Lecture 9

Math Foundations Team

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Introduction

- We will look at continuous optimization concepts in this lecture.
- There are two main branches of continuous optimization - constrained and unconstrained optimization.
- We seek the minimum of an objective function which we assume is differentiable.
- This is like finding the valleys of the objective function, and since the objective function is differentiable, the gradient tells us the direction to move to get the maximum increase in the objective function

Motivation

Consider the data in the given table

x	y
1	3.1
2	4.9
3	7.3
4	9.1

Easiest model of y one can think of is $y = ax + b$.

Let $\hat{y} = ax + b$ be the y predicted. Our aim is to find a and b such that the difference between actual y and the predicted y is minimum. So the loss function can be defined as

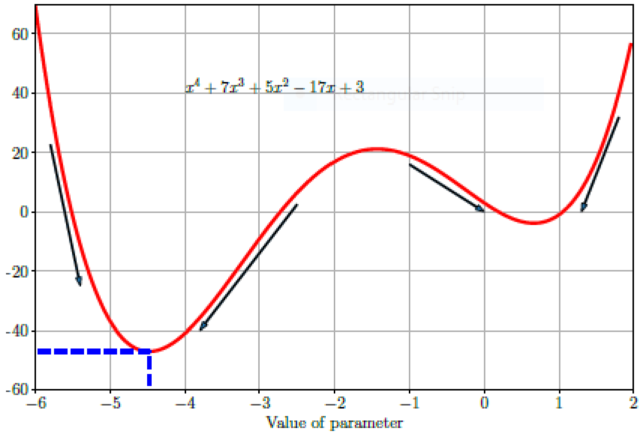
$$L(a, b) = \sum_{i=1}^{4} (y - \hat{y})^2 = \sum_{i=1}^{4} (y - (ax + b))^2$$

Then the problem will be to find a and b such that $L(a, b)$ is minimized which is an optimization problem.

Unconstrained Optimization

- We move in the direction of the negative gradient to decrease the objective function.
- We move until we encounter a point at which the gradient is zero.
- This constitutes a valley of the objective function, known as a local minimum
- For unconstrained optimization, this is enough for the big picture.

Example



$x^4 + 7x^3 + 5x^2 - 17x + 3$
60
40
20
0
-20
-40
-60
−6
−5
−4
−3
−2
−1
0
1
2
Value of parameter

Example

- Let $l(x) = x^4 + 7x^3 + 5x^2 - 17x + 3$.
- The gradient is $\frac{dl(x)}{dx} = 4x^3 + 21x^2 + 10x - 17$.
- Setting the gradient to zero identifies points corresponding to a local minimum or local maximum - there are three such points since this is a cubic equation.
- How do we check if we are dealing with a local minimum or local maximum? Look at the second derivative!
- If the second derivative at the point where the gradient vanishes is negative we are talking about a local maximum, otherwise it is a local minimum.
- The second derivative is $12x^2 + 42x + 10$

Optimization using gradient descent

- For low-order polynomials we can solve the equations analytically and find points at which the gradient is zero. Then we can do the second derivative test and identify whether these points are local minima/local maxima.
- In general we need to follow the negative gradient.
- Consider the problem of solving for the minimum of a real-valued function $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ where $f : R^d \rightarrow R$ is an objective loss function that captures the cost of using the current set of parameters.
- We assume our function f is differentiable but that the minimum cannot be found analytically in closed form.

Optimization using gradient descent

- The main idea of gradient-descent, a first-order optimization algorithm, is to take a step from the current point of magnitude proportional to the negative gradient of the function at the current point.
- In mathematical terms if $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma((\nabla f)(\boldsymbol{x}_0))^T$ for a small step-size $\gamma > 0$ then $f(\boldsymbol{x}_1) \le f(\boldsymbol{x}_0)$.
- The above suggests that in order to find the optimum of f, say at $f(\boldsymbol{x}_*)$, we can start at some initial point $\boldsymbol{x}_0$ and then iterate according to $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^T$
- For a suitable step-size γ_i, the sequence of points $f(\boldsymbol{x}_0) \ge f(\boldsymbol{x}_1) \ge \dots$ converges to some local minimum.

Example of gradient descent

- Let $f = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x}$ where $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\boldsymbol{A} = \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} -5 \\ -3 \end{bmatrix}$.
- From a previous lecture on Vector Calculus, we know that $\nabla f = \boldsymbol{x}^T\boldsymbol{A} + \boldsymbol{b}^T$
- Starting at $\boldsymbol{x}_0 = \begin{bmatrix} -3 \\ -1 \end{bmatrix}$ and iterating according to the equation given on a previous slide, we will eventually encounter the minimum value. (Refer the code grad_descent1.m.)

Choosing step-size

- If the step-size is too small, gradient descent will become too slow.
- If the step-size is too large, gradient descent might overshoot the target, fail to coverge, or even diverge.
- One way to handle the issue of choosing step-sizes is to choose a different step-size at each step. If a given step-size leads to an increase in the value of the cost function, we have been too aggressive, so a better step-size would be half the current step-size. On the other hand, if the value of the function has decreased, the step-size could be larger. In each case we undo the current step and change the step-size.
- This heuristic guarantees monotonic convergence.

Example of gradient descent

One reasonable way of choosing step size is finding $\gamma_i = \gamma$ such that $h(\gamma) = f(\boldsymbol{x}_{i+1}) = f(\boldsymbol{x}_i - \gamma((\nabla f)(\boldsymbol{x}_i))^T)$ is minimum. Here h is an univariate function of γ.
Consider $f(x, y) = x^2 + 3y^2$. Find minimum using gradient descent method with initial point (2,2).
Now $\nabla f(x, y) = [2x, 6y]$. Then $h(\gamma) = f(\boldsymbol{x} - \gamma((\nabla f)(\boldsymbol{x}))^T)$ $= (1-2\gamma)^2x^2 + 3(1-6\gamma)^2y^2$. Differentiating with respect to γ equating it to 0, we get
$h'(\gamma) = -4(1-2\gamma)x^2 - 36(1-6\gamma)y^2 = 0$. This gives us $\gamma = \frac{x^2+9y^2}{2x^2+54y^2}$. (Refer the code grad_descent2.m.) In some case finding step size can be expensive and different methods are used.

Gradient correctness

- Let $J(\boldsymbol{w}) = J(w_1, w_2, \dots w_d)$.
- We can compute the gradient of the function via a finite-difference approximation.
- We sample a few of the parameters $w_1, w_2, \dots w_d$ and compute the partial derivative $\frac{\partial J}{\partial w_i}$ using the finite-difference approximation.
- We compute the partial derivative approximately as follows:

$$\frac{\partial J}{\partial w_i} = \frac{J(w_1, w_2, \dots w_i + \Delta, \dots w_d) - J(w_1, w_2, \dots, w_i, \dots w_d)}{\Delta}$$

Learning rate decay

- How are we to decide the value of the learrning rate?
- What happens if we choose a large value for the learning rate and let it be constant?
- In this case, the algorithm might come close to the optimal answer in the very first iteration but it will then oscillate around the optimal point.
- What happens if we choose a small value for the learning rate and let it be constant?
- In this case, it will take a very long time for the algorithm to converge to the optimal point.

Learning rate decay

- The solution to the learning rate problem is to choose a variable learning-rate - large initially but decaying with time.
- This will enable the algorithm to make large strides towards the optimal point and then slowly converge.
- With a learning-rate dependent on time, the update step becomes $\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \alpha_t \nabla J$.
- The t in the update step is usually measured in terms of the number of cycles over all the training points, so the learning rate remains a constant during a particular cycle.

Learning rate decay

- The two most common decay functions are exponential decay and inverse decay, expressed mathematically as follows:

$$\text{exponential decay: } \alpha_t = \alpha_0 e^{-kt}$$
$$\text{inverse decay: } \alpha_t = \frac{\alpha_0}{1 + kt}$$

- In both of the above functions k controls the rate of decay.
- Another kind of decay function is the step decay where we reduce the learning rate by a constant factor every few steps of gradient descent.

Bold-driver algorithm

- In the bold-driver algorithm, the learning rate changes based on whether the objective function is improving or worsening.
- The learning rate is increased by a factor of around 5% in each iteration as long as the steps improve the objective function.
- As soon as the objective function worsens in a given step, the step is undone and the learning rate is reduced by a factor of 50%, and the new learning rate is used in a fresh update step.
- This process is continued to convergence.
- When the objective function is estimated using samples of data, we can only get a noisy estimate of the gradient, and the bold driver algorithm may not work properly.
- In such cases we can test the objective function and adjust the learning rate after m steps, rather than a single step.

Line search

- Line search uses the optimum step-size directly in order to provide the best improvement.
- It is rarely used in vanilla gradient descent because of its computational expense, but is helpful in some specialized variations of gradient descent.
- Let $J(\boldsymbol{w})$ be the function being optimized, and let $\boldsymbol{g}_t$ be the descent direction at the beginning of the tth step with the parameter vector being $\boldsymbol{w}_t$.
- In the steepest-descent method, the direction $\boldsymbol{g}_t$ is the same as $-\nabla J(\boldsymbol{w}_t)$.
- In light of the above the update step is $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t$.

Line search

- In line search the learning rate α_t is chosen at the tth step so as to minimize the value of the objective function at $\boldsymbol{w}_{t+1}$.
- Therefore the step-size α_t is computed as $\alpha_t = \min_\alpha J(\boldsymbol{w}_t + \alpha \boldsymbol{g}_t)$.
- After performing this step the gradient is computed at $\boldsymbol{w}_{t+1}$.
- The gradient at $\boldsymbol{w}_{t+1}$ will be perpendicular to the search direction at $\boldsymbol{g}_t$. Otherwise α_t will not be optimal and there will exist another point at which the objective function is minimized.

Line search

- To see this result, note that if the gradient at $\boldsymbol{w}_{t+1}$ is not perpendicular to the search direction $\boldsymbol{g}_t$, we can improve the objective function by moving a further $+\delta$ or $-\delta$ along $\boldsymbol{g}_t$ at $\boldsymbol{w}_{t+1}$.
- Using Taylor's series expansion we have $J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t \pm \delta \boldsymbol{g}_t) \approx J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_T) \pm \delta \boldsymbol{g}_t^T \nabla J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t)$.
- If $\boldsymbol{g}_t^T \nabla J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t) \neq 0$ then depending on its sign we can choose δ to be positive or negative to ensure that $J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t + \delta \boldsymbol{g}_t) < J(\boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t)$.

Line search

- One question remains - how do we perform the optimization $\min_\alpha J(\boldsymbol{w}_t + \alpha \boldsymbol{g}_t)$?
- An important property that we exploit of typical line-search settings is that the objective function is a unimodal function of α.
- This is especially true if we do not use the original objective function but quadratic or convex approximations of it.
- The first step in optimization is to identify a range $[0, \alpha_{\text{max}}]$ in which to perform the search for the optimum α.

Line search

- We can sweep evaluate the objective function values at geometrically increasing values of α.
- It is then possible to narraow the search interval by using binary-search, golden-section search method, or the Armijo rule.
- The first two of these methods are exact methods and need for the objective function to be unimodal in α, and the last of the methods is an inexact method that does not rely on unimodality.
- The Armijo rule therefore has broader applicability than either the binary search or golden-section search methods.

Binary search

- We start by initialising the search interval α to $[a, b] = [0, \alpha_{\max}]$.
- In binary search over $[a, b]$, we evaluate the objective function at two points close to the midpoint of the interval, ie at $\frac{a+b}{2}$ and $\frac{a+b+\epsilon}{2}$ and find out whether the function is increasing or decreasing at $\frac{a+b}{2}$. Here ϵ is a small value such as 10^{-6}.
- If the objective function is found to be increasing at $\frac{a+b}{2}$, we narrow the interval to $[a, \frac{a+b+\epsilon}{2}]$ and continue the search.
- Otherwise we narrow the interval to $[\frac{a+b}{2}, b]$ and continue the search.

Golden-section search

- We initialise the search interval to $[a, b] = [0, \alpha_{\text{max}}]$.
- The process of narrowing the search interval is different from the one adopted for binary search - here we use the fact that for any mid-samples m_1, m_2 in the region $[a, b]$ where $a < m_1 < m_2 < b$, at least one of the intervals $[a, m_1]$ or $[m_2, b]$ can be dropped. Sometimes we can go so far as to drop $[a, m_2]$ and $[m_1, b]$.
- When $\alpha = a$ yields the minimum for the objective function, i.e $H(\alpha)$, we can drop the interval $(m_1, b]$. Similarly when $\alpha = b$ yields the minimum for $H(\alpha)$ we can drop the interval $[a, m_2)$. When $\alpha = m_1$ is the value at which the minimum is achieved we can drop $(m_2, b]$. When $\alpha = m_2$ is the value at which the minimum is achieved we can drop $[a, m_1)$.

Golden-section search

- The new bounds on the search interval $[a, b]$ are reset based on the exclusions mentioned in the previous slide.
- At the end of the process we are left with an interval containing 0 or 1 evaluated point.
- If we have an interval containing no evaluated point, we select a random point $\alpha = p$ in the reset interval $[a, b]$, and then another point q in the larger of the intervals $[a, p]$ and $[p, b]$.
- On the other hand if we are left with an interval $[a, b]$ containing a single evaluated point $\alpha = p$, then we select $\alpha = q$ in the larger of the intervals $[a, p]$ and $[p, b]$.
- This yields another four points on which to continue the golden-section search. We continue until we achieve the desired accuracy.

Armijo rule

- The basic idea of the Armijo rule is that as one goes down the direction of improvement $\boldsymbol{g}_t$ at the starting point $\boldsymbol{w}_t$, the rate of improvement of the objective function comes down.
- The rate of improvement of the objective function along the search direction at the starting point is $\boldsymbol{g}_t^T \nabla F(\boldsymbol{w})$.
- The typical improvement of the objective function for a particular α can be expected to be $\alpha \boldsymbol{g}_t^T \nabla F(\boldsymbol{w})$ for most real-world objective functions.
- The Armijo rule is satisfied for a fraction $\mu \in (0, 0.5)$ of this improvement. A typical value of μ is 0.25.
- We want to find the largest step-size α such that $F(\boldsymbol{w}_t) - F(\boldsymbol{w}_t + \alpha \boldsymbol{g}_t) \geq \mu \alpha \boldsymbol{g}_t^T \nabla F(\boldsymbol{w}_t)$.

Armijo rule

- At first sight, it seems that the Armijo rule is not saying much since for small enough values of α, we can see that the finite-difference approximation ensures that the rule is true for $\mu = 1.0$.
- However we want a larger step-size to get faster convergence.
- What is the largest step-size we can use?
- We test successively decreasing values of α and stop the first time the condition $F(\boldsymbol{w}_t) - F(\boldsymbol{w}_t + \alpha \boldsymbol{g}_t) \geq \mu \alpha \boldsymbol{g}_t^T \nabla F(\boldsymbol{w}_t)$ is satisfied.
- In backtracking line search we start by checking $H(\alpha_{\text{max}})$, then $H(\beta\alpha_{\text{max}}) \ldots H(\beta^r \alpha_{\text{max}})$. We then use $\alpha = \beta^r \alpha_{\text{max}}$. Note that a typical value of β is 0.25.

When do we use line search?

- The line-search method can be shown to converge to a local optimum, but it is computationally expensive. For this reason, it is rarely used in vanilla gradient descent.
- Some methods like Newton's method, however, require exact line search.
- Fast inexact methods like Armijo's rule are used in vanilla gradient descent.
- One advantage of using exact line search is that fewer steps are needed to achieve convergence to a local optimum. This might more than compensate for the computational expense of individual steps.

Stochastic gradient descent

- Let us consider a machine learning problem consisting of loss functions incurred at N data points. Let the loss function at the nth data point be $L_n(\theta)$, and let the total loss be $L(\theta) = \sum_{n=1}^{n=N} L_n(\theta)$. Here θ is the parameter vector of interest and the goal of machine learning is to find the parameter vector θ that minimizes the loss function.
- The standard gradient descent procedure is a batch optimization method in that the update step considers the gradient of the entire loss function $L(\theta)$., i,e $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i \nabla L(\boldsymbol{\theta}_i)^T = \boldsymbol{\theta}_i - \gamma_i \sum_{n=1}^{n=N} \nabla L_n(\boldsymbol{\theta}_i)^T$.
- This step is expensive since we may have many data points in the batch.

Stochastic gradient descent

- Stochastic gradient descent is especially useful when the data set is very large and one can get good descent directions using modest samples of the data.
- Let there be n datapoints $\boldsymbol{X}_1, \boldsymbol{X}_2 \ldots \boldsymbol{X}_n$ and let S be a subset of the indices $\{1, 2, \ldots n\}$.
- The set S of data points can be treated as a sample and a sample-centric objective function can be constructed as follows: $J(S) = \sum_{i \in S} (\boldsymbol{w}^T \boldsymbol{X}_i - y_i)^2$
- The key idea in stochastic gradient descent is that the gradient of the sample-specific objective function, $J(S)$ with respect to the parameter vector $\boldsymbol{w}$ is an excellent approximation of the true gradient.

Stochastic gradient descent

- The update equation in case of stochastic gradient descent can be written as

$$[w_1, w_2, \ldots w_d]^T <= [w_1, w_2, \ldots w_d]^T - \alpha[\frac{\partial J(S)}{\partial w_1}, \frac{\partial J(S)}{\partial w_2}, \ldots \frac{\partial J(S)}{\partial w_d}]^T$$

- This approach is referred to as mini-batch stochastic gradient.
- In the extreme case S can contain only one index chosen at random, and the approach is then called as stochastic gradient descent.

Stochastic gradient descent

- We can perform a mini-batch gradient descent by taking only a small subset of the data points.
- In the extreme case we can choose only one L_n to estimate the gradient.
- Why does taking only a subset of data points work? It works because we get an unbiased estimate of the true gradient.
- We can show that when the learning rate decreases at a suitable rate and some mild assumptions can be made, stochastic gradient descent almost surely converges to a local minimum.
- Estimating the gradient out of large mini-batches will lead to lower variance in the estimate, in contrast to estimating the gradient from small batch sizes.