# ITCS 6156/8156 Fall 2024
# Machine Learning

# Linear Regression

Instructor: Hongfei Xue
Email: hongfei.xue@charlotte.edu
Class Meeting: Tue & Thu, 4:00 PM – 5:15 PM, WWH 130

Some content in the slides is based on Dr. Razvan's lecture

# Recap

- Spam email classification:
  - Binary classification of emails:
    Spam vs. Ham (Legitimate message)


- Expert Systems approach (Rule-based)
  - A group of experts write rules determining whether an email is spam or not.
  - A programmer implement the rules into computer code

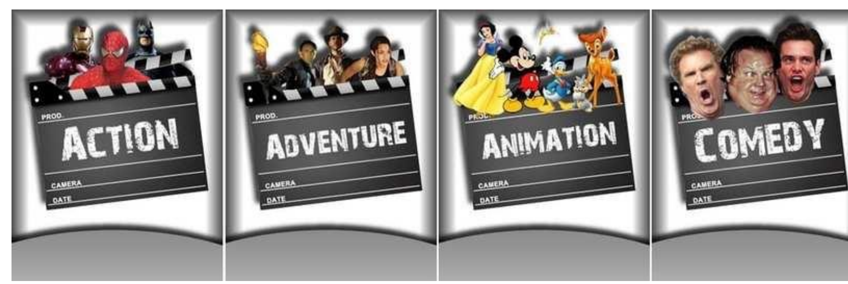
- Machine Learning algorithms can automatically learn the weights to combine features.

# Machine Learning

- Function is everywhere!
  - Function → $f$ ; Input instance → $x$; Output Targe → $y$

$f($  $)$ → 

$f($ 

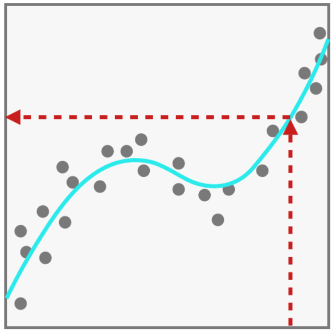
$)$ → 


- Machine Learning Task:
  - learn an (unknown) function $f: X \rightarrow Y$ that maps input instances x $\in X$ to output targets $f(x) \in Y$.

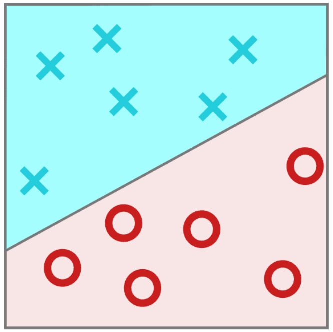
# Classification vs. Regression

- Machine Learning Task:
  - learn an (unknown) function $f: X \rightarrow Y$ that maps input instances x $\in X$ to $\boxed{\text{output targets } f(x) \in Y}$.
- Linear Regression is a Regression algorithm.

- **Regression**:
  - The output targets $f(x) \in Y$ is continuous, or has a continuous component (e.g., stock price prediction).



- **Classification**:
  - The output targets $f(x) \in Y$ is one of a finite set of discrete categories (e.g., mail classification).



# Supervised Learning

- Machine Learning Task:
  - learn an (unknown) function $f: \boxed{X} \rightarrow \boxed{Y}$ that maps input instances x $\in X$ to output targets $f(x) \in Y$.
- Linear Regression is a Supervised Learning algorithm.
- Supervised Learning: The output targets are known in the set of training examples:
  - $(x_1, y_1), (x_2, y_2) \ldots (x_i, y_i)$ [e.g., predict stock price]
- Training data vs. Test data

| All Data | |
|---|---|
| Training Data | Test Data |

# Supervised Learning

- Training data vs. Test data

All Data

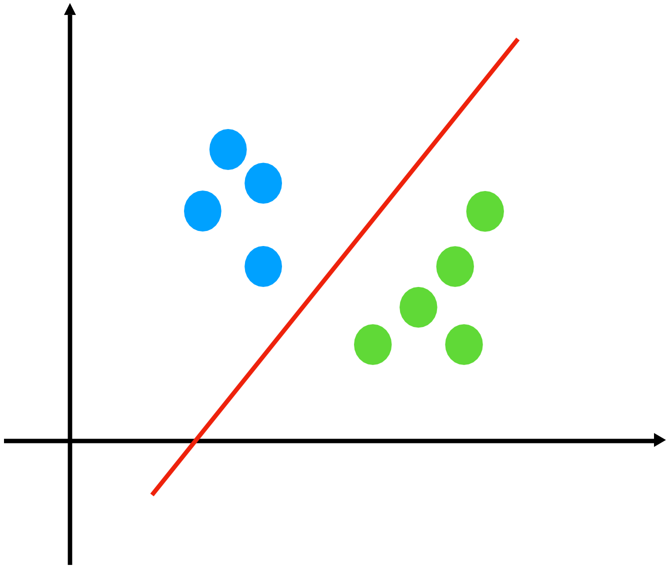
Training Data | Test Data

- The goal of machine learning algorithms is to build a function (hypothesis) $h(x)$ such that:
  - $h$ matches $f$ well on the training data → $h$ is able to fit data that it has see
  - $h$ also matches $f$ well on the test data → $h$ is able to generalize to unseen data
- To achieve the goal, we want to choose $h$ from a "nice" class of functions that depends on a vector of parameters w:
  - $h(x) \equiv h_w(x) \equiv h(w, x)$

# Do we need to make assumptions on the data?

- **No free lunch theorem:** we must make such assumptions.

- **Informal theorem:** for any machine learning algorithm $\mathcal{A}$, there must exist a task $\mathcal{P}$ on which it will fail

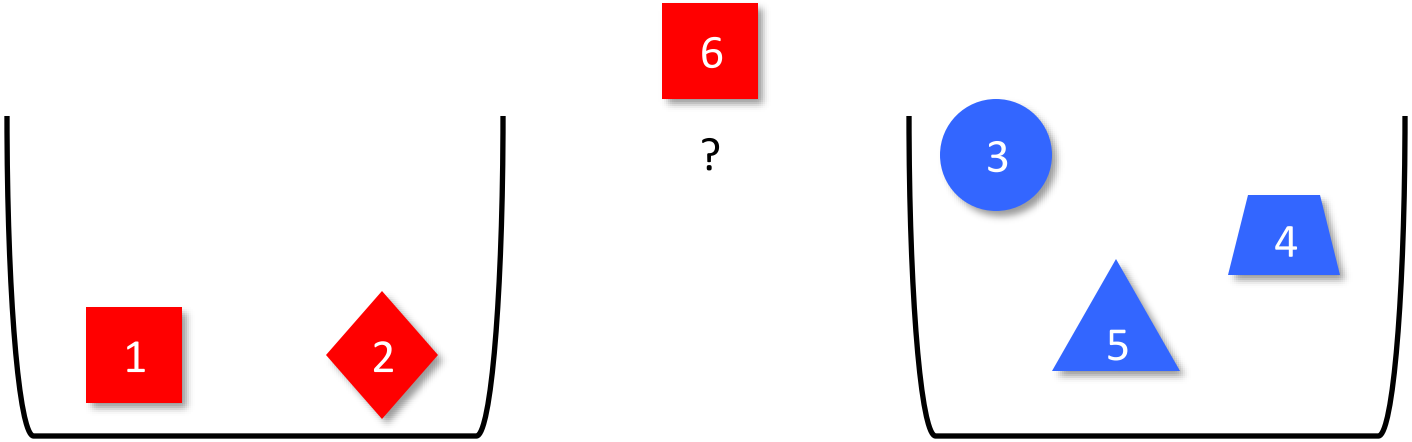


- We use prior knowledge (i.e., we believe linear function is enough) to design an ML algorithm here

# Which Hypothesis?

- Hypothesis class: $\mathcal{H} = \{h\}$
  - E.g.: linear models, quadratic models, neural networks, etc.

- An example:
  - $M_1$: $x$ is Red => $x \in C_1$
  - $M_2$: $x$ is a Square or $x$ is a Diamond => $x \in C_1$
  - $M_3$: $x$ is Red and $x$ is a Quadrilateral => $x \in C_1$




Container $C_1$ Container $C_2$

# Occam's Razor

**William of Occam** (1288 – 1348)

English Franciscan friar, theologian and philosopher.

"*Entia non sunt multiplicanda praeter necessitatem*"

- Entities must not be multiplied beyond necessity.

i.e. Do not make things needlessly complicated.

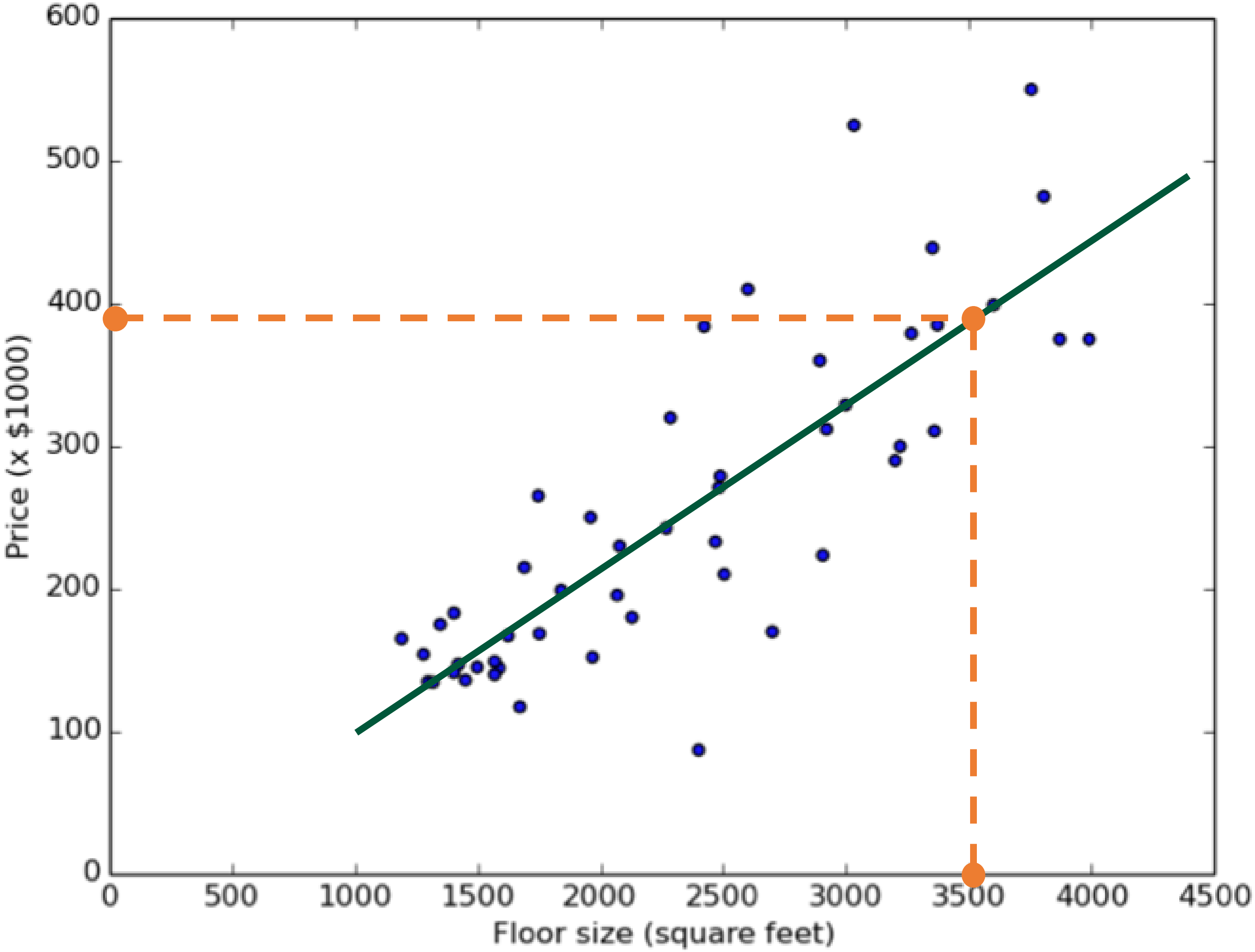
i.e. Prefer the simplest hypothesis that fits the data.

# House Price Prediction

- Given the floor size in square feet, predict the selling price:
  - Input $x$: the floor size of the house
  - Output $y$: the selling price of the house
  - Need to learn a function (hypothesis) $h$ such that $h(x) \approx f(x)$

- Is this a classification or regression task?
  - Regression, because the house price is real-valued.
  - Would a problem with only two labels $y_1 = 0.5$ and $y_2 = 1.0$ still be regression?

# House Price Prediction




- **Another hypothesis?**
- **Pros and cons of the hypothesis?**

# Linear Regression

- Use a linear function to approximate the real (unknown) function:
    - $h_{\mathbf{w}}(X) = \mathbf{w}^T X = [w_0, w_1]^T [1, x] = w_1 x + w_0$
    - In our case, $h_{\mathbf{w}}(X)$ is a straight line
        - $w_0$ is the intercept (or the bias term)
        - $w_1$ controls the slope
- Actually, the floor size of the house is not the only factor determining its sell price. There are many factors: $(x_1, \dots, x_d)$.
    - $$h_{\mathbf{w}}(X) = \mathbf{w}^T X = [w_0, w_1, \dots, w_d]^T [1, x_1, \dots, x_d]$$ $$= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$
- Assumption: the prediction results are the linear combination of input attributes (features).

# Error Measurement

- Our linear approximation function:
  - $h_{\mathbf{w}}(X) = \mathbf{w}^T X = [w_0, w_1, \dots, w_d]^T [1, x, \dots, x_d]$
    $= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- Error Measurement: Find **w** that obtains the best fit on the training data, i.e. find **w** that minimizes the sum of square errors:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(X_n) - y_n)^2$$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$$

  - $N$: Total number of samples in training set
  - $J(\mathbf{w})$: Error function
  - $\hat{\mathbf{w}}$: Optimimal $\boldsymbol{w}$ that minimize $J(\mathbf{w})$

- Why do we use the square errors?

# Inductive Learning Hypothesis

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \dots, w_d]$
  - Find $\mathbf{w}$ that minimizes an error function $J(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_i, \mathbf{w})$ and $t_i$.
  - Expect that $h(\mathbf{x}, \mathbf{w})$ performing well on training examples $\mathbf{x}_i \Longrightarrow h(\mathbf{x}, \mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x}_\mathrm{j} \in \mathbf{X}$.

**Inductive Learning Hypothesis**

# Matrix Notation

- **Linear Regression Learning Task**
  - learn $\mathbf{w}$ given training examples $< \mathbf{X}, \boldsymbol{y} >$.
  - The training data is denoted as $< \mathbf{X}, \boldsymbol{y} >$, where $\mathbf{X}$ is a $N \times D$ data matrix consisting of $N$ data examples such that each data example is a $D$ dimensional vector. $\boldsymbol{y}$ is a $N \times 1$ vector consisting of corresponding target values for the examples in $\mathbf{X}$.

- The derivation of the least squares estimate can be done by first converting the expression of the squared loss into matrix notation, i.e.,

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

# Analytical Solution

- To minimize the error, we first compute its derivative with respect to $\mathbf{w}$:

$$\frac{\partial LL(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2}\frac{\partial}{\partial \mathbf{w}}(\boldsymbol{y} - \mathbf{X}\mathbf{w})^T(\boldsymbol{y} - \mathbf{X}\mathbf{w})$$

- Note that, we use the fact that $(\mathbf{X}\mathbf{w})^T\boldsymbol{y} = \boldsymbol{y}^{\boldsymbol{T}}\mathbf{X}\mathbf{w}$, since both quantities are scalars, and the transpose of a scalar is equal to itself. Continuing with the derivative:

$$\frac{\partial LL(\mathbf{w})}{\partial \mathbf{w}} = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{2}\mathbf{w}^{\boldsymbol{T}}\mathbf{X}^{\boldsymbol{T}}\mathbf{X} - \mathbf{2}\boldsymbol{y}^{\boldsymbol{T}}\mathbf{X})$$

# Analytical Solution

- Setting the derivation to 0, we get:

$$\begin{aligned}
2\mathbf{w}^\top\mathbf{X}^\top\mathbf{X} - 2\mathbf{y}^\top\mathbf{X} &= 0 \\
\mathbf{w}^\top\mathbf{X}^\top\mathbf{X} &= \mathbf{y}^\top\mathbf{X} \\
(\mathbf{X}^\top\mathbf{X})^\top\mathbf{w} &= \mathbf{X}^\top\mathbf{y} \text{ (Taking transpose both sides)} \\
(\mathbf{X}^\top\mathbf{X})\mathbf{w} &= \mathbf{X}^\top\mathbf{y} \\
\mathbf{w} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}
\end{aligned}$$

The Moore-Penrose pseudo-inverse

# Summary

- Our linear approximation function:
  - $h_{\mathbf{w}}(X) = \mathbf{w}^T X = [w_0, w_1, \dots, w_d]^T [1, x, \dots, x_d]$
    $= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- Error Measurement: Find **w** that minimizes the sum of square errors:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_w(X_n) - y_n)^2$$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$$

- Analytical Solution:

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

# Machine Learning as Optimization

At this point, we move away from the situation where a perfect solution exists, and the learning task it to reach the perfect solution. Instead, we focus on finding the *best possible* solution which optimizes certain criterion.

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

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

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

# Convexity

*Convexity* is a property of certain functions which can be exploited by optimization algorithms. The idea of convexity can be understood by first considering *convex sets*. A convex set is a set of points in a coordinate space such that every point on the line segment joining any two points in the set are also within the set. Mathematically, this can be written as:

$$w_1, w_2 \in S \Rightarrow \lambda w_1 + (1-\lambda) w_2 \in S$$

where $\lambda \in [0,1]$. A *convex function* is defined as follows:

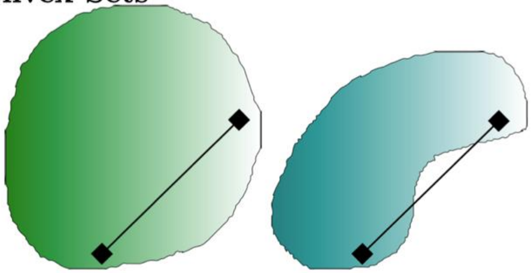
- $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function if the domain of $f$ is a convex set and for all $\lambda \in [0,1]$:

$$f(\lambda w_1 + (1-\lambda) w_2) \leq \lambda f(w_1) + (1-\lambda) f(w_2)$$

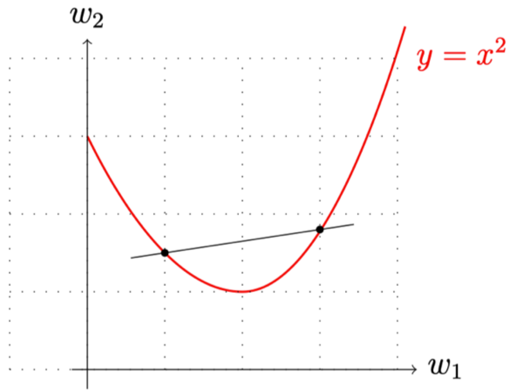
Some examples of convex functions are:

- Affine functions: $w^\top x + b$
- $||w||_p$ for $p \geq 1$
- Logistic loss: $\log(1 + e^{-y w^\top x})$

**Convex Sets**



**Convex Functions**



# Convex Optimization

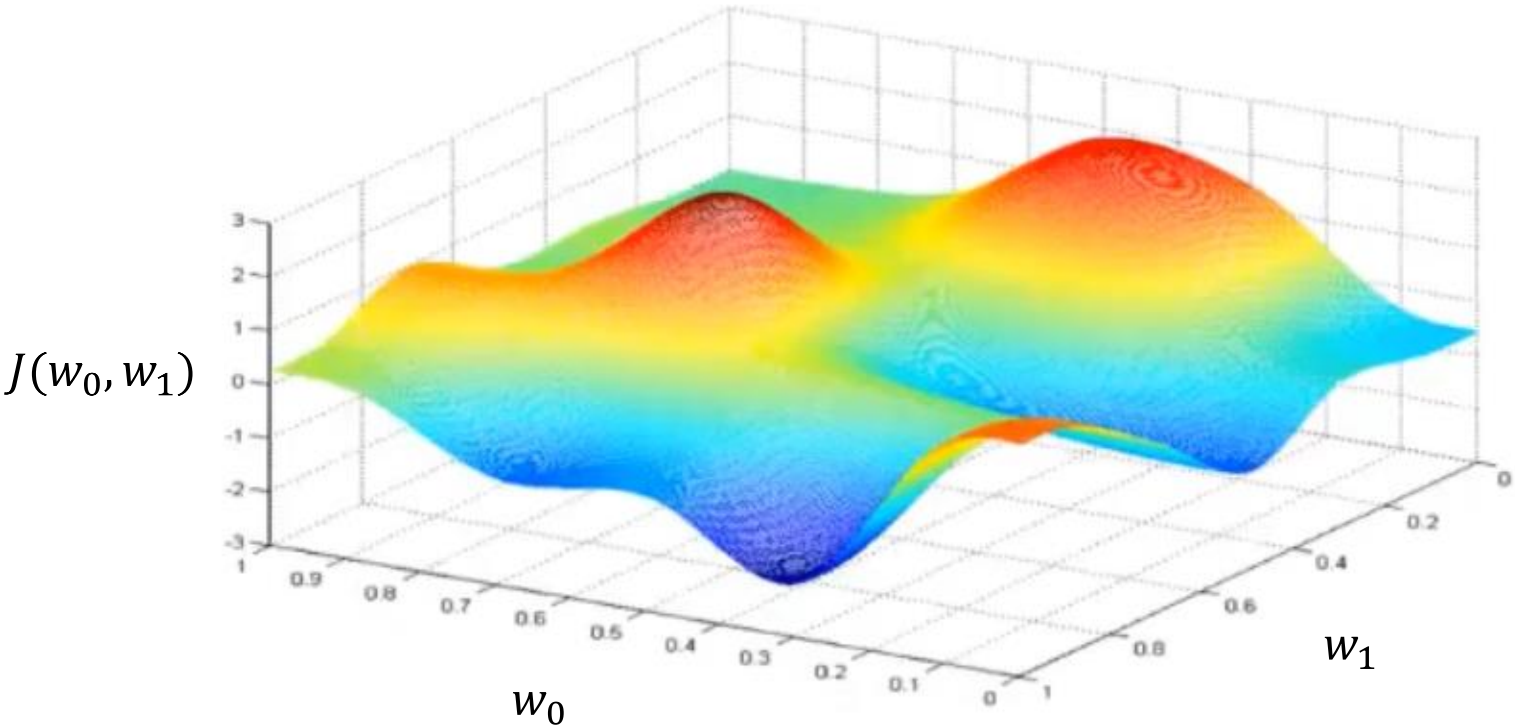
- Optimality Criterion

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

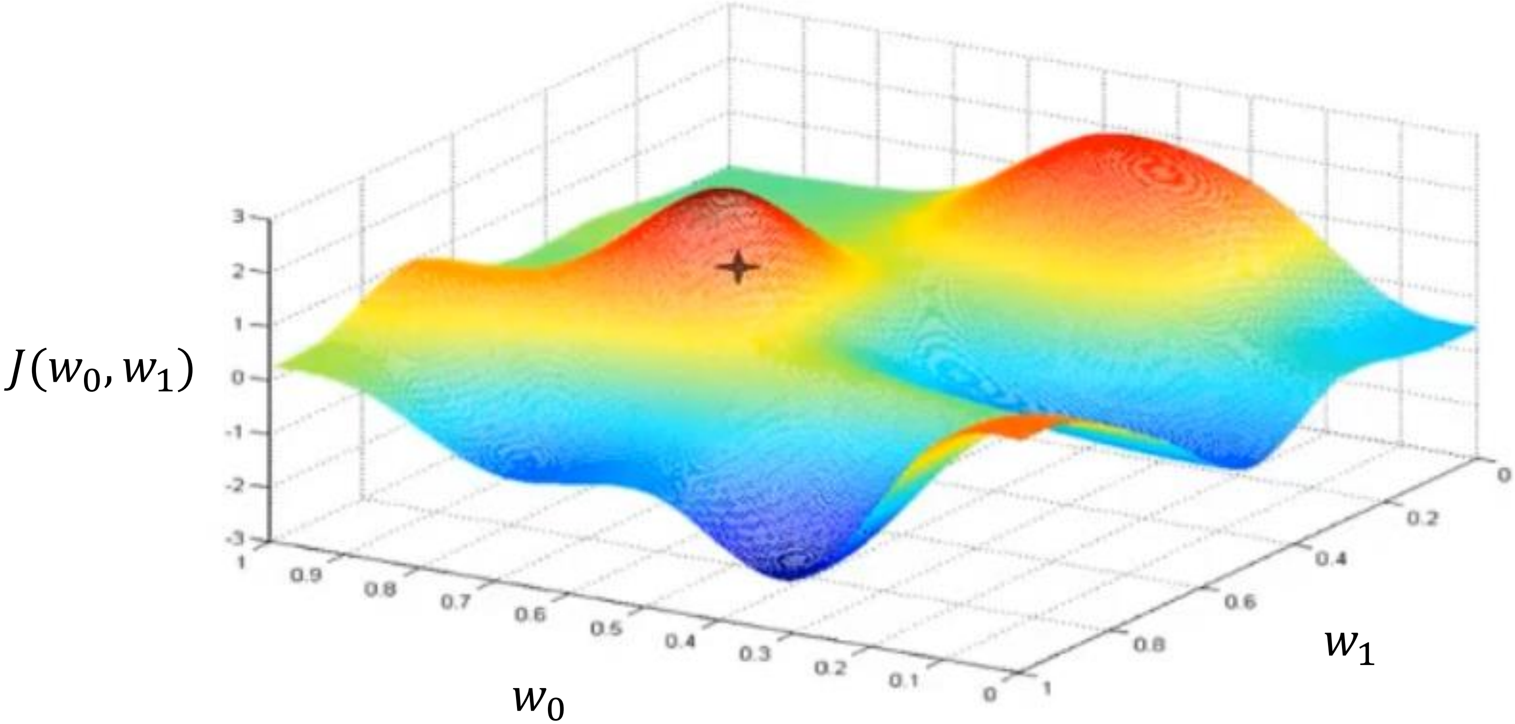
  where all $f_i(w)$ are **convex functions**.

- $w_0$ is feasible if $w_0 \in Dom\ f_0$ and all constraints are satisfied
- A feasible $w^*$ is optimal if $f_0(w^*) \leq f_0(w)$ for all $w$ satisfying the constraints

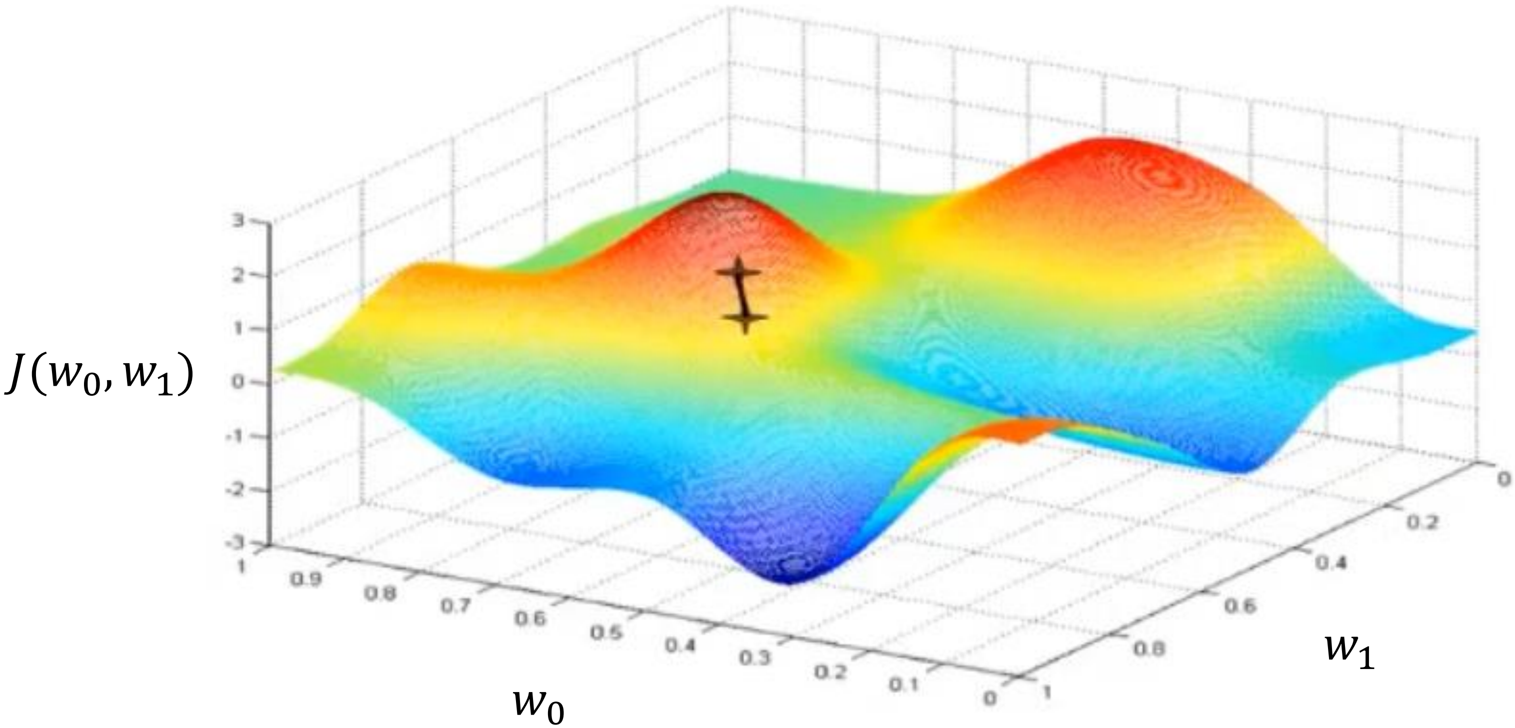
# Gradient Descent



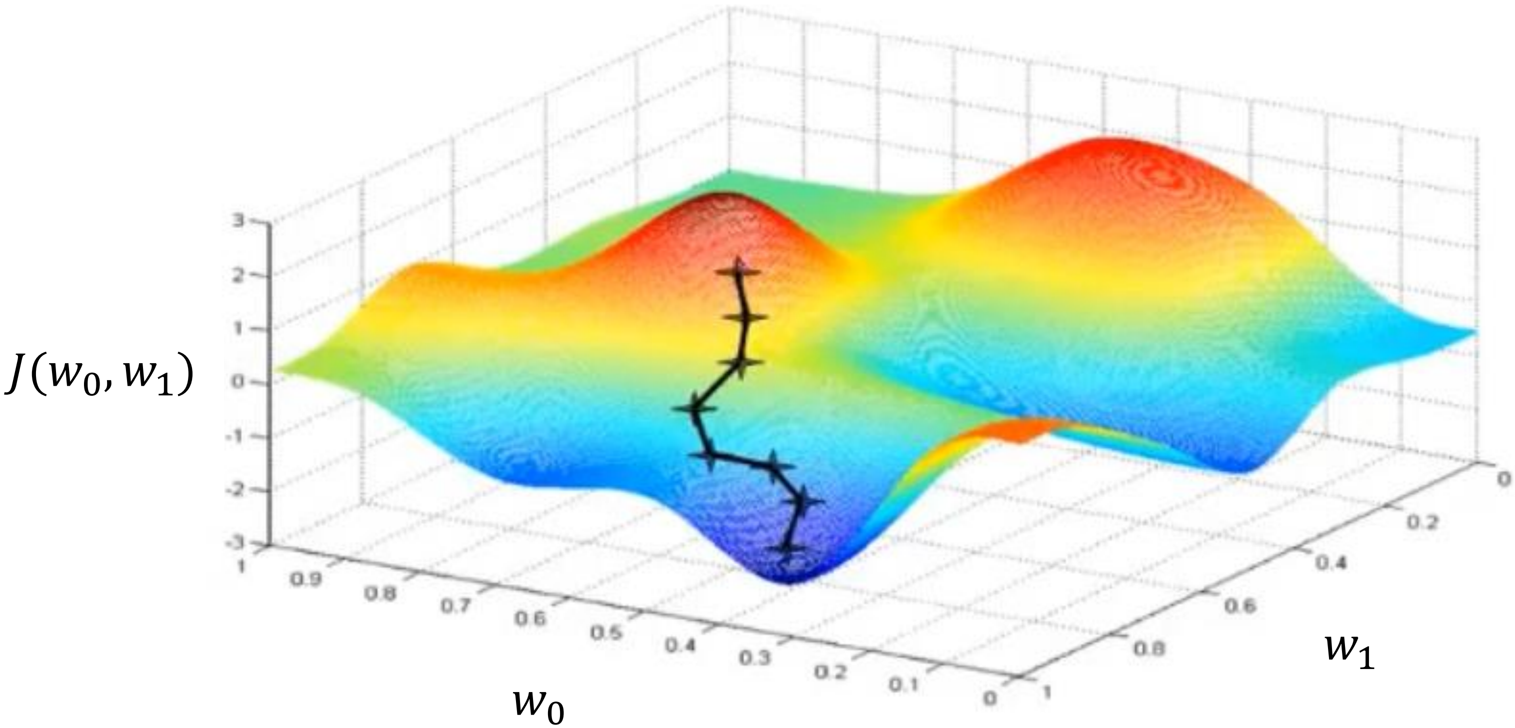
# Gradient Descent



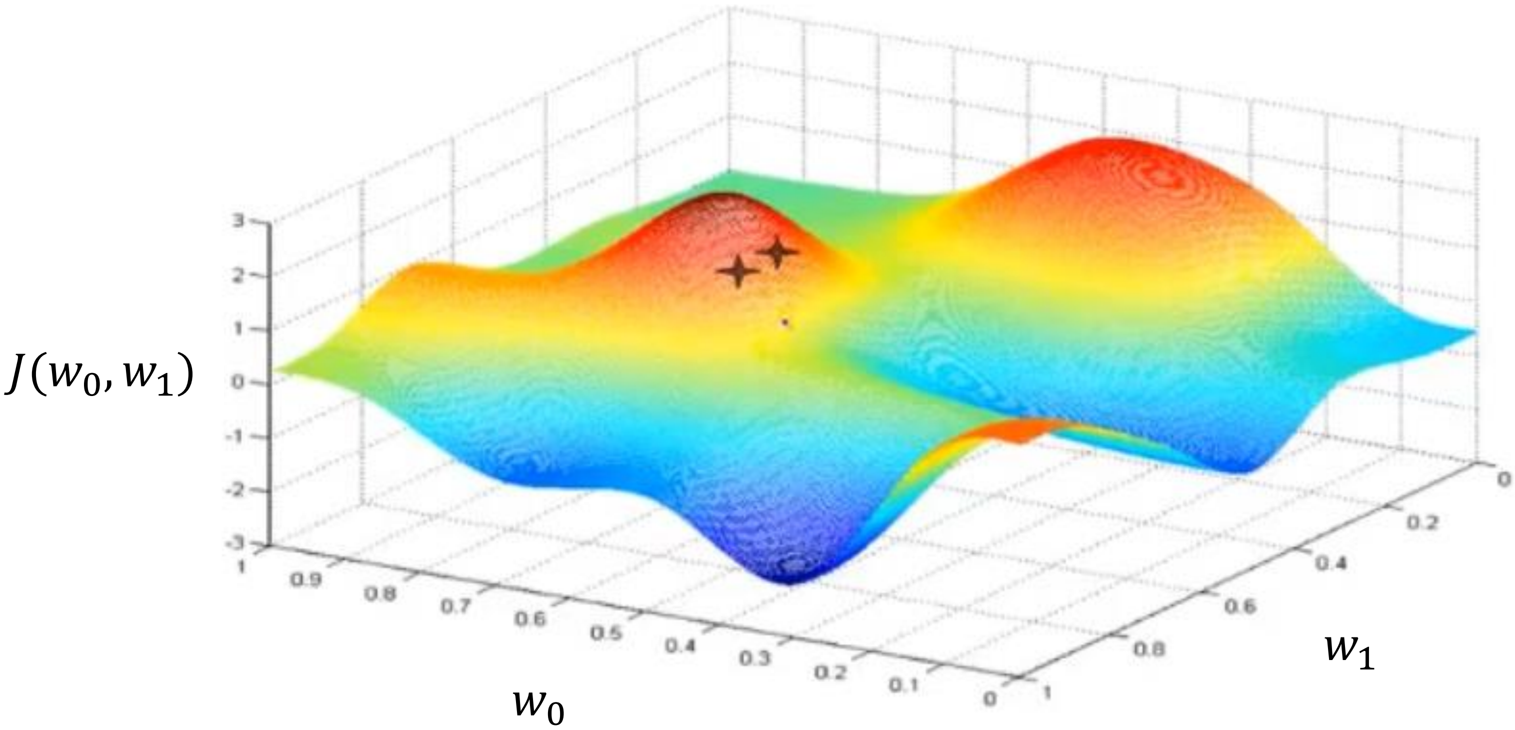
# Gradient Descent



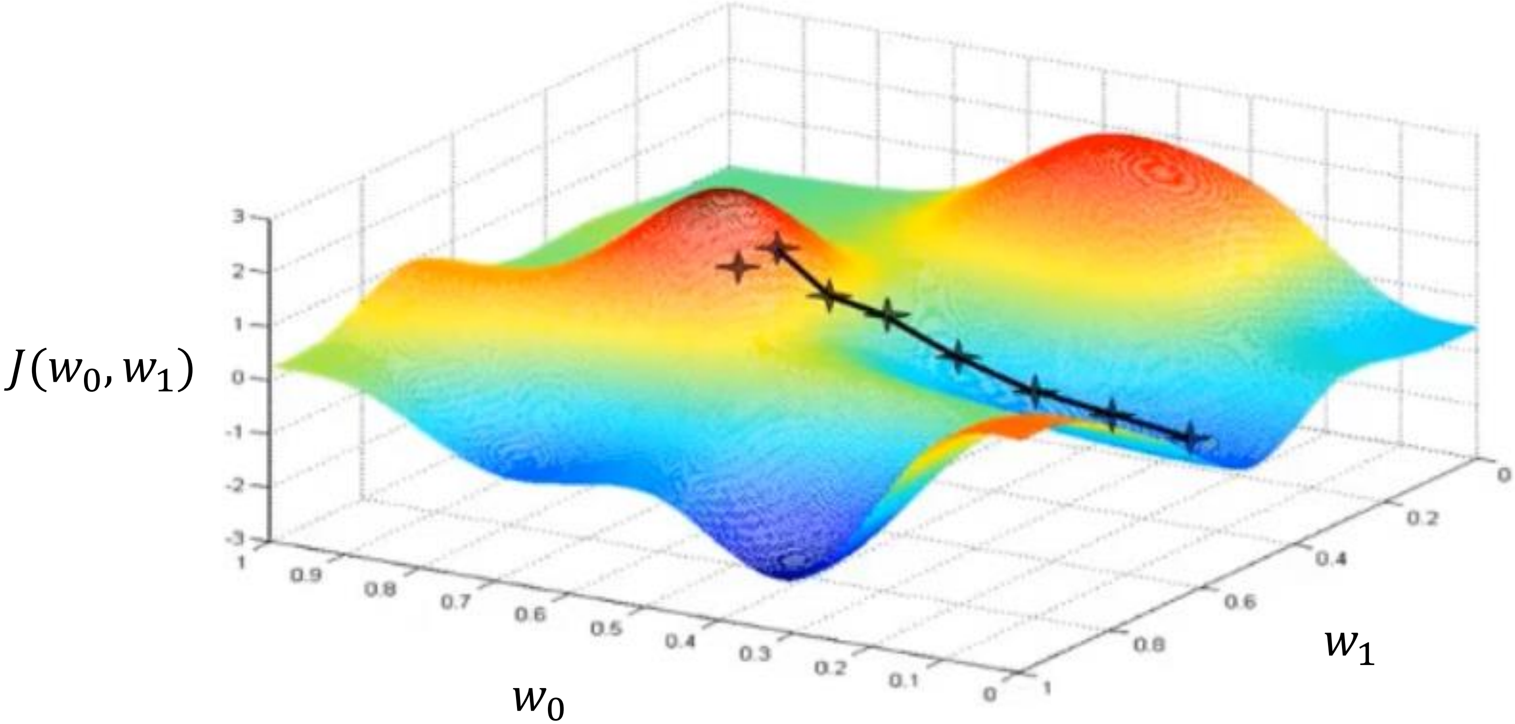
# Gradient Descent



# Gradient Descent



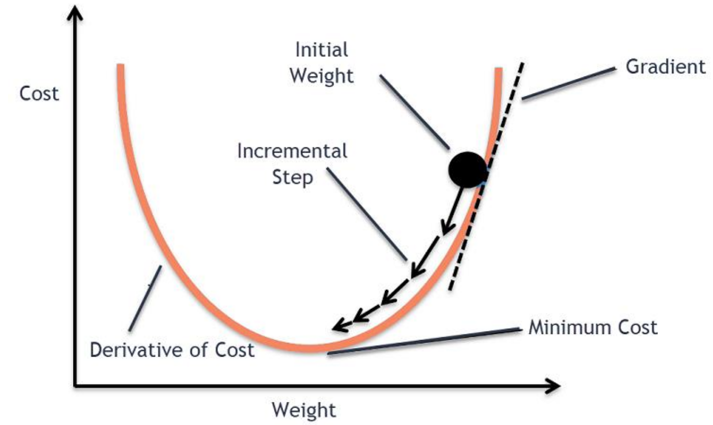
# Gradient Descent



# Gradient Descent

- Denotes the direction of steepest ascent

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




- A small step in the weight space from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$ changes the objective (or error) function. This change is maximum if $\delta\mathbf{w}$ is along the direction of the gradient at $\mathbf{w}$ and is given by:

$$\delta E \simeq \delta\mathbf{w}^\top \nabla E(\mathbf{w})$$

- Since $E(\mathbf{w})$ is a smooth continuous function of $\mathbf{w}$, the extreme values of $E$ will occur at the location in the input space ($\mathbf{w}$) where the gradient of the error function vanishes, such that:

$$\nabla E(\mathbf{w}) = 0$$

- The vanishing points can be further analyzed to identify them as saddle, minima, or maxima points.

# Taylor Expansion

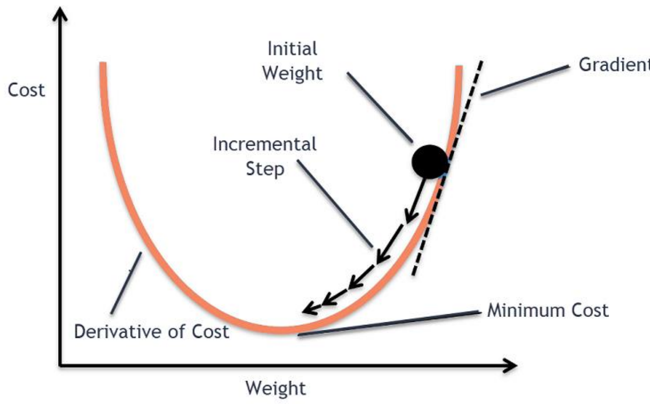
One can also derive the local approximations done by first order and second order methods using the Taylor expansion of $E(\mathbf{w})$ around some point $\mathbf{w}'$ in the weight space.

$$E(\mathbf{w}') \simeq E(\mathbf{w}) + (\mathbf{w}' - \mathbf{w})^\top \nabla + \frac{1}{2}(\mathbf{w}' - \mathbf{w})^\top \mathbf{H}(\mathbf{w}' - \mathbf{w})$$

For first order optimization methods, we ignore the second order derivative (denoted by $\mathbf{H}$ or the *Hessian*). It is easy to see that for $\mathbf{w}$ to be the local minimum, $E(\mathbf{w}) - E(\mathbf{w}') \leq 0$, $\forall \mathbf{w}'$ in the vicinity of $\mathbf{w}$. Since we can choose any arbitrary $\mathbf{w}'$, it means that every component of the gradient $\nabla$ must be zero.

# Gradient Decent

1. Start from any point in variable space
2. Move along the direction of the steepest descent
    - By how much?
    - A learning rate ($\eta$)
    - What is the direction of steepest descent?




- Gradient descent is a first-order optimization method for convex optimization problems. It is analogous to “hill-climbing” where the gradient indicates the direction of steepest ascent in the local sense.

# Gradient Decent

- **Training Rule for Gradient Descent**

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

- For each weight component:

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

  - The key operation in the above update step is the calculation of each partial derivative.

$$\begin{aligned} \frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= \frac{1}{2} \sum_i 2(y_i - \mathbf{w}^\top \mathbf{x}_i) \frac{\partial}{\partial w_j} (y_i - \mathbf{w}^\top \mathbf{x}_i) \\ &= \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)(-x_{ij}) \end{aligned}$$

  - where $x_{ij}$ denotes the $j^{th}$ attribute value for the $i^{th}$ training example.

# Gradient Decent

- The final weight update rule:

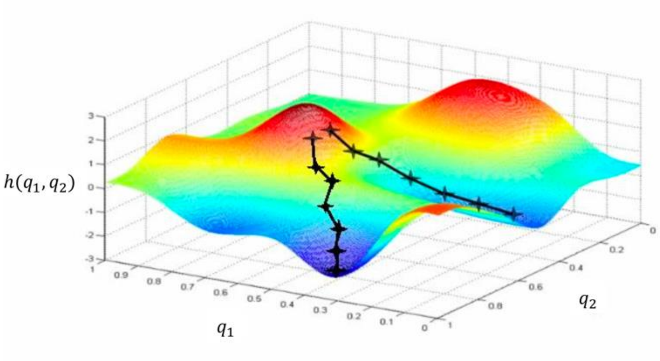
$$w_j = w_j + \eta \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i) x_{ij}$$

- 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 Decent

Non-convex Example

- Issues with Gradient Decent:
  - Slow convergence
  - Stuck in local minima




- One should note that the second issue will not arise in the case of convex problem as the error surface has only one global minima.

- More efficient algorithms exist for batch optimization, including Conjugate Gradient Descent and other quasi-Newton methods. Another approach is to consider training examples in an online or incremental fashion, resulting in an online algorithm called Stochastic Gradient Descent.

# Stochastic Gradient Descent (SGD)

- **Update weights after every (or a small subset of) training example(s).**
- 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 |

- Why SGD?

The analytical approach discussed earlier involves a matrix inversion ($(\mathbf{X}^\top \mathbf{X})^{-1}$) which is a $(D+1) \times (D+1)$ matrix. Alternatively, one could solve a system of equations. When $D$ is large, this inversion can be computationally expensive ($O * D^3$) for standard matrix inversion. Moreover, often, the linear system might have singularities and inversion or solving the system of equations might yield numerically unstable results.

# Stochastic Gradient Descent (SGD)

To compute the gradient update rule one can differentiate the error with respect to each entry of $\mathbf{w}$.

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

1 or K (a small number)

$$= \sum_{i=1}^{\cancel{N}}(\mathbf{w}^\top \mathbf{x}_i - y_i)x_{ij}$$

1 or K (a small number)

Using the above result, one can perform repeated updates of the weights:

$$w_j := w_j - \eta\frac{\partial J(\mathbf{w})}{\partial w_j}$$

# Questions?