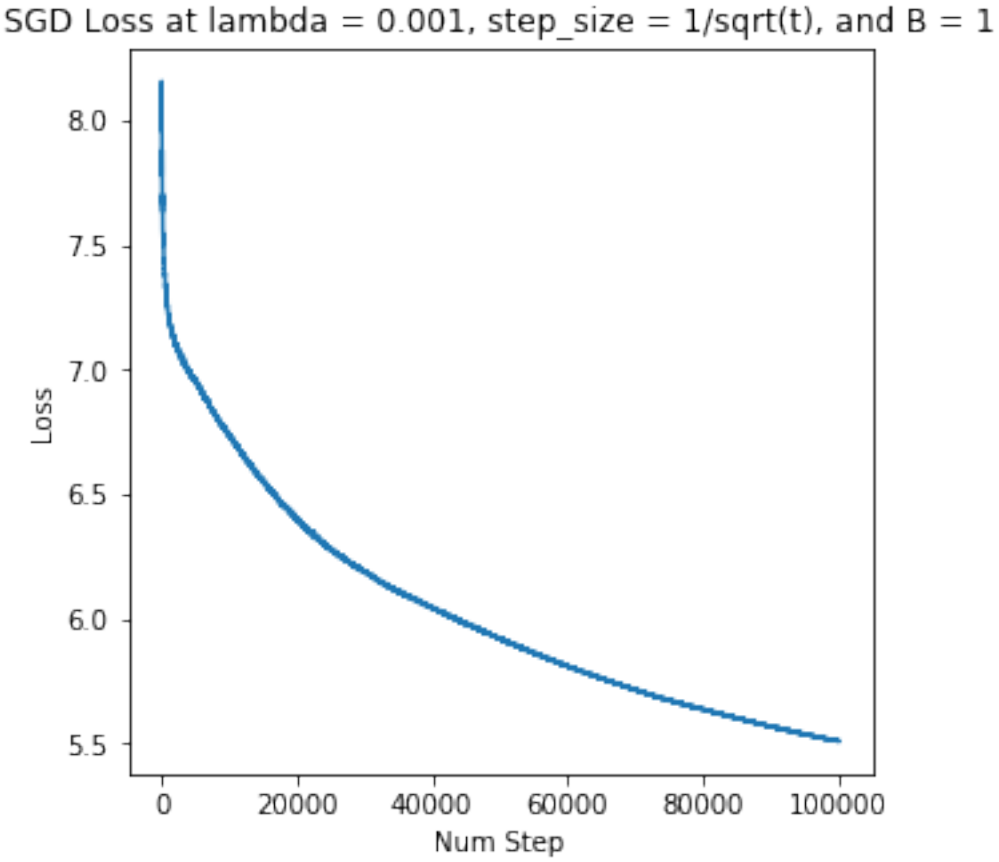
4. Implement Stochastic Grad Descent [11]

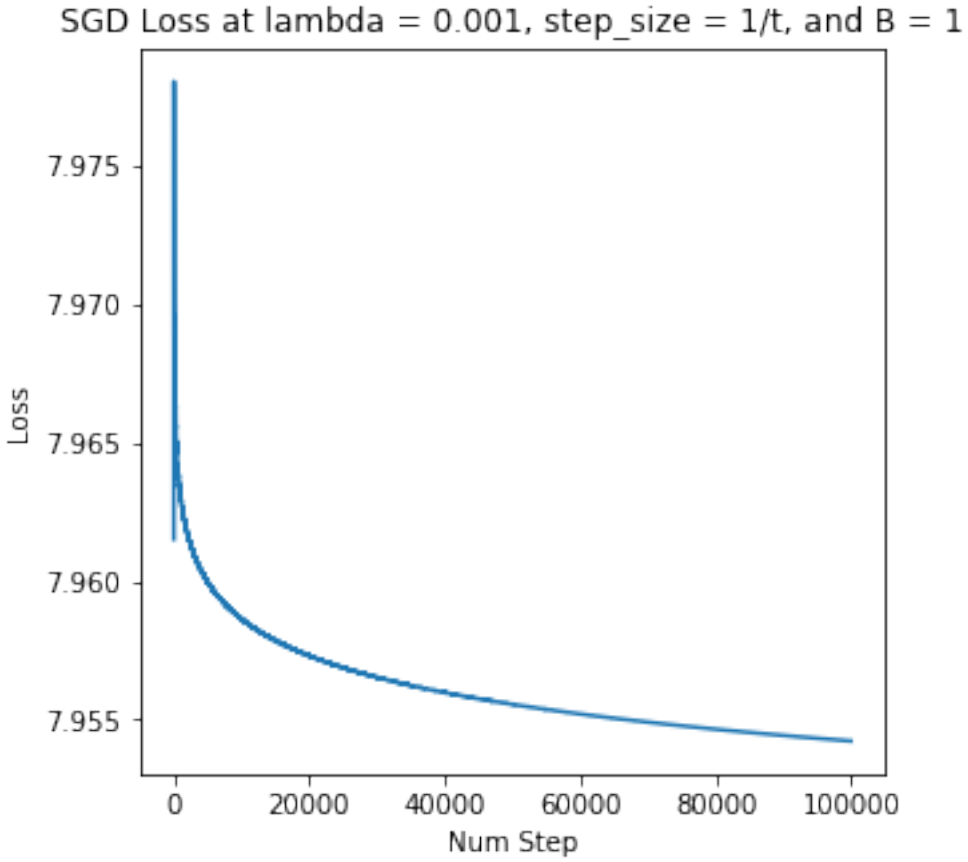
5. Use SGD to find θ_λ^* that minimizes the ridge regression objective for the λ and B selected in the previous problem.

6. Step Size: Try step sizes that decrease with the step number according to the following schedules: $\eta_t = \frac{C}{t}$ and $\eta_t = \frac{C}{\sqrt{t}}$, $C \leq 1$. Please include $C = 0.1$ in your submissions.

[11] Refer to `stochastic_grad_descent` from HW1 - 2019


SGD Loss at lambda = 0.001, step_size = 0.005, and B = 1
Loss
Num Step
8
7
6
5
4
3
0
20000
40000
60000
80000
100000


SGD Loss at lambda = 0.001, step_size = 1/sqrt(t), and B = 1
8.0
7.5
7.0
6.5
6.0
5.5
Loss
0
20000
40000
60000
80000
100000
Num Step



As shown above, the min loss is 3.131 achieve when step size is 0.05.
The θ is: [-1.341736 0.74105522 1.15087591 2.26252857 -2.47698603 -0.78870877 -0.66170294 -0.66170294 0.69995027 1.00150192 1.35592862 0.20464685 -1.49757454 -2.45934331 1.25037301 1.87837869 0.75761022 0.60828829 -0.0038115 -0.0038115 -0.0038115 0.05589895 0.05589895 0.05589895 0.07780999 0.07780999 0.07780999 0.0886481 0.0886481 0.0886481 0.09487685 0.09487685 0.09487685 -0.06370046 -0.06370046 -0.06370046 0.11790856 0.11790856 0.11790856 0.11413693 0.11413693 0.11413693 0.11280427 0.11280427 0.11280427 0.11218991 0.11218991 0.11218991 -1.33892822]

Some things to note:

- In this case we are investigating the convergence rate of the optimization algorithm with different step size schedules, thus we're interested in the value of the objective function, which includes the regularization term.
- Sometimes the initial step size (C for C/t and $C/\sqrt{t}$) is too aggressive and will get you into a part of parameter space from which you can't recover. Try reducing C to counter this problem.

- As we'll learn in an upcoming lecture, SGD convergence is much slower than GD once we get close to the minimizer. (Remember, the SGD step directions are very noisy versions of the GD step direction). If you look at the objective function values on a logarithmic scale, it may look like SGD will never find objective values that are as low as GD gets. In terminology we'll learn in Lecture 2, GD has much smaller "optimization error" than SGD. However, this difference in optimization error is usually dominated by other sources of error (estimation error and approximation error). Moreover, for very large datasets, SGD (or minibatch GD) is much faster (by wall-clock time) than GD to reach a point that's close [enough] to the minimizer.
- (Optional) There is another variant of SGD, sometimes called **averaged SGD**, in which rather than using the last parameter value we visit, say θ^T, we use the average of all parameter values we visit along the optimization path: $\theta = \frac{1}{T}\sum_{t=1}^{T}\theta^t$, where T is total number of steps taken. Try this approach[12] and see how it compares.

7. (Optional) Try a stepsize rule of the form $\eta_t = \frac{\eta_0}{1+\eta_0\lambda t}$, where λ is your regularization constant, and η_0 a constant you can choose. How do the results compare?

[12] Some theory for averaged SGD is given on page 191 of Understanding Machine Learning: From Theory to Algorithms. Refer to page 195 of the same book for other averaging techniques you can try.