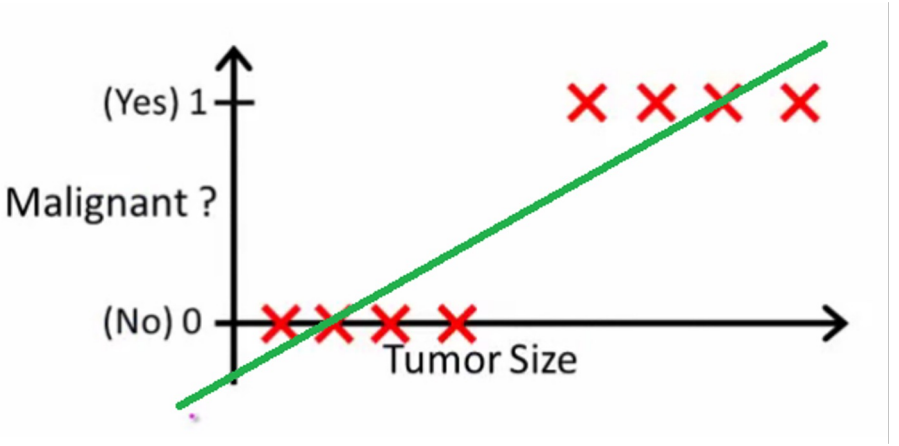
Introduction to Gradient Descent

Dr. Puneet Gupta

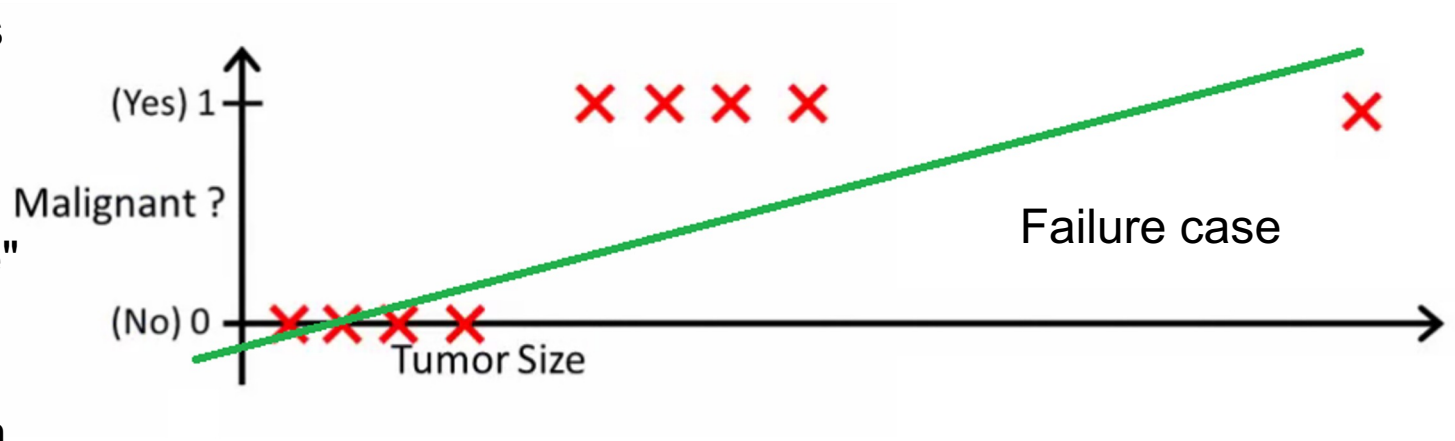
Using Linear Regression for Binary Classification

Example

- Using linear regression, fit a polynomial (line in this case) through the training data of type {tumor size, tumor type}.
- Malignant tumors means 1 and non-malignant means 0.
- Intuition: All tumors larger certain threshold are malignant
- Green line is the model $h(x)$.
- Prediction Rule: If $h(x)$ > 0.5, for any given tumor size x, predict malignant tumor and benign otherwise.



Linear regression gives a raw number but logistic regression tells probability that x belongs to the "positive" class. Hence, it is a regression algorithm but, by setting a rule on the probability, we can perform classification.

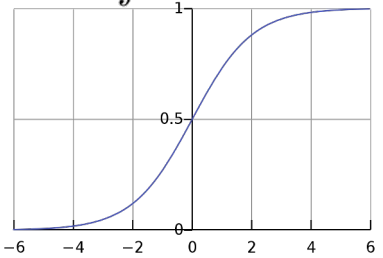


Logistic Regression

Turning scores of input $\mathbf{x}$, $\mathbf{w}^T\mathbf{X}$ *into probabilities using sigmoid function*

$$p\left(y=1|\mathbf{x},\mathbf{w}\right)=\mu=\sigma\left(\mathbf{w}^T\mathbf{x}\right)=\frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}}=\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$p\left(y=0|\mathbf{x},\mathbf{w}\right)=1-\mu=1-\sigma\left(\mathbf{w}^T\mathbf{x}\right)=\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$$



$$\Rightarrow p\left(y|\mathbf{x},\mathbf{w}\right)=\frac{1}{1+e^{-y\mathbf{w}^T\mathbf{x}}} \qquad If\ y\in\{-1,+1\}\ instead\ of\ y\in\{0,+1\}$$

Decision boundary (where classes are equiprobable)

$$p\left(y=1|\mathbf{x},\mathbf{w}\right)=p\left(y=0|\mathbf{x},\mathbf{w}\right)\quad\Rightarrow\quad\mathbf{w}^T\mathbf{x}=0$$

- *The decision boundary of Logistic Regression is a linear hyperplane and rule is* $(y=1)$, *if* $\mathbf{w}^T\mathbf{x}\geq 0$; *otherwise* 0.
- *High positive or negative scores of* $\mathbf{w}^T\mathbf{x}$ *mean high or low probabilities or confidence of* $(y=1)$

Basics: Derivatives

Magnitude of derivative at a point is the rate of change of the function at that point

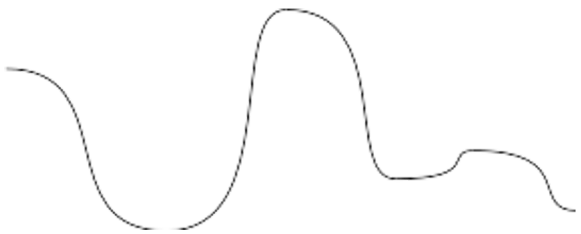
$$f : \mathbb{R} \to \mathbb{R} \qquad f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{\Delta f(x)}{\Delta x}$$

Positive derivative means f is **increasing** at *x* if we increase the value of *x* by a very small amount; negative derivative means it is **decreasing**

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

Derivative becomes zero at stationary points (optima or saddle points)

- The function becomes “flat” (f’(x)=0 if we change x by a very little at such points
- These are the points where the function has its maxima/minima (unless they are saddles). At saddle points, derivative is zero but neither minima nor maxima. These are common in deep learning models.



$$f'(x) = 0; f''(x) < 0 \Rightarrow x \; is \; maxima$$
$$f'(x) = 0; f''(x) > 0 \Rightarrow x \; is \; minima$$
$$f'(x) = 0; f''(x) = 0 \Rightarrow x \; may \; be \; saddle$$

Basics: Derivatives

$$\text{If } f : \mathbb{R}^D \rightarrow \mathbb{R}$$
$$\text{then, } \nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots \frac{\partial f}{\partial x_D} \right)$$

Each element in this gradient vector tells us how much f will change if we move a little along the corresponding direction.

Optima and saddle points are defined similar to one-dim case. It require the properties that we saw for one-dim case must be satisfied along all the directions.

The second derivative in this case is known as the Hessian

$$\text{If } f : \mathbb{R}^D \rightarrow \mathbb{R} \text{ then, } \nabla^2 f(x) \; is \; D \times D \; matrix.$$
$$\text{If } f : \mathbb{R}^D \rightarrow \mathbb{R}^k \text{ then, } \nabla^2 f(x) \; is \; k \; matrices, of \; size \; D \times D \; matrix.$$

$$\nabla f(x) = 0; \nabla^2 f(x) \; is \; p.s.d \Rightarrow x \; is \; minima$$
$$\nabla f(x) = 0; \nabla^2 f(x) \; is \; n.s.d \Rightarrow x \; is \; maxima$$

Analyzing optimal solutions for loss functions

Usually, optimization problems in ML looks like :

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\ \mathcal{L}(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} l\left(y_n, \mathbf{w}^T \mathbf{x_n}\right) + \lambda R(\mathbf{w})$$

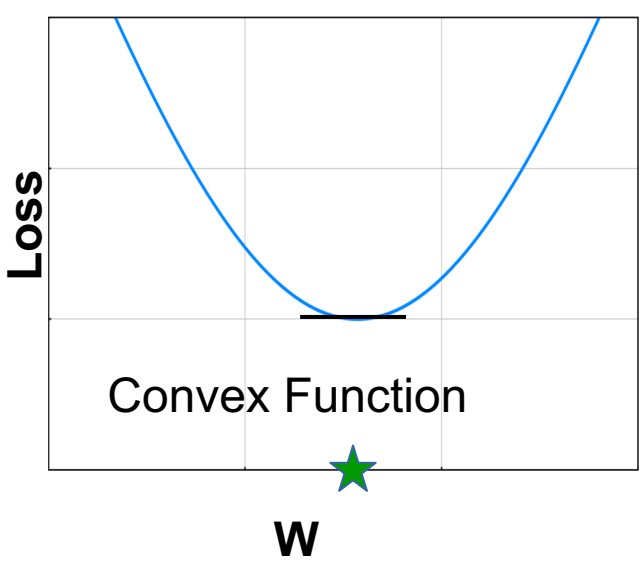
where, $l\left(y_n, \mathbf{w}^T \mathbf{x_n}\right)$ *is the loss function for* n^{th} *training sample and* $R(\mathbf{w})$ *is the optional regularizer on parameters.*

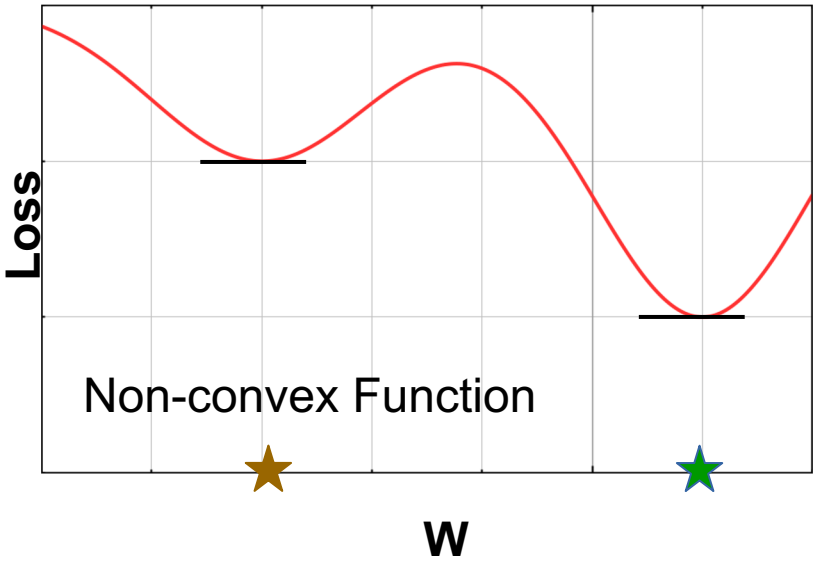
For example : Ridge Regression

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\ \mathcal{L}_{reg}(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} \left(y_n - \mathbf{w}^T \mathbf{x_n}\right)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right]$$

Finding the optima (minima) of loss function by visualizing the loss function as a function of weights in terms of curves or surfaces.

In convex functions, local minima and global minima are same but in non-convex minima, they differ.





★ Global Optima

★ Local Optima

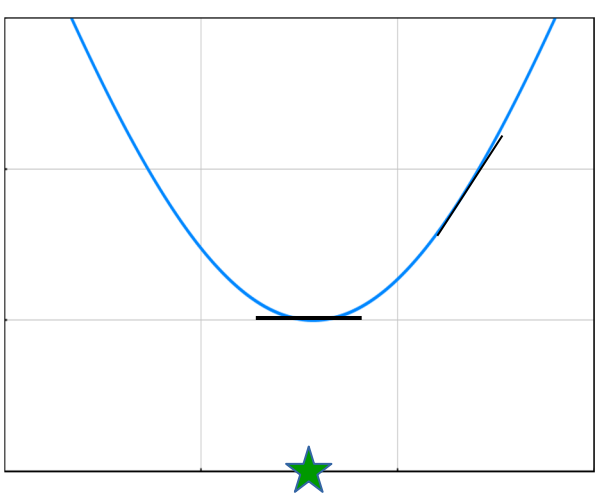
First-order optimality condition

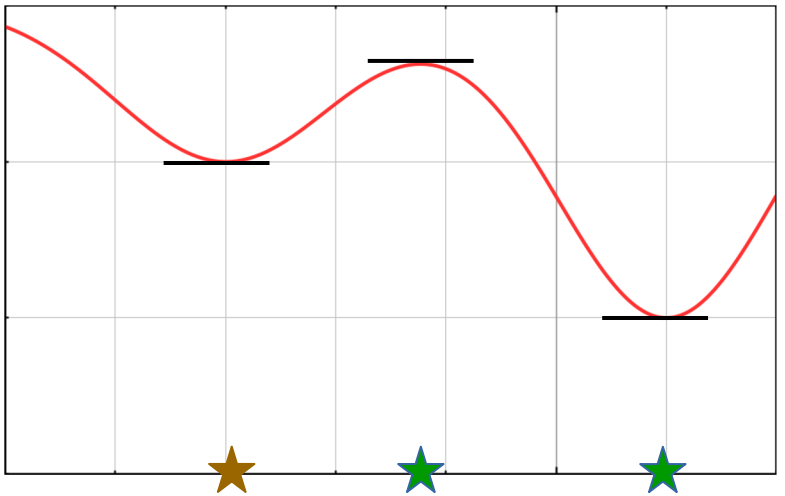
Usually, ML problems are non-convex in nature and they can be solved by non-convex optimization, which is a research area and outside the scope of our course.

Approach, we have used: The gradient g must be zero at each optima (local or global). Also known as first-order optimality condition. That is, set g=0 and evaluate unknown parameters (like w for hyperplane based learning) to find optima.

It may or may not provide closed form solution as in linear regression or logistic regression respectively. Even if it does not provide close form solution, the gradient g can still be helpful by utilizing it in iterative optimization methods.

$$g = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial \left[\sum_{n=1}^{N} l\left(y_n, \mathbf{w}^T \mathbf{x_n}\right) + \lambda R(\mathbf{w}) \right]}{\partial \mathbf{w}} = 0$$





★ Optimal Solution

★ Wrong Solution

Iterative Optimization using gradients: Gradient Descent

Gradient Descent :

1) Initialize $\mathbf{w}$ as $\mathbf{w}^0$

2) Update $\mathbf{w}$ using :

$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \mathbf{g}^t$

3)Repeat until convergence

- η_t known as learning rate, can be constant or vary at each time step.
- The effective step size (how much w moves) depends on both η_t and current gradient g^t

When to stop: Many criteria, e.g., gradients become negligible, or validation error starts increasing.

What happen for convex function?

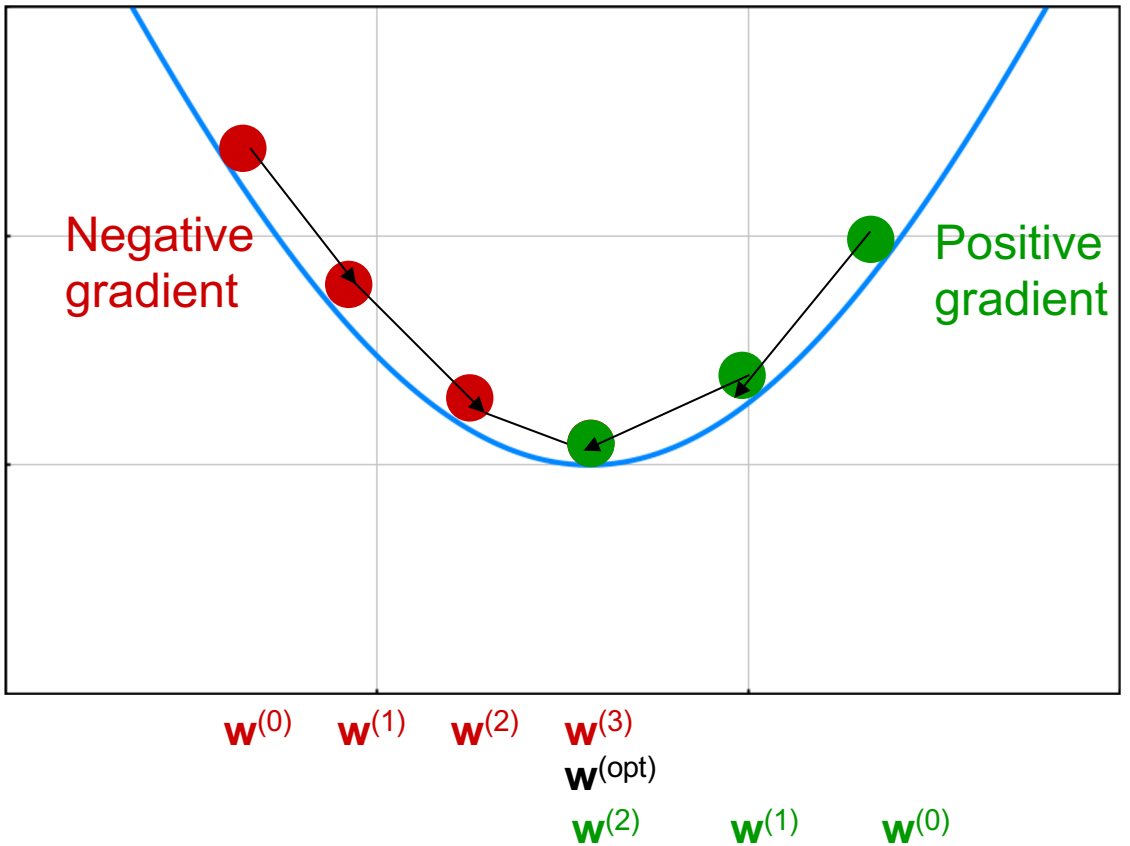
- Guaranteed to provide to local optima (which is global optima for convex functions).

First-order method (utilizing only the gradient **g** of the objective)

Basic idea: Start at some location $\mathbf{w}^{(0)}$ and move in the opposite direction of the gradient.

By how much?

Till when?

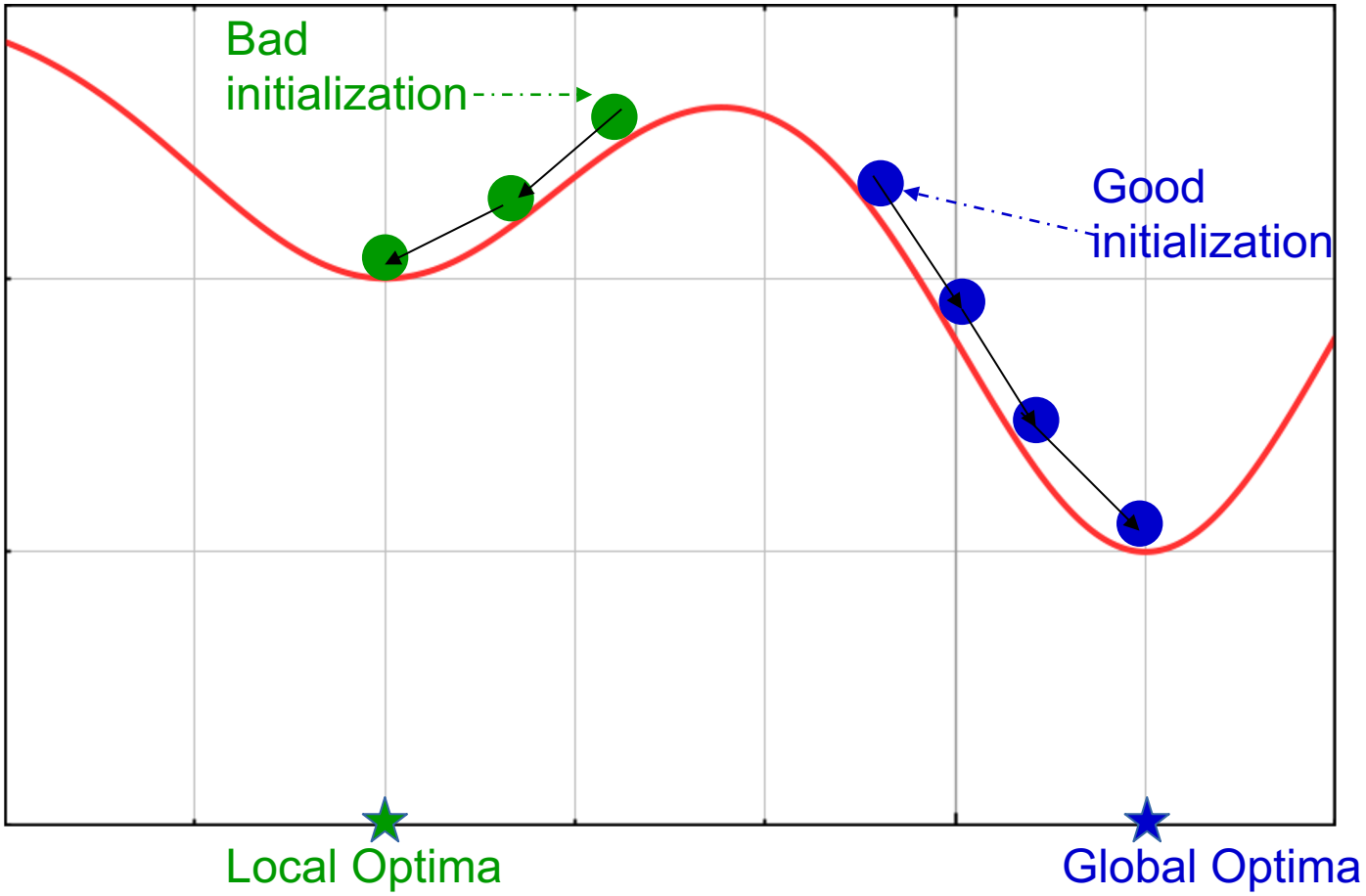


Importance of Weight Initializations

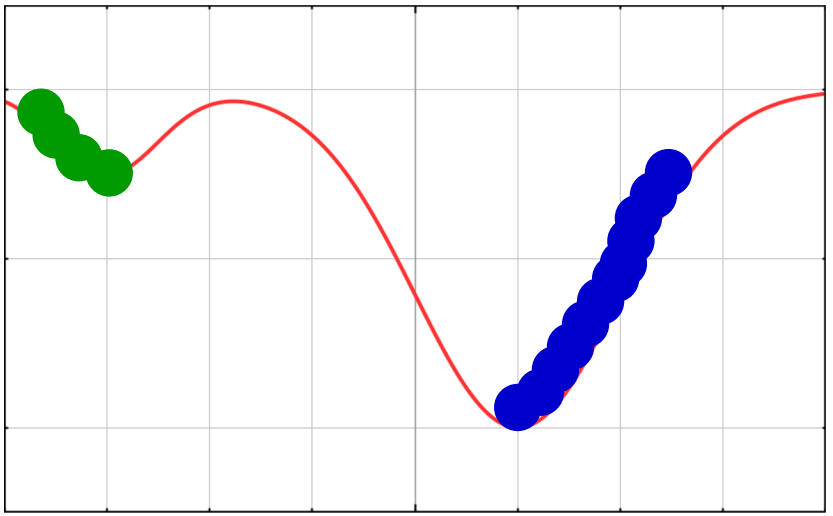
A good initialization $\mathbf{w}^{(0)}$ plays a crucial role. We may get trapped in a bad local optima for non-convex function.

Remedy

Run multiple times with different initialization and select the best one.

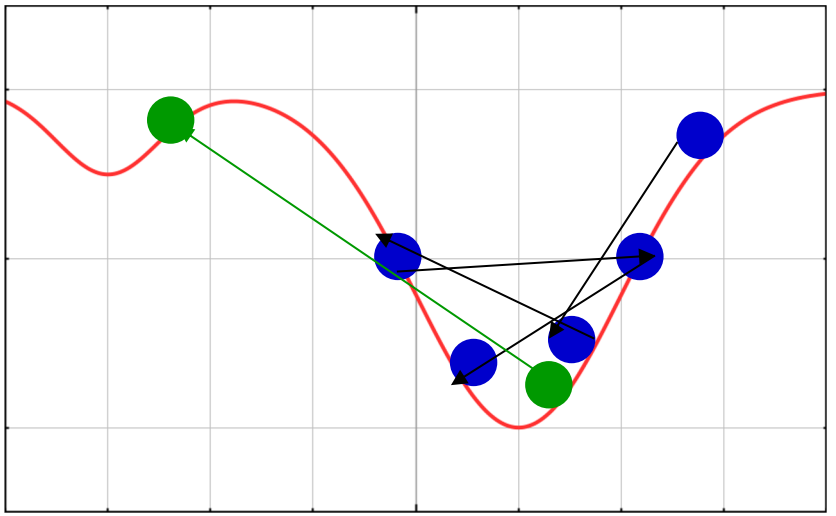


Importance of Learning Rates



Problems with small learning rates

- May take too long to converge
- May not be able to “cross” bad optima and reach towards good optima



Problems with Large learning rates

- May Keep Oscillating
- Jump from good region to bad region

The learning rate can defined as:

- Constant (Require proper tuning for good convergence and proper optima estimation)
- Adaptively decreasing as time step increases (like, divide η by a constant factor)
- Use adaptive learning rates (e.g., using methods such as Adagrad, Adam) (Revisit later).

Stochastic Gradient Descent

Computing gradient in GD is higly time expensive when N is very large.

$$\mathbf{g} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial \left[l\left(y_n, \mathbf{w}^T \mathbf{x_n} \right) \right]}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial l_n}{\partial \mathbf{w}} = \sum_{n=1}^{N} \mathbf{g}_n \quad [Avoiding\ regularization]$$

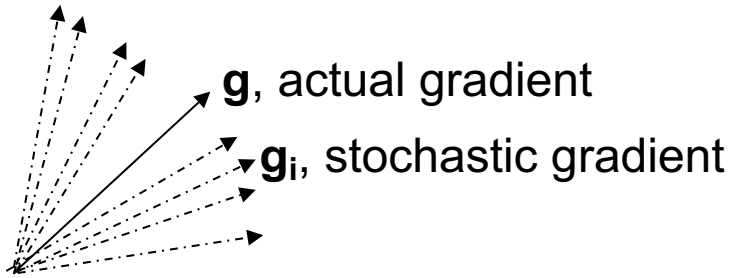
It can be handled by using stochastic gradient descent (SGD)
It picks a random $i \in \{1, 2, ..., N\}$ *and set* $\mathbf{g} \approx \mathbf{g}_i = \frac{\partial l_i}{\partial \mathbf{w}}$.

SGD updates use gradient at one sample for approximating the actual gradient

Stochastic Gradient Descent :
1) Initialize $\mathbf{w}$ as $\mathbf{w}^0$
2) Pick a random $i \in \{1, 2, ..., N\}$. Update $\mathbf{w}$ using :
$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \mathbf{g}_i^t$
3)Repeat until convergence

Mini-batch SGD

SGD uses a single example to approximate the gradient. It is a reasonable estimate of g but will have large variance.



One way to control the variance in the gradient's approximation is ***mini-batch SGD*** where mini-batch containing more than one sample is used for approximating the gradients. Actual gradient is approximated in mini-batch SGD using:

$$\mathbf{g} \approx \frac{1}{B} \sum_{i=1}^{B} \mathbf{g}_i$$

where B is the batch size.

Gradient Descent: Observations

Solving linear regression in closed form :

$$\mathbf{w} = \left(\sum_{n=1}^{N} \mathbf{x_n}\mathbf{x_n}^T\right)^{-1} \sum_{n=1}^{N} y_n \mathbf{x_n} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

requires $O(ND^2+D^3)$ *operations, where* D *is the number of attributes or dimensions.*

- *GD for linear regression avoids matrix inversion.*
- *GD, SGD and mini−batch SGD require* $O(ND)$, $O(D)$ *and* $O(BD)$ *operations respectively.*

GD updates for linear regression are : $\mathbf{w}^{t+1} = \mathbf{w}^t + 2\eta_t \sum_{n=1}^{N} \left(y_n - \mathbf{w}^{t^T}\mathbf{x_n}\right)\mathbf{x_n}$

The updates rectify w *by moving it in the right direction.*

Assume $N = 1$, *then :*

- *If* $\mathbf{w}^{\mathbf{t}^\mathbf{T}}\mathbf{x_n} < y_n$, *then after update* $\mathbf{w}^{(\mathbf{t+1})^\mathbf{T}}\mathbf{x_n} > \mathbf{w}^{\mathbf{t}^\mathbf{T}}\mathbf{x_n}$
- *If* $\mathbf{w}^{\mathbf{t}^\mathbf{T}}\mathbf{x_n} > y_n$, *then after update* $\mathbf{w}^{(\mathbf{t+1})^\mathbf{T}}\mathbf{x_n} < \mathbf{w}^{\mathbf{t}^\mathbf{T}}\mathbf{x_n}$

Same principle is used by logistic regression where reasoning is in terms of probabilities.