EECE 5644: Numerical Optimization & Gradient Descent for Linear Regression

Mark Zolotas

E-mail: m.zolotas@northeastern.edu
Webpage: https://coe.northeastern.edu/people/zolotas-mark/

Tentative Course Outline (Wks. 1-2)

Topics	Dates	Assignments	Additional Reading
~~Course Overview~~ ~~Machine Learning Basics~~	~~07/05~~	**Optional Homework 0** released on Canvas on 07/08 but please do NOT submit on Canvas	~~Chpt. 1~~ ~~Murphy 2012~~
Foundations: ~~Linear Algebra~~, ~~Probability~~, Numerical Optimization (Gradient Descent), Regression	07/06-12		~~Stanford LA Review~~ ~~Stanford Prob. Review~~ Chpt. 8 Murphy 2022
Quick Python Tutorial	07/12	**Homework 1** released on Canvas on 07/15 **Due 07/25**	N/A
Linear Classifier Design, Linear Discriminant Analysis and Principal Component Analysis (PCA)	07/13-14		Chpts. 9.2 & 20.1 Murphy 2022
Bayesian Decision Theory: Empirical Risk Min, Max Likelihood (ML), Max a Posteriori	07/14-15		Chpt. 2 Duda & Hart 2001 Deniz Erdogmus Notes

Probability Recap

- Independent implies zero covariance and uncorrelated:

$$X \perp\!\!\!\perp Y \implies \text{Cov}[X, Y] = 0 \iff \text{Corr}[X, Y] = 0$$

- Uncorrelated does NOT imply independent

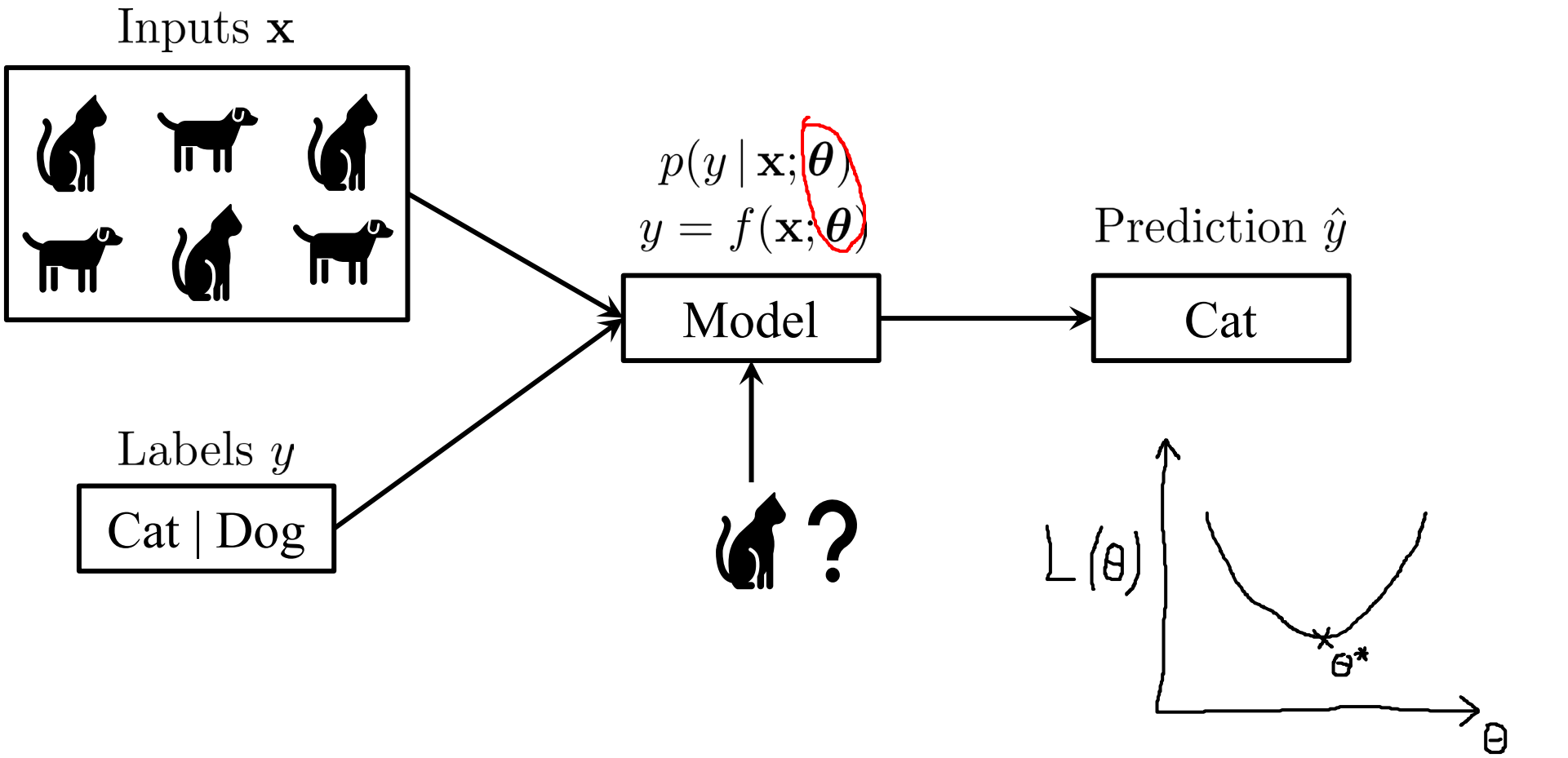
$$\text{Corr}[X, Y] = 0 \not\implies p_{XY}(X, Y) = p_X(X)p_Y(Y)$$

- Unless multivariate Gaussian:

$$p_{X_1,\ldots,X_n}(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\exp\Big(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big)$$

$$\boldsymbol{\mu} = \begin{bmatrix} E[X_1] \\ \vdots \\ E[X_n] \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \text{Var}[X_1] & \cdots & \text{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \text{Cov}[X_n, X_1] & \cdots & \text{Var}[X_n] \end{bmatrix}$$

Numerical Optimization



Find parameter values $\boldsymbol{\theta} \in \Theta$ that minimize a loss/cost/objective function $\mathcal{L}(\boldsymbol{\theta})$

Optimization Problem

Parameters: Unknown variables of a model/function

Continuous Optimization: $\Theta \subseteq \mathbb{R}^n$ for n variables

Find parameter values $\boldsymbol{\theta} \in \Theta$ that minimize a loss function $\mathcal{L} : \Theta \to \mathbb{R}$:

$$\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta})$$

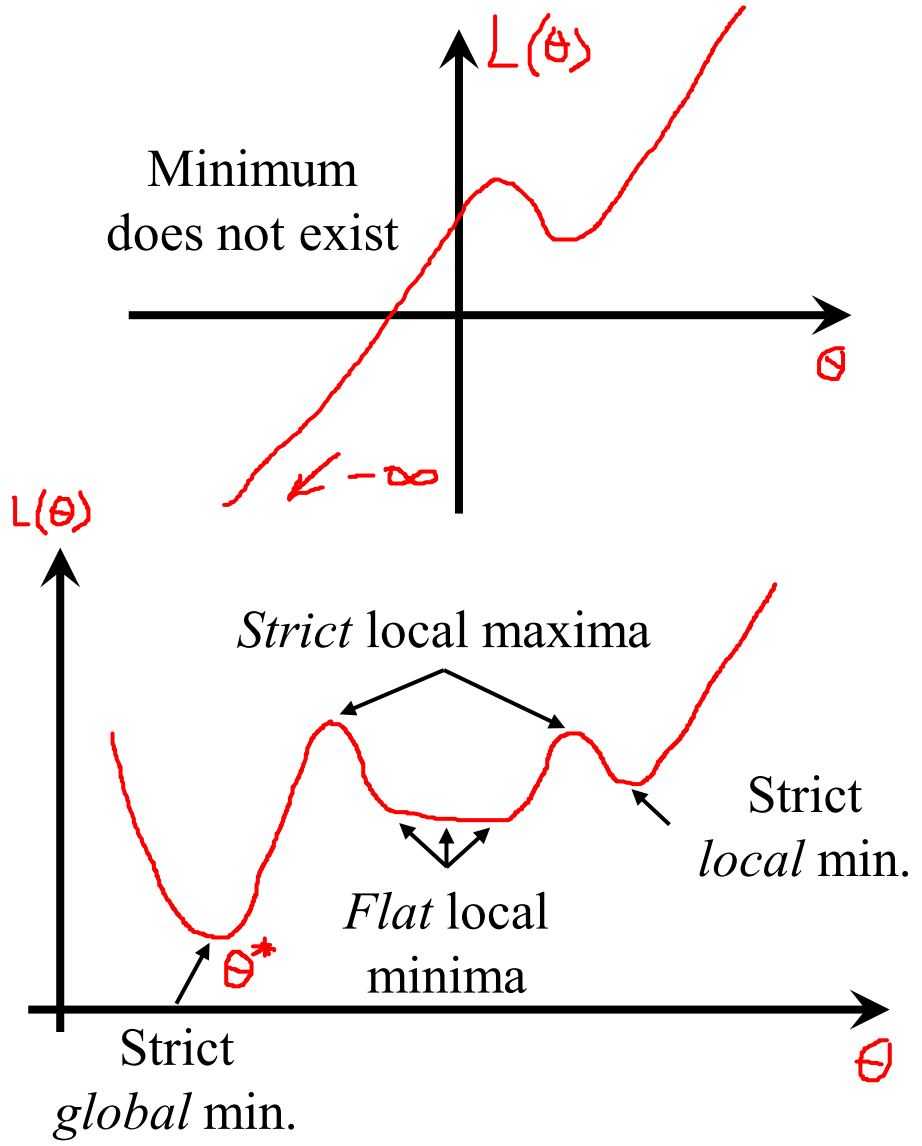
$\min L(\theta) = \max -L(\theta)$

Optimize: Find **set** of parameters that **minimize/maximize** objective

Objective/Loss Function: Quantitative measure of performance, e.g., error, profit, time etc.

Local vs Global Optimization

- Optimal solutions may <u>not exist</u> or may <u>not be unique</u>
- Even if they exist, **global** optima θ^* are difficult to recognize and locate
- **Local** optima assumed acceptable, especially in non-linear cases
- Local solutions may be **strictly lower** or **equally (flat)** to nearby competitors



Constrained vs Unconstrained Optimization

- **Unconstrained:** No constraints on $\mathcal{L}$, can choose any parameter value $\boldsymbol{\theta} \in \Theta$
- **Constrained:** Exist a set of constraints on allowable parameters
  - Partition into **inequality** and **equality** constraints:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \quad \text{s.t.} \quad c_i(\boldsymbol{\theta}) = 0, i \in \mathcal{E}$$

$$c_i(\boldsymbol{\theta}) \geq 0, i \in \mathcal{I}$$

  - **Feasible set** is the subset $\mathcal{C} \subseteq \Theta$ that satisfies these constraints
  - Constrained optimization formulation: $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta} \in \mathcal{C}} \mathcal{L}(\boldsymbol{\theta})$
  - *Example:*

$$\min_{\theta_1, \theta_2} (\theta_1 - 2)^2 + (\theta_2 - 1)^2 \quad \text{s.t.} \quad \theta_1^2 - \theta_2 \leq 0, \quad \theta_1 + \theta_2 \leq 2$$

θ $L(\theta)$ $c(\theta) = \begin{bmatrix} -\theta_1^2 + \theta_2 \\ -\theta_1 - \theta_2 + 2 \end{bmatrix}$

What Makes a Local Minimum?

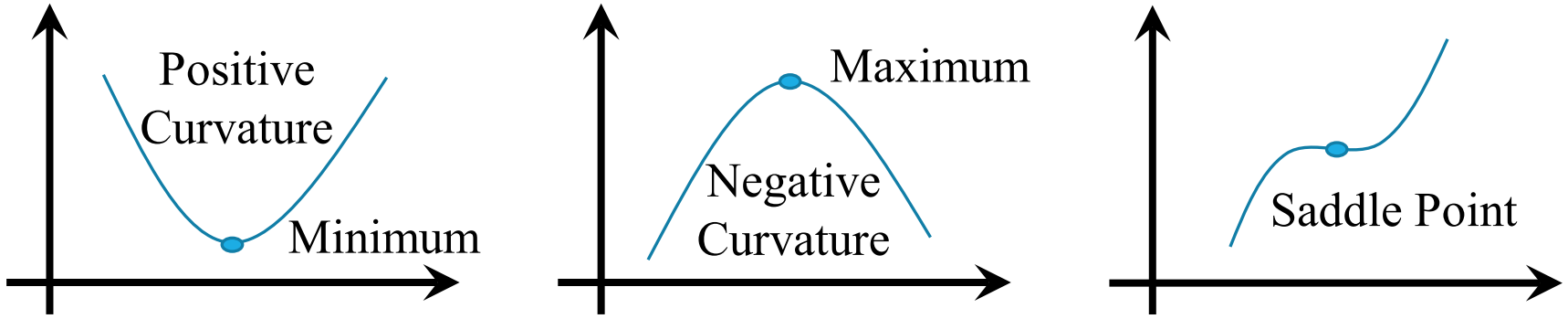
Two properties need to be satisfied to fulfil a local minimum:

1. $\mathbf{g}(\boldsymbol{\theta}^*) = \mathbf{0}$, *i.e.* $\boldsymbol{\theta}^*$ is a stationary point
2. $\mathbf{H}(\boldsymbol{\theta}^*) \geq 0$ *i.e.* $\mathbf{H}$ is a positive semi-definite matrix

Measures *curvature*

$$\mathbf{g}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\delta \mathcal{L}}{\delta \theta_1} \\ \frac{\delta \mathcal{L}}{\delta \theta_2} \\ \vdots \\ \frac{\delta \mathcal{L}}{\delta \theta_n} \end{bmatrix}$$

$$\mathbf{H}(\boldsymbol{\theta}) = \nabla^2 \mathcal{L}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\delta^2 \mathcal{L}}{\delta \theta_1^2} & \cdots & \frac{\delta^2 \mathcal{L}}{\delta \theta_1 \delta \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta^2 \mathcal{L}}{\delta \theta_n \delta \theta_1} & \cdots & \frac{\delta^2 \mathcal{L}}{\delta \theta_n^2} \end{bmatrix}$$



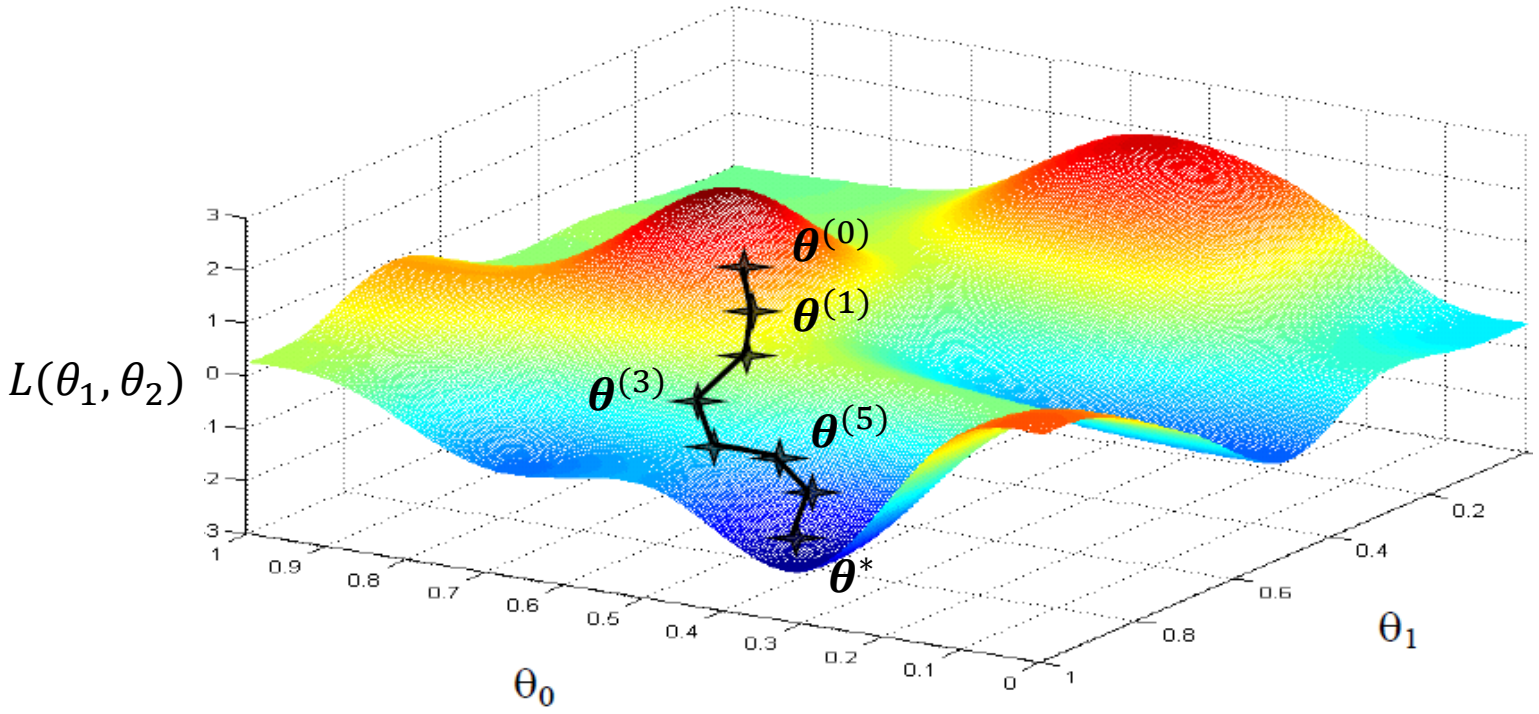
OK… But HOW do we Optimize $\mathcal{L}(\theta)$?

Follow the slope

Gradient Descent

Gradient Descent – Intuition

- Choose an initial value $\boldsymbol{\theta}^{(0)}$
- At each iteration, choose a new $\boldsymbol{\theta}^{(t+1)}$ to decrease $\mathcal{L}(\boldsymbol{\theta})$
- Repeat until stationary point (**minimum**) where $\mathbf{g}(\boldsymbol{\theta}) = \mathbf{0}$



Gradient Descent – Algorithm

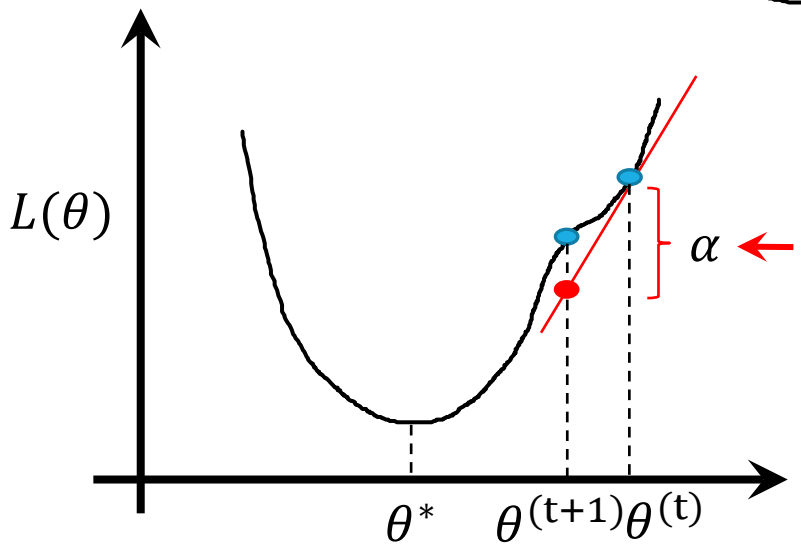
1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})$$

Vectorized

Move in direction of negative gradient hence name *descent*

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \frac{\delta \mathcal{L}}{\delta \theta_j}$$
$$\text{for } j = 1, \ldots, n$$



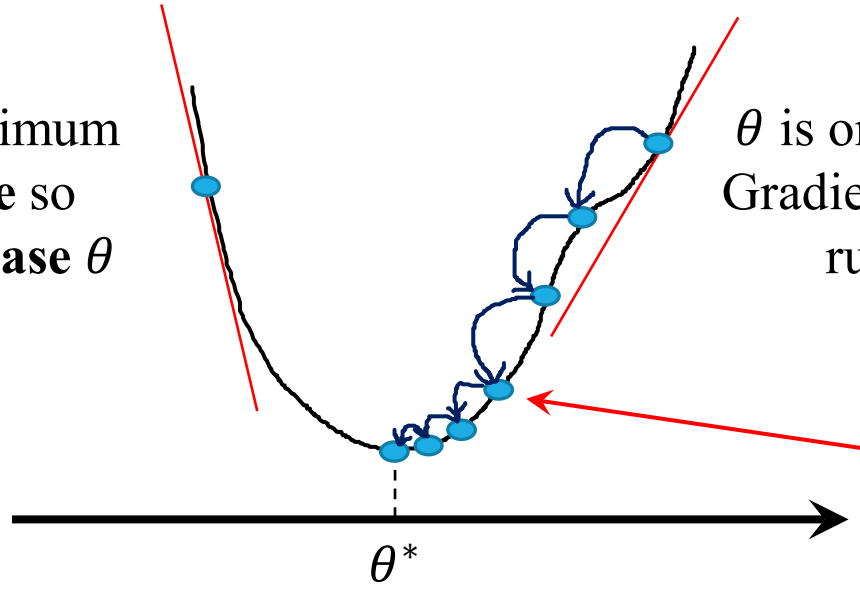
Step Size/Learning Rate:
A small value that controls how far we move along gradient

Gradient Descent – Derivative Direction

1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})$$

θ is on the left of minimum
Gradient is **negative** so update rule will **increase** θ



θ is on the right of minimum
Gradient is **positive** so update rule will **decrease** θ

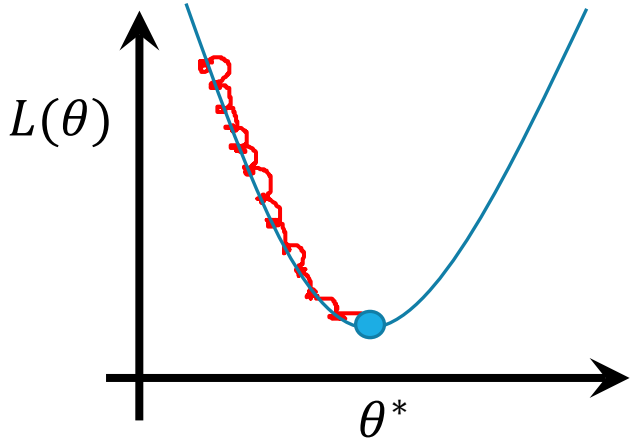
As we approach θ^* slope gets smaller and thus steps are smaller

Gradient Descent – Choosing Step Size

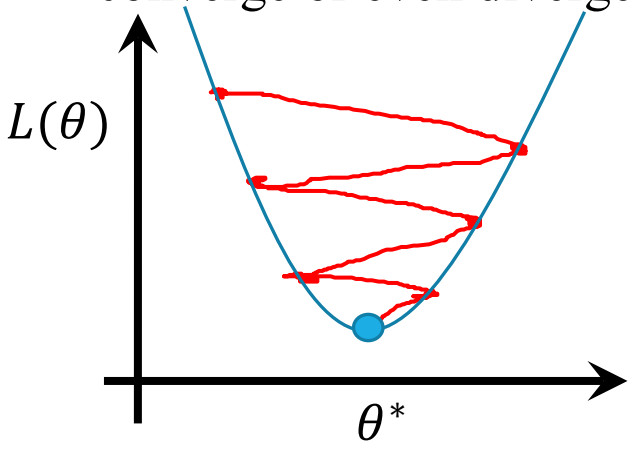
1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})$$

α too small then gradient descent can be slow



α too large then we can overshoot minimum, fail to converge or even diverge

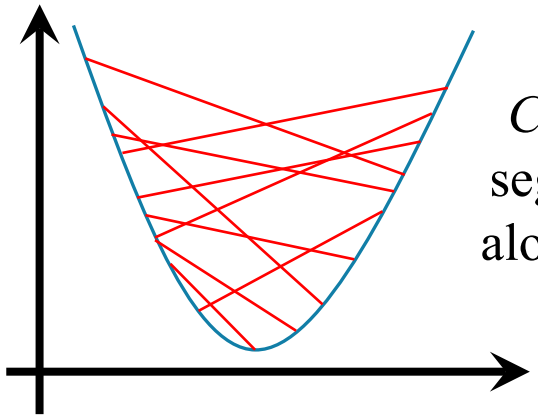


Gradient Descent – Convergence

1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence: When? $\mathbf{g}(\boldsymbol{\theta}) = \mathbf{0}$

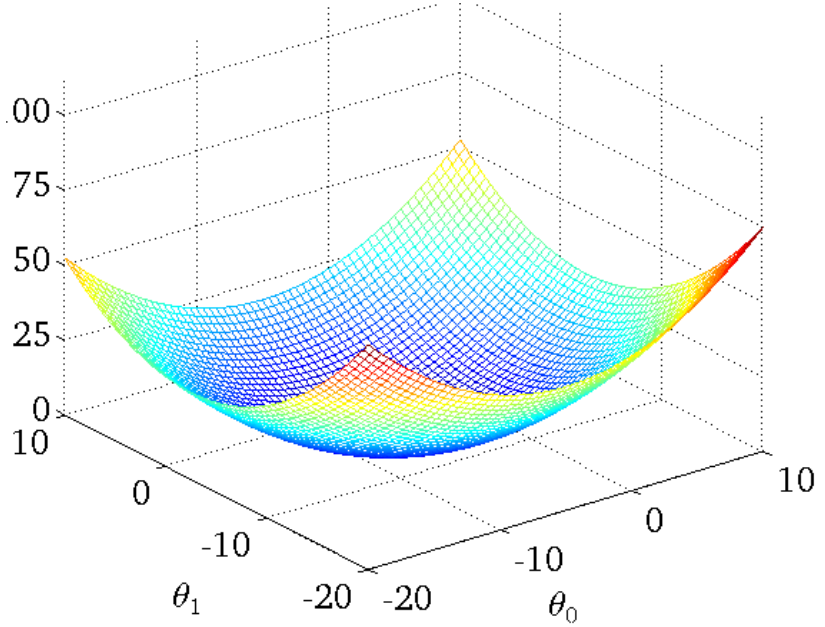
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})$$

- **Strictly convex** functions have **one global** minimum



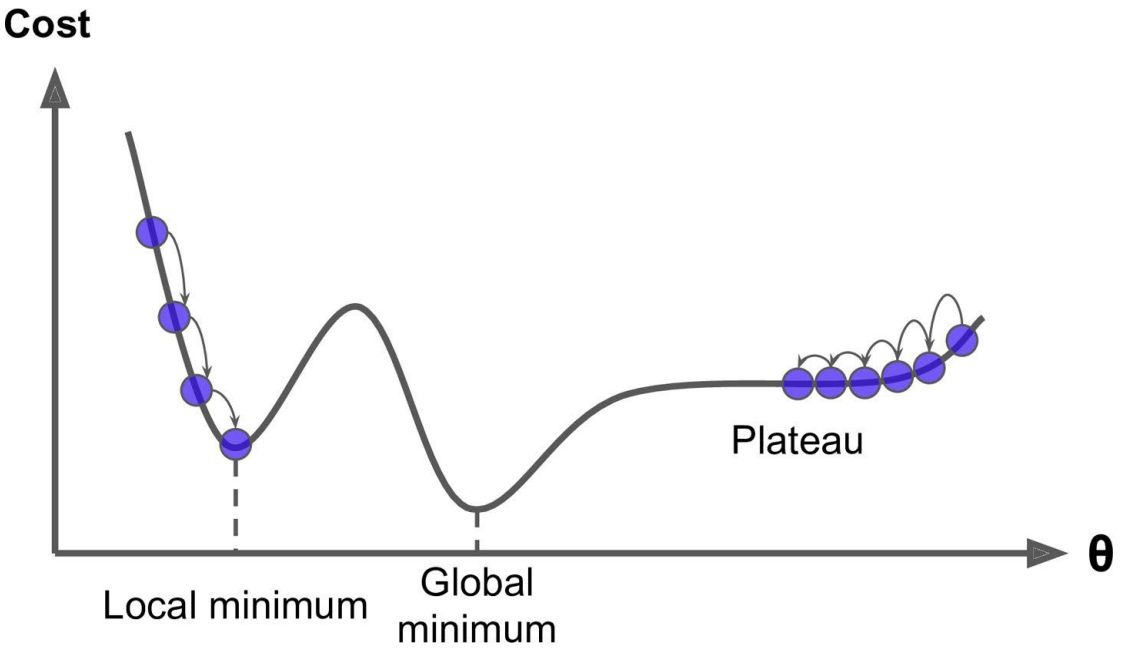
Convex Function: no line segment joining two points along the function lie below the graph at any point

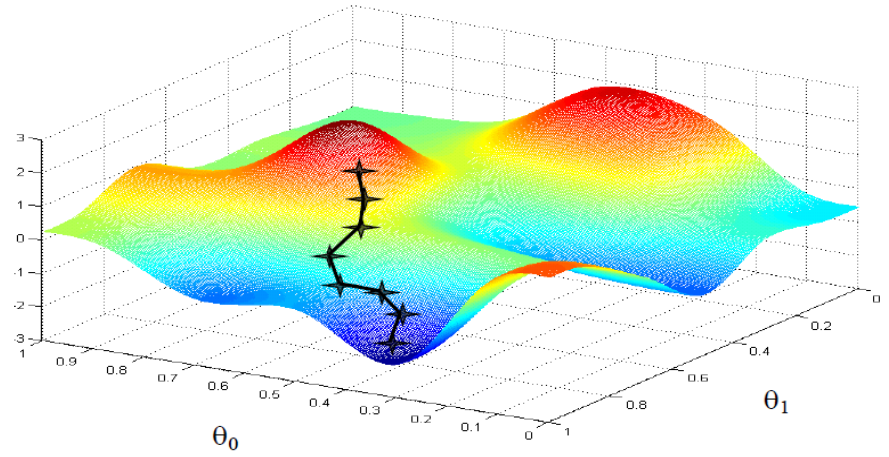
Convex functions
"bowl" shaped



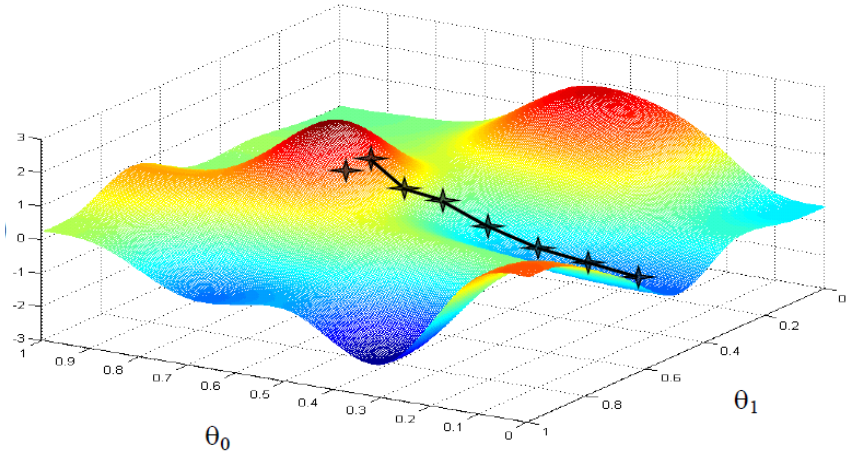
Gradient Descent – Convergence Issues

- Most functions more complex than convex and can cause convergence issues:
  - Stop at a local minimum
  - Plateau points bring descent to a slow halt





Try multiple random starting locations $\boldsymbol{\theta}^{(0)}$



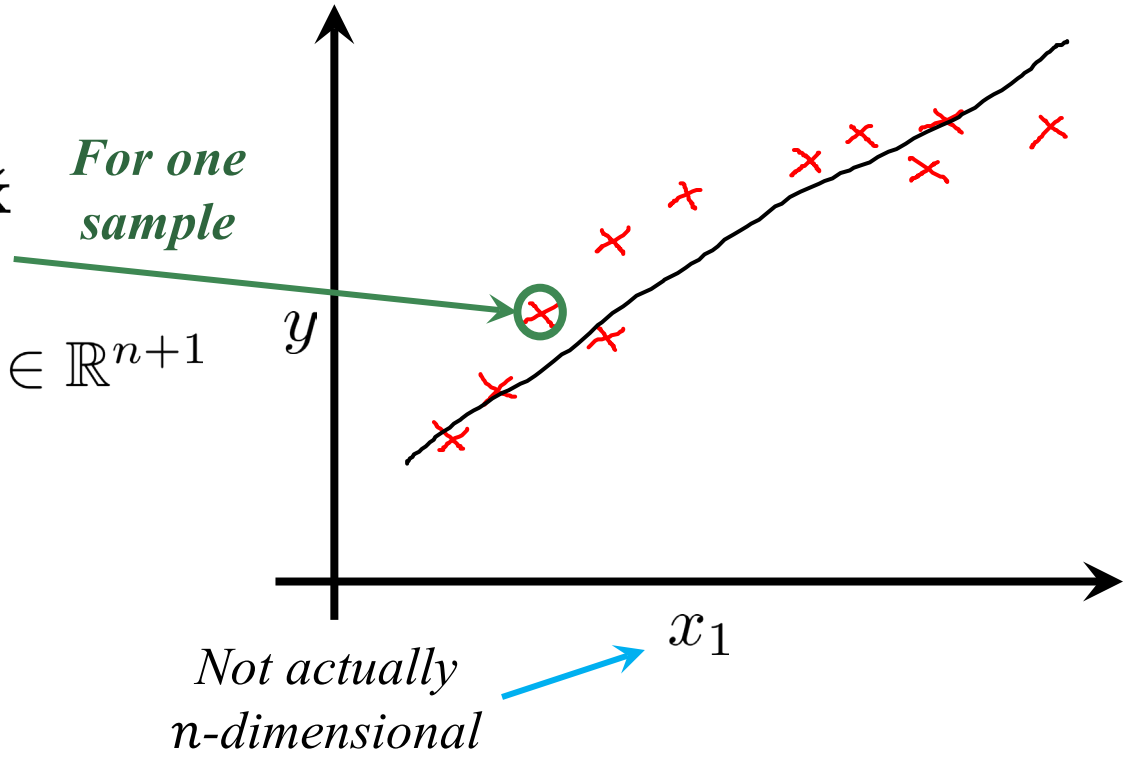
Example: Least Squares Regression – Sample Predictions

Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, N training samples
Inputs or **features** $\mathbf{x} \in \mathcal{X} = \mathbb{R}^n$ and **real**-valued responses $y \in \mathbb{R}$
Regression **coefficients** or **weights** $\mathbf{w} \in \mathbb{R}^n$ and a **bias** term $w_0 \in \mathbb{R}$

Model **predictions/hypotheses**:

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\theta}) = w_0 + \sum_{j=1}^{n} x_j w_j = \boldsymbol{\theta}^\intercal \tilde{\mathbf{x}}$$

where $\boldsymbol{\theta} = [w_0, \mathbf{w}] \in \mathbb{R}^{n+1}, \tilde{\mathbf{x}} = [1, \mathbf{x}] \in \mathbb{R}^{n+1}$



Example: Least Squares Regression – Dataset Predictions

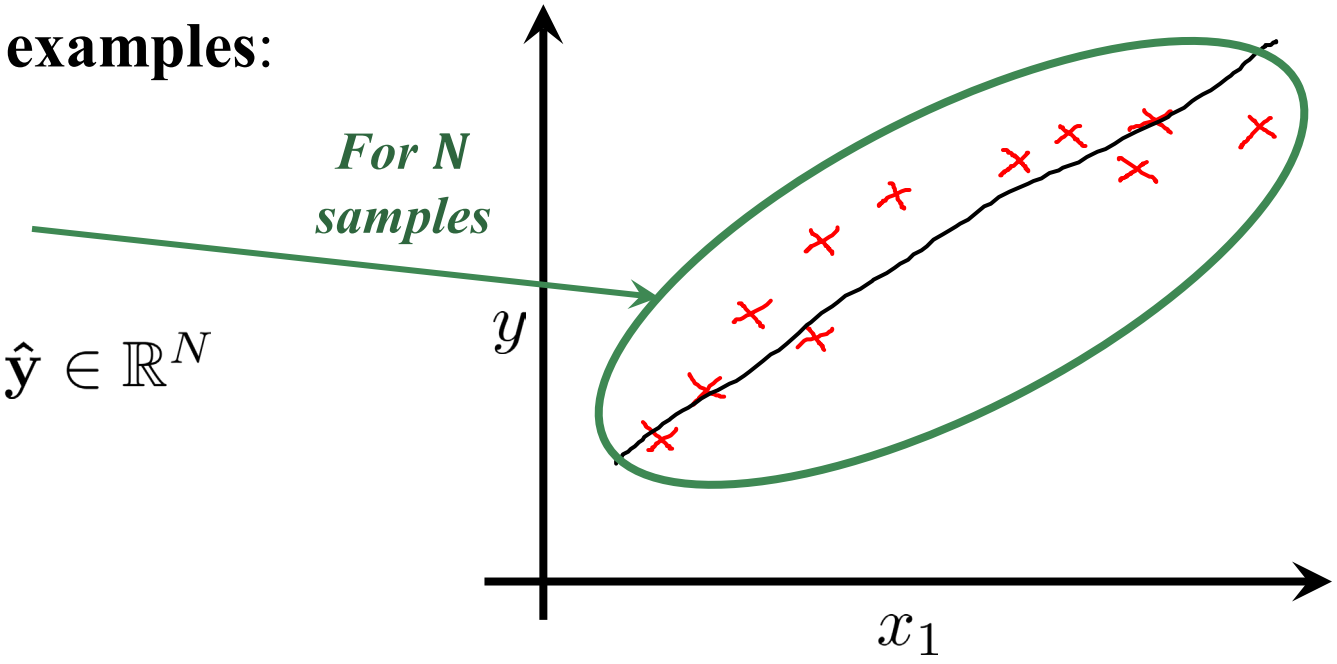
Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, N training samples
Inputs or **features** $\mathbf{x} \in \mathcal{X} = \mathbb{R}^n$ and **real**-valued responses $y \in \mathbb{R}$
Regression **coefficients** or **weights** $\mathbf{w} \in \mathbb{R}^n$ and a **bias** term $w_0 \in \mathbb{R}$

For full dataset with *N* **training examples**:

Design Matrix

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta}$$

where $\mathbf{X} \in \mathbb{R}^{N \times (n+1)}, \hat{\mathbf{y}} \in \mathbb{R}^N$



Example: Least Squares Regression – Loss Function

Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, N training samples
Inputs or **features** $\mathbf{x} \in \mathcal{X} = \mathbb{R}^n$ and **real**-valued responses $y \in \mathbb{R}$
Regression **coefficients** or **weights** $\mathbf{w} \in \mathbb{R}^n$ and a **bias** term $w_0 \in \mathbb{R}$

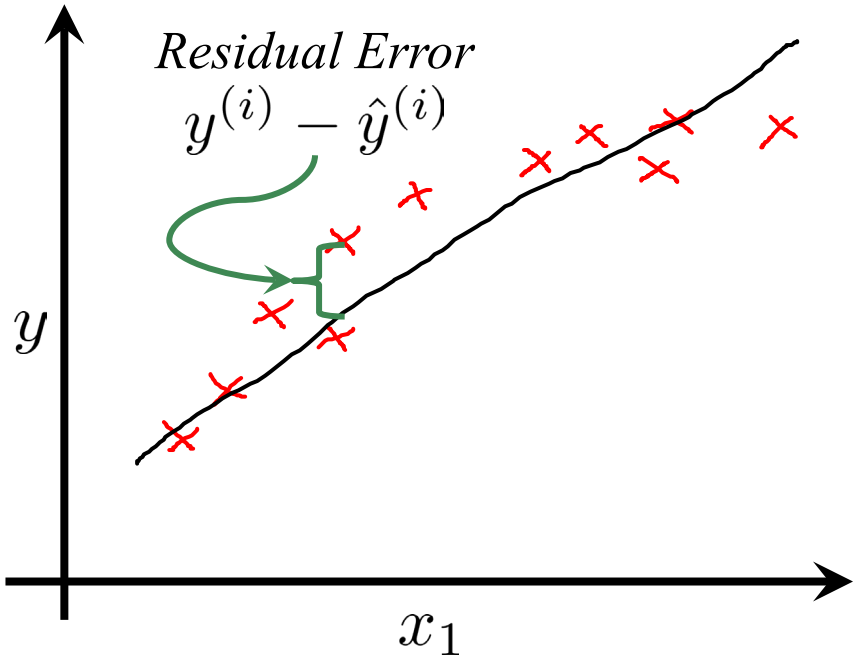
Define a **loss function** that measures how close $\hat{y}$ is to y, i.e., a function of the **residual error**:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})^2 = \sum_{i=1}^{N} (\boldsymbol{\theta}^\intercal \tilde{\mathbf{x}}^{(i)} - y^{(i)})^2$$

Written in matrix form

$$= ||\mathbf{X}\boldsymbol{\theta} - \mathbf{y}||_2^2$$

$$= (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\intercal (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$



Example: Least Squares Regression – Gradient

Find parameter values $\boldsymbol{\theta}_{\text{MSE}}$ that minimize our least squares loss function scaled by number of examples N, *i.e.*, the Mean Squared Error (MSE):

$$\boldsymbol{\theta}_{\text{MSE}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2N}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)} - y^{(i)})^2$$

Require gradient of $\mathcal{L}(\boldsymbol{\theta})$ to perform Gradient Descent:

$$\nabla\frac{1}{2N}\mathcal{L}(\boldsymbol{\theta}) = \frac{\delta}{\delta\boldsymbol{\theta}^{\intercal}}\left[\frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)} - y^{(i)})^2\right] = \frac{1}{2N}\sum_{i=1}^{N}\frac{\delta}{\delta\boldsymbol{\theta}^{\intercal}}\left[(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)} - y^{(i)})^2\right]$$

$$= \frac{1}{2N}\sum_{i=1}^{N}2(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)} - y^{(i)})\frac{\delta(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)})}{\delta\boldsymbol{\theta}^{\intercal}} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{\theta}^{\intercal}\tilde{\mathbf{x}}^{(i)} - y^{(i)})\tilde{\mathbf{x}}^{(i)}$$

Example: Least Squares Regression – Gradient Update

1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence:

Average scaling for MSE, 2 cancels out in derivative

Vector derivative

Batch GD update rule →

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \alpha \nabla \frac{1}{2N} \mathcal{L}(\boldsymbol{\theta}^{(t)}) \\ &= \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{\theta}^{\intercal(t)} \tilde{\mathbf{x}}^{(i)} - y^{(i)}) \tilde{\mathbf{x}}^{(i)} \right) \\ &= \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{N} \mathbf{X}^{\intercal} (\mathbf{X}\boldsymbol{\theta}^{(t)} - \mathbf{y}) \right)\end{aligned}$$

Equivalent matrix derivative ←

3. When $\mathbf{g}(\boldsymbol{\theta}) = \mathbf{0}$, then we have derived $\boldsymbol{\theta}^*$ using GD

Example: Least Squares Regression – Analytical Solution

Alternatively, we can solve the least squares regression problem by setting the gradient to 0:

$$\begin{aligned}\nabla \mathcal{L}(\boldsymbol{\theta}) &= \mathbf{X}^\intercal(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\ &= \mathbf{X}^\intercal\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^\intercal\mathbf{y} = 0 \implies \mathbf{X}^\intercal\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^\intercal\mathbf{y}\end{aligned}$$

Hence can directly compute the optimal $\boldsymbol{\theta}^*$ using the closed-form solution:

$$\boldsymbol{\theta}^* = (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{y}$$

Least Squares GD vs Analytical Solution

- Analytical solution to directly compute optimal minimum

$$\boldsymbol{\theta}^* = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y}$$

- Least Squares GD
  - ✓ Linear complexity $O(N)$
  - ✓ Generally applicable
  - × Need to select parameters, e.g., learning rate/step size a
  - × Might get stuck in local optima

- Analytical Solution
  - ✓ No parameter tuning
  - ✓ Gives global optimum
  - × Not generally applicable
  - × Poor complexity $O(Nn^2)$

Coding Break

Stochastic Gradient Descent – Motivation

- **Batch GD:** update rule at every step is based on **entire training set** (size N)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)}) = \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{N} \sum_{i=1}^{N} \nabla \mathcal{L}_i(\boldsymbol{\theta}^{(t)}) \right)$$

Per-example loss, e.g., residual error

- Each update step has $O(N)$ complexity

- What if N is huge, e.g., ~billions of examples?
  - **Memory intensive** to process training set in one step (terabytes to store)
  - **Computationally expensive** to evaluate loss for N examples at once

Stochastic Gradient Descent – Noisy Estimate

- **Stochastic Optimization:** Minimize average value of loss

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{i \sim \text{Uniform}\{1,\ldots,N\}}[\mathcal{L}_i(\boldsymbol{\theta})]$$

- Instead of computing $\nabla\mathcal{L}(\boldsymbol{\theta})$ over all N examples, sample a **mini-batch** set $\mathcal{B}$ of size M (~one to hundreds of examples) uniformly from training set

$$\mathcal{B} = \{\mathbf{x}^{(i)}\} \sim \text{Uniform}\{1, \ldots, N\} \quad \text{for } i \in \{1, \ldots, M\}$$

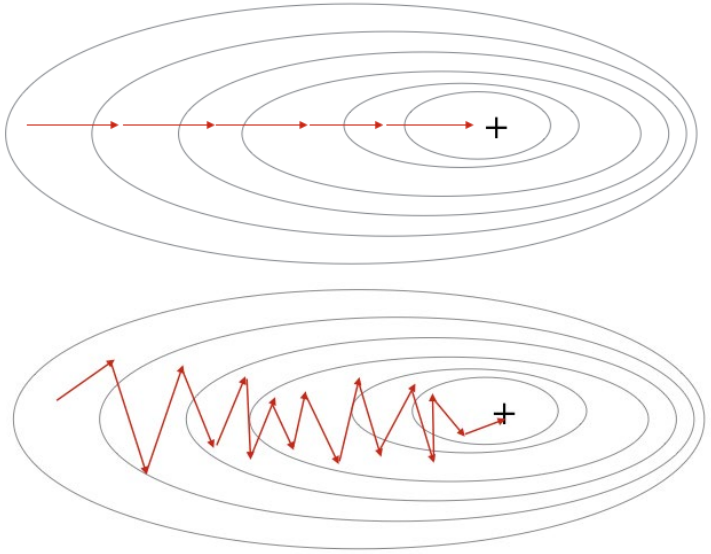
Batch size

- Compute a **noisy estimate** of derivative in GD update:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla\, \mathbb{E}_{i \in \mathcal{B}}[\mathcal{L}_i(\boldsymbol{\theta}^{(t)})] = \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{M} \sum_{i \in \mathcal{B}} \nabla \mathcal{L}_i(\boldsymbol{\theta}^{(t)}) \right)$$

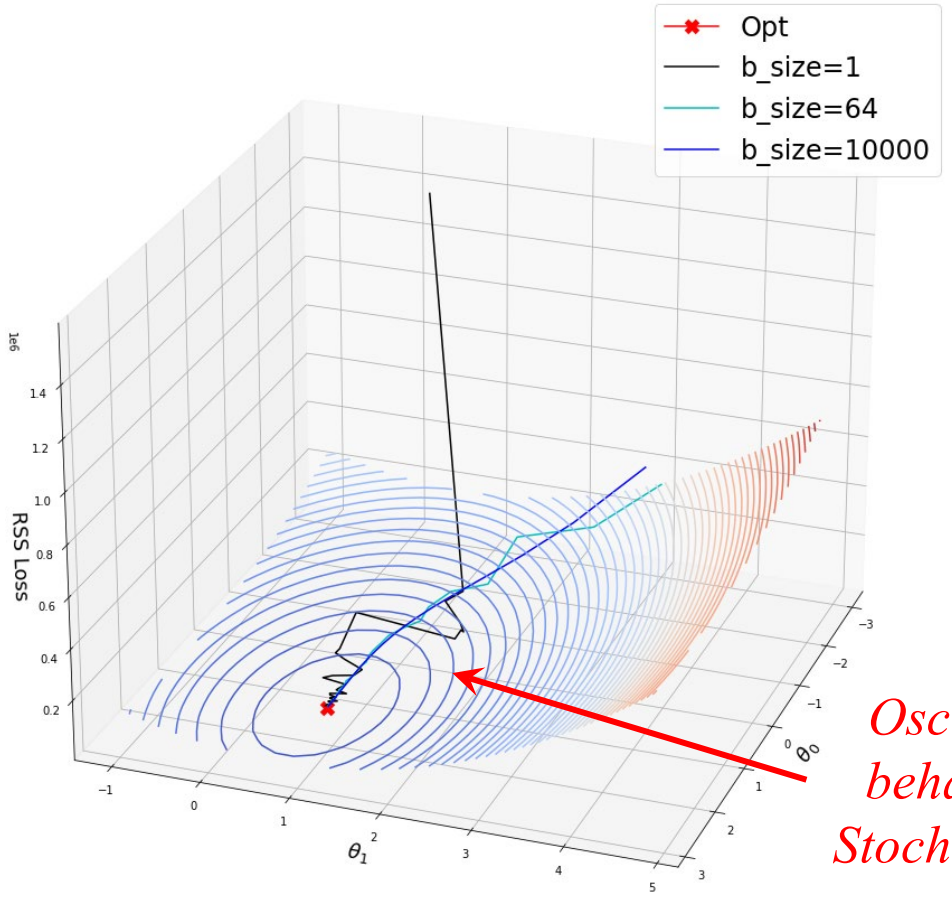
Stochastic Gradient Descent – Notes

- Terminology depending on batch size M
  - If $M = N$ then **Batch GD**
  - If $M \ll N$ but $M \neq 1$ then **Mini-batch GD**
  - If $M = 1$ then **Stochastic GD**
  - An **epoch** is a single sweep over all N samples

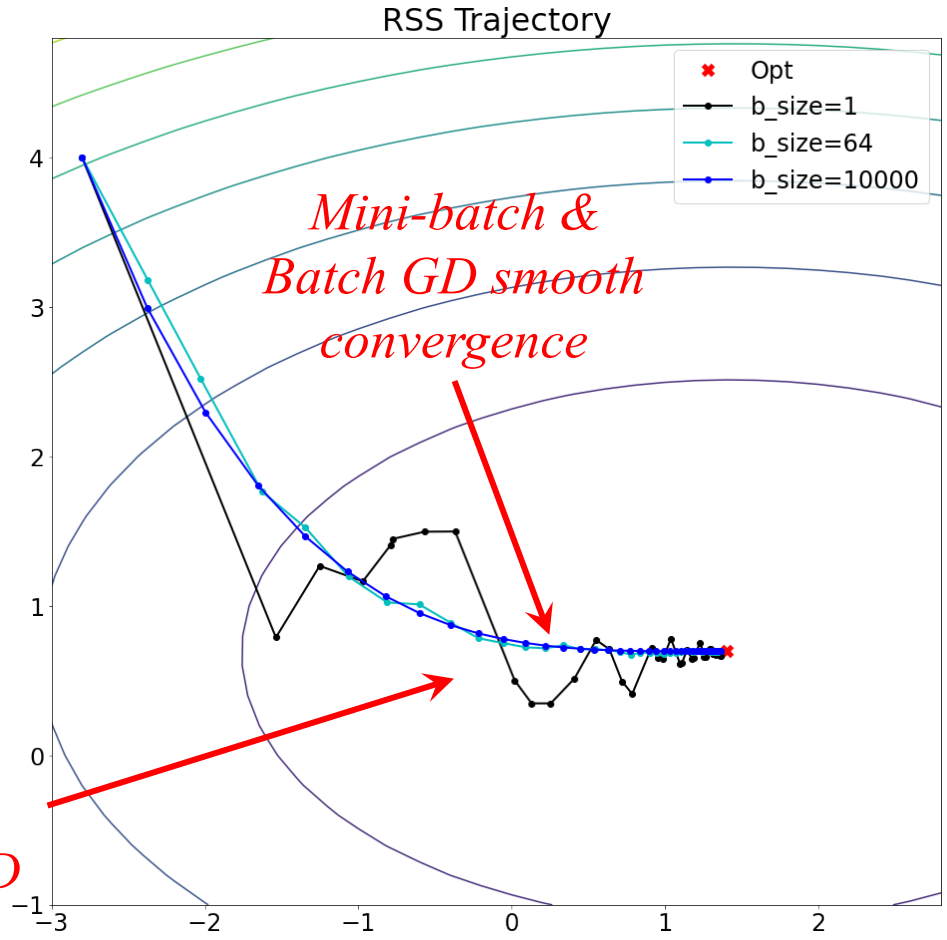


- **Unbiased estimate** of gradient; may never exactly "converge" to minimum
- Mini-batch/Stochastic GD makes progress to minimum with each new batch of examples → GD update complexity is NOT dependent on N
- Draw samples **without replacement**, i.e., each sample drawn once per epoch

Batch vs Stochastic Gradient Descent (1)



Oscillatory behavior of Stochastic GD



Batch vs Stochastic Gradient Descent (2)

- **Memory Efficiency:** Stochastic > Batch GD
- **Computational Efficiency:** Varies
  - Stochastic GD fast processing per sample but poor use of compute resources
  - Batch GD updates all at once (vectorization) but will be slow if N very large
- **Convergence Speed:** Varies
  - For very large N, Stochastic > Batch GD as it updates more frequently
  - Batch GD stable so less oscillations could lead to faster convergence
- **Convergence Guarantees:** Varies
  - Stochastic oscillates around minima but it can also escape shallow local minima

Mini-batch GD best of both worlds

Example: Least Squares Regression – Mini-batch GD

1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Batchify dataset: $\mathcal{B}_b \sim \text{Uniform}\{1, \ldots, N\}$
3. Repeat until convergence:
   — For each b in total number of batches:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla \mathbb{E}_{i \in \mathcal{B}_b}[\mathcal{L}_i(\boldsymbol{\theta}^{(t)})]$$

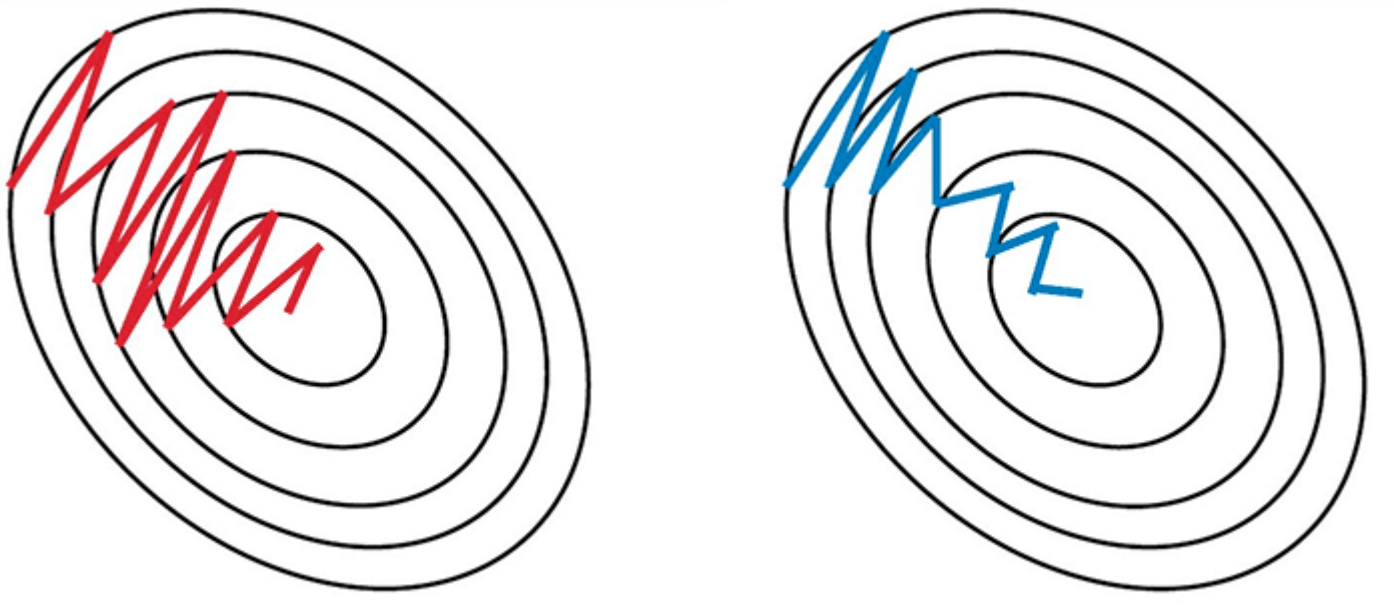
$$= \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{|\mathcal{B}_b|} \sum_{i \in \mathcal{B}_b} (\boldsymbol{\theta}^{\intercal(t)} \tilde{\mathbf{x}}^{(i)} - y^{(i)}) \tilde{\mathbf{x}}^{(i)} \right)$$

Batch size M may not be constant so write as cardinality $|B_b|$ instead

4. When $\mathbf{g}(\boldsymbol{\theta}) \approx \mathbf{0}$, then we have derived $\boldsymbol{\theta}^*$ using mini-batch GD

Coding Break

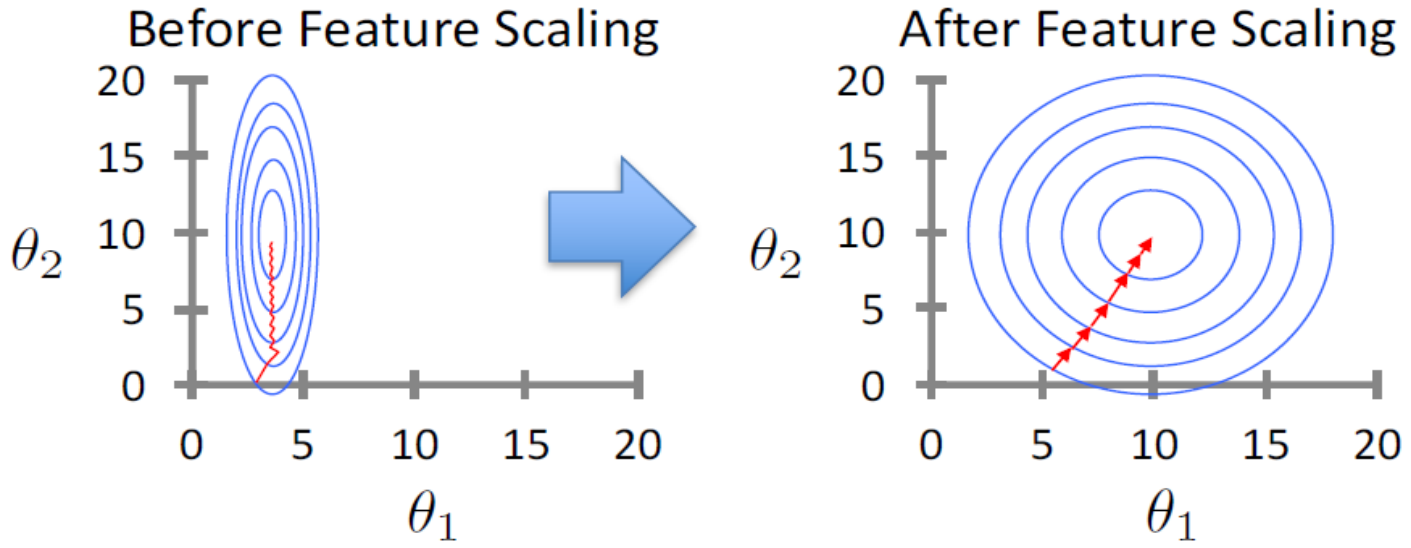
Improving on Gradient Descent



Feature Scaling

- Ensure input features $x_1, x_2, \ldots, x_n$ are **similarly scaled** → Faster GD

Apply transformation to both training & test data



Imagine x_1 in range [100, 10000] while x_2 in range [0,10]

- **Min-max normalization:**

$$x_j^{(i)} = \frac{x_j^{(i)} - \min x_j}{\max x_j - \min x_j} \in [0, 1]$$

- **Standardization:**

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$ mean 0, stddev 1

Momentum

- GD is slow in flat regions → how to speed up?
- Add a **momentum** term to update:

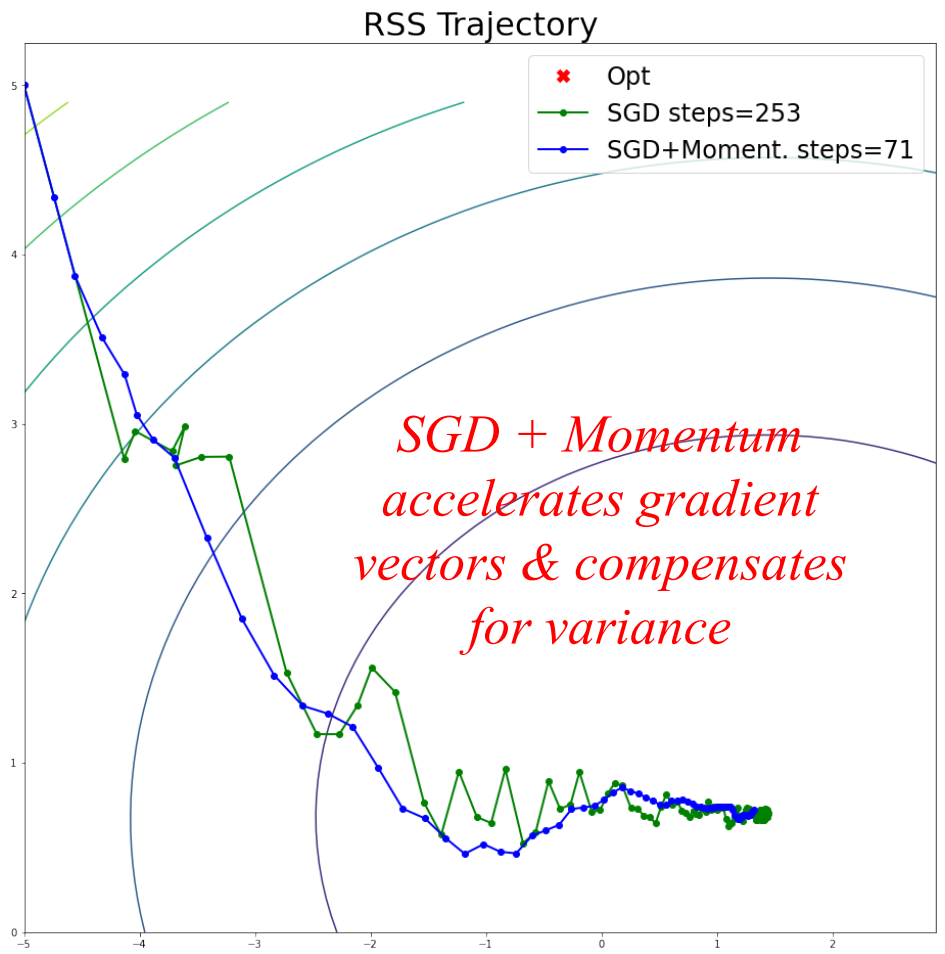
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha m^{(t+1)}$$

$$m^{(t+1)} = \beta m^{(t)} + \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})$$

- Simple to implement in practice

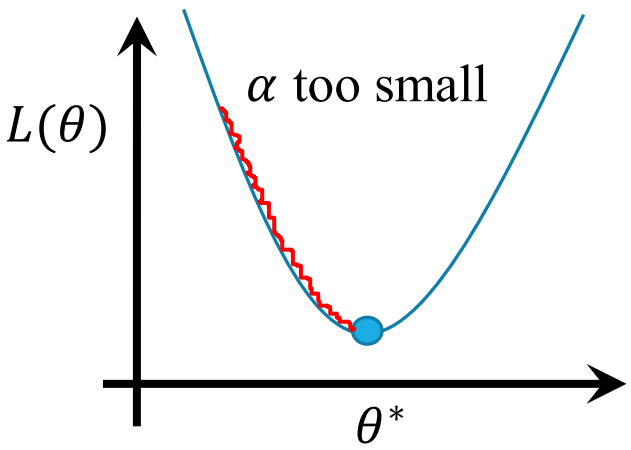
(1 − β)

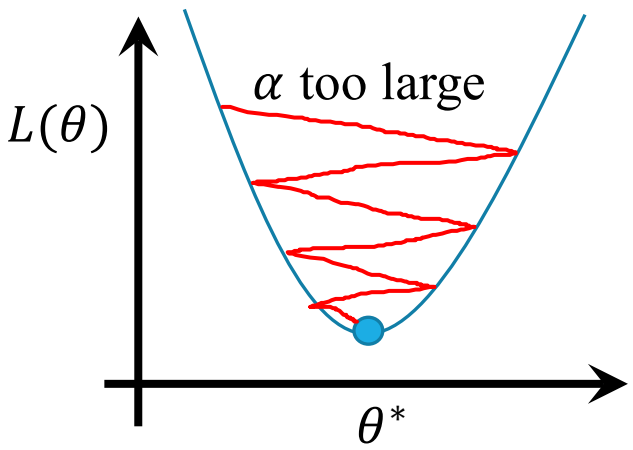
- View $m^{(t+1)}$ like a moving average of past gradients with new β scaling factor

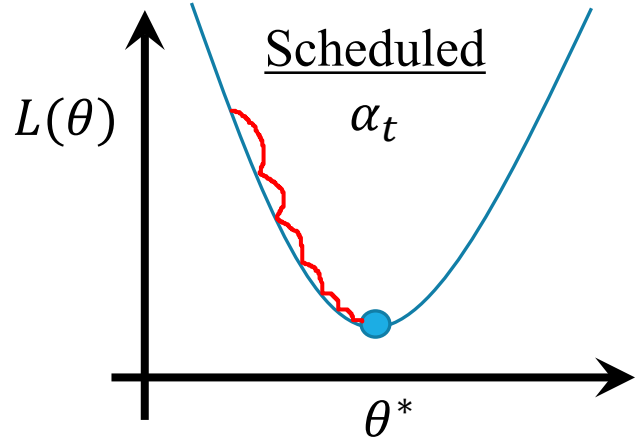


Learning Rate Scheduling

- Vary parameter α_t over time (**learning rate scheduling**)

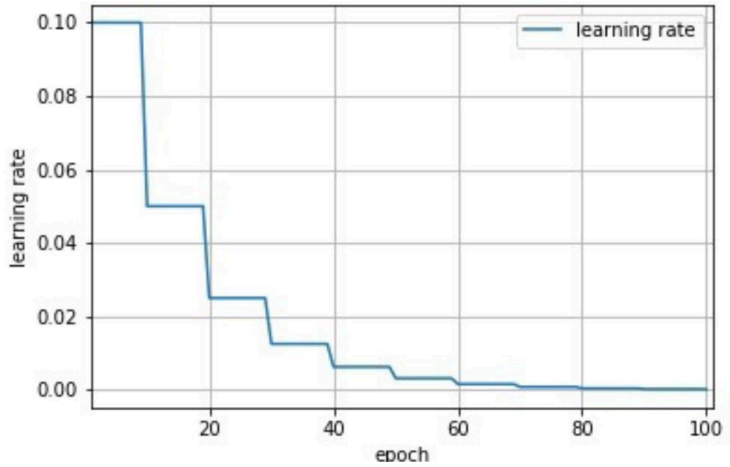






- Example: **piecewise constant** schedule

$$\alpha_t = \alpha_i \text{ if } t_i \leq t \leq t_{i+1}$$



Second Order Methods

- **First-order** methods computationally cheap as only use **gradient** but **slow**
- **Second-order** methods incorporate **curvature** to for **faster** convergence
- Example: **Newton's** method

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \overbrace{\left(\nabla^2 \mathcal{L}(\boldsymbol{\theta}^{(t)})\right)^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}^{(t)})}^{\textit{Newton Step } \mathbf{H}^{-1} \times \mathbf{g}}$$

- Fast convergence but inverse $\mathbf{H}^{-1}$ is expensive for high n
- **Quasi-Newton** methods approx. $\mathbf{H}^{-1}$
- Still difficult to estimate Hessian for noisy gradient estimates as in SGD

Stochastic/Mini-batch GD most popular in machine/deep learning

Constrained Optimization

- Can solve **equality** constrained problems by forming a **Lagrangian**

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \quad \text{s.t.} \quad c_{i\in\mathcal{E}}(\boldsymbol{\theta}) = 0 \quad \Longrightarrow \quad \mathcal{L}(\boldsymbol{\theta}, \lambda) = \mathcal{L}(\boldsymbol{\theta}) + \sum_{i\in\mathcal{E}}^{m} \lambda_i c_i$$

Lagrange multiplier λ *for m constraints*

- At stationary point: $\mathcal{L}(\boldsymbol{\theta}, \lambda) = 0$

- Example on optimizing **quadratic forms**:

$$\max_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^\intercal \mathbf{A}\mathbf{x} \quad \text{s.t.} \quad ||\mathbf{x}||_2^2 = 1 \quad \Longrightarrow \quad \mathcal{L}(\mathbf{x}, \lambda) = \mathbf{x}^\intercal \mathbf{A}\mathbf{x} + \lambda(1 - \mathbf{x}^\intercal \mathbf{x})$$

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) = 2\mathbf{A}^\intercal \mathbf{x} - 2\lambda\mathbf{x} = 0 \quad \Longrightarrow \quad \mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Optimal x^* to min/max quadratic forms are ***eigenvectors*** *of A*

- **KKT** conditions generalize to also handle **inequality** constraints

Gradient Descent in Practice

- **Mini-batch** GD is an effective optimization algorithm
  - Most widely used training algorithm in machine learning
  - Strikes a good balance between Stochastic and Batch GD
- Watch out for **convergence issues** and **getting stuck** in local optima
  - Poorly set hyperparameters can cause both effects
  - Scheduling α_t is good practice
- **Scale features**
  - Better & faster GD solution as one feature is not dominating the loss objective
- Use **momentum**
  - Simple heuristic that almost always works better than standard GD
  - Improves convergence speed and may even help escape saddle points

Concluding Remarks

- Today we explored **Batch & Stochastic Gradient Decent (GD)**
- Look at *"ls_regression_batch_gd.ipynb"* for batch GD and *"ls_regression_stochastic_gd.ipynb"* for stochastic GD in the context of least squares (LS) linear regression:

https://github.com/mazrk7/EECE5644_IntroMLPR_LectureCode/blob/main/notebooks/linear_regression/ls_regression_batch_gd.ipynb

https://github.com/mazrk7/EECE5644_IntroMLPR_LectureCode/blob/main/notebooks/linear_regression/ls_regression_stochastic_gd.ipynb

- Contains examples of **momentum** & **second-order** methods
- Questions?