# Introduction to Machine Learning

## Linear Regression

Varun Chandola

Computer Science & Engineering
State University of New York at Buffalo
Buffalo, NY, USA
chandola@buffalo.edu

University at Buffalo
Department of Computer Science and Engineering
School of Engeering and Applied Sciences

# Outline

# Basics

- Data - scalar ($x$), vector ($\mathbf{x}$), Matrix ($\mathbf{X}$)

## Scalars

- Numeric ($x \in \mathbb{R}$)
- Categorical (e.g., $x \in \{0, 1\}$)
- Constants will be denoted as $D$, $M$, etc.

## Vector

- Length of a vector $\mathbf{x} \in \mathbb{R}^D$
- Vector *dot* product ($\mathbf{x} \cdot \mathbf{y}$)
- Norm of a vector ($|\mathbf{x}|$, $\|\mathbf{x}\|, \|\mathbf{x}\|_p$)

## Matrix

- Size of a matrix ($\mathbf{X} \in \mathbb{R}^{M \times N}$)
- Transpose of a matrix ($\mathbf{X}^\top$)
- Matrix product ($\mathbf{XY}$))

- A vector is a special matrix with only one column

$$\mathbf{x} \cdot \mathbf{y} \equiv \mathbf{x}^\top \mathbf{y}$$

# Linear Regression

- There is one scalar **target** variable $y$
- There is one vector **input** variable $x$
- Inductive bias:

$$y = \mathbf{w}^\top \mathbf{x}$$

### Linear Regression Learning Task

Learn $\mathbf{w}$ given training examples, $\langle \mathbf{X}, \mathbf{y} \rangle$.

# Geometric Interpretation

- Fitting a straight line to $d$ dimensional data

$$y = \mathbf{w}^\top \mathbf{x}$$

$$y = \mathbf{w}^\top \mathbf{x} = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

- Will pass through origin
- Add intercept

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

- Equivalent to adding another column in $\mathbf{X}$ of 1s.

# Incorporating Bias/Intercept

## Explicit Bias

$$\begin{aligned}\mathbf{x} &\equiv \{x_1, x_2, \ldots, x_d\}\\ \mathbf{w} &\equiv \{w_1, w_2, \ldots, w_d\}\\ y &= w_0 + \mathbf{w}^\top \mathbf{x}\end{aligned}$$

## Implicit Bias

$$\begin{aligned}\mathbf{x} &\equiv \{1, x_1, x_2, \ldots, x_d\}\\ \mathbf{w} &\equiv \{w_0, w_1, w_2, \ldots, w_d\}\\ y &= \mathbf{w}^\top \mathbf{x}\end{aligned}$$

# Learning Parameters - Least Squares Approach

- Minimize *squared loss*

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- or,

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

- Make prediction ($\mathbf{w}^\top \mathbf{x}_i$) as close to the target ($y_i$) as possible
- Least squares estimate

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

- *We will derive this expression in class.*

# Machine Learning as Optimization Problem[1]

- Learning is optimization
- Faster optimization methods for faster learning
- Let $\mathbf{w} \in \mathbb{R}^d$ and $S \subset \mathbb{R}^d$ and $f_0(\mathbf{w}), f_1(\mathbf{w}), \ldots, f_m(\mathbf{w})$ be real-valued functions.
- Standard optimization formulation is:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & f_0(\mathbf{w}) \\ \text{subject to} \quad & f_i(\mathbf{w}) \leq 0, \;\; i = 1, \ldots, m. \end{aligned}$$

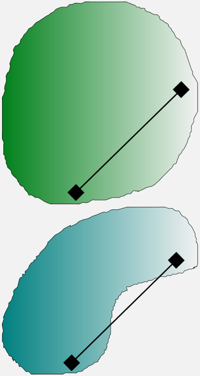
[1] Adapted from http://ttic.uchicago.edu/~gregory/courses/ml2012/lectures/tutorial_optimization.pdf. Also see, http://www.stanford.edu/~boyd/cvxbook/ and http://scipy-lectures.github.io/advanced/mathematical_optimization/.

# Solving Optimization Problems

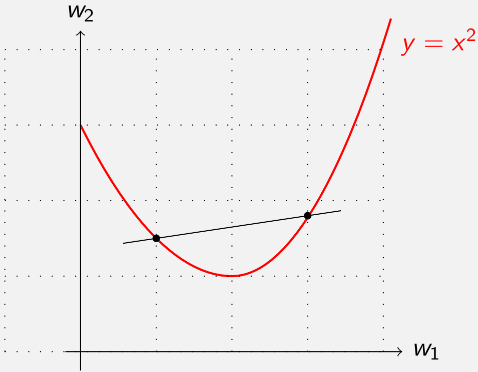
- Methods for **general optimization problems**
  - Simulated annealing, genetic algorithms
- Exploiting *structure* in the optimization problem
  - **Convexity**, Lipschitz continuity, smoothness

# Convexity

## Convex Sets



## Convex Functions



# Convex Optimization

- Optimality Criterion

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & f_0(\mathbf{w}) \\ \text{subject to} \quad & f_i(\mathbf{w}) \leq 0, \; i = 1, \ldots, m. \end{aligned}$$

  where all $f_i(\mathbf{w})$ are **convex functions**.
- $\mathbf{w}_0$ is feasible if $\mathbf{w}_0 \in \mathit{Dom}\ f_0$ and all constraints are satisfied
- A feasible $\mathbf{w}^*$ is optimal if $f_0(\mathbf{w}^*) \leq f_0(\mathbf{w})$ for all $\mathbf{w}$ satisfying the constraints

# Matrix Calculus Basics

$$\frac{\partial \mathbf{a}^\top \mathbf{b}}{\partial \mathbf{a}} = \frac{\partial \mathbf{b}^\top \mathbf{a}}{\partial \mathbf{a}} = \mathbf{b}$$

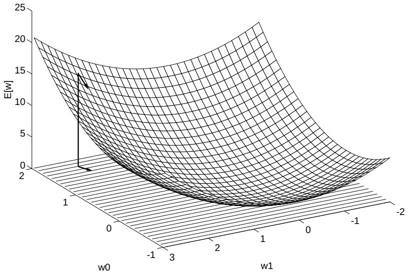
$$\frac{\partial \mathbf{a}^\top \mathbf{M}\mathbf{a}}{\partial \mathbf{a}} = 2\mathbf{M}\mathbf{a}$$

where $\mathbf{M}$ is a symmetric matrix.

# Gradient of a Function

- Denotes the direction of steepest ascent

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_d} \end{bmatrix}$$



# Finding Extremes of a Single Variable Function

1. Set derivative to 0
$$\nabla J(\mathbf{w}) = 0$$
2. Check second derivative for minima or maxima or saddle point

# Finding Extremes of a Multiple Variable Function - Gradient Descent

1. Start from any point in variable space
2. Move along the direction of the steepest descent (or ascent)
   - By how much?
   - A learning rate ($\eta$)
   - What is the direction of steepest descent?
     - Gradient of $J$ at $\mathbf{w}$

## Training Rule for Gradient Descent

$$\mathbf{w} = \mathbf{w} - \eta \nabla J(\mathbf{w})$$

For each weight component:

$$w_j = w_j - \eta \frac{\partial J}{\partial w_j}$$

# Convergence Guaranteed?

- Error surface contains only one global minimum
- Algorithm *will* converge
  - Examples need not be linearly separable
- $\eta$ should be *small enough*
- Impact of too large $\eta$?
- Too small $\eta$?

# Issues with Gradient Descent

- Slow convergence
- Stuck in local minima

# Stochastic Gradient Descent [1]

- **Update weights after every training example.**
- For sufficiently small $\eta$, closely approximates Gradient Descent.

| **Gradient Descent** | **Stochastic Gradient Descent** |
|---|---|
| Weights updated after summing error over all examples | Weights updated after examining each example |
| More computations per weight update step | Significantly lesser computations |
| Risk of local minima | Avoids local minima |

# Gradient Descent Based Method

- Minimize the squared loss using *Gradient Descent*

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- Why?

# References

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel.
Backpropagation applied to handwritten zip code recognition.
*Neural Comput.*, 1(4):541–551, Dec. 1989.