OPTIMIZATION AND GRADIENT DESCENT BASICS

Shingchern D. You

Optimization

- Unconstrained optimization problem

  Ex: $\min f(x, y) = 3x - y^2$

- Constrained optimization problem

  Ex: $\min f(x, y) = x + y$ subject $x^2 + y^2 = 1$

- It is easier to solve unconstrained optimization problem
- Some constrained optimization problems can be converted into unconstrained optimization problems

Optimization

- Finding optimal solution
  - Closed form
  - Iterative method (eg. gradient descent)
  - Randomized method (eg. Simulated annealing)

Unconstrained Optimization

- Closed form approach for one variable
- It is known that if a_0 is a local minimum/maximum, then $f'(a_0) = 0$
- We can find a_0 by solving $f'(x) = 0$ with additional verification steps
- Multiple variables: If (a_0, b_0) is a local minimum/maximum, then $\frac{\partial}{\partial x} f(x, y)|_{(a_0, b_0)} = 0$ and $\frac{\partial}{\partial y} f(x, y)|_{(a_0, b_0)} = 0$

Gradient

- A multivariable function $f(x_1, \ldots, x_p)$ can also be written as $f(\mathbf{x})$ where $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$
- We use $\nabla = \begin{bmatrix} \frac{\partial}{\partial x_1} \\ \vdots \\ \frac{\partial}{\partial x_p} \end{bmatrix}$ as the differential operator (also called Laplace operator or Laplacian, read as del)

Gradient

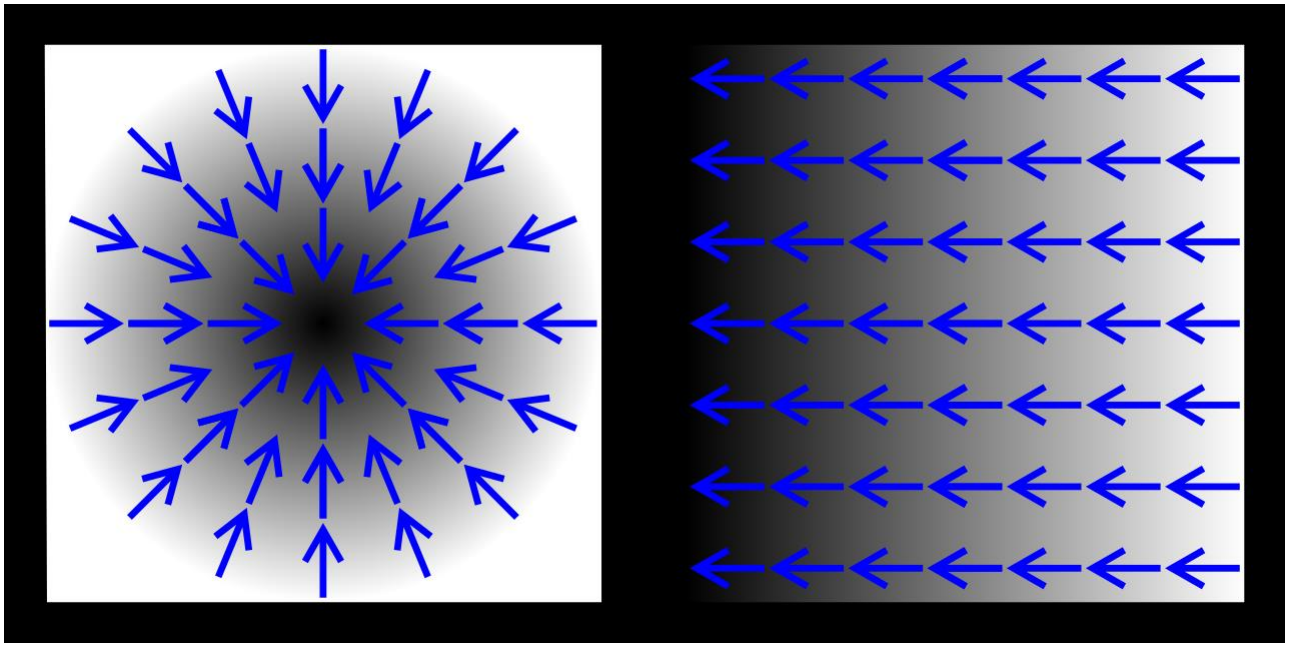
- If $f(\mathbf{x})$ has an extremum at $\mathbf{x}_0$, then $\nabla f(\mathbf{x}_0) = \mathbf{0}$
- So we can find $\mathbf{x}_0$ by solving $\nabla f(\mathbf{x}) = \mathbf{0}$ with additional verification steps
- The term ∇f is also called **gradient** of f

Gradient

- Theorems (from P V O'Neil, 3rd Ed, pp. 860)
- $f: R^p \to R$ and $\nabla f: R^p \to R^p$ and $\nabla f(\boldsymbol{p}_0) \neq \mathbf{0}$
- The direction from point $\boldsymbol{p}_0$ where f has **max rate of change** is the direction of $\nabla f(\boldsymbol{p}_0)$, with the rate of $\|\nabla f(\boldsymbol{p}_0)\|$
- The direction from $\boldsymbol{p}_0$ where f has min rate of change is the direction of $-\nabla f(\boldsymbol{p}_0)$, with the rate of $-\|\nabla f(\boldsymbol{p}_0)\|$

Gradient

- Let $f(\mathbf{x}) = c_0$ be a level surface, then $\nabla f(\boldsymbol{p}_0)$ is **normal** to the point $\boldsymbol{p}_0$ on the surface (2D projection of 3D function)
- The gradient denote the direction of greatest change of a scalar function. The values in greyscale and increase from white (low) to dark (high) – wiki gradient

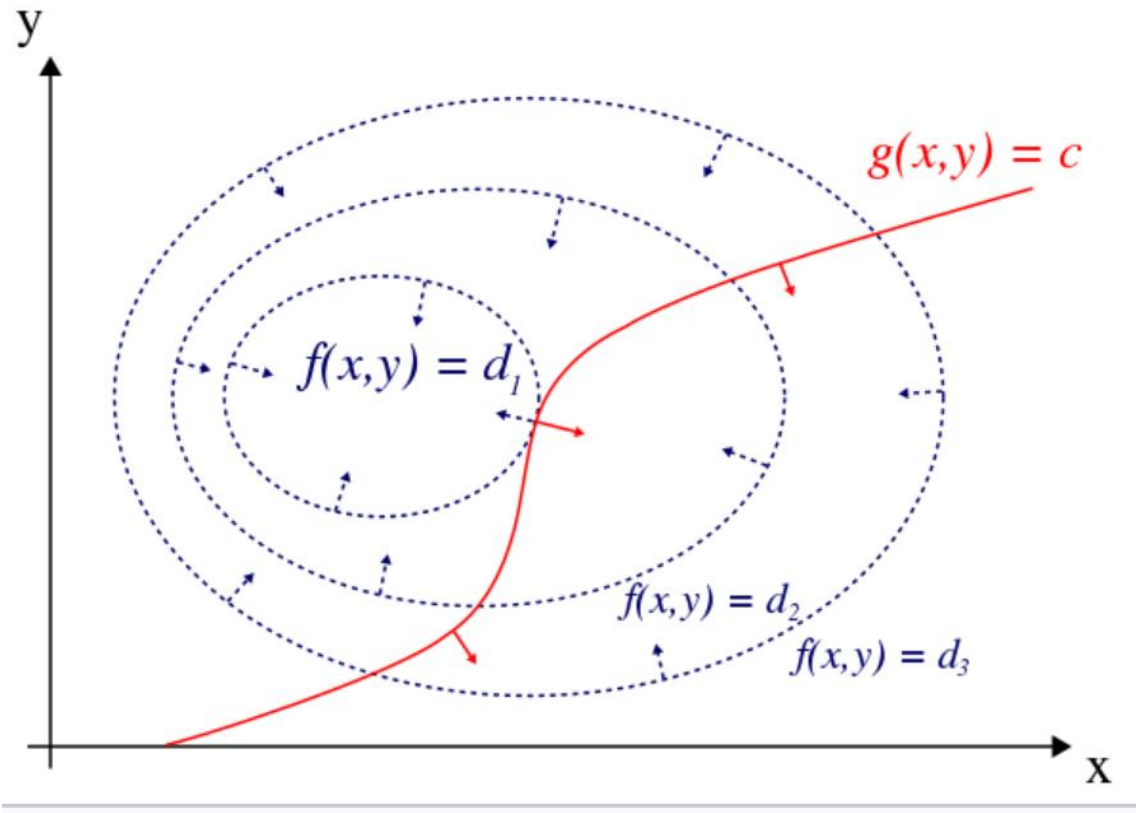


Unconstrained Optimization

- Iterative solution widely used: gradient descent
- To be covered later
- Gradient descent has many variations
  - Example: Conjugate gradient descent, gradient descent with momentum, Newton's method, etc.
- Also need auto differentiation to be fully automatic

Constrained Optimization

- Want to maximize $f(x, y)$ subject to $g(x, y) = 0$
- Figure from https://en.wikipedia.org/wiki/Lagrange_multiplier



Lagrange multiplier

- Assume that contours of *f* are an uphill and $d_1 > d_2$
- Consider the contour $f(x, y) = d_2$
- If d_2 increases a little bit, $g(x, y)$ curve still intersect with $f(x, y)$ in two points (i.e., solution exists)
- That is, $d_2 \uparrow$, then $f(x, y) \uparrow$ (OK)
- Extreme value doesn't exist in contour $f(x, y) = d_2$

Lagrange multiplier

- Therefore, the extreme value is the point where the red line **tangentially** touches the blue contour
- They have the same tangent vector
- Tangent vector is orthogonal to normal vector
- Recall normal vector is computed by ∇f

Lagrange multiplier

- Thus, gradients of f and g are parallel and opposite to each other (can be formally proved)
- Therefore, $\nabla f = \lambda \nabla g$ for some λ (λ is called **Lagrange multiplier**)
- Recall that maximal (minimal) point, $\nabla f = \mathbf{0}$. Thus, $\lambda \nabla g = \mathbf{0}$, too

Lagrange multiplier

- Another explanation
  (http://episte.math.ntu.edu.tw/entries/en_lagrange_mul/index.html)
- Think that y is an implicit function of x
- To find extreme value, we want

$$\frac{d}{dx} f\big(x, y(x)\big) = 0$$

- Therefore,

$$f_x + f_y \frac{dy}{dx} = 0$$

Lagrange multiplier

- In addition, we have

$$g(x,y) = g\big(x, y(x)\big) = 0$$

- Therefore,

$$\frac{d}{dx} g\big(x, y(x)\big) = 0$$

$$g_x + g_y \frac{dy}{dx} = 0$$

Lagrange multiplier

- In matrix form

$$\begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix} \begin{bmatrix} 1 \\ \frac{dy}{dx} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Therefore,

$$\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix} = 0$$

- Or, $f_x = \lambda g_x$ and $f_y = \lambda g_y$
- General form: $\nabla f = \lambda \nabla g$

Lagrange multiplier

- To use the Lagrange multiplier method, we write the auxiliary equation as

$$\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda g(x, y)$$

and solve the followings

$$\frac{\partial}{\partial x}\mathcal{L} = 0, \frac{\partial}{\partial y}\mathcal{L} = 0, \frac{\partial}{\partial \lambda}\mathcal{L} = 0$$

- Note: $\frac{\partial}{\partial \lambda}\mathcal{L} = 0$ is the original constraint
- Lagrange multiplier method is only a **necessary** condition for optimality

Lagrange multiplier example

- Example from wiki
- Maximize $f(x, y) = x + y$ subject to $x^2 + y^2 = 1$
  - $\mathcal{L}(x, y, \lambda) = x + y + \lambda(x^2 + y^2 - 1)$
  - $\frac{\partial}{\partial x}\mathcal{L} = 1 + 2\lambda x = 0$
  - $\frac{\partial}{\partial y}\mathcal{L} = 1 + 2\lambda y = 0$
  - $\frac{\partial}{\partial \lambda}\mathcal{L} = x^2 + y^2 - 1 = 0$

Lagrange multiplier example

- Finally, we have the following stationary points

in the form of (x, y, λ)

$$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) \text{ and } \left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$$

The first point is the max point

- Note: the auxiliary equation is NOT exactly the cost (loss) function, so it may not be directly used with gradient descent

Numerical solution

- What if we want to find solutions numerically?
- Note that we have a system of (nonlinear) equations if we use hands to do partial derivatives
- Therefore, we may use existing methods, such as Newton's method, to find roots of the equations
- We also have alternative methods

Numerical solution

- To use gradient descent for constrained optimization problem (given later),
  - Treat constraint as a penalty function
  - Use projected gradient descent
- For example, we may use the following as the cost function for the previous example

$$J(x, y) = x + y - 100(x^2 + y^2 - 1)^2$$

- Of course, the solution in this case is not exact due to the use of the penalty function

KKT-conditions

- Extension of Lagrange multiplier method to include inequality
- Optimize $f(x)$ subject to $g_i(x) \leq 0$ and $h_j(x) = 0$ $(i = 1..m, j = 1..n)$
- We write down auxiliary equation $\nabla \mathcal{L}(\ldots) = 0$ as well as additional conditions to solve the problem

KKT-conditions

- Ex: $\min f(x)$ subject to $h(x) \leq 0$ and $g(x) = 0$
- We do a similar trick as in Lagrange multipliers

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda g(x) + \mu h(x)$$

- Therefore, we have (detailed omitted)

$$\nabla_x \mathcal{L}(x, \lambda, \mu) = 0$$
$$g(x) = 0$$
$$h(x) \leq 0$$
$$\mu \geq 0$$
$$\mu h(x) = 0$$

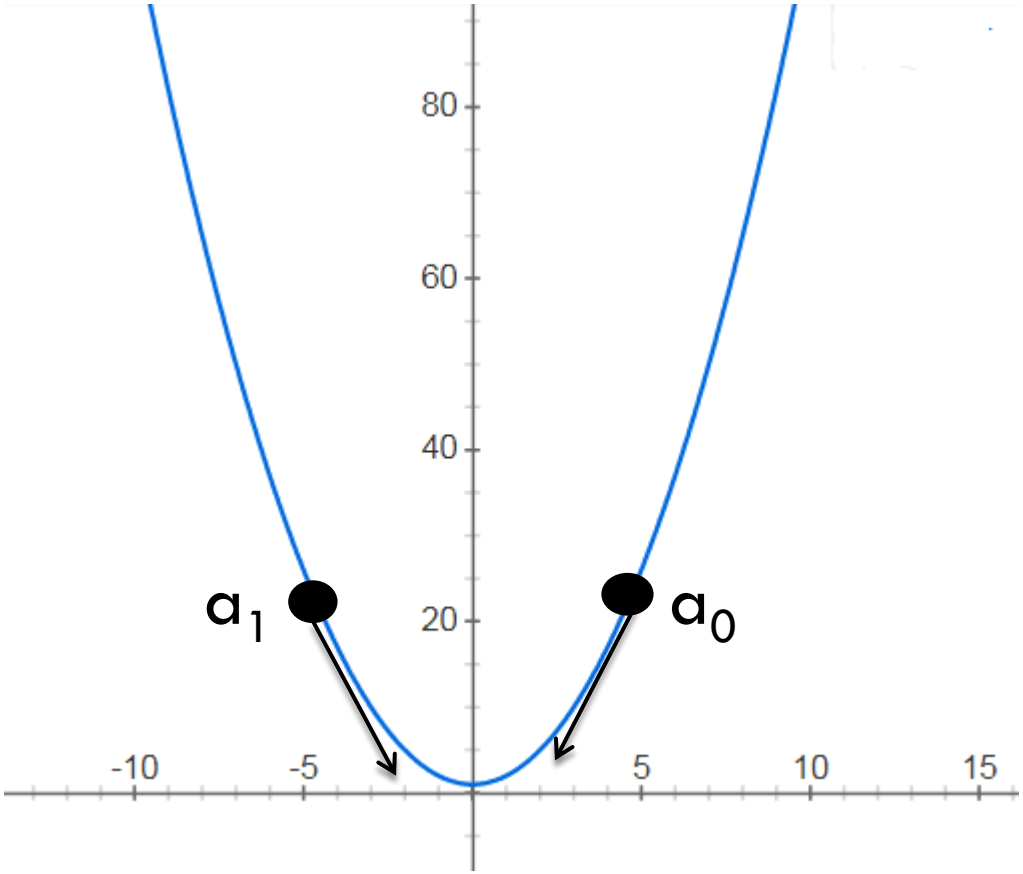
- Known as **Karush–Kuhn–Tucker (KKT) conditions** (first-order necessary conditions)

KKT-conditions

- KKT conditions can be extended to handle multivariable functions
- Usually solved by iterative methods
  - Use tricks (such as projection, active set, or interior point method) to deal with inequality pieces
  - The rest equations are nonlinear equations (same as Lagrange case)
- Support vector machine (SVM) is based on KKT conditions

Unconstrained optimization

- Consider simple **unconstrained optimization** problem: $f(x) = x^2 + 1$



Gradient Descent Method

- Let Δ be a small number
- At point a_0, $f'(a_0) > 0$, if we want $f(a_0 + \Delta) < f(a_0)$, we know $\Delta < 0$
- At point a_1, $f'(a_1) < 0$, if we want $f(a_1 + \Delta) < f(a_1)$, we know $\Delta > 0$
- To find **minimum** for function f, at $x = a_k$ we need to set $\Delta = -\eta f'(a_k)$ where η is a small positive number

Gradient Descent Method

- Extend this idea, we have

$$\mathbf{x}(k+1) \leftarrow \mathbf{x}(k) - \eta \nabla f(\mathbf{x}(k))$$

where k is iteration index

- This approach is called gradient descent method (or gradient search)
- η is a small positive value
- How to find a "good" η for higher convergence is called "line search"

Gradient Descent Method

- Ref: http://theory.stanford.edu/~tim/s15/l/l15.pdf
- Another method to explain gradient descent method
- Consider a simple linear case ($\mathbf{x} \in R^p$)

$$f(\mathbf{x}) = \boldsymbol{w} \cdot \mathbf{x} + b$$

Where $\boldsymbol{w} \in R^p$ and $b \in R$ are constants and $\cdot$ denotes inner product, or $\boldsymbol{w} \cdot \mathbf{x} = \boldsymbol{w}^T \mathbf{x}$

- Want to find vector $\boldsymbol{u} \in R^p$ with $\|\boldsymbol{u}\| = 1$ such that $f(\mathbf{x} + \boldsymbol{u})$ is minimal

Gradient Descent Method

- We know $f(\mathbf{x}+\boldsymbol{u}) = \boldsymbol{w}^T(\mathbf{x}+\boldsymbol{u}) + b = f(\mathbf{x}) + \boldsymbol{w}^T\boldsymbol{u}$
- Thus, $f(\mathbf{x}+\boldsymbol{u})$ is minimal only if $\boldsymbol{u} = -\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$
- In general, a multivariable function is not linear
- However, we can make it almost linear if considering small length of $\boldsymbol{u}$

Gradient Descent Method

- For small $\boldsymbol{u}$ (in length), we have Taylor's expansion for f about $\mathbf{x} = \mathbf{x}_0$ as

$$f(\mathbf{x}_0 + \boldsymbol{u}) \approx f(\mathbf{x}_0) + \boldsymbol{u}^T f(\mathbf{x}_0)'$$

- As the above equation is also a linear function, when compared with eq in previou slice, we have $\boldsymbol{w} = \nabla f(\mathbf{x}_0)$
- Because we want $\boldsymbol{u}$ to be small, minimal $f(\mathbf{x}_0 + \boldsymbol{u})$ occurs if $\boldsymbol{u}$ is in opposite direction of $\nabla f(\mathbf{x}_0)$,i.e., $\boldsymbol{u} = -\eta \nabla f(\mathbf{x}_0)$

Estimate Gradient

- Sometimes it is not easy to compute gradient, we may use the following to estimate gradient

$$\frac{\partial}{\partial x_j} f(\mathbf{x}) \approx$$

$$\{f(x_1, \ldots, x_{j-1}, x_j + h, x_{j+1}, \ldots, x_p) - f(\mathbf{x})\}/h$$

- It is also useful to monitor the differences between true gradient and estimated gradient during iteration

Estimate Gradient

- Another (better) way to estimate gradient is through centered difference formula:

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$$ with small h

Gradient Descent Algorithm

- Algorithm to find (candidate) $\mathbf{w}$ to minimize $f(\mathbf{w})$
  - Step 1: Find initial point $\mathbf{w}(k)$. Let $k \leftarrow 0$
  - Step 2: While $\|\nabla f(\mathbf{w}(k))\| > \epsilon$ do

$$\mathbf{w}(k+1) \leftarrow \mathbf{w}(k) - \eta \nabla f(\mathbf{w}(k))$$
$$k \leftarrow k+1$$

- In the algorithm, ϵ is a small positive number to determine the termination of the algorithm
- η is called **step size** (**also called learning rate**) and should be small to prevent problems

Gradient Descent Method

- Demo of learning rate: here
- Gradient descent method only finds (candidates of) **local** extreme values (not global ones)
- The selection of η affect the convergence speed
  - Simplest algorithm: use a constant η
  - More complicated method: line search is a binary search algorithm to find the value of η that minimizing f over the line $\mathbf{w} - \eta \nabla f(\mathbf{w})$
- Exercise: How to perform **gradient ascent**

Gradient Descent for Regression

- Given a known dataset of $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(n)} \in R^p$ with corresponding labels $d_{(1)}, \ldots, d_{(n)} \in R$, we define the error in the linear case as

$$\mathcal{E}_{(k)} = \boldsymbol{w}^T \boldsymbol{x}_{(k)} - d_{(k)}, 1 \leq k \leq n$$

- If we want to extend to nonlinear function, we may use $\mathcal{E}_{(k)} = f\left(\boldsymbol{w}^T \boldsymbol{x}_{(k)}\right) - d_{(k)}$

  where $f(\cdot)$ is a nonlinear function, such as **sigmoid**

Gradient Descent for Regression

- We can think a 2-class classification as a regression problem with $d_{(i)} \in \{0,1\}$, 0 for one class and 1 for another class
- What if we have multiple classes? A typical method is to use multiple 2-class classifiers (recall one-vs-all)
- Alternatively, we may use softmax (covered later)

Gradient Descent for Regression

- We may define the cost function as

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{k=0}^{n} \varepsilon_{(k)}^2 = \frac{1}{n}\sum_{k=1}^{n} (\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{(\boldsymbol{k})} - d_{(k)})^2$$

- Want to minimize the cost function by using gradient descent

Batch Gradient Descent

- It is easy to know

$$\nabla J(\boldsymbol{w}) = \frac{2}{n}\sum_{k=0}^{n}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{(\boldsymbol{k})} - d_{(k)})\boldsymbol{x}_{(\boldsymbol{k})} = \frac{2}{n}\sum_{k=0}^{n}\varepsilon_{(k)}\boldsymbol{x}_{(\boldsymbol{k})}$$

- Note: average errors are used to determine gradient. Thus, it is called batch gradient descent
- **Batch mode** updates weights infrequently (**low efficiency), also called vanilla gradient descent**
- If we have a nonlinear function $f(\cdot)$, we still can compute gradient by using chain rule

Stochastic Gradient Descent

- Another possible method to compute gradient for each instance (observation) $\boldsymbol{x}_{(k)}$ is

$$\nabla J(\boldsymbol{w}) = \varepsilon_{(k)}\boldsymbol{x}_{(k)}$$

- In **stochastic gradient descent**, weights updates for every instance
- In a typical situation, we nee to present the dataset to gradient descent many times
- Algorithm "learn" all data samples in training set one time is called one **epoch**

Stochastic Gradient Descent

- In stochastic gradient descent, step size must keep small. A large step size may **fail to converge**
- The adaptive signal processing version of stochastic gradient descent is called LMS (least mean squares) algorithm

Mini-batch Gradient Descent

- Recall
  - Batch gradient descent uses average error over entire dataset for one iteration (one gradient updating)
  - Stochastic gradient descent uses error on one instance for one iteration to update gradient
- We can do somewhere in between: Use average error of, say, 128 instances for one weight update (called mini-batch gradient descent)

Regularization

- L1 norm (Lasso)
- L2 norm
- Dropout (used in neural networks)

Regularization

- We can avoid overfitting by using regularization
- We can do it in gradient descent with a modified cost function (batch gradient descent because of summing over training samples)

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{k=1}^{n} \varepsilon_{(k)}^2 + \lambda g(\boldsymbol{w})$$

In the case of L-2 regularization, we have

$$g(\boldsymbol{w}) = \sum_{j=1}^{p} w_j^2$$

Regularization

- We may also apply L-2 regularization to stochastic gradient descent (with slight modification)
- Note: If "bias" term, i.e., w_0 in $(\boldsymbol{w}^T\mathbf{x} + w_0)$, is used, we will not penalize w_0
- Other than L-2 regularization, we may also try L-1 regularization (Lasso)
- Additional ref: https://arxiv.org/pdf/1609.04747.pdf

L2 Regularization

- Let $f(\boldsymbol{w}) = \frac{1}{n}\sum_{k=1}^{n} \varepsilon_{(k)}^2$, then

  $\nabla J(\boldsymbol{w}) = \nabla f(\boldsymbol{w}) + \nabla \lambda g(\boldsymbol{w})$

  where $\nabla \lambda g(\boldsymbol{w}) = \frac{1}{2}\lambda \boldsymbol{w}$

- Therefore,

$$\mathbf{w}(k+1) \leftarrow \mathbf{w}(k) - \eta \nabla f\big(\mathbf{w}(k)\big) - \frac{1}{2}\eta\lambda \boldsymbol{w}(k)$$

L2 Regularization

- Equivalently,

$$\mathbf{w}(k+1) \leftarrow (1 - \frac{1}{2}\eta\lambda)\mathbf{w}(k) - \eta\nabla f\big(\mathbf{w}(k)\big)$$

- This term is less than one, so weights tends to go to zero if gradient is very small
- Exercise: repeat the above procedure to find L1 effect