Optimization for ML

CS771: Introduction to Machine Learning

Plan today

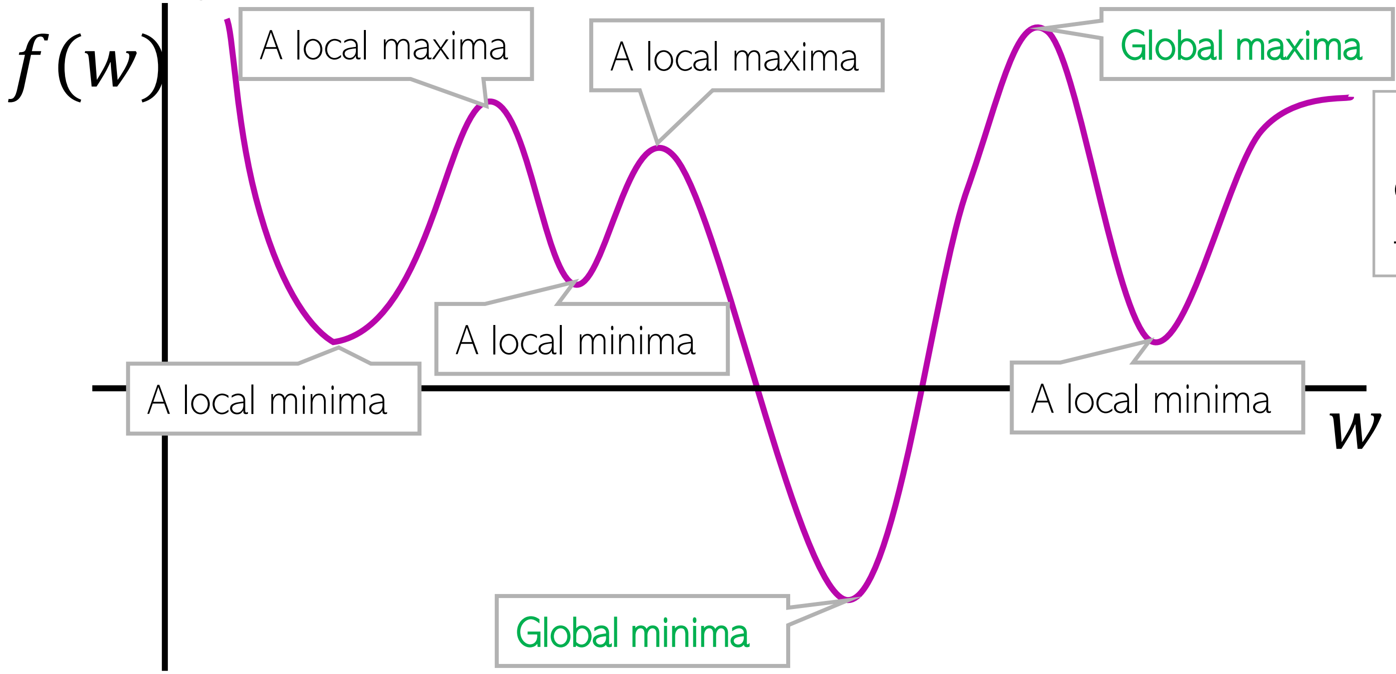
- Basics of optimization of functions
  - Global and local optima
  - First derivative (gradient) and second derivative (Hessian)
- Convex and non-convex functions
- Popular optimization methods
  - Gradient descent
  - Stochastic gradient descent

Functions and their optima

The objective function of the ML problem we are solving (e.g., squared loss for regression)

Assume unconstrained for now, i.e., just a real-valued number/vector

- Many ML problems require us to optimize a function f of some variable(s) w
- For simplicity, assume f is a scalar-valued function of a scalar w ($f: \mathbb{R} \rightarrow \mathbb{R}$)



Usually interested in global optima but often want to find local optima, too

For deep learning models, often the local optima are what we can find (and they usually suffice) – more later

Will see what these are later

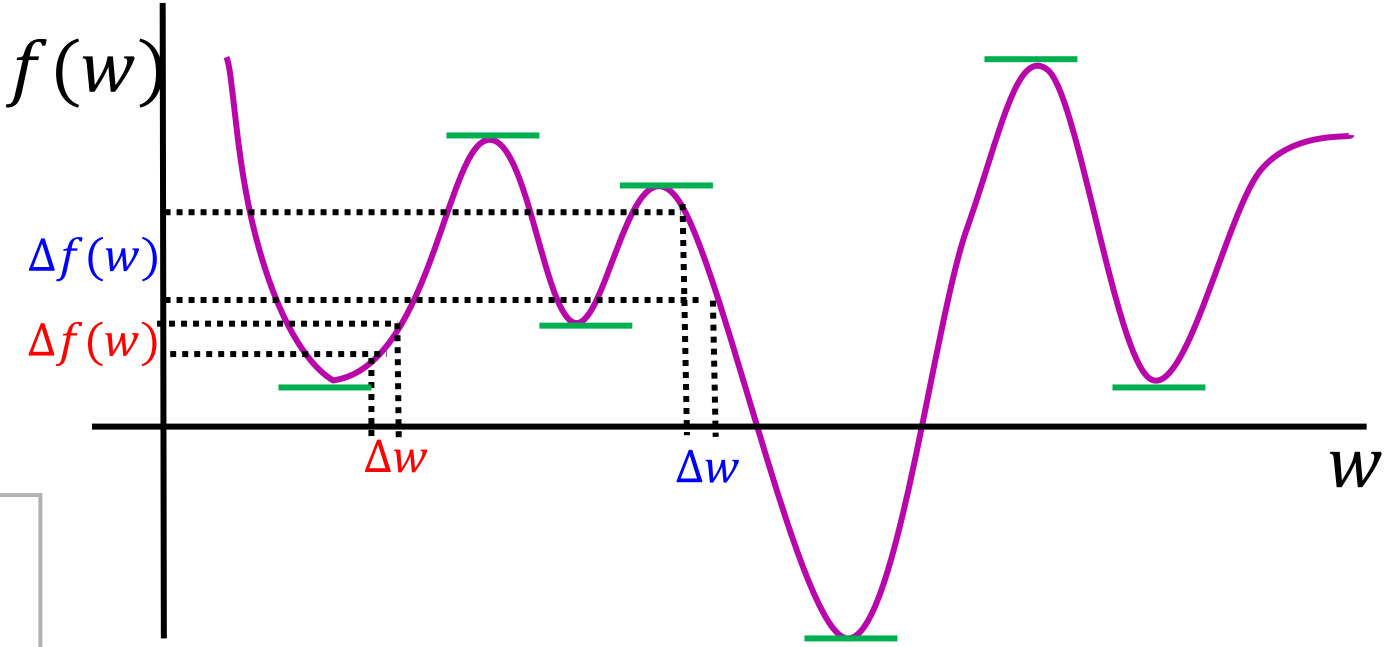
- Any function has one/more optima (maxima, minima), and maybe saddle points
- Finding the optima or saddles requires derivatives/gradients of the function

Derivatives

Will sometimes use $f'(w)$ to denote the derivative

- Magnitude of derivative at a point is the rate of change of the func at that point

$$\frac{df(w)}{dw} = \lim_{\Delta w \to 0} \frac{\Delta f(w)}{\Delta w}$$



Sign is also important: Positive derivative means f is increasing at w if we increase the value of w by a very small amount; negative derivative means it is decreasing

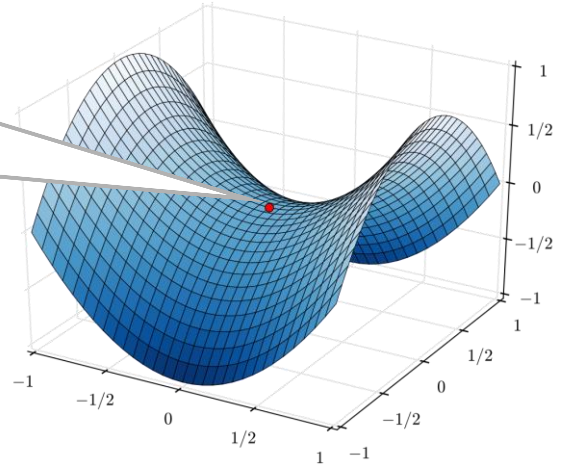
Understanding how f changes its value as we change w is helpful to understand optimization (minimization/maximization) algorithms

- Derivative becomes zero at stationary points (optima or saddle points)
  - The function becomes "flat" ($\Delta f(w) = 0$ if we change w by a very little at such points)
  - These are the points where the function has its maxima/minima (unless they are saddles)

Saddle Points

- Points where derivative is zero but are neither minima nor maxima

Function might increase along one direction but decrease in a different direction



- Saddle points are very common for loss functions of deep learning models
  - Need to be handled carefully during optimization

- Second or higher derivative may help identify if a stationary point is a saddle

Rules of Derivatives

Some basic rules of taking derivatives

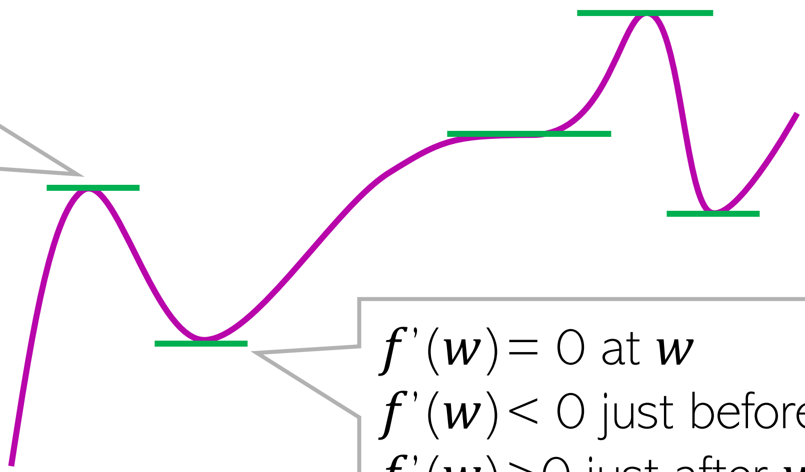
- Sum Rule: $\left(f(w)+g(w)\right)' = f'(w)+g'(w)$
- Scaling Rule: $\left(a \cdot f(w)\right)' = a \cdot f'(w)$ if a is not a function of w
- Product Rule: $\left(f(w) \cdot g(w)\right)' = f'(w) \cdot g(w) + g'(w) \cdot f(w)$
- Quotient Rule: $\left(f(w)/g(w)\right)' = \left(f'(w) \cdot g(w) - g'(w)f(w)\right)/\left(g(w)\right)^2$
- Chain Rule: $\left(f\left(g(w)\right)\right)' \stackrel{\text{def}}{=} (f \circ g)'(w) = f'\left(g(w)\right) \cdot g'(w)$

We already used some of these (sum, scaling and chain) when calculating the derivative for the linear regression model

Derivatives

- How the derivative itself changes tells us about the function's optima

$f'(w) = 0$ at w,
$f'(w) > 0$ just before w
$f'(w) < 0$ just after w
w is a maxima



$f'(w) = 0$ at w
$f'(w) < 0$ just before w
$f'(w) > 0$ just after w
w is a minima

$f'(w) = 0$ and $f''(w) < 0$
w is a maxima

$f'(w) = 0$ and $f''(w) > 0$
w is a minima

But, the behavior of second derivative alone may not be enough to tell whether it's a saddle point

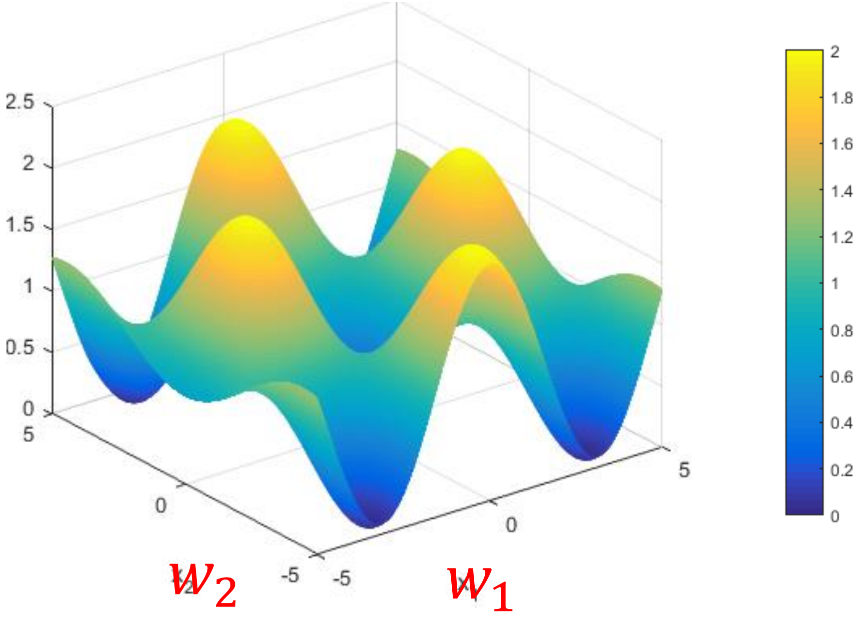
- The second derivative $f''(w)$ can provide this information
  - It is the rate of change of the first derivative

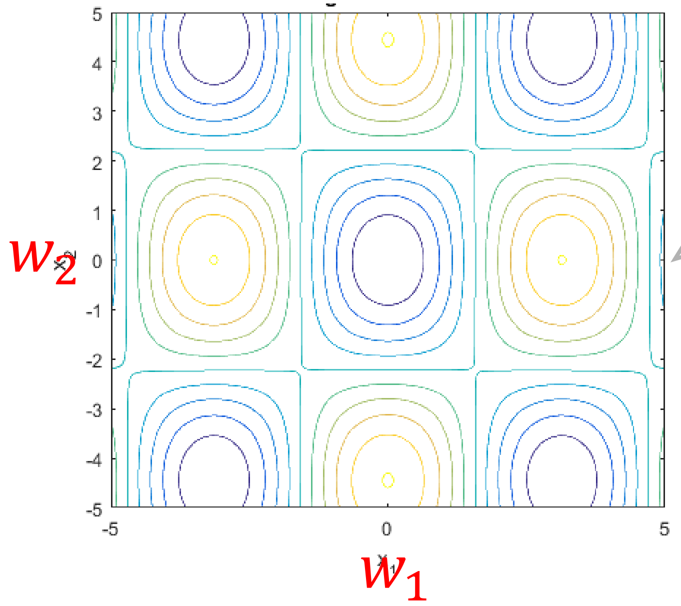
Multivariate Functions

- Most functions that we see in ML are multivariate function
- Example: Loss fn $\boldsymbol{L(w)}$ in lin-reg was a multivar function of $\boldsymbol{D}$-dim vector $\boldsymbol{w}$

$$L(\boldsymbol{w}): \mathbb{R}^D \to \mathbb{R}$$

- Here is an illustration of a function of 2 variables (4 maxima and 5 minima)





Two-dim contour plot of the function (i.e., what it looks like from the above)

Each curve here is the set of points at which the function has the same value

Plot courtesy: http://benchmarkfcns.xyz/benchmarkfcns/griewankfcn.html

Derivatives of Multivariate Functions

- Can define derivative for a multivariate functions as well via the gradient
- Gradient of a function $\boldsymbol{f(w)}: \mathbb{R}^D \rightarrow \mathbb{R}$ is a $\boldsymbol{D \times 1}$ vector of partial derivatives

$$\nabla f(\boldsymbol{w}) = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_D}\right)^\top$$

Each element in this gradient vector tells us how much f will change if we move a little along the corresponding (akin to one-dim case)

- Conditions for optima defined similar to one-dim case
  - Required properties for one-dim case must be satisfied for all components of $\boldsymbol{w}$
- If the function itself is vector valued, e.g., $\boldsymbol{f(w)}: \mathbb{R}^D \rightarrow \mathbb{R}^K$ then we will have $\boldsymbol{K}$ such gradient vectors, one for each output dimension of $\boldsymbol{f}$
- The second derivative in the multivariate case is known as the Hessian matrix

The Hessian

- For a multivar scalar valued function $f(\boldsymbol{w}): \mathbb{R}^D \rightarrow \mathbb{R}$, Hessian is a $D \times D$ matrix

$$\nabla^2 f(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 w_D} \\ \frac{\partial^2 f}{\partial w_2 w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 w_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_D w_1} & \frac{\partial^2 f}{\partial w_D w_2} & \cdots & \frac{\partial^2 f}{\partial w_D^2} \end{bmatrix}$$

Gives information about the curvature of the function at point $\boldsymbol{w}$

Note: If the function itself is vector valued, e.g., $f(\boldsymbol{w}): \mathbb{R}^D \rightarrow \mathbb{R}^K$ then we will have K such $D \times D$ Hessian matrices, one for each output dimension of f

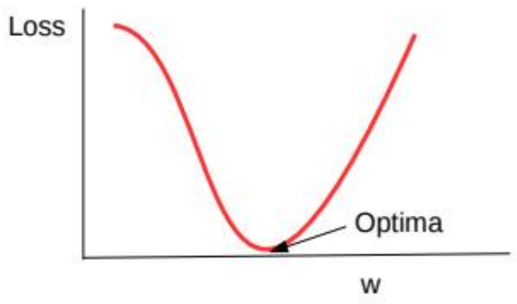
A square, symmetric $D \times D$ matrix $\boldsymbol{M}$ is PSD if $\boldsymbol{w}^\top \boldsymbol{M} \boldsymbol{w} \geq \boldsymbol{0} \ \forall \boldsymbol{w} \in \mathbb{R}^D$
Will be NSD if $\boldsymbol{w}^\top \boldsymbol{M} \boldsymbol{w} \leq \boldsymbol{0} \ \forall \boldsymbol{w} \in \mathbb{R}^D$

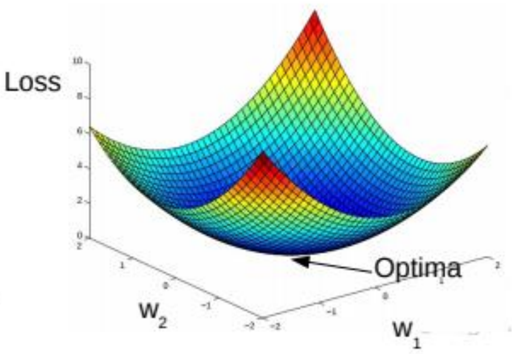
PSD if all eigenvalues are non-negative

- The Hessian matrix can be used to assess the optima/saddle points
  - $\nabla f(\boldsymbol{w}) = 0$ and $\nabla^2 f(\boldsymbol{w})$ is a positive semi-definite (PSD) matrix then $\boldsymbol{w}$ is a minima
  - $\nabla f(\boldsymbol{w}) = 0$, and $\nabla^2 f(\boldsymbol{w})$ is a negative semi-definite (NSD) matrix then $\boldsymbol{w}$ is a maxima
  - The determinant of the Hessian is zero at saddle point (has at least one zero eig-val)

Convex and Non-Convex Functions

- A function being optimized can be either convex or non-convex
- Here are a couple of examples of convex functions

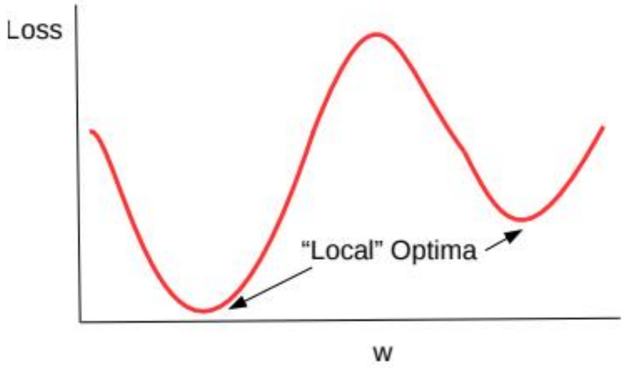


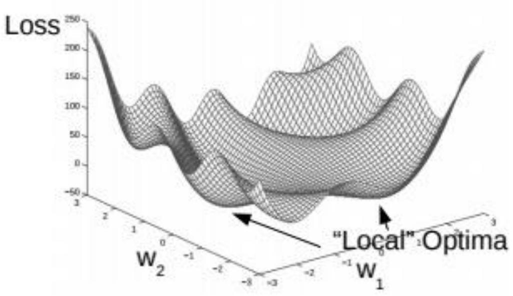


Convex functions are bowl-shaped. They have a unique optima (minima)

Negative of a convex function is called a concave function, which also has a unique optima (maxima)

- Here are a couple of examples of non-convex functions





Non-convex functions have multiple minima. Usually harder to optimize as compared to convex functions

Loss functions of most deep learning models are non-convex

Convex Sets

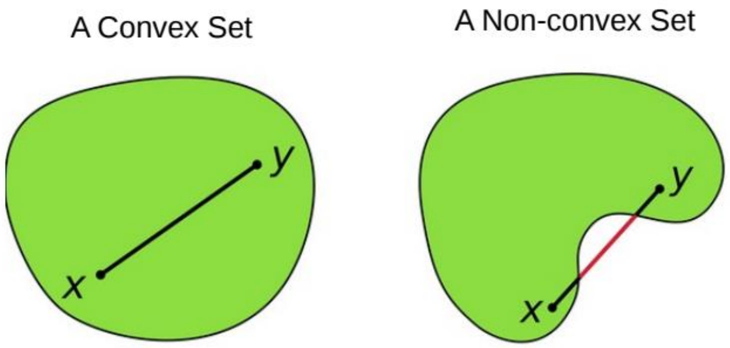
- A set $\boldsymbol{S}$ of points is a convex set, if for any two $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{S}$, and $0 \leq \lambda \leq 1$

z is also called a "convex combination" of two points

$$z = \lambda x + (1 - \lambda) y \in S$$

Can also define convex combination of N points $x_1, x_2, \ldots, x_N$ using non-neg $\lambda_1, \lambda_2, \ldots, \lambda_N$ s.t., $\sum_{i=1}^{N} \lambda_i = 1$ as $z = \sum_{i=1}^{N} \lambda_i x_i$

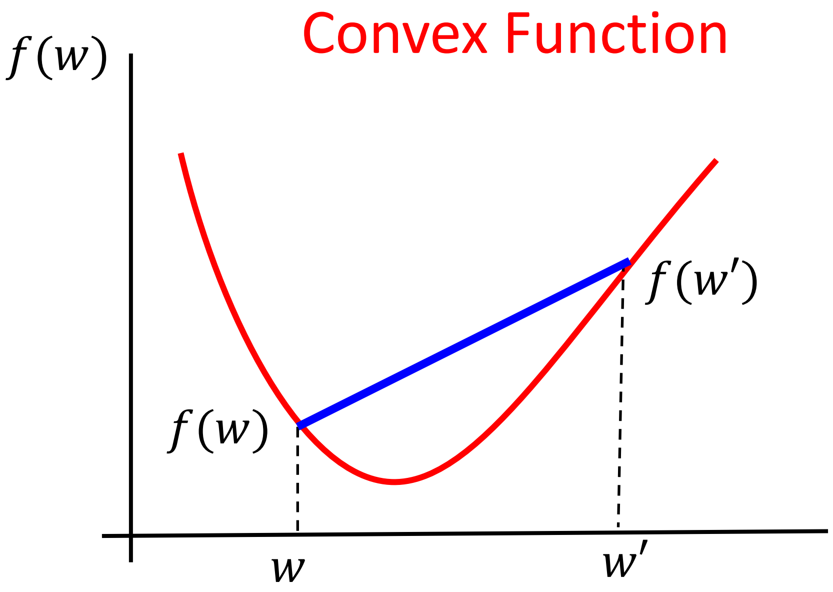
- Above means that all points on the line-segment between $\boldsymbol{x}$ and $\boldsymbol{y}$ lie within $\boldsymbol{S}$

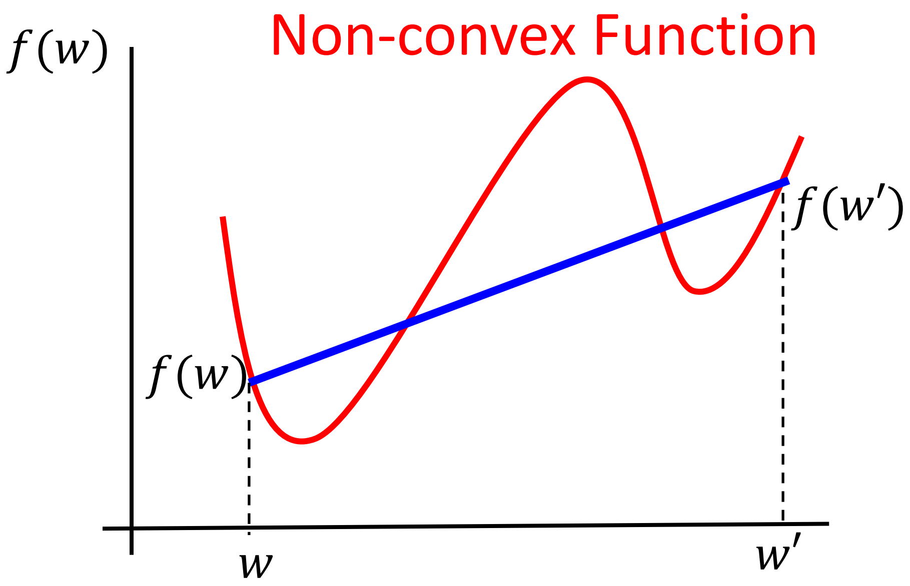


- The domain of a convex function needs to be a convex set

Checking if a function is convex

- Informally, $f(w)$ is convex if ALL of its chords lie above the function everywhere





- If $f(w)$ is convex then for any $\lambda \in (0,1)$ and all pairs of points w and w'

"Chord lies above" condition, mathematically

$$f(\lambda w + (1-\lambda)w') \leq \lambda f(w) + (1-\lambda)f(w')$$

Even more general, holds for convex combination of any n points

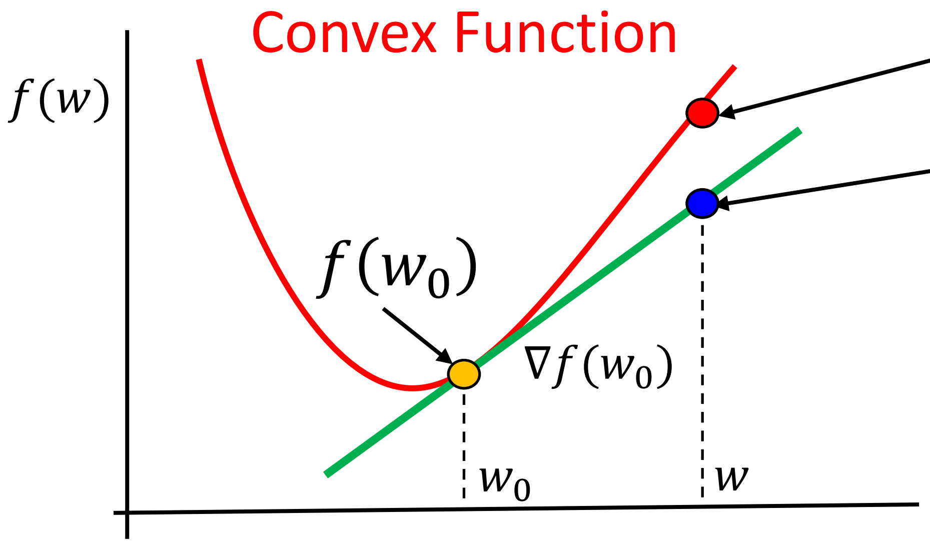
$$f\left(\sum_{i=1}^{n} \lambda_i w_i\right) \leq \sum_{i=1}^{n} \lambda_i f(w_i)$$

Jensen's Inequality

Checking for Convexity using Derivatives

- First-order convexity (graph of f must be above all the tangents)

$$f(w) \geq f(w_0) + \nabla f(w_0)^\top (w - w_0)$$



Exercise: Use second derivative test to show that the ridge regression objective is (strictly) convex

- Second derivative a.k.a. Hessian matrix (if exists) must be positive semi-definite

$$\nabla^2 f(\boldsymbol{w}) \succcurlyeq \mathbf{0}$$

- Strictly convex: Above inequalities are strict (i.e., not $\geq$ but $>$)

Some Basic Rules for Convex Functions

- All linear or affine functions $(aw + b)$ are convex in w for any $a, b \in \mathbb{R}$
- $\exp(aw)$ is convex in $w \in \mathbb{R}$ for any $a \in \mathbb{R}$
- $\log(w)$ is concave in $w > 0$ $(-\log(w)$ is convex)
- w^a is convex in $w > 0$ for any $a \geq 1$, and concave for $0 \leq a \leq 1$
- $|w|^a$ is convex in $w \in \mathbb{R}$f or any $a \geq 1$
- All norm functions (e.g., ℓ_2-squared norm, ℓ_1 norm) are convex
- Nonnegative weighted sum of convex functions is also convex function
- If $f(w)$ is convex then $f(aw + b)$ is also convex
- Rules regarding compositions of two functions h and g: $f(w) = h(g(w))$

h convex and nondecreasing, g convex: f convex	h convex and nonincreasing, g convex: f concave
h convex and nondecreasing, g concave: f concave	h convex and nonincreasing, g concave: f convex

Optimization Problems in ML

- The general form of an optimization problem in ML will usually be

$$w_{opt} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$$

$L(\boldsymbol{w})$ may denote the training loss, or training loss + regularizer term

or

$$w_{opt} = \arg\min_{w \in \boldsymbol{C}} L(\boldsymbol{w})$$

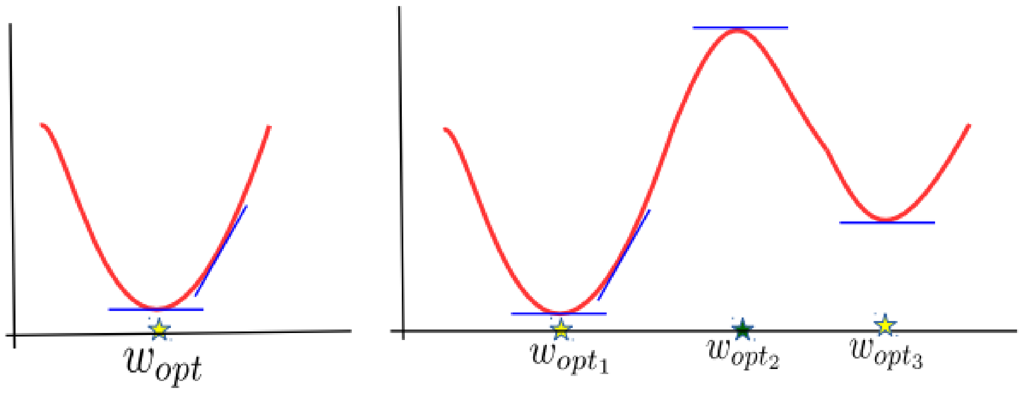
Training loss with a constraint on $\boldsymbol{w}$

- $\boldsymbol{C}$ is the constraint set that the solution must belong to, e.g.,
  - Non-negativity constraint: All entries in $\boldsymbol{w}_{opt}$ must be non-negative
  - Sparsity constraint: $\boldsymbol{w}_{opt}$ is a sparse vector with at most $\boldsymbol{K}$ non-zeros
- Constrained opt. probs can be converted into unconstrained opt. (will see later)
- For now, assume we have an unconstrained optimization problem

Methods for Solving Optimization Problems

Method 1: Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regression



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

E.g., linear/ridge regression, but not for logistic/softmax regression

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution

- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent

Method 2: Iterative Optimiz. via Gradient Descent

Can I used this approach to solve maximization problems?

For max. problems we can use gradient ascent
$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{g}^{(t)}$

Iterative since it requires several steps/iterations to find the optimal solution

Fact: Gradient gives the direction of steepest change in function's value

Will move in the direction of the gradient

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

Gradient Descent

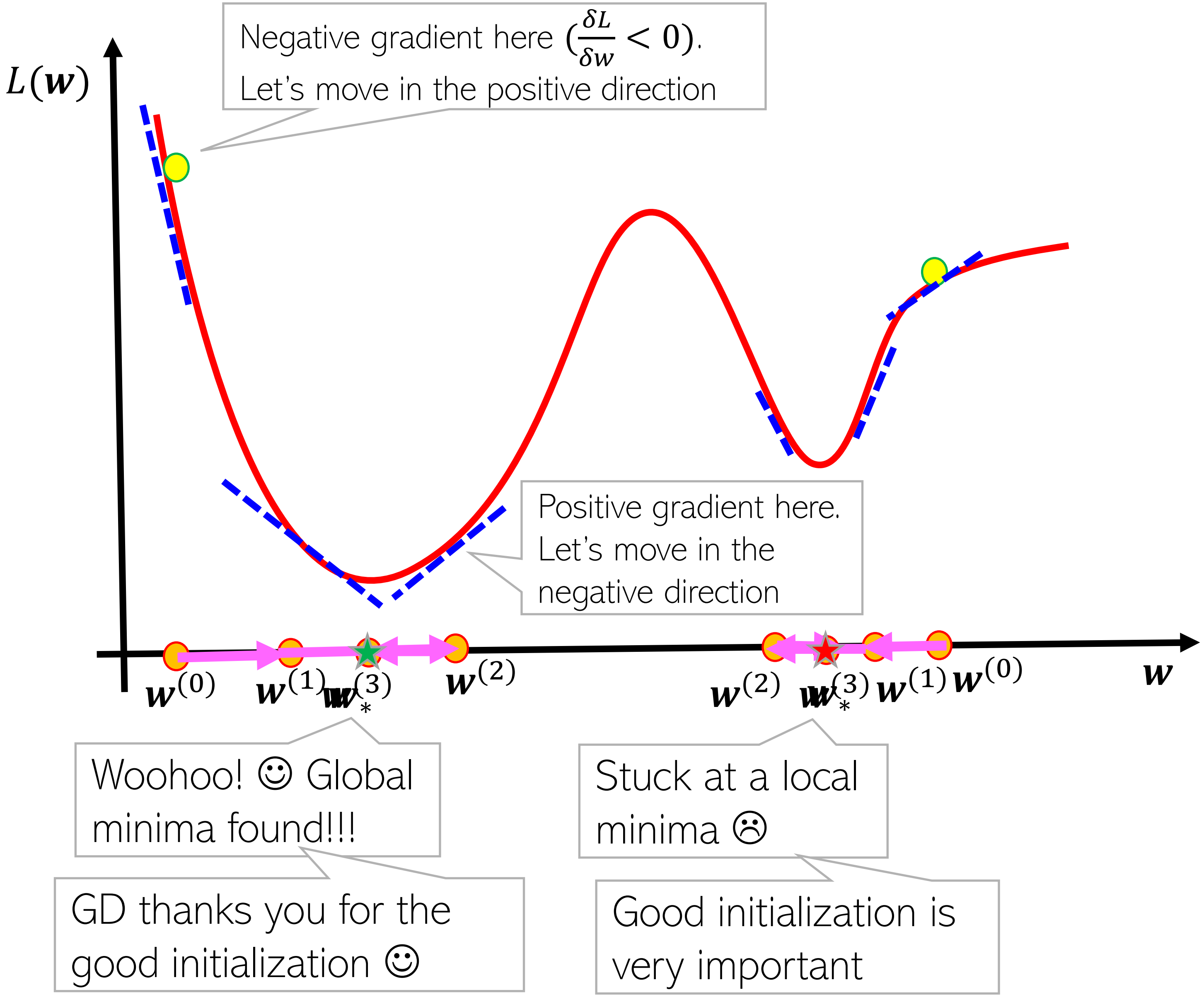
- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \dots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

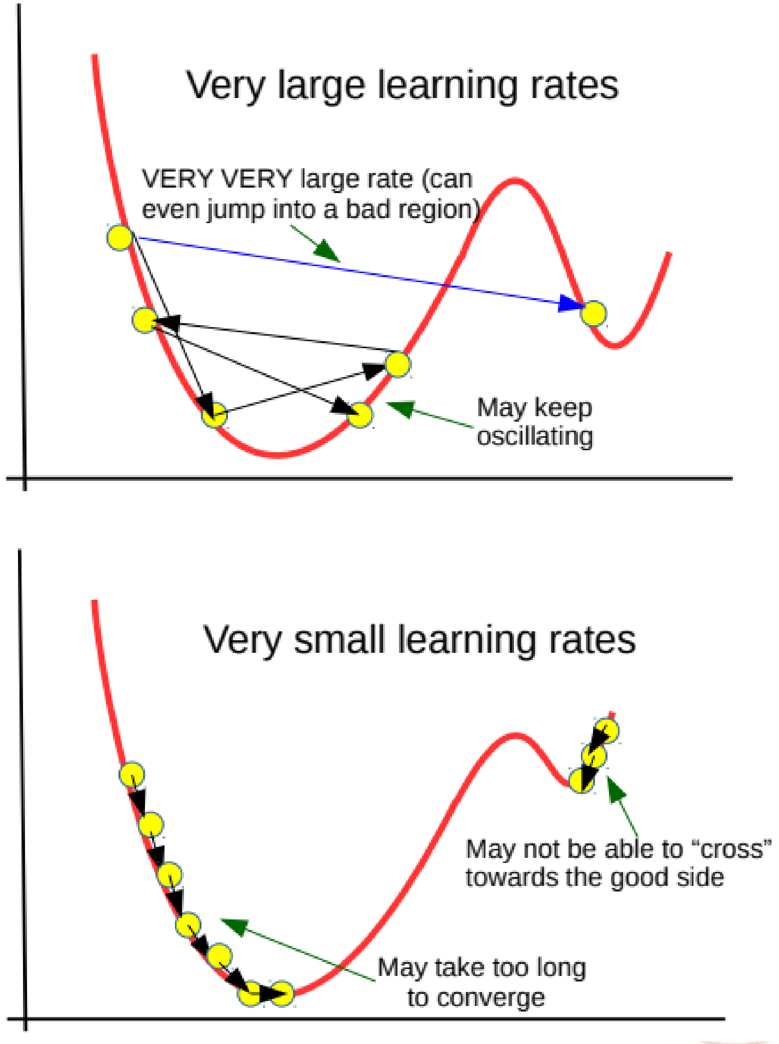
Will see the justification shortly

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

Gradient Descent: An Illustration



Learning rate is very important



Faster GD: Stochastic Gradient Descent (SGD)

- Consider a loss function of the form $\boldsymbol{L(w)} = \frac{1}{N}\sum_{n=1}^{N} \ell_n(\boldsymbol{w})$

Writing as an average instead of sum. Won't affect minimization of $L(\boldsymbol{w})$

- The gradient in this case can be written as

Expensive to compute – requires doing it for all the training examples in each iteration ☹

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} L(w) = \nabla_{\boldsymbol{w}}[\frac{1}{N}\sum_{n=1}^{N} \ell_n(\boldsymbol{w})] = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{g}_n$$

Gradient of the loss on n^{th} training example

- Stochastic Gradient Descent (SGD) approximates $\boldsymbol{g}$ using a <u>single</u> training example

- At iter. $\boldsymbol{t}$, pick an index $\boldsymbol{i} \in \{1,2,\ldots,N\}$ uniformly randomly and approximate $\boldsymbol{g}$ as

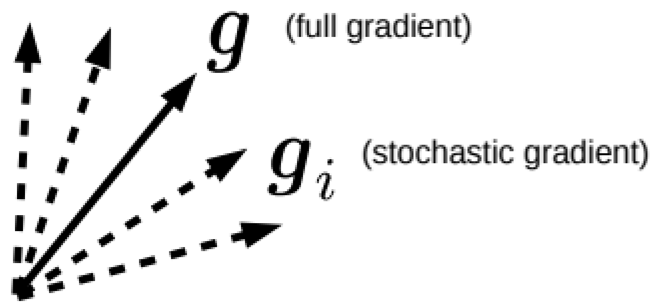
$$\boldsymbol{g} \approx \boldsymbol{g}_i = \nabla_{\boldsymbol{w}} \ell_i(\boldsymbol{w})$$

Can show that $\boldsymbol{g}_i$ is an unbiased estimate of $\boldsymbol{g}$, i.e., $\mathbb{E}[\boldsymbol{g}_i] = \boldsymbol{g}$

- May take more iterations than GD to converge but each iteration is much faster ☺
  - SGD per iter cost is $O(D)$ whereas GD per iter cost is $O(ND)$

Minibatch SGD

- Gradient approximation using a single training example may be noisy



The approximation may have a high variance – may slow down convergence, updates may be unstable, and may even give sub-optimal solutions (e.g., local minima where GD might have given global minima)

- We can use $B > 1$ unif. rand. chosen train. ex. with indices $\{i_1, i_2, \ldots, i_B\} \in \{1,2,\ldots,N\}$
- Using this "minibatch" of examples, we can compute a minibatch gradient

$$\boldsymbol{g} \approx \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{g}_{i_b}$$

- Averaging helps in reducing the variance in the stochastic gradient
- Time complexity is $O(BD)$ per iteration in this case