6. Linear Regression
STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

Mar 19, 2024

Recap: Policy Summary

- 5 Assignments (**65**), Midterm (**25**), Attendance (**10**)
  - No final exam
- Attendance: [-1] for each absent; no late check
  - **Option 1 (Default)**: [3 free absences] without any report
  - **Option 2**: [no free absence]; docs to make up your score
- Assignment: [-25%] additive for each late day (not counting seconds)
  - **Option 1 (Default)**: Submit your own work
  - **Option 2**: Refer to others' solution/code and get [x70%]
- Study group size [≤ 5]
- Intuitively, we do not want to spend time for any non-academic issue

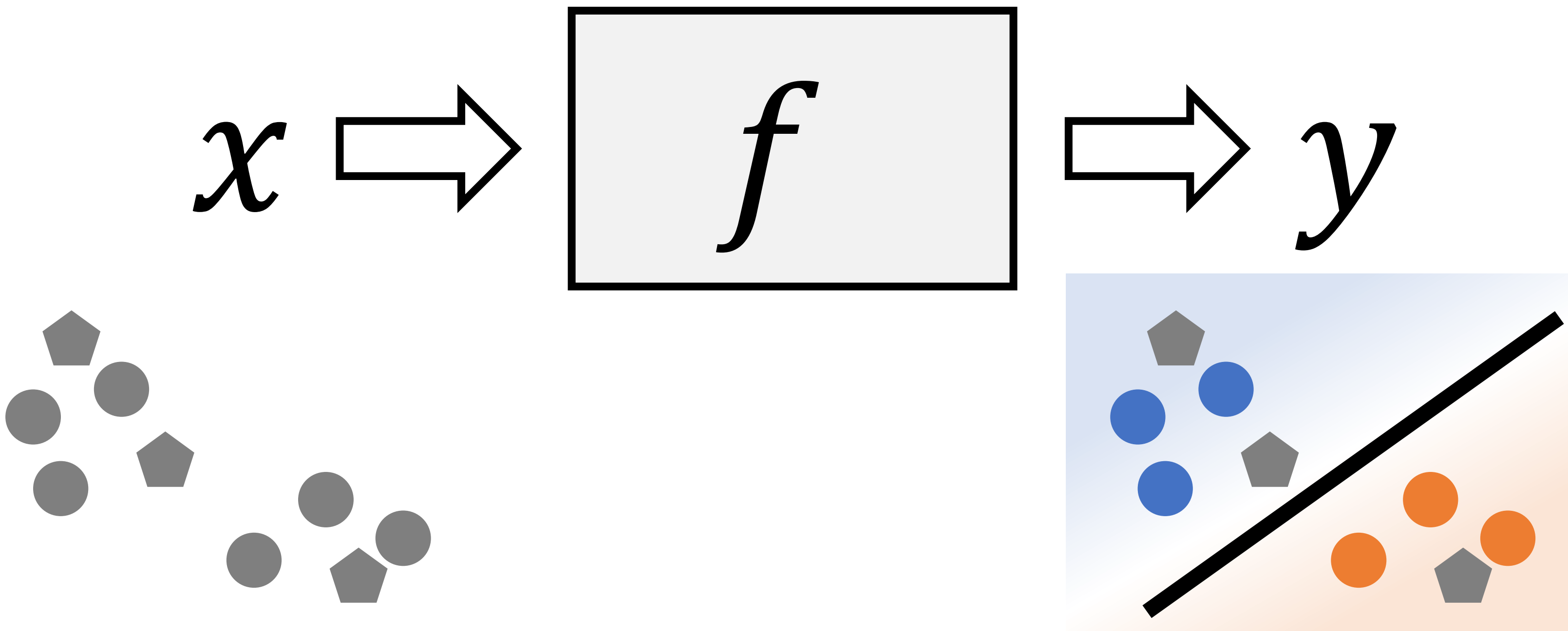
Assignment 1

- Due **Friday 3/29, 11:59pm**
- Topics
  - (Programming) NumPy basics
  - (Programming) Linear regression on a polynomial
  - (Math) Derivation and proof for linear regression

- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take **option 2**.

Recap: Machine Learning

- Learning a model $f: x \to y$ with training data $\{(x_i, y_i)\}_{i=1}^{N}$ s.t. it generalizes well on unseen data
  - Training data:
  - Labels: or
  - Test data:



Recap: Machine Learning Tasks

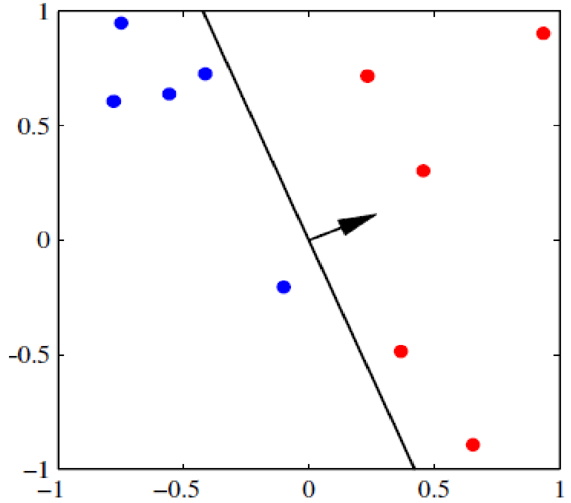
- **Supervised Learning**
  - Classification
  - Regression
- **Unsupervised Learning**
  - Clustering
  - Density estimation
  - Embedding / Dimensionality reduction
- **Reinforcement Learning**
  - Learning to act
    (e.g., robot control, decision making, etc.)

Supervised Learning

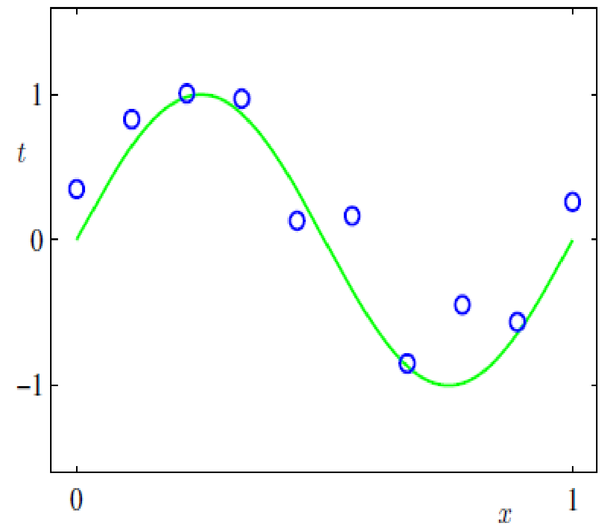
- Given a dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where
  - $x_i \in \mathcal{X}$: input (feature)
  - $y_i \in \mathcal{Y}$: output (label)
- A black box ML algorithm produces a prediction function $h: \mathcal{X} \to \mathcal{Y}$, such that $h(x)$ can predict the y values for all x
  - Not only for all training data $x_i \in D$,
    but also for unseen test data $x^* \in \mathcal{X}$.
- Labels could be discrete or continuous
  - Discrete labels: **classification**
  - Continuous labels: **regression**

Supervised Learning

- Learning a function $h: \mathcal{X} \to \mathcal{Y}$
- Labels could be discrete or continuous
  - Discrete labels: **classification**
  - Continuous labels: **regression** (today's topic)



classification



regression

Outline

- Linear Regression
  - Basis Functions
  - Objective Function: Least Squares
  - Gradient
- Optimization Methods
  - Gradient Descent
    - Batch Gradient Descent
    - Stochastic Gradient Descent
  - Closed-Form Solution
  - Newton’s Method

Linear Regression

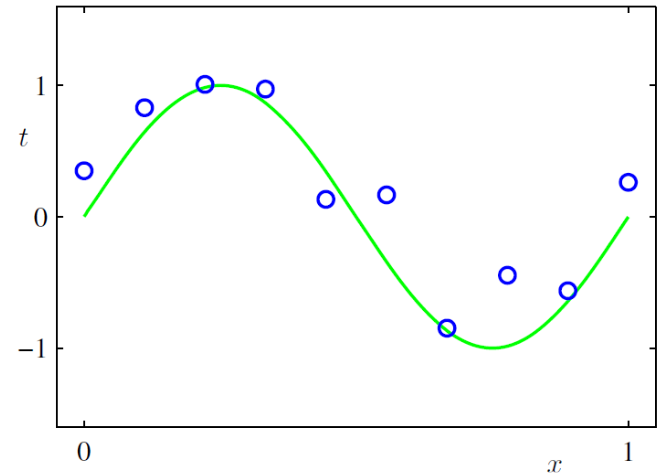
Notations

In this lecture, we will use the following notation:

- $\mathbf{x} \in \mathbb{R}^D$: data (scalar or vector)
- $\phi(\mathbf{x}) \in \mathbb{R}^M$: features for $\mathbf{x}$ (vector)
- $\phi_j(\mathbf{x}) \in \mathbb{R}$: j-th feature for $\mathbf{x}$ (scalar)
- $y \in \mathbb{R}$: continuous-valued label (i.e., target value)
- $\mathbf{x}^{(n)}$: denotes the n-th training example.
- $y^{(n)}$: denotes the n-th training label.

Linear regression (with 1d inputs)

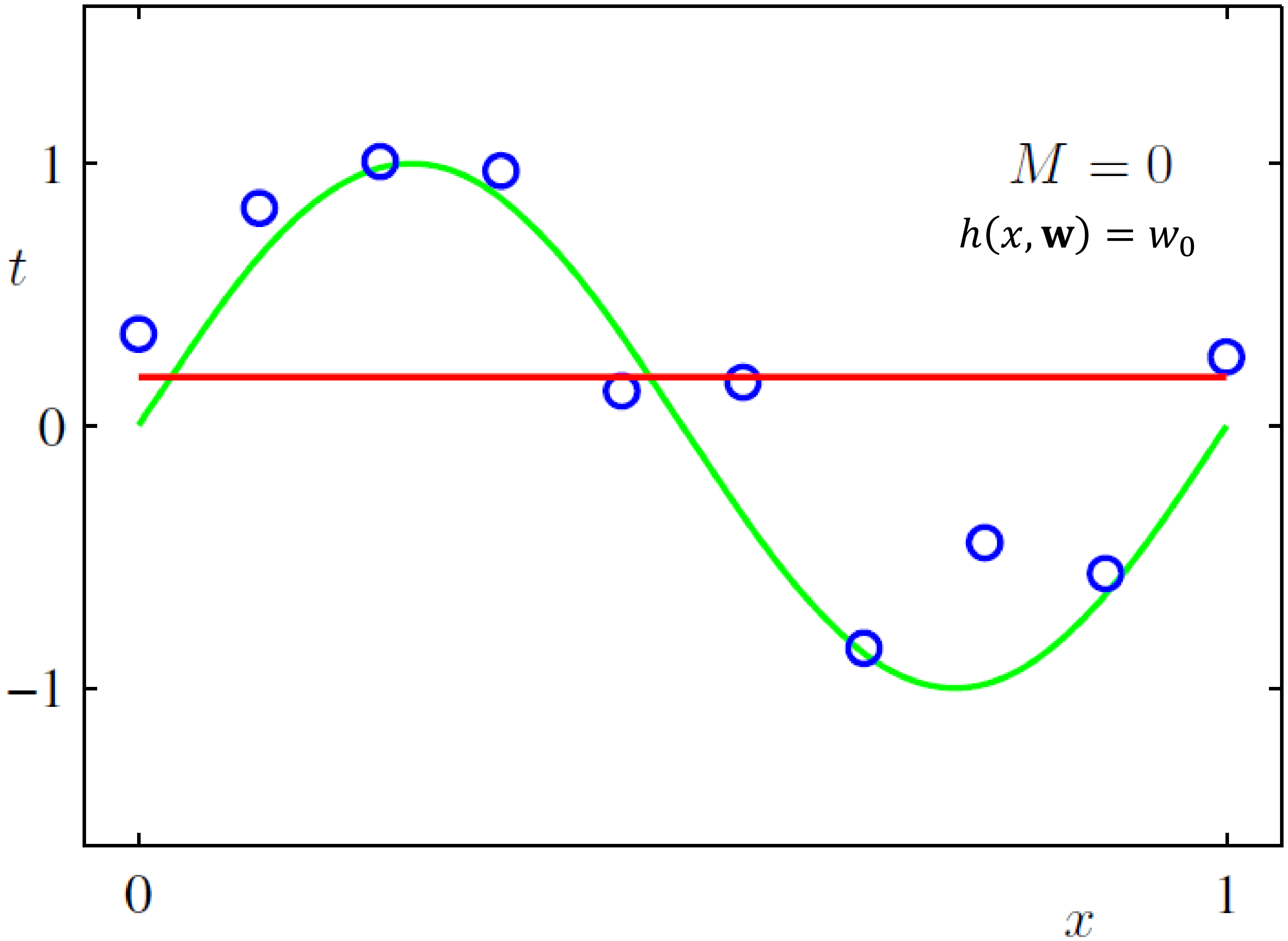
- Consider 1d case: $x, y \in \mathbb{R}$
- Given a set of observations: $\{x^{(1)}, \dots, x^{(N)}\}$
- And corresponding target values: $\{y^{(1)}, \dots, y^{(N)}\}$



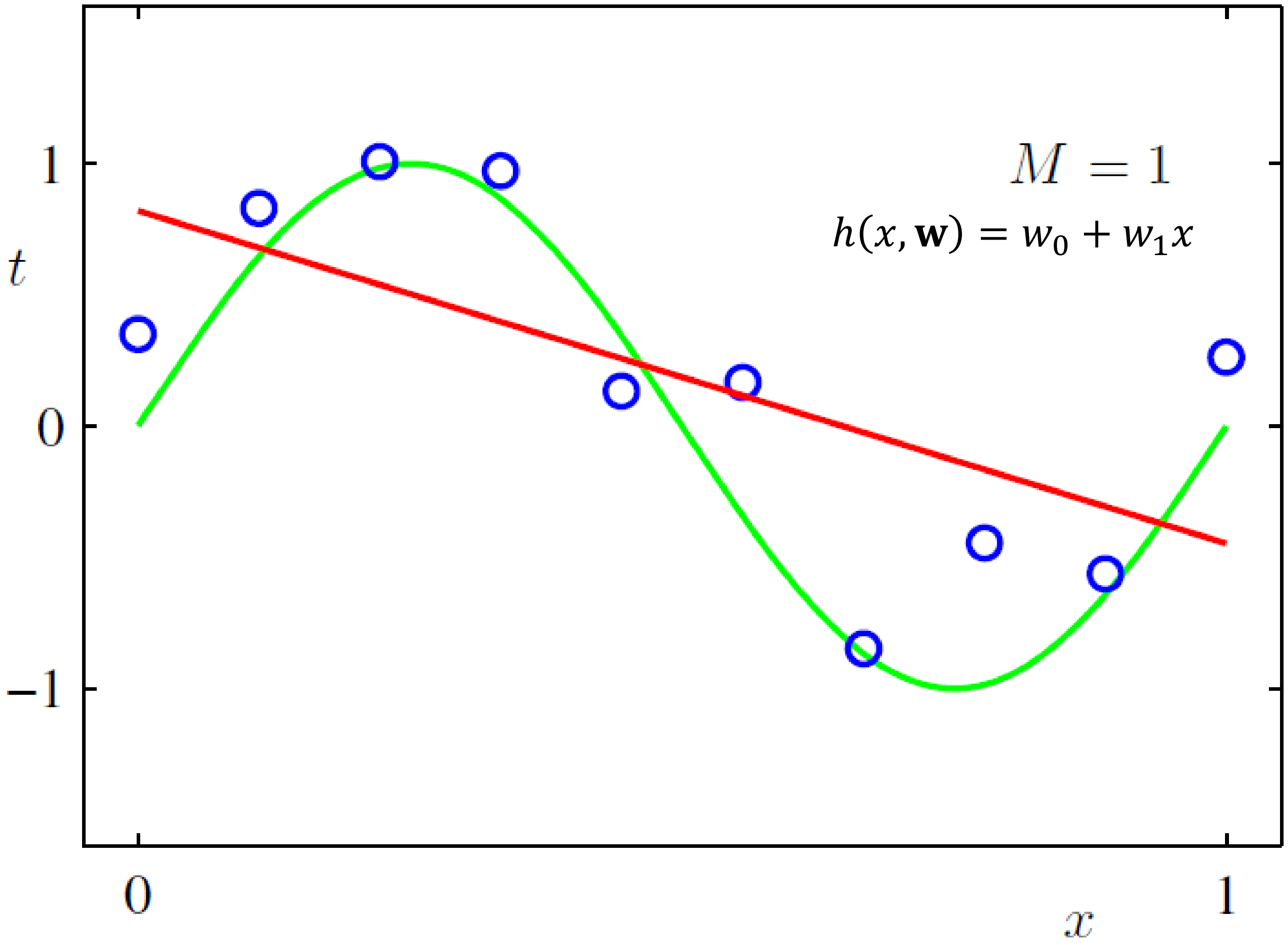
- We want to learn a function $h(x, \mathbf{w}) \simeq y$ to predict future values using a polynomial basis functions.

$$h(x, \mathbf{w}) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

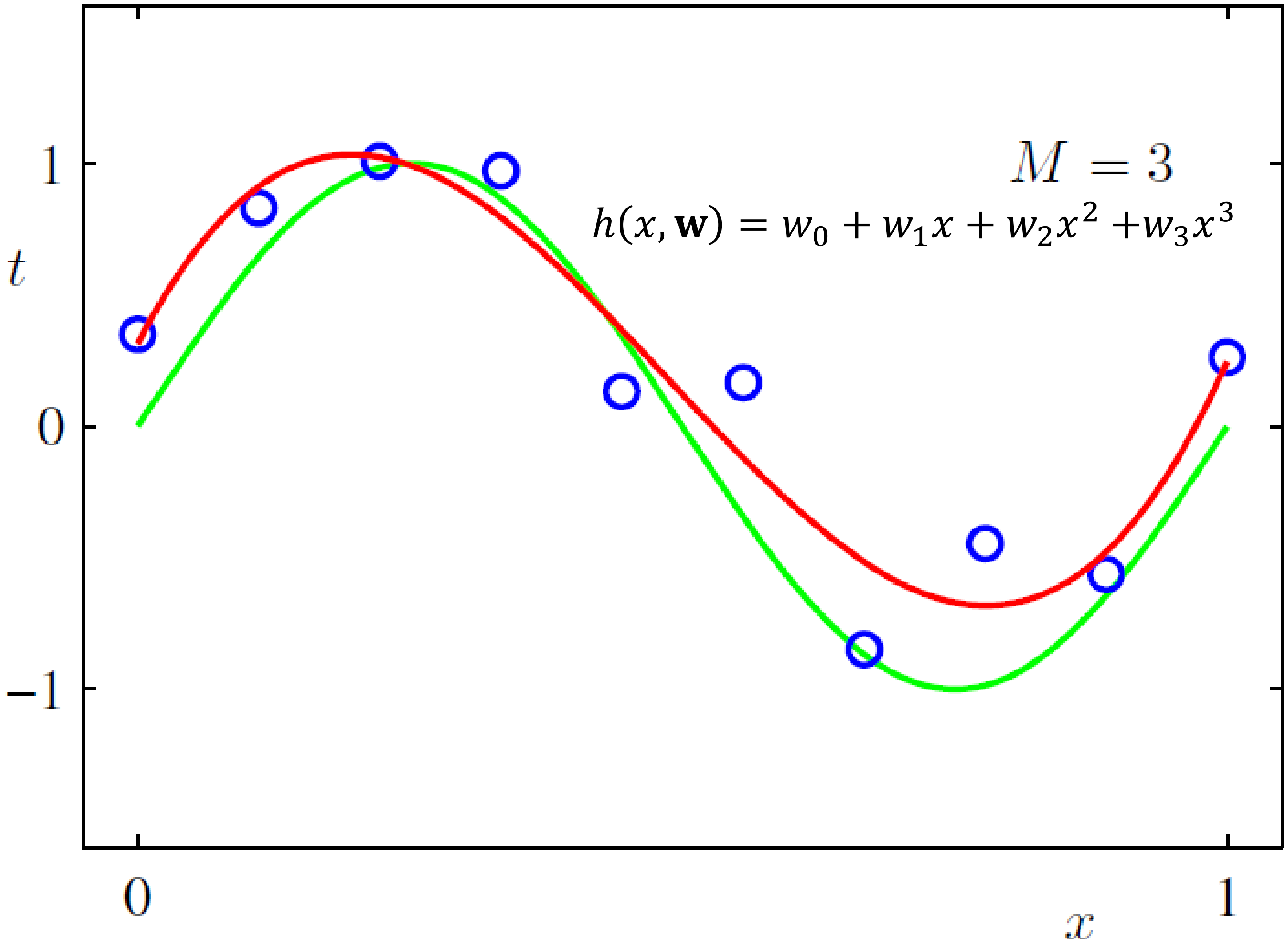
0^{th} Order Polynomial



1^{st} Order Polynomial



3rd Order Polynomial



Linear Regression (general case)

- The function $h(\mathbf{x}, \mathbf{w})$ with arbitrary kernels ϕ:

$$h(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

- The function $h(\mathbf{x}, \mathbf{w})$ is linear w.r.t. $\mathbf{w}$.
  - Goal: find the best values for the parameters/weights $\mathbf{w}$.
- For simplicity, add a bias term (a.k.a. bias trick)

The upper limit is reduced to $M-1$ to make the length of ϕ to be M for simplicity; don't be confused

$$h(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{\boxed{M-1}} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

where $\mathbf{w} = [w_0, \dots, w_{M-1}]^T$,

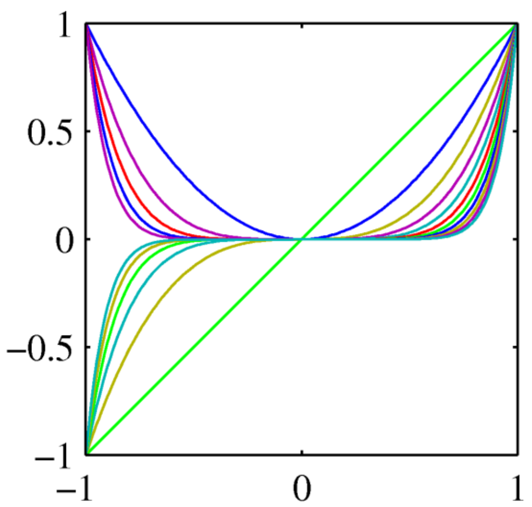
$$\phi(\mathbf{x}) = [\phi_o(\mathbf{x}), \dots, \phi_{M-1}(\mathbf{x})]^T, \phi_o(\mathbf{x}) = 1$$

Basis Functions

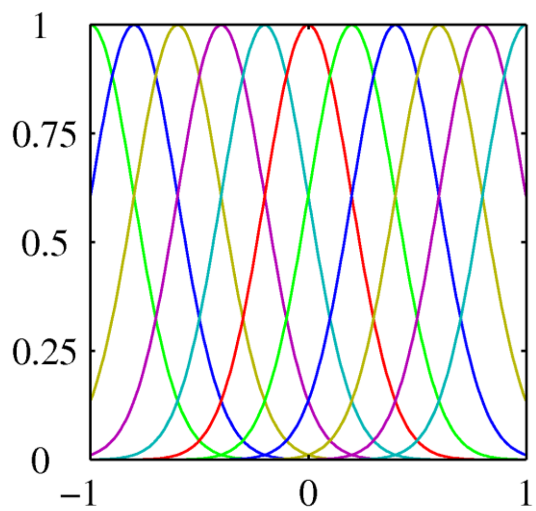
- Basis functions do not have to be linear

$$\phi_j(x) = x^j \qquad \phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\} \qquad \phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

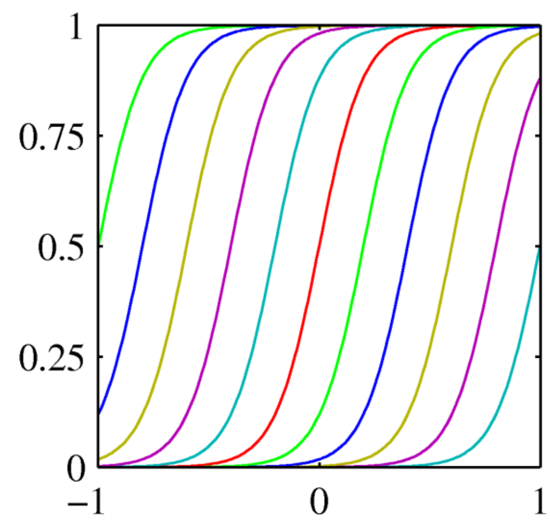
$$\sigma(a) = \frac{1}{1+\exp(-a)}$$



Polynomial

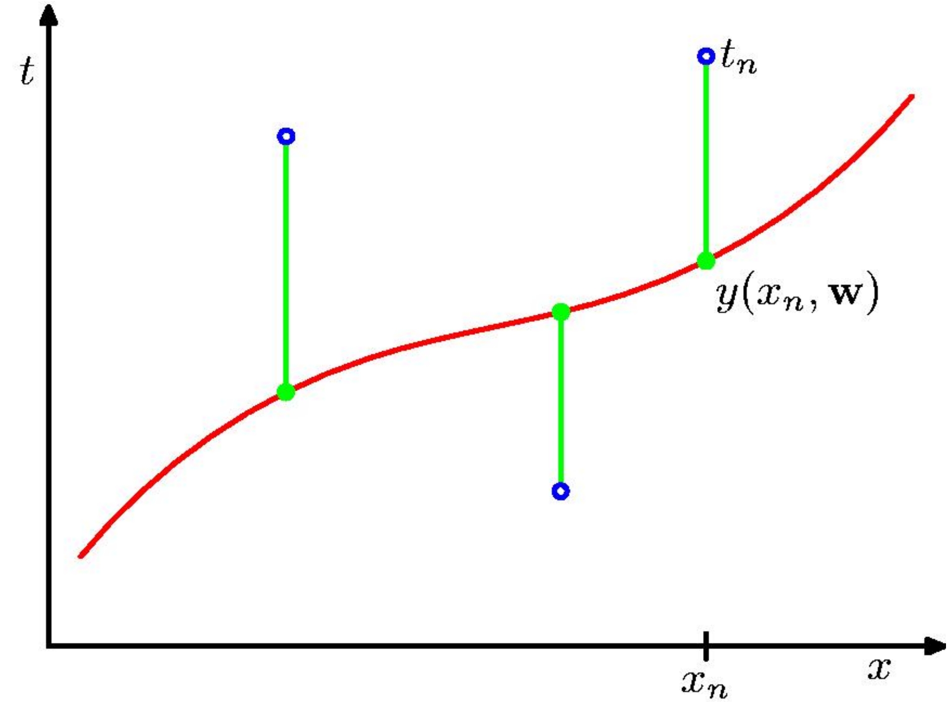


Gaussian



Sigmoid

Objective: Sum-of-Squares Error



$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{h(\mathbf{x}^{(n)}, \mathbf{w}) - y^{(n)}\right\}^2$$

- We want to find $\mathbf{w}$ that minimizes $E(\mathbf{w})$ over the training data.

Least Squares Problem

- Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- Gradient

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

Least Squares Problem

- Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- Gradient

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$\frac{\partial}{\partial a}[f(a)]^2 = 2f(a)f'(a)$$

$$= \sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})\frac{\partial}{\partial w_j}(\sum_{j'=0}^{M-1} w_{j'}\phi_{j'}(\mathbf{x}^{(n)}) - y^{(n)})$$

Not to conflict with j in derivative

Least Squares Problem

- Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- Gradient

$$\begin{aligned}\frac{\partial E(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2 \\ &= \sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})\frac{\partial}{\partial w_j}(\sum_{j'=0}^{M-1} w_{j'}\phi_{j'}(\mathbf{x}^{(n)}) - y^{(n)}) \\ &= \sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})\phi_j(\mathbf{x}^{(n)})\end{aligned}$$

$$\frac{\partial w_{j'}}{\partial w_j} = \begin{cases} 1 \text{ if } j = j' \\ 0 \text{ otherwise}\end{cases}$$

Concatenate each component of the gradient:

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N} \left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi_j(\mathbf{x}^{(n)})$$

We get a vectorized form of the gradient:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} E(\mathbf{w}) \\ \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} E(\mathbf{w}) \end{bmatrix} \qquad \phi(\mathbf{x}^{(n)}) = \begin{bmatrix} \phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)}) \end{bmatrix}$$

Concatenate each component of the gradient:

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)}) \phi_j(\mathbf{x}^{(n)})$$

We get a vectorized form of the gradient:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \qquad\qquad \phi(\mathbf{x}^{(n)}) =$$

$$\begin{bmatrix} \frac{\partial}{\partial w_0} E(\mathbf{w}) \\ \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} E(\mathbf{w}) \end{bmatrix} = \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)}) \begin{bmatrix} \phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)}) \end{bmatrix}$$

Concatenate each component of the gradient:

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N} \Big(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \Big) \phi_j(\mathbf{x}^{(n)})$$

We get a vectorized form of the gradient:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} E(\mathbf{w}) \\ \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} E(\mathbf{w}) \end{bmatrix} = \sum_{n=1}^{N} \Big(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \Big) \begin{bmatrix} \phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)}) \end{bmatrix}$$

$$\phi(\mathbf{x}^{(n)}) =$$

Vectorization

$$= \sum_{n=1}^{N} \Big(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \Big) \phi(\mathbf{x}^{(n)})$$

Concatenate each component of the gradient:

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)}) \phi_j(\mathbf{x}^{(n)})$$

We get a vectorized form of the gradient:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \qquad\qquad \phi(\mathbf{x}^{(n)}) =$$

$$\begin{bmatrix} \frac{\partial}{\partial w_0} E(\mathbf{w}) \\ \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} E(\mathbf{w}) \end{bmatrix} = \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)}) \begin{bmatrix} \phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)}) \end{bmatrix}$$

$$= \sum_{n=1}^{N} (\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)}) \phi(\mathbf{x}^{(n)})$$

$$= \sum_{n=1}^{N} (\mathbf{w}^T \phi(\mathbf{x}^{(n)}) - y^{(n)}) \phi(\mathbf{x}^{(n)})$$

Vectorization

Gradient (Vectorized Form)

- In summary, we have:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N} \left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

$$= \sum_{n=1}^{N} \left(\mathbf{w}^T \phi(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} E(\mathbf{w}) \\ \frac{\partial}{\partial w_1} E(\mathbf{w}) \\ \vdots \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} E(\mathbf{w}) \end{bmatrix} \qquad \phi(\mathbf{x}^{(n)}) = \begin{bmatrix} \phi_0(\mathbf{x}^{(n)}) \\ \phi_1(\mathbf{x}^{(n)}) \\ \vdots \\ \vdots \\ \phi_{M-1}(\mathbf{x}^{(n)}) \end{bmatrix}$$

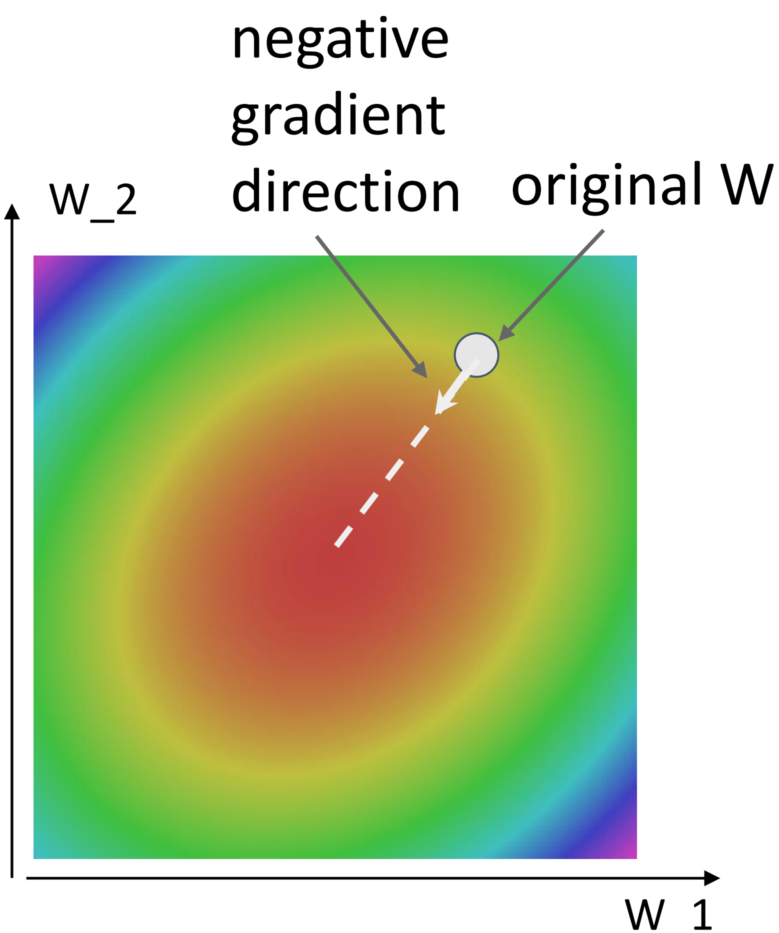
Optimization Methods

Gradient Descent

- Iteratively step in the direction of the negative gradient (direction of local steepest descent)

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```

- Hyperparameters:
  - Weight initialization method
    - w = 0 is fine for now
      (not for non-convex optim)
  - Number of steps
    - Long enough
  - Learning rate
    - You need to tune this



Batch Gradient Descent (BGD)

- Given data $(\mathbf{x}, y)$, initial $\mathbf{w}$
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \boxed{\eta} \nabla_{\mathbf{w}} E(\mathbf{w})$$

Learning rate

where

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N} \Big(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \Big) \phi(\mathbf{x}^{(n)})$$

$$= \sum_{n=1}^{N} (\mathbf{w}^T \phi(\mathbf{x}^{(n)}) - y^{(n)}) \phi(\mathbf{x}^{(n)})$$

Stochastic Gradient Descent (SGD)

- Compute the gradient for an individual example instead of the **entire** training data

- Repeat until convergence
  - For $n = 1, \dots, N$,

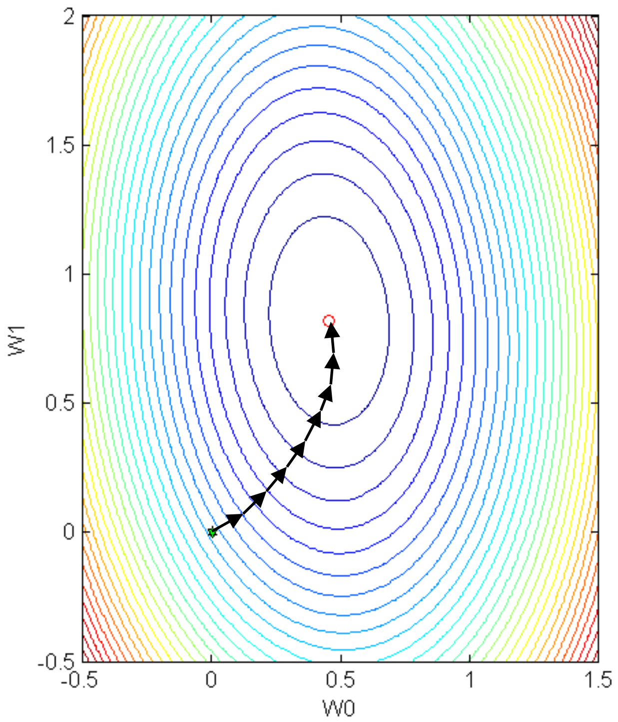
$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)})$$

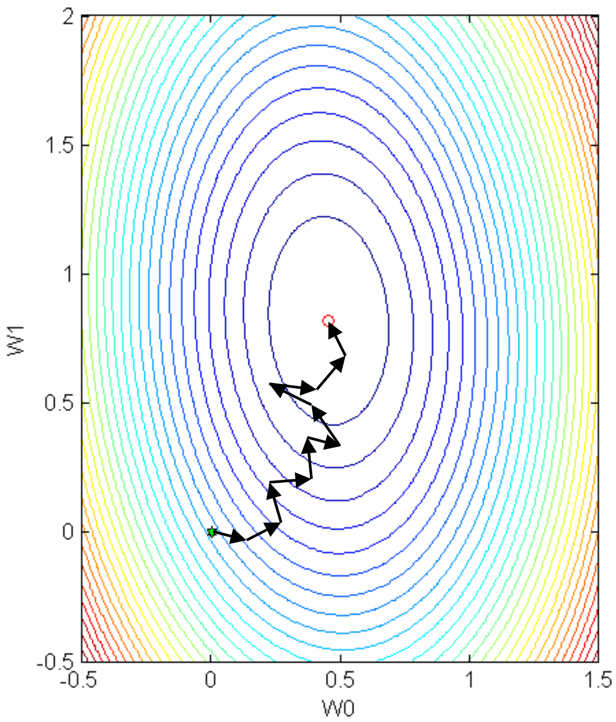
where

$$\nabla_{\mathbf{w}} E(\mathbf{w}|\mathbf{x}^{(n)}) = \left(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

$$= \left(\mathbf{w}^T \phi(\mathbf{x}^{(n)}) - y^{(n)} \right) \phi(\mathbf{x}^{(n)})$$

BGD vs. SGD





BGD vs. SGD

- **Batch gradient**
  - Stable or accurate direction
  - Slow update

- **Stochastic gradient**
  - Noisy or inaccurate direction
  - Fast update
  - Additional hyperparameter: **data sampling**
    - For A1, retrieving in the index order is fine.

Cf. Mini-Batch Gradient Descent

- Subsample and compute the batch gradient with $N_s \subseteq N$ samples
  - Moderately stable direction
  - Moderately fast update
  - Additional hyperparameter: **batch size**
    - Power of 2 is common (32, 64, 128, …)

- = Stochastic gradient descent (SGD) in some context (especially in deep learning)

Aside: Uniform Sampling

- Commonly used for stochastic learning
  - Epoch: the number of times retrieving the dataset
  - Iteration: the number of times retrieving the batch

```
# bsz: batch size
# num_iters = num_epochs*(num_data // bsz)
for h in range(num_epochs):
  order = randperm(num_data)
  for i in range(num_data // bsz):
    x = train_data[order[i*bsz:(i+1)*bsz]]
```

Closed-Form Solution

Closed-Form Solution

- Main idea:
  - Compute gradient and set it to 0.
    (condition for optimal solution)
  - Solve the equation in a closed form

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- We will derive the gradient from matrix calculus

Closed-Form Solution

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

Closed-Form Solution

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}) - y^{(n)})^2$$

Vectorization

Closed-Form Solution

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$(a-b)^2 = a^2 - 2ab + b^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}))^2 - \sum_{n=1}^{N} y^{(n)}\mathbf{w}^T\phi(\mathbf{x}^{(n)}) + \frac{1}{2}\sum_{n=1}^{N} y^{(n)2}$$

Closed-Form Solution

- Objective function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}) - y^{(n)})^2$$

$$= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}))^2 - \sum_{n=1}^{N} y^{(n)}\mathbf{w}^T\phi(\mathbf{x}^{(n)}) + \frac{1}{2}\sum_{n=1}^{N} y^{(n)2}$$

$$= \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y}$$

Vectorization by defining data matrix

The Data Matrix

- The data matrix is an $N \times M$ matrix, applying
  - the M basis functions (columns)
  - to N data points (rows)

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$

$$\Phi\mathbf{w} \approx \mathbf{y}$$

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}(\sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}^{(n)}) - y^{(n)})^2 \\ &= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}) - y^{(n)})^2 \\ &= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}^{(n)}))^2 - \sum_{n=1}^{N} y^{(n)}\mathbf{w}^T\phi(\mathbf{x}^{(n)}) + \frac{1}{2}\sum_{n=1}^{N} y^{(n)2} \\ &= \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}^T\Phi^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} \end{aligned}$$

Useful Trick: Matrix Calculus

- Main idea so far:
  - Compute gradient and set it to 0.
    (condition for optimal solution)
  - Solve the equation in a closed form using matrix calculus

- Need to compute the first derivative in matrix form

Recap: The Gradient

Suppose that $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ is a function that takes as input a matrix A of size $m \times n$ and returns a real value. Then the **gradient** of f (with respect to $A \in \mathbb{R}^{m\times n}$) is the matrix of partial derivatives, defined as:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix} \in \mathbb{R}^{m\times n},$$

i.e., an $m \times n$ matrix with

$$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}.$$

Recap: The Gradient

Note that the size of $\nabla_A f(A)$ is always the same as the size of A. So if, in particular, A is just a vector $x \in \mathbb{R}^n$,

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}.$$

It follows directly from the equivalent properties of partial derivatives that:

- $\nabla_x (f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$.
- For $t \in \mathbb{R}, \nabla_x (t \cdot f(x)) = t \nabla_x f(x)$

Gradient of Linear Functions

- Linear function:

$$f(x) = \sum_{i=1}^{n} b_i x_i$$

- Gradient:

$$\frac{\partial f(x)}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k$$

- Compact form:

$$\nabla_x b^\top x = b$$

Gradient of Linear Functions

- Quadratic function (A is symmetric):

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j$$

- Gradient:

$$\frac{\partial f(x)}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j = 2 \sum_{i=1}^{n} A_{ki} x_i$$

- Compact form:

$$\nabla_x x^\top A x = 2Ax$$

Putting Together: Solution via Matrix Calculus

- Compute gradient and set to zero

$$
\begin{aligned}
\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \left(\frac{1}{2} \mathbf{w}^T \Phi^T \Phi \mathbf{w} - \mathbf{w}^T \Phi^T \mathbf{y} + \frac{1}{2} \mathbf{y}^T \mathbf{y} \right) \\
&= \Phi^T \Phi \mathbf{w} - \Phi^T \mathbf{y} \\
&= 0
\end{aligned}
$$

- Solve the resulting equation (normal equation)

$$\Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{y}$$

$$\mathbf{w}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

This is the *Moore-Penrose pseudo-inverse*: $\mathbf{\Phi}^\dagger = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T$

applied to: $\Phi \mathbf{w} \approx \mathbf{y}$

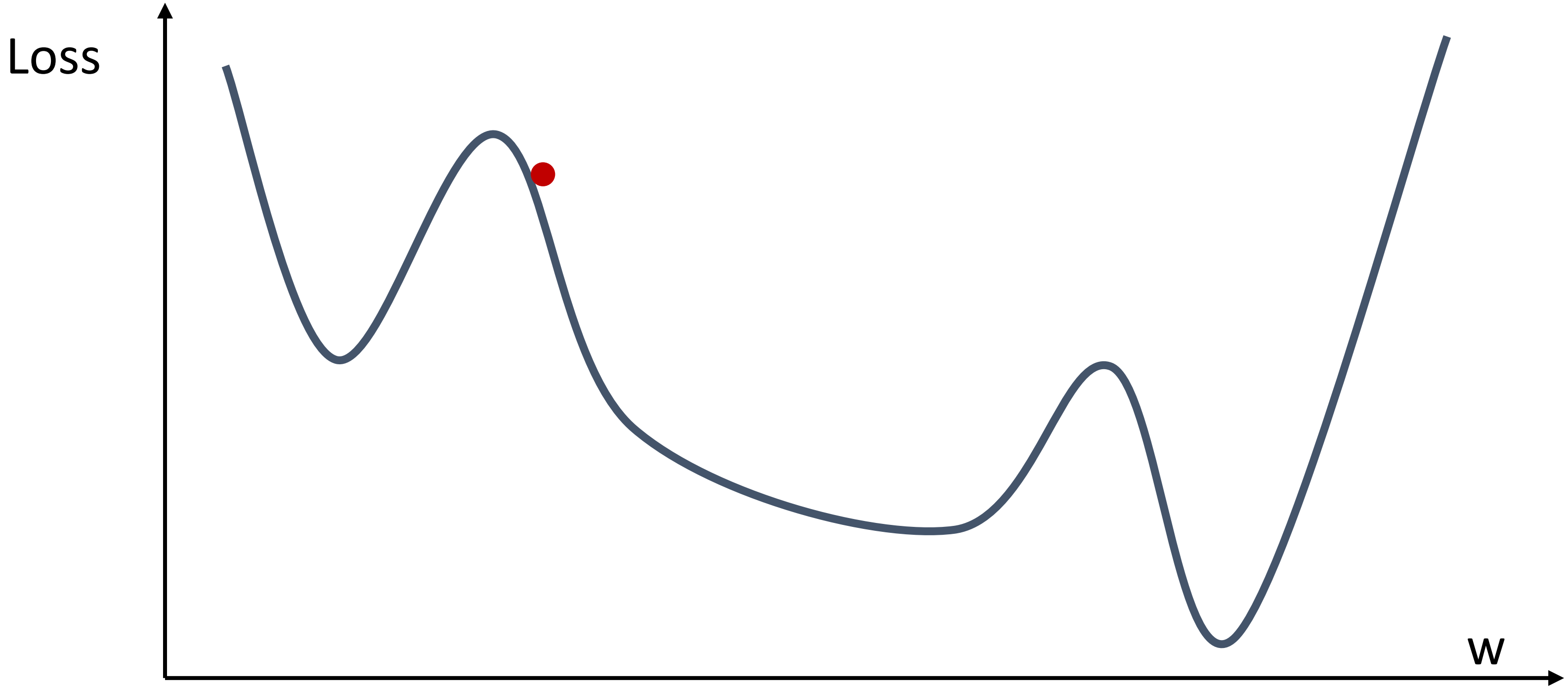
Gradient Descent vs. Closed-Form

- **Gradient Descent**
  - **Fast**: quadratic form requires $O(N^2)$
  - Need to choose an appropriate learning rate η

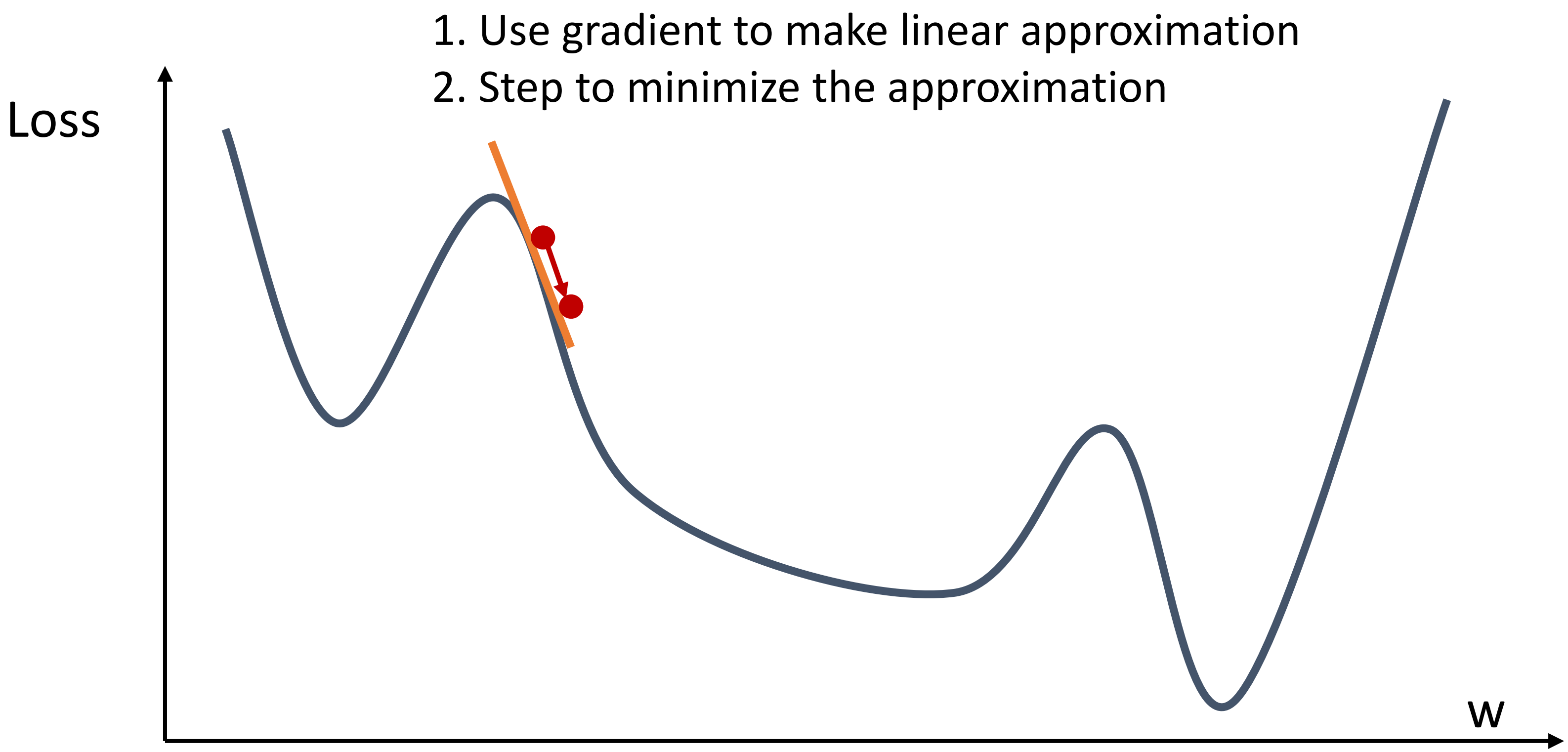
- **Closed-Form Solution**
  - **Slow**: matrix inversion requires $O(N^3)$
    - Faster version: $O(N^{2.373})$; out of scope of this course
  - Matrix inversion is numerically unstable

Newton’s Method

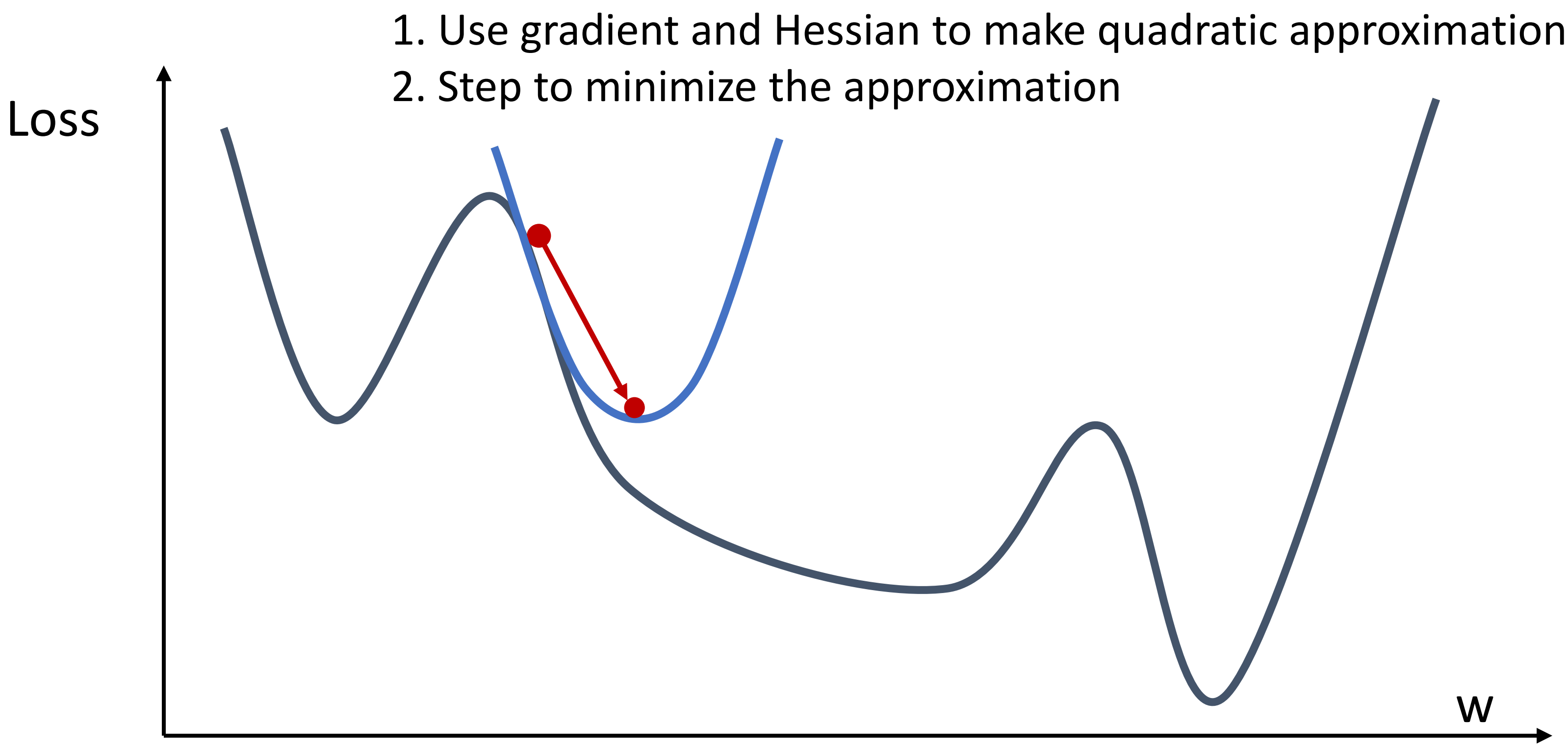
First-Order Optimization



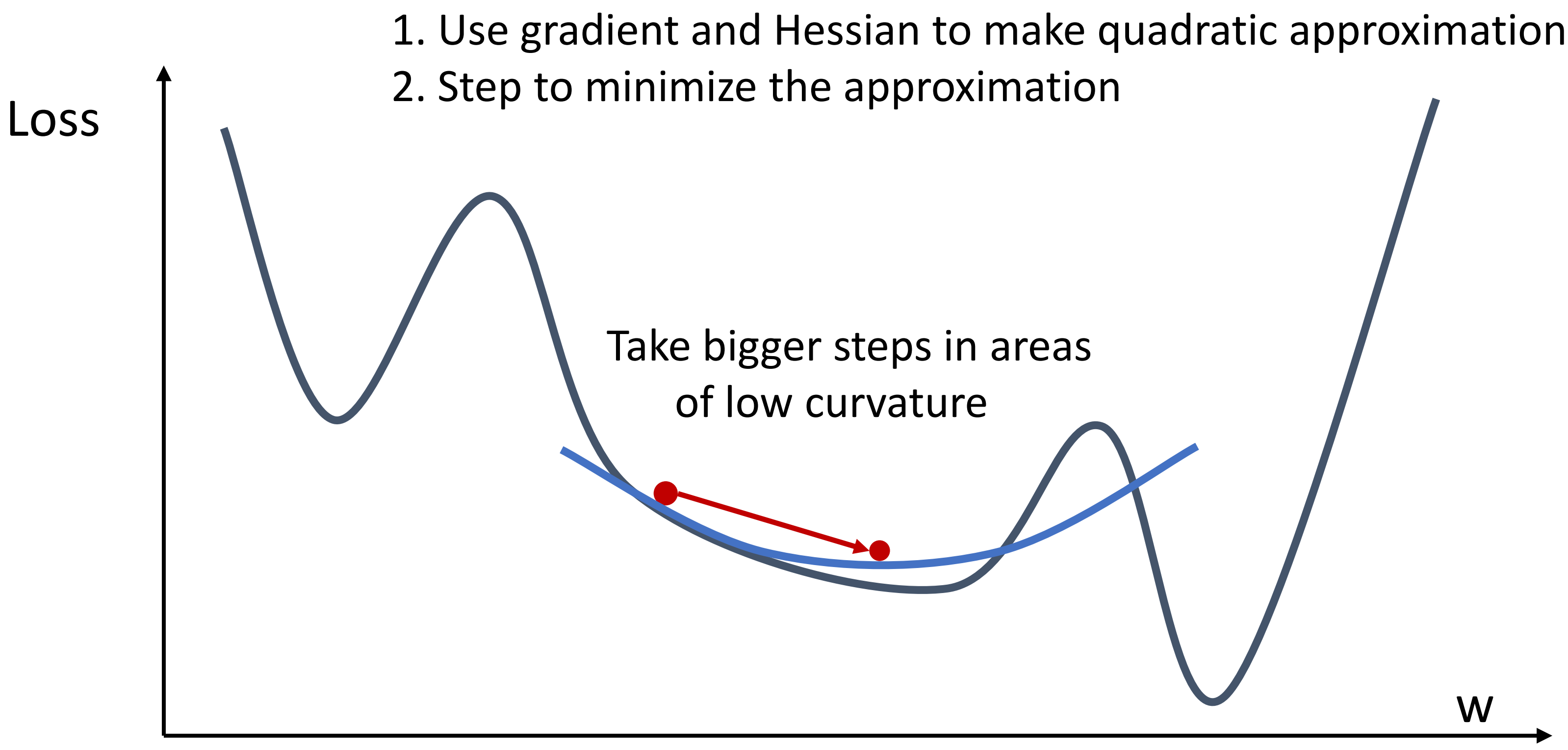
First-Order Optimization



Second-Order Optimization



Second-Order Optimization



Newton's Method

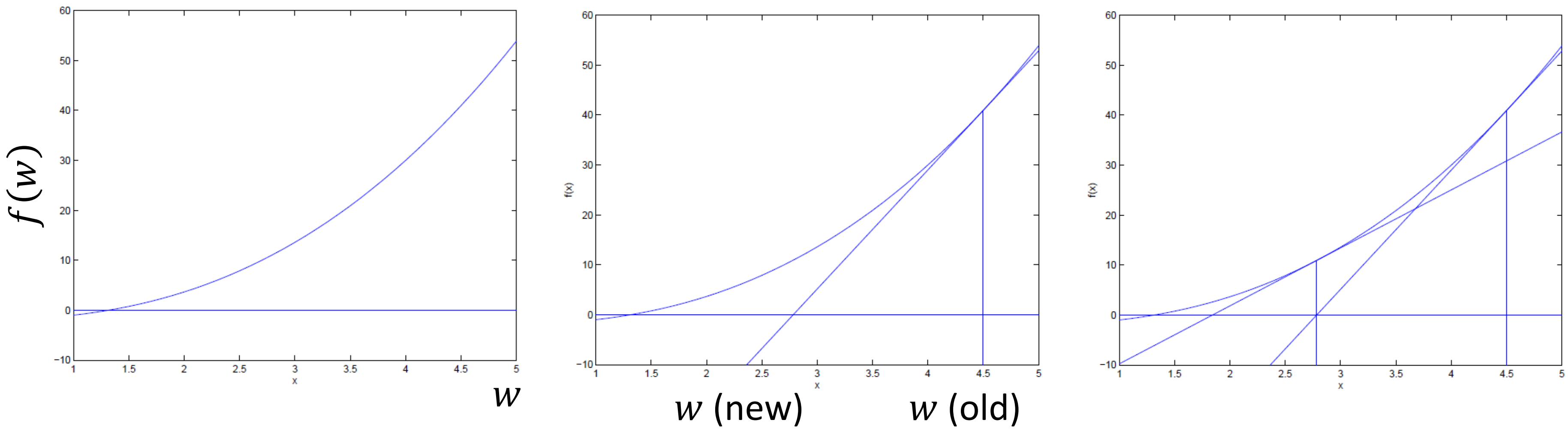
- Goal: Minimizing a general function $E(\mathbf{w})$
  - Assume scalar $\mathbf{w}$ for simplicity.
  - Approach: solve for $f(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = 0$

- Newton's method (aka Newton-Raphson method)
  - Repeat until convergence:

$$\mathbf{w} := \mathbf{w} - \frac{f(\mathbf{w})}{f'(\mathbf{w})}$$

Newton’s Method

- Interactively solve until we get $f(\mathbf{w}) = 0$.



- Geometric intuition

$$\mathbf{w} := \mathbf{w} - \frac{f(\mathbf{w})}{f'(\mathbf{w})}$$

Current value

“Slope”

Newton's Method

- We want to minimize $E(\mathbf{w})$
  - Convert $E'(\mathbf{w}) = f(\mathbf{w})$
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \frac{E'(\mathbf{w})}{E''(\mathbf{w})}$$

Newton update when $\mathbf{w}$ is a scalar

- This method can be extended for multivariate case:

$$\mathbf{w} := \mathbf{w} - H^{-1}\nabla_{\mathbf{w}} E$$

Newton update when $\mathbf{w}$ is a vector

where H is a hessian matrix (evaluated at $\mathbf{w}$)

$$H_{ij}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}_i \partial \mathbf{w}_j}$$

Gradient Descent vs. Newton's Method

- **Gradient Descent**
  - First-order optimization
  - Computing the update by linear approximation of the loss curve
  - Parametric; need to set the size of update (learning rate)
- **Newton's Method**
  - Second-order optimization
  - Computing the update by quadratic approximation of the loss curve
  - Non-parametric
  - Reduced to closed-form solution if the learning objective is quadratic with respect to learnable parameters

Next: Other Topics on Linear Regression