STA 314: Statistical Methods for Machine Learning I

Lecture - Gradient Descent

Xin Bing

Department of Statistical Sciences
University of Toronto

A general problem of solving a minimization problem

Suppose we want to solve the following problem

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\Theta} \mathcal{J}(\mathbf{w}; \mathcal{D}^{train}) := \operatorname*{argmin}_{\mathbf{w}\in\Theta} \mathcal{J}(\mathbf{w})$$

where

- $\mathcal{J}(\mathbf{w})$ is a differentiable function in $\mathbf{w} = (w_1, \ldots, w_p)$
- Θ is the parameter space of $\mathbf{w}$, typically chosen as a subspace of $\mathbb{R}^p$

- The **optimal solution** (if exists) must be a **critical point**, i.e. point to which the derivative is zero (partial derivatives to zero for multi-dimensional parameter).

Finding the optimal solution requires to solve the equations

- **Partial derivatives**: derivatives of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial w_1}\mathcal{J}(w_1, w_2) = \lim_{h\to 0} \frac{\mathcal{J}(w_1 + h, w_2) - \mathcal{J}(w_1, w_2)}{h}$$

- The minimum must occur at a point where the partial derivatives are zero

$$\begin{bmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_p} \end{bmatrix} = 0$$

- This implies a system of equations, which we can solve analytically in some scenarios.
- We may also use optimization techniques that iteratively get us closer to the solution.

Direct solution

- OLS:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}} \mathcal{J}(\mathbf{w}; \mathcal{D}^{train}) = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}} ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2.$$

  The partial derivatives w.r.t. $\mathbf{w}$ are

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}).$$

  (If not familiar with multi-dimensional derivatives, calculate $\frac{\partial \mathcal{J}}{\partial w_j}$ and stack them together).
  Setting the above equal to zero results

$$\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}, \qquad \Longrightarrow \qquad \hat{\mathbf{w}} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}.$$

Direct solution

- Ridge:

$$\hat{\mathbf{w}}_\lambda^R = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w}; \mathcal{D}^{train}) = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}}\, ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda||\mathbf{w}||_2^2.$$

  The partial derivatives w.r.t. $\mathbf{w}$ are

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w}.$$

  Setting the above equal to zero results

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)\hat{\mathbf{w}}_\lambda^R = \mathbf{X}^\top\mathbf{y}, \qquad \Rightarrow \qquad \hat{\mathbf{w}}_\lambda^R = \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p\right)^{-1}\mathbf{X}^\top\mathbf{y}.$$

Gradient Descent

- Now let's see a second way to solve

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \mathcal{J}(\mathbf{w})$$

which is more broadly applicable: **gradient descent**.

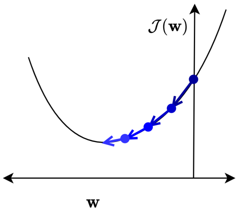
- Many times, we do not have a direct solution to

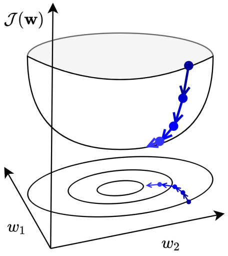
$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = 0.$$

- Gradient descent is an **iterative algorithm**, which means we apply an update repeatedly until some criterion is met.

Gradient Descent

We **initialize** **w** to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of **steepest descent** of the loss function $\mathcal{J}$.





What is the direction of the steepest descent of $\mathcal{J}(\mathbf{w})$ at **w**?

Gradient Descent

- By definition, the direction of the greatest increase in $\mathcal{J}(\mathbf{w})$ at $\mathbf{w}^{(0)}$ is its gradient

$$\left.\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(0)}} \in \mathbb{R}^p$$

- So, we update $\mathbf{w}$ in the **opposite** direction of the gradient at $\mathbf{w}^{(0)}$:

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \alpha \cdot \left.\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{(0)}}$$

  for some $\alpha > 0$.

- If α is chosen small, then

$$\mathcal{J}(\mathbf{w}^{(1)}) < \mathcal{J}(\mathbf{w}^{(0)})$$

  unless $\partial \mathcal{J}(\mathbf{w})/\partial \mathbf{w}$ at $\mathbf{w}^{(0)}$ is zero.

Gradient descent: coordinatewise viewpoint

By repeating the above procedure: for $k = 0, 1, 2, \cdots$,

- at the $(k+1)$th iteration, for each $j \in \{1, 2, \ldots, p\}$,

$$w_j^{(k+1)} \leftarrow w_j^{(k)} - \alpha \cdot \frac{\partial \mathcal{J}}{\partial w_j}\bigg|_{\mathbf{w}=\mathbf{w}^{(k)}}$$

- $\alpha > 0$ is a **learning rate** (or step size).
  - The larger it is, the faster $\mathbf{w}^{(k+1)}$ changes relative to $\mathbf{w}^{(k)}$
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.

Gradient descent for OLS

Example

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^p}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{w}), \qquad \mathcal{J}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2.$$

Update rule in vector form at the $k+1$th iteration:

$$\begin{aligned}\mathbf{w}^{(k+1)} &\leftarrow \mathbf{w}^{(k)} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{(k)}} \\ &= \mathbf{w}^{(k)} + 2\alpha \mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}^{(k)}).\end{aligned}$$

Initialization: $\mathbf{w}^{(0)} = 0$.

Gradient descent for Ridge

Example

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w}), \qquad \mathcal{J}(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda||\mathbf{w}||_2^2.$$

Update rule in vector form at the $k+1$th iteration:

$$\begin{aligned} \mathbf{w}^{(k+1)} &\leftarrow \mathbf{w}^{(k)} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{(k)}} \\ &= \mathbf{w}^{(k)} + 2\alpha\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}^{(k)}) - 2\alpha\lambda\mathbf{w}^{(k)}. \end{aligned}$$

Initialization: $\mathbf{w}^{(0)} = 0$.

Subgradient descent for convex but nondifferentiable functions

When $\mathcal{J}(\cdot)$ is convex but not differentiable at $\mathbf{x}$, one can use its subgradient at $\mathbf{x}$, defined as

$$\partial \mathcal{J}(\mathbf{x}) := \{v : \mathcal{J}(\mathbf{y}) \geq \mathcal{J}(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\top v, \ \forall \mathbf{y} \in \Theta\}.$$

For $f(x) = |x|$,

$$\partial f(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \\ (-1, 1) & x = 0 \end{cases}$$

For $f(\mathbf{x}) = ||\mathbf{x}||_1$,

$$[\partial f(\mathbf{x})]_j = \begin{cases} 1 & x_j > 0 \\ -1 & x_j < 0 \\ (-1, 1) & x_j = 0 \end{cases}$$

Gradient descent for Lasso

Example

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^p} \mathcal{J}(\mathbf{w}), \qquad \mathcal{J}(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda||\mathbf{w}||_1.$$

Update rule in vector form at the $k+1$th iteration:

$$\begin{aligned}\mathbf{w}^{(k+1)} &\leftarrow \mathbf{w}^{(k)} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{(k)}} \\ &= \mathbf{w}^{(k)} + 2\alpha\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}^{(k)}) - \alpha\lambda\partial(||\mathbf{w}||_1) \mid \mathbf{w}^{(k)}.\end{aligned}$$

Initialization: $\mathbf{w}^{(0)} = 0$.

Stopping criteria

When do we stop?

- The objective value stops changing:

$$|\mathcal{J}(\mathbf{w}^{(k+1)}) - \mathcal{J}(\mathbf{w}^{(k)})| \text{ is small,} \qquad \text{e.g. } \leq 10^{-6}.$$

- The parameter stops changing: $||\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}||_2$ is small or $||\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}||_2 / ||\mathbf{w}^{(k)}||_2$ is small.

- When we reach the maximum number (M) of iterations, e.g. $M = 1000$.

When do we expect Gradient Descent (GD) to work?

Recall that we try to solve

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \Theta} \mathcal{J}(\mathbf{w}).$$

- Obviously, $\mathcal{J}$ needs to be differentiable.
- If $\mathcal{J}$ is also a **convex function** and Θ is a convex set, then GD with a suitable choice of step size guarantees to find the optimal solution.
- In many cases, $\Theta = \mathbb{R}^p$ which is convex.

Convex Sets

A set $\mathcal{S}$ is convex if for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{S}$,

$$(1-\lambda)\mathbf{x}_0 + \lambda \mathbf{x}_1 \in \mathcal{S} \quad \text{for all } 0 \leq \lambda \leq 1.$$

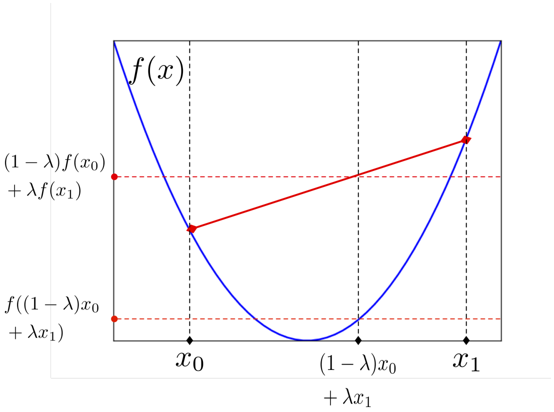
The Euclidean space $\mathbb{R}^p$ is a convex set.

Convex Sets and Functions

- A function f is **convex** if for any $\mathbf{x}_0, \mathbf{x}_1$ in the domain of f,

$$f((1-\lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1) \leq (1-\lambda)f(\mathbf{x}_0) + \lambda f(\mathbf{x}_1), \quad \forall \lambda \in [0,1].$$

- Equivalently, the set of points lying above the graph of f is convex.
- Intuitively: the function is bowl-shaped.



How to tell a loss is convex?

1. Verify the definition.
2. If f is twice differentiable and $f''(x) \geq 0$ for all x, then f is convex.
   - the least-squares loss function $(y - t)^2$ is convex as a function of t
   - the function
   $$-yt + \log\left(1 + e^t\right)$$
   is convex in t.
3. There are other sufficient conditions for convex, but non-differentiable, functions!

4 A composition rule: **linear functions preserve convexity**.
If f is a convex function and g is a linear function, then both $f \circ g$ and $g \circ f$ are convex.

- the least-square loss $(y - \mathbf{x}^\top \mathbf{w})^2$ is convex in $\mathbf{w}$
- the negative log-likelihood under logistic regression

$$-y\mathbf{x}^\top \mathbf{w} + \log\left(1 + e^{\mathbf{x}^\top \mathbf{w}}\right)$$

  is convex in $\mathbf{w}$.
- Both $\sum_i (y_i - \mathbf{x}_i^\top \mathbf{w})^2$ and $\sum_i \left[-y_i \mathbf{x}_i^\top \mathbf{w} + \log\left(1 + e^{\mathbf{x}_i^\top \mathbf{w}}\right)\right]$ are convex in $\mathbf{w}$.

5 There are more composition rules!

6 A great book:

Convex Optimization, Stephen Boyd and Lieven Vandenberghe.

Gradient Descent for Linear Regression

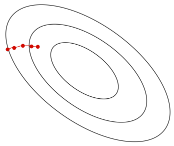
- The squared error loss
$$\sum_{i=1}(y_i - \mathbf{x}_i^\top \mathbf{w})^2$$
of linear regression is a convex function. So there is a unique solution.

- Even in the case when a closed-form solution exists, we sometimes need to use GD.

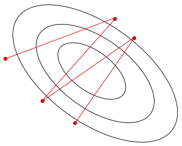
- Why gradient descent, if we can find the optimum directly?
  - When p is large, GD is more efficient than direct solution
    - Linear regression solution: $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$
    - Matrix inversion is an $\mathcal{O}(p^3)$ algorithm
    - Each GD update costs $\mathcal{O}(np)$
    - Or less with stochastic GD (Stochastic GD, later)
    - Huge difference if $p \gg \sqrt{n}$

Effect of the learning rate (step size)

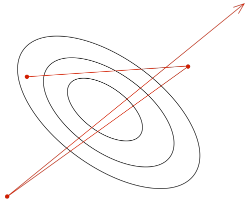
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small: slow progress



α too large: oscillations

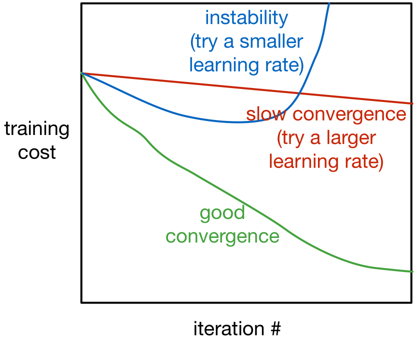


α much too large: instability

- Good values are typically small. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

Training Curves

- To diagnose optimization problems, it's useful to look at the **training cost**: plot the training cost as a function of iteration.



- **Warning**: the training cost could be used to check whether the optimization problem reaches certain convergence. But
  - It does not tell whether we reach the global minimum or not
  - It does not tell anything on the performance of the fitted model

Batch Gradient Descent

- Recall for OLS that:

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \sum_{i=1}^{n} \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i.$$

- Computing the gradient requires summing over **all** of the training examples, which can be done via matrix / vector operations.
  The fact that it uses all training samples is known as **batch training**.

Stochastic Gradient Descent

- Batch training is impractical if you have a large dataset (e.g. millions of training examples, $n \approx 10$ millions)!

- **Stochastic gradient descent (SGD)**: update the parameters based on the gradient for a single training example.

  For each iteration $k \in \{1, 2, \ldots\}$,

  1. Choose $i \in \{1, \ldots, n\}$ uniformly at random
  2. Update the parameters by ONLY using this ith sample,

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i$$

Stochastic Gradient Descent

$$\hat{\mathbf{w}}^{(k+1)} = \hat{\mathbf{w}}^{(k)} + \alpha \left[y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}^{(k)} \right] \mathbf{x}_i$$

Pros:

- Computational cost of each SGD update is independent of n!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: the gradients between SGD and GD have the same expectation for i.i.d. data.

Stochastic Gradient Descent

Cons: using single training example to estimate gradient:

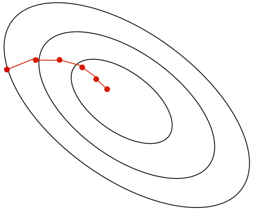
- Variance in the estimate may be high

Compromise approach:

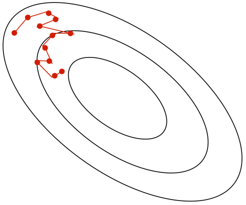
- compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, n\}$, called a **mini-batch**.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.

Stochastic Gradient Descent

- Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.



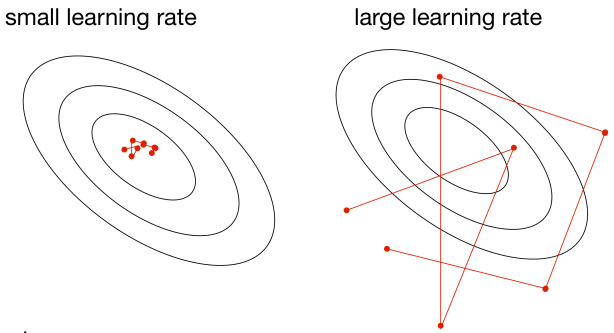
batch gradient descent



stochastic gradient descent

SGD Learning Rate

- In stochastic training, the learning rate also influences the **fluctuations** due to the stochasticity of the gradients.



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations