Machine Learning (CE 40477)

Fall 2025

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 1, 2025

Course Overview

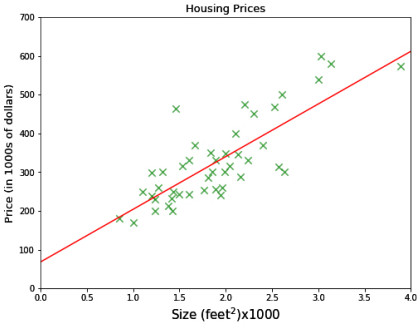
Course Overview

- Supervised Learning
- Unsupervised Learning
- Neural Networks
- Computer Vision
- Natural Language Processing
- Additional Chapter (Agentic AI)

Supervised Learning

Learn to predict outcomes using labeled data.

- Predict house prices from past sales data.
- Use features such as size, location, and room count.

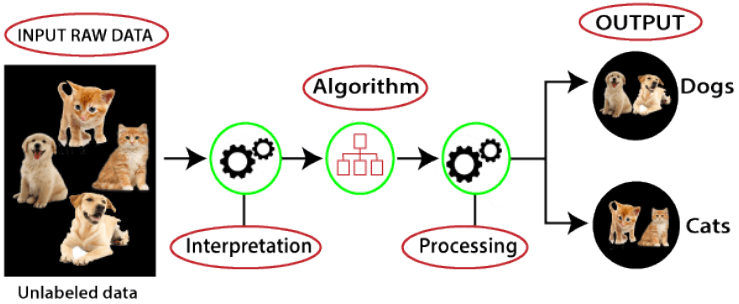


Learn More: Predicting House Prices with Linear Regression

Unsupervised Learning

Discover hidden structures within unlabeled data.

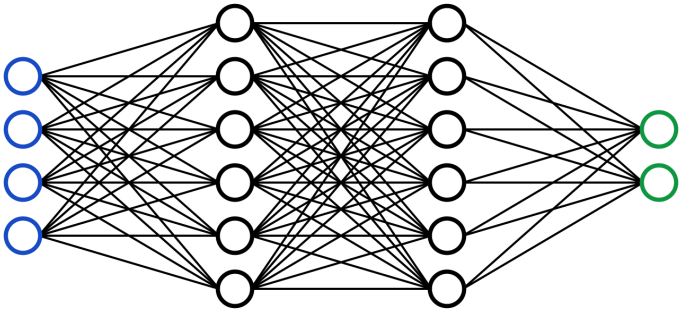
- Identify groups or patterns without predefined labels.



Example: Image clustering.

Neural Networks

- Designed to mimic the human brain in order to solve complex tasks



Examples: Facial recognition, voice assistants.

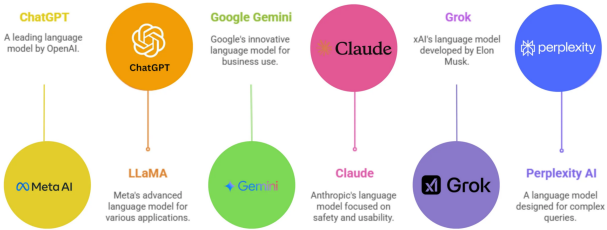
Computer Vision

- Enables machines to perceive and interpret visual information

Examples: Factory quality control, medical image analysis.

Natural Language Processing (NLP)

- Understand and generate human language



Example: Chat-bots, language translation.

What is Machine Learning (ML)?

- **Machine Learning**: Enables computers to learn patterns from data without explicit programming.
- Focuses on predicting outcomes, classification, or discovering hidden structures.

- **Goal**: Build models that make accurate predictions from past data.

Tom M. Mitchell's Definition of ML

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

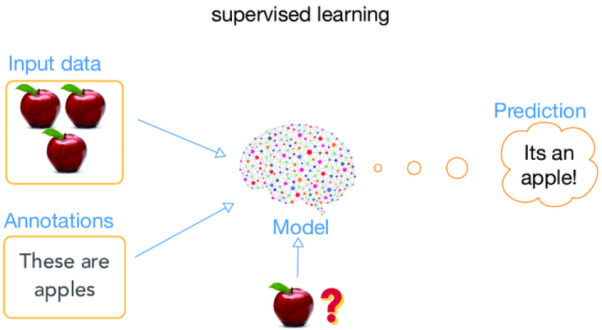
- Formally: Learning Problem = (T, P, E)
- Example: Predicting house prices using past data.

Paradigms of ML

- **Supervised learning** (regression, classification)
  - predicting a target variable for which we get to see examples.
- **Unsupervised learning**
  - revealing structure in the observed data
- **Reinforcement learning**
  - partial (indirect) feedback, no explicit guidance
  - given rewards for a sequence of moves to learn a policy and utility functions
- **Other paradigms:** semi-supervised learning, active learning, online learning, etc.

Supervised Learning

- **Definition**: A form of machine learning where the model learns from labeled data $\{(x_i, y^{(i)})\}$ to predict an output y given an input x.
- **Goal**: Estimate a function $f : \mathbb{R}^d \to \mathbb{R}$, such that $y = f(\mathbf{x}) + \varepsilon$, where ε is noise.



Components of (Supervised) Learning

- **Unknown target function:** $f : \mathcal{X} \to \mathcal{Y}$
  - **Input space:** $\mathcal{X}$
  - **Output space:** $\mathcal{Y}$
- **Training data:** $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
- **Pick a formula** $h : \mathcal{X} \to \mathcal{Y}$ **that approximates the target function** f
  - selected from a set of hypotheses $\mathcal{H}$

Components of (Supervised) Learning (cont.)

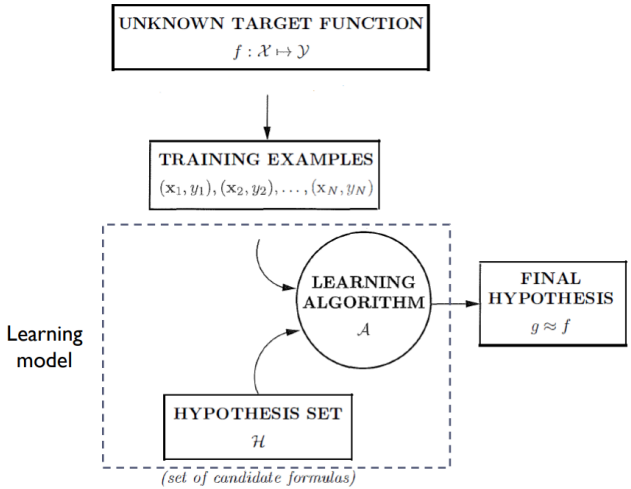
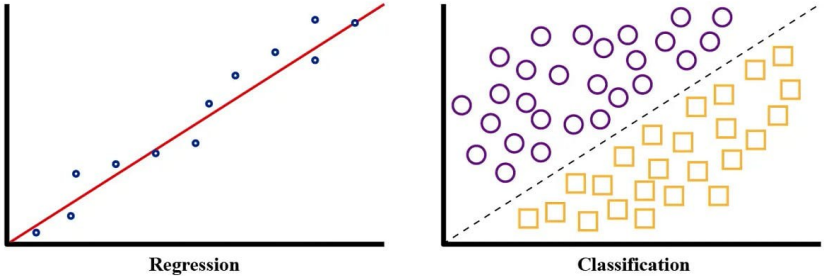


Figure adapted from Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. Learning from Data: A Short Course

Supervised Learning: Regression vs. Classification

- **Regression**: predict a continuous target variable
  - E.g., $y \in [0, 1]$
- **Classification**: predict a discrete target variable
  - E.g., $y \in \{1, 2, \ldots, C\}$



Solution Components - Learning Model

- The **Learning Model** consists of:
  - **Hypothesis Set**: Defines the possible functions $\mathcal{H} = \{h_\theta(\mathbf{x}) | \theta \in \Theta\}$, where $h_\theta(\mathbf{x})$ represents candidate functions and θ is the learning parameters of problem.
  - **Learning Algorithm**: Find $\theta^* \in \Theta$ such that $h_{\theta^*}(\mathbf{x}) \approx f(\mathbf{x})$.
- Both work together to map inputs x to outputs y with minimized error.
- In other words, θ^* is best parameters to predict outputs using chosen hypothesis.

Hypothesis Space Overview

- **Hypothesis (h)**: A mapping from input space $\mathscr{X}$ to output space $\mathscr{Y}$.
- **Linear Regression Hypothesis**:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_d x_d = \mathbf{w}^\top \mathbf{x}$$

- **Input Vector x**:

$$\mathbf{x} = \left[x_0 = 1, x_1, x_2, \ldots, x_d\right]$$

- **Parameter Vector w**:

$$\mathbf{w} = \left[w_0, w_1, w_2, \ldots, w_d\right]$$

Linear Hypothesis Representation

- **Linear Hypothesis Space**:
  - **Simplest form**: Linear combination of input features.

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

- **Linear Hypothesis Examples**:
  - **Single Variable**: $h_{\mathbf{w}}(x) = w_0 + w_1 x$
  - **Multivariate**: $h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$

Understanding Cost Functions

- In **hypothesis space**, we select a function $h_{\mathbf{w}}(x)$ to approximate the true relationship between input x and output y.
- The objective is to minimize the difference between predicted values $h(x)$ and actual values y.
- This difference is quantified using **cost functions**, which guide us in choosing the optimal hypothesis.

What is a Cost Function?

- A **cost function** measures how well the hypothesis $h_{\mathbf{w}}(x)$ fits the training data.
- In regression problems, the most common error function is the **Squared Error (SE)**:

$$SE : \left(y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)})\right)^2$$

- Cost function should measure all predictions. Thus a choice could be **Sum of Squared Errors (SSE)**:

$$J(\mathbf{w}) = \sum_{i=1}^{N} \left(y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)})\right)^2$$

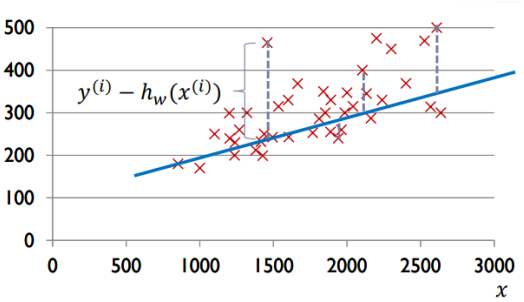
- **Objective:** Minimize the cost function to find the best parameters $\mathbf{w}$.

SSE: Sum of Squared Errors

- **SSE** is widely used due to its simplicity and differentiability.
- Intuitively, it represents the squared distance between predicted and true values.
- Penalizes larger errors more severely than smaller ones (due to the square).
- For linear regression, it can be written as:

$$SSE = \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} \right)^2$$

How to measure the error



$$J(w) = \sum_{i=1}^{n} \left(y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}) \right)^2$$

$$= \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)} \right)^2$$

Figure adapted from slides of Dr. Soleymani, Machine Learning course, Sharif university of technology.

The learning algorithm

- **Objective:** Choose $\mathbf{w}$ so as to minimize the $J(\mathbf{w})$
- **The learning algorithm:** optimization of the cost function
  - Explicitly taking the cost function derivative with respect to the w_i's, and setting them to zero.
- Parameters of the best hypothesis for the training set:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

Cost function optimization: univariate

$$J(\mathbf{w}) = \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)} \right)^2$$

- **Necessary conditions for the "optimal" parameter values:**

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = 0, \qquad \frac{\partial J(\mathbf{w})}{\partial w_1} = 0$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \sum_{i=1}^{n} 2 \left(y^{(i)} - w_0 - w_1 x^{(i)} \right) (-x^{(i)}) = 0$$

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \sum_{i=1}^{n} 2 \left(y^{(i)} - w_0 - w_1 x^{(i)} \right) (-1) = 0$$

  - A system of 2 linear equations

Linear regression example: TV Advertising and Sales

This is a real-world example of how businesses can use linear regression to make decisions about marketing budgets. The following table shows the amount of TV advertising budget spend and respective average sales of houses in Boston:

TV Advertising Spend ($1000)	Sales (Units)
230.1	22.1
44.5	10.4
17.2	9.3
151.5	18.5
180.8	12.9

Table 1: TV Advertising Spend vs. Sales Dataset

Learn More: Advertising Sales Dataset

Linear regression example: TV Advertising and Sales (cont.)

- In this problem, Sales per TV advertising is need thus the amount spend is considered as input x and sales is considered as output y.
- Using linear regression, cost function can be written as:

$$J(\mathbf{w}) = \sum_{i=1}^{5} \left(y^{(i)} - w_0 - w_1 x^{(i)} \right)^2$$

- Applying necessary conditions for the optimal parameters:

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \sum_{i=1}^{5} 2 \left(y^{(i)} - w_0 - w_1 x^{(i)} \right) (-x^{(i)}) = 0 \implies 110863 w_1 + 624.1 w_0 - 10843.04 = 0$$

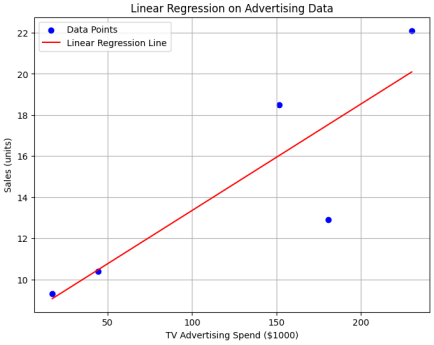
$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \sum_{i=1}^{5} 2 \left(y^{(i)} - w_0 - w_1 x^{(i)} \right) (-1) = 0 \implies 624.1 w_1 + 5 w_0 - 73.2 = 0$$

Linear regression example: TV Advertising and Sales (cont.)

- Solving the system of two equations described, we get:

$$w_1 \approx 0.052$$

$$w_0 \approx 8.18$$



Cost function optimization: multivariate

- Remembering **SSE cost function** of multivariate linear regression

$$J(\mathbf{w}) = \sum_{i=1}^{n} \left(y^{(i)} - h_{\mathbf{w}}(x^{(i)}) \right)^2 = \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} \right)^2$$

- We usually write matrix form of the problem as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

In which $x_m^{(i)}$ indicates m'th feature of data point i

Cost function optimization: multivariate

- Using the matrix forms suggested, we can rewrite cost function:

$$J(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

- Explicitly taking the cost function derivative with respect to the $\mathbf{w}$, and setting them to zero:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -2\mathbf{X}^T\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0 \implies \mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \implies \mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

- $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$ is called pseudo-inverse of matrix $\mathbf{X}$
- The matrix $\mathbf{X}$ is often not square yet not invertible.
- The pseudo-inverse can be computed for any matrix, regardless of its shape.

Computational limitations of analytical solution

- **Scalability:** Analytical solutions do not scale well with very large datasets, making them impractical for big data applications.
- **Finding the inverse of a matrix:**
  - Simplest way of finding inverse: Gaussian elimination, having complexity of $O(n^3)$.
  - Other methods: LU decomposition, which also has a complexity of $O(n^3)$ but is more stable.
  - Numerical methods: Iterative methods like Conjugate Gradient, which can be more efficient for large, sparse matrices.

Cost function optimization

- **Another approach,**
  - Start from an initial guess and iteratively change $\mathbf{w}$ to minimize $J(\mathbf{w})$.
    - The gradient descent algorithm
- **Steps:**
  - Start from $\mathbf{w}^0$
  - Repeat:
    - Update $\mathbf{w}^t$ to $\mathbf{w}^{t+1}$ in order to reduce J
    - $t \leftarrow t+1$

  until we hopefully end up at a minimum.

Gradient Descent

- In each step, takes steps proportional to the negative of the gradient vector of the function at the current point $\mathbf{w}^t$:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla J(\mathbf{w}^t)$$

- $J(\mathbf{w})$ **decreases fastest** if one goes from $\mathbf{w}^t$ in the direction of $-\nabla J(\mathbf{w}^t)$
- **Assumption**: $J(\mathbf{w})$ is defined and differentiable in a neighborhood of a point $\mathbf{w}^t$.
- **Gradient ascent** takes steps proportional to (the positive of) the gradient to find a local maximum of the function.

Gradient Descent (cont.)

- In gradient descent, we only use the gradient (1st order Taylor approximation).
- In other words, we assume that the function ℓ around $\mathbf{w}$ is linear and behaves like:

$$\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s}$$

  where $g(\mathbf{w}) = \nabla \ell(\mathbf{w})$.
- Our goal is to find a vector $\mathbf{s}$ that minimizes this function.
- In gradient descent we simply set:

$$\mathbf{s} = -\eta g(\mathbf{w}),$$

  for some small $\eta > 0$.
- It is straightforward to prove that in this case $\ell(\mathbf{w}+\mathbf{s}) < \ell(\mathbf{w})$:

$$\underbrace{\ell(\mathbf{w}+(-\eta g(\mathbf{w})))}_{\text{after one update}} \approx \ell(\mathbf{w}) \underbrace{-\eta g(\mathbf{w})^\top g(\mathbf{w})}_{<0} < \underbrace{\ell(\mathbf{w})}_{\text{before}}$$

Gradient Descent (cont.)

- **Minimize** $J(\mathbf{w})$

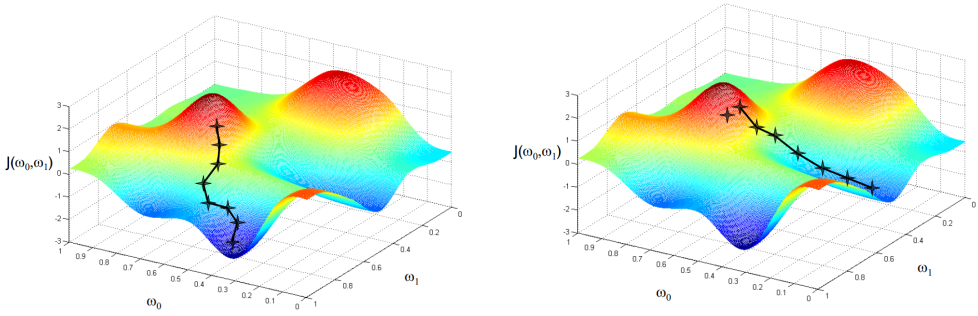
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^t)$$

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

- If η is small enough, then $J(\mathbf{w}^{t+1}) \leq J(\mathbf{w}^t)$.
- η can be allowed to change at every iteration as η_t.

Gradient descent disadvantages

- **Local minima problem**
- **However, when J is convex, all local minima are also global minima** $\Rightarrow$ gradient descent can converge to the global solution.



Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

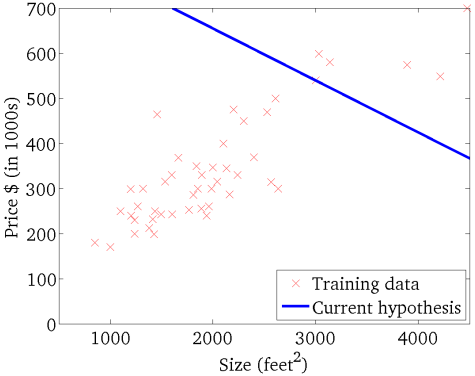
Cost function optimization

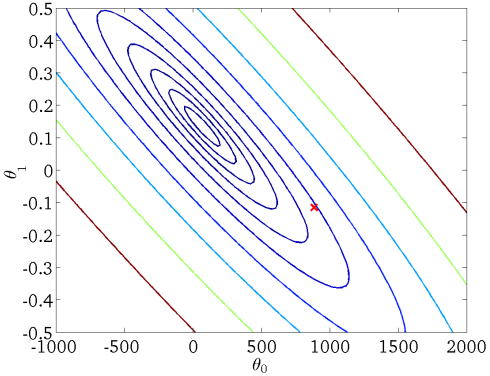
- Weight update rule having $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)} \right) \mathbf{x}^{(i)}$$

- η: too small $\Rightarrow$ gradient descent can be slow.
- η: too large $\Rightarrow$ gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

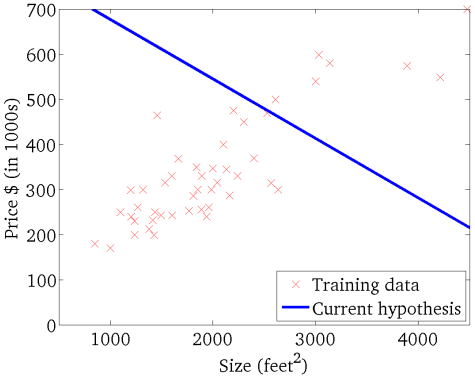
Gradient descent overview

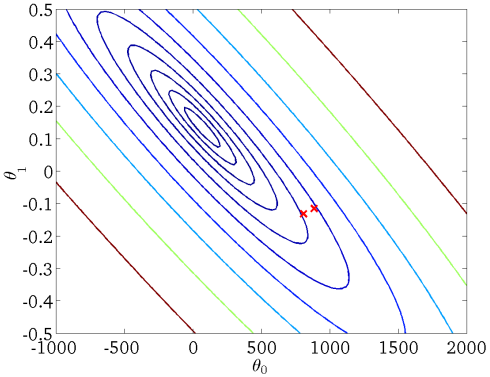




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

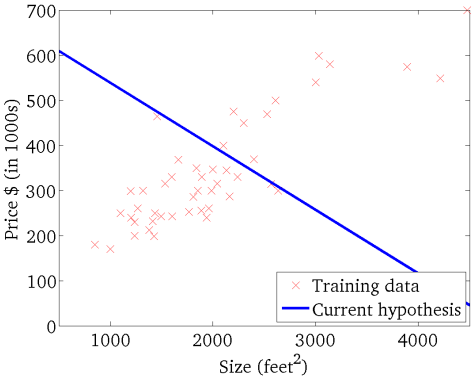
Gradient descent overview (cont.)

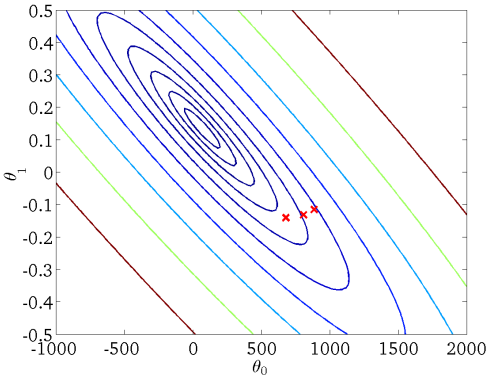




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

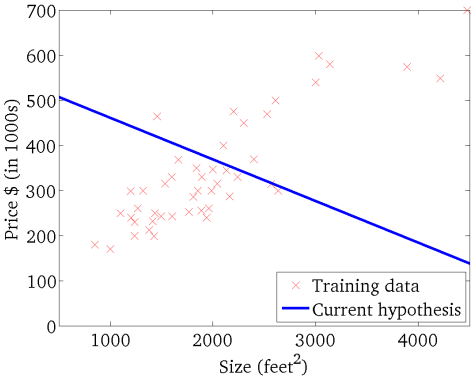
Gradient descent overview (cont.)

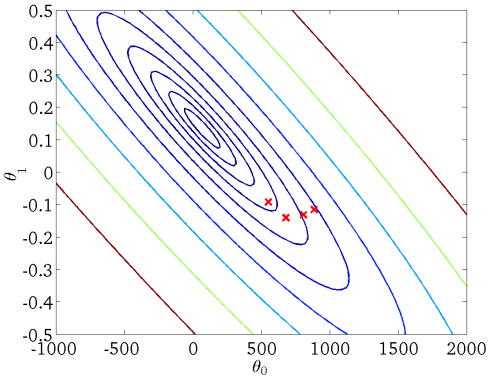




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

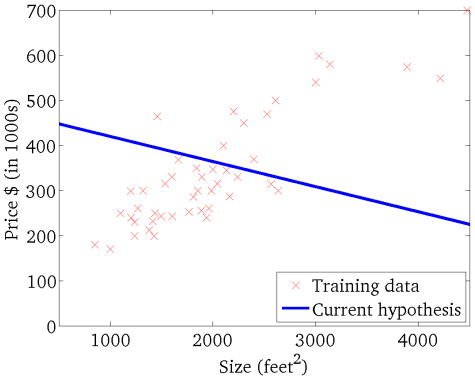
Gradient descent overview (cont.)

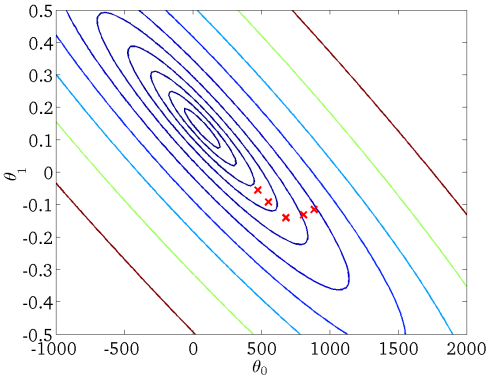




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

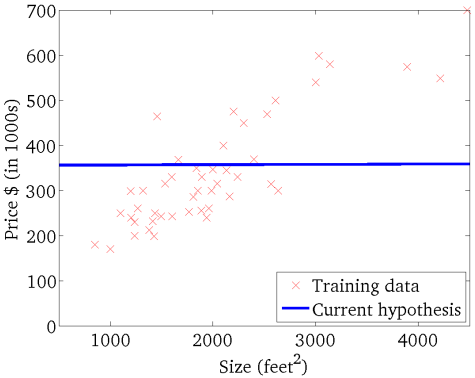
Gradient descent overview (cont.)

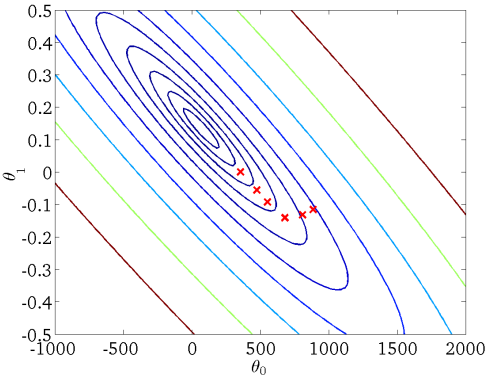




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

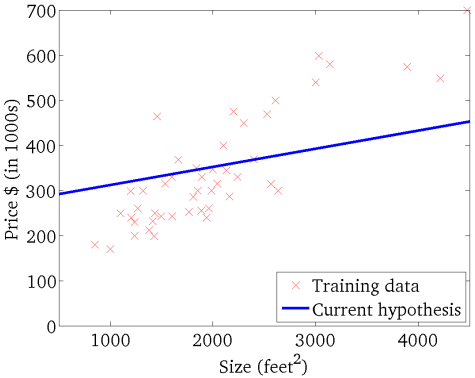
Gradient descent overview (cont.)

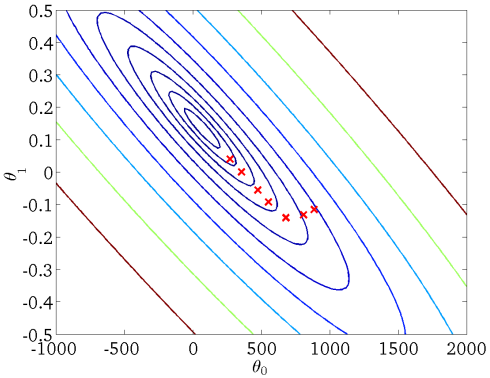




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

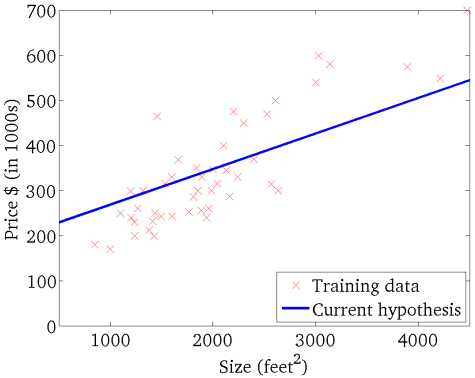
Gradient descent overview (cont.)

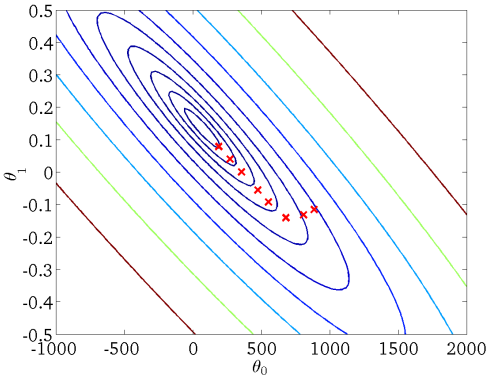




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

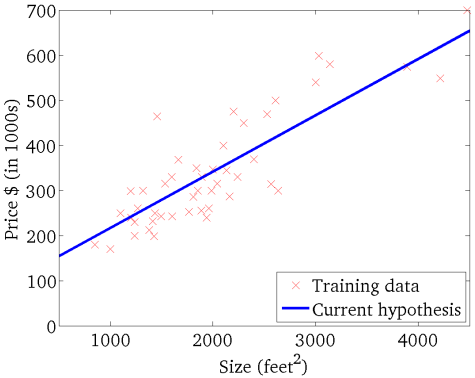
Gradient descent overview (cont.)

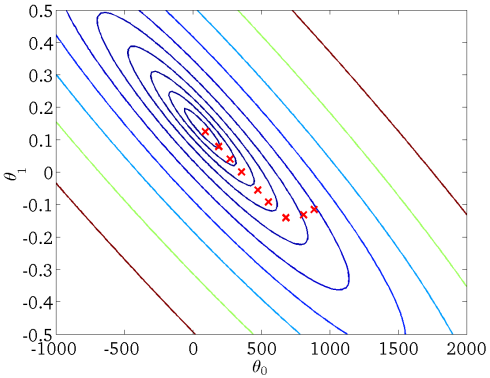




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

Gradient descent overview (cont.)





Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

Variants of gradient descent

- **Batch gradient descent:** processes the entire training set in one iteration
  - It can be computationally costly for large data sets and practically impossible for some applications (such as online learning).
- **Mini-batch gradient descent:** processes small, random subsets (mini-batches) of the training set in each iteration
  - Balances the efficiency of batch gradient descent.
- **Stochastic gradient descent:** processes one training example per iteration
  - Updates the model parameters more frequently, which can lead to faster convergence.

Stochastic gradient descent

- **Example:** Linear regression with SSE cost function

$$J(\mathbf{w}) = \sum_{i=1}^{n} J^{(i)}(\mathbf{w})$$

$$J^{(i)}(\mathbf{w}) = \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} J^{(i)}(\mathbf{w})$$

$$\Longrightarrow \mathbf{w}^{t+1} = \mathbf{w}^t + \eta \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right) \mathbf{x}^{(i)}$$

In which $x^{(i)}$ indicates the i'th observation arrived.

Beyond Linear Regression

- How can we extend linear regression to model non-linear relationships?
  - **Transforming Data Using Basis Functions:**
    - Basis functions allow us to transform the original features into a new feature space.
    - Common basis functions include polynomial and Gaussian functions.
  - **Learning a Linear Regression on Transformed Features:**
    - By applying linear regression to the transformed feature vectors, we can model complex, non-linear relationships.
    - This approach maintains the simplicity and interpretability of linear regression while extending its flexibility.

Beyond Linear Regression: Polynomial Regression

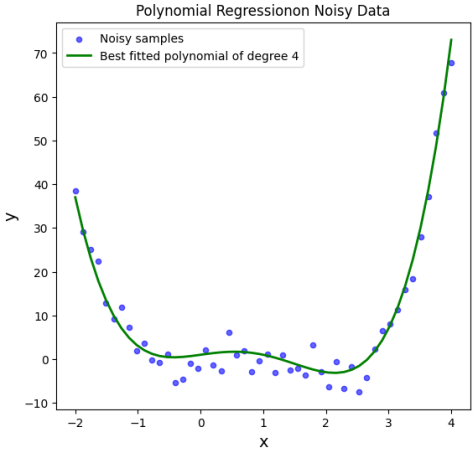


Figure 1: Best fitted polynomial of degree 4 can generalize well on samples

Beyond Linear Regression: change of basis

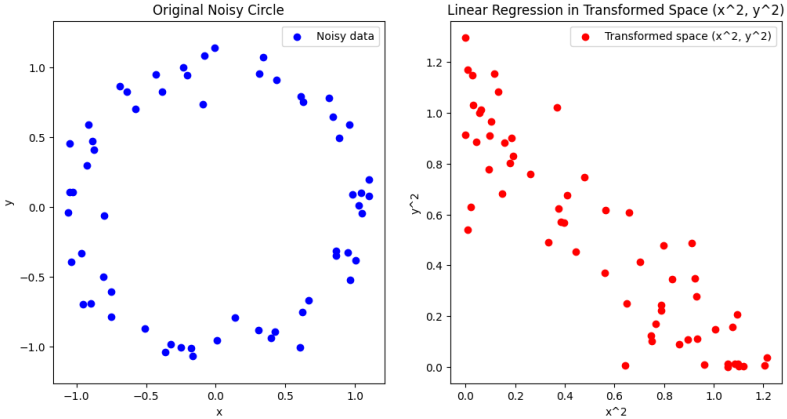


Figure 2: Changing the basis of $[1, x, y]$ to $[1, x^2, y^2]$, we can use linear regression

Polynomial regression: Univariate

- **Polynomial Regression Hypothesis:** m'th order regression

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x^1 + \cdots + w_{m-1} x^{m-1} + w_m x^m$$

- **Objective:** Fit a polynomial of degree m to data points.
- Similar to what we did for univariate linear regression, we can define:

$$\mathbf{X}' = \begin{bmatrix} 1 & x^{(1)^1} & \cdots & x^{(1)^m} \\ 1 & x^{(2)^1} & \cdots & x^{(2)^m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{(n)^1} & \cdots & x^{(n)^m} \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_m \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

In which $x^{(i)}$ indicates the i-th data point.

Polynomial regression analytical solution: Univariate

Rewriting the SSE cost function using matrix form we have:

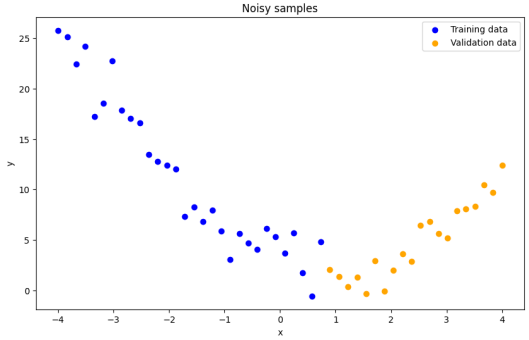
$$J(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}'\mathbf{w}\|_2^2$$

Analytical solution: It has closed form solution as

$$\hat{\mathbf{w}} = \left(\mathbf{X}'^T\mathbf{X}'\right)^{-1}\mathbf{X}'^T y$$

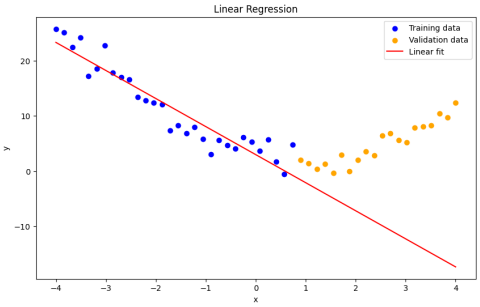
Polynomial regression: example

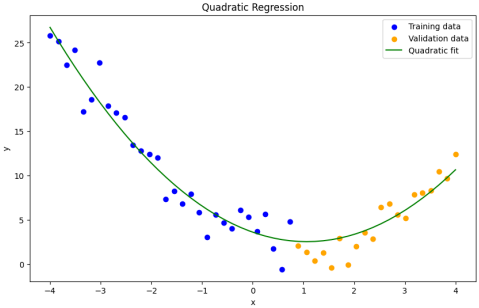
- Consider the following noisy samples



Splitting samples to train and validation sets

Quadratic regression vs Linear regression





Fitting both quadratic and linear regression shows the power of polynomial regression on generalizing for more complex patterns.

Generalization Overview

Main Idea: The ability of a model to perform well on unseen data

- **Training Set**: $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
- **Test Set**: New data not seen during training
- **Cost Function**: Measures how well the model fits data

$$J(\mathbf{w}) = \sum_{i=1}^{n} (y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}))^2$$

- **Objective**: Minimize the cost function on unseen data (generalization error)

Expected Test Error

Definition: Expected performance on unseen data

- Test data sampled from the same distribution $\mathbb{P}(\mathbf{x}, y)$

$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}}[(y - h_{\mathbf{w}}(\mathbf{x}))^2]$$

- Approximate using test set $\hat{J}(\mathbf{w})$
- Generalization error is the gap between training and test performance.

Training vs Test Error

Key Concept: Training error measures fit on known data, test error on unseen data

- **Training (empirical) error**:

$$J_{\text{train}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)})\right)^2$$

- **Test error**:

$$J_{\text{test}}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\left(y_{\text{test}}^{(i)} - h_{\mathbf{w}}(\mathbf{x}_{\text{test}}^{(i)})\right)^2$$

- **Goal**: Minimize the test error (generalization).

Overfitting Definition

Concept: A model fits the training data well but performs poorly on the test set

$$J_{\text{train}}(\mathbf{w}) \ll J_{\text{test}}(\mathbf{w})$$

- Causes: Model too complex, high variance
- Consequence: Captures noise in training data, fails on unseen data

Underfitting Definition

Concept: The model is too simple and cannot capture the structure of the data

$$J_{\text{train}}(\mathbf{w}) \approx J_{\text{test}}(\mathbf{w}) \gg 0$$

- Causes: Model lacks complexity, high bias
- Consequence: Poor fit on both training and test data

Regime 1: High Variance

Cause: Poor performance due to high variance.

Symptoms:

- Training error $\ll$ Test error
- Training error $< \varepsilon$
- Test error $> \varepsilon$

Remedies:

- Add more training data
- Reduce model complexity (simpler models are less prone to high variance)
- Use Bagging (covered later)

Note: ε is a predefined threshold for acceptable training error.

Regime 2: High Bias

Cause: The model is too simple and underfits the data.

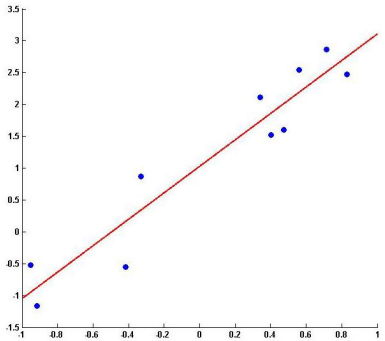
Symptoms:

- Training error $> \varepsilon$

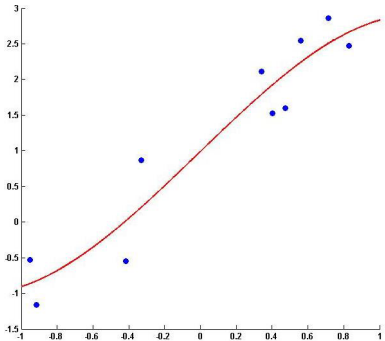
Remedies:

- Use a more complex model (e.g., kernelized or non-linear)
- Add more features
- Apply Boosting (covered later)

Generalization: polynomial regression



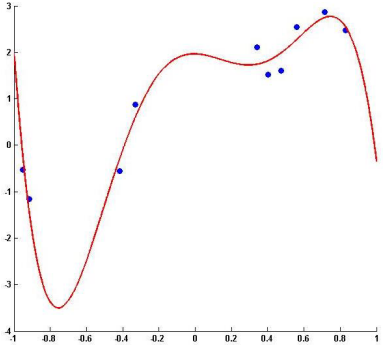
Degree of 1



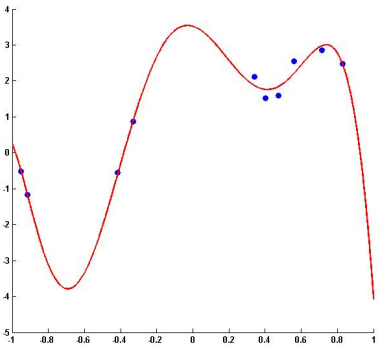
Degree of 3

Example adapted from slides of Dr. Soleymani, ML course, Sharif University of technology.

Overfitting: polynomial regression



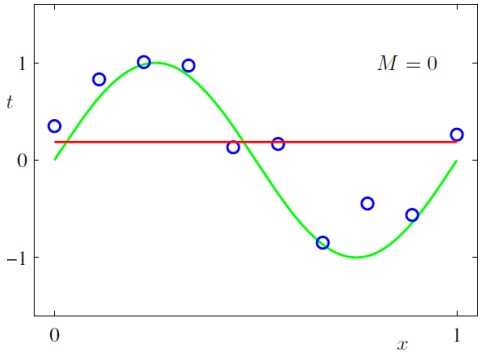
Degree of 5

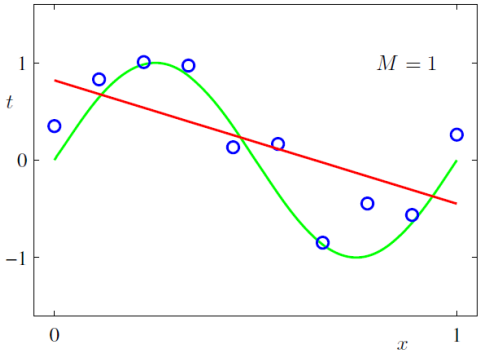


Degree of 7

Example adapted from slides of Dr. Soleymani, ML course, Sharif University of technology.

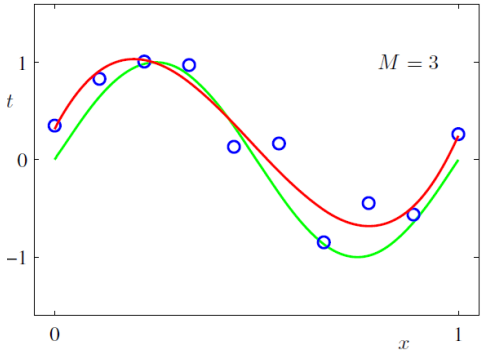
Polynomial regression with various degrees: example

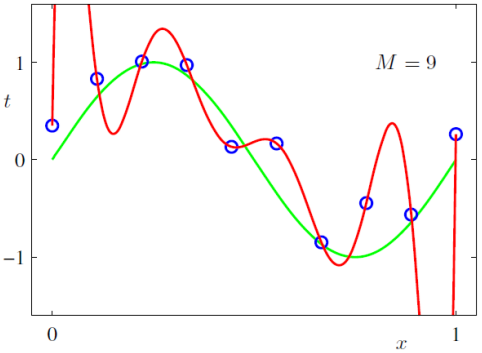




Figures adapted from Machine Learning and Pattern Recognition, Bishop

Polynomial regression with various degrees: example (cont.)





Figures adapted from Machine Learning and Pattern Recognition, Bishop

Bias-Variance Decomposition

Generalization error decomposition:

$$\mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}}[(y - h_{\mathbf{w}}(\mathbf{x}))^2] = (\text{Bias})^2 + \text{Variance} + \text{Noise}$$

- **Bias**: Error due to simplifying assumptions in the model

$$\text{Bias}(\mathbf{x}) = \mathbb{E}[h_{\mathbf{w}}(\mathbf{x})] - f(\mathbf{x})$$

- **Variance**: Sensitivity of the model to training data

$$\text{Variance}(\mathbf{x}) = \mathbb{E}[h_{\mathbf{w}}(\mathbf{x}) - \mathbb{E}[h_{\mathbf{w}}(\mathbf{x})])^2]$$

- **Noise**: Irreducible error from the inherent randomness in data

Proof: Decomposition of Expected Test Error

Notations:

- $\mathbf{x}$: input (feature) vector
- y: true output (label)
- D: training dataset
- $h(\mathbf{x}|D)$: predictor learned from D
- $\bar{h}(\mathbf{x}) = \mathbb{E}_D[h(\mathbf{x}|D)]$: expected prediction
- $\bar{y}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}]$: true regression function
- $\mathbb{E}_{\mathbf{x},y,D}[\cdot]$: expectation over data and datasets

For the sake of brevity, we ommit the $\mathbf{w}$ subscript from h in the proof. All the expectations are taken over the distribution $\mathbb{P}$.

Proof: Decomposition of Expected Test Error

Starting Point: Decompose the expected test error

$$\mathbb{E}_{\mathbf{x},y,D}\left[\left(h(\mathbf{x}|D)-y\right)^2\right]=\mathbb{E}_{\mathbf{x},y,D}\left[\left((h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))+(\bar{h}(\mathbf{x})-y)\right)^2\right]$$

$$=\mathbb{E}_{\mathbf{x},D}\left[(h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))^2\right]+2\,\mathbb{E}_{\mathbf{x},y,D}\left[(h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x})-y)\right]+\mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-y)^2\right]$$

The middle term of the above equation is 0 as we show below:

$$\begin{aligned}\mathbb{E}_{\mathbf{x},y,D}\left[(h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x})-y)\right]&=\mathbb{E}_{\mathbf{x},y}\left[\mathbb{E}_D\left[h(\mathbf{x}|D)-\bar{h}(\mathbf{x})\right](\bar{h}(\mathbf{x})-y)\right]\\&=\mathbb{E}_{\mathbf{x},y}\left[\left(\mathbb{E}_D[h(\mathbf{x}|D)]-\bar{h}(\mathbf{x})\right)(\bar{h}(\mathbf{x})-y)\right]\\&=\mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-\bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x})-y)\right]=0\end{aligned}$$

Proof: Decomposition of Expected Test Error

Returning to the earlier expression, we're left with the variance and another term:

$$\mathbb{E}_{\mathbf{x},y,D}\left[(h(\mathbf{x}|D)-y)^2\right] = \underbrace{\mathbb{E}_{\mathbf{x},D}\left[(h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))^2\right]}_{\text{Variance}} + \mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-y)^2\right]$$

We can break down the second term in the above equation as follows:

$$\mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-y)^2\right] = \mathbb{E}_{\mathbf{x},y}\left[((\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))+(\bar{y}(\mathbf{x})-y))^2\right]$$

$$= \underbrace{\mathbb{E}_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x})-y)^2\right]}_{\text{Noise}} + \underbrace{\mathbb{E}_{\mathbf{x}}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))^2\right]}_{\text{Bias}^2} + 2\mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x})-y)\right]$$

Proof: Decomposition of Expected Test Error

The third term in the equation above is 0, as we show below:

$$\begin{aligned}\mathbb{E}_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x})-y)\right] &= \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{y|\mathbf{x}}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x})-y)\right]\right] \\ &= \mathbb{E}_{\mathbf{x}}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x})-\mathbb{E}_{y|\mathbf{x}}[y])\right] \\ &= \mathbb{E}_{\mathbf{x}}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x})-\bar{y}(\mathbf{x}))\right] = 0\end{aligned}$$

This gives us the decomposition of expected test error as follows:

$$\underbrace{\mathbb{E}_{\mathbf{x},y,D}\left[(h(\mathbf{x}|D)-y)^2\right]}_{\text{Expected Test Error}} = \underbrace{\mathbb{E}_{\mathbf{x},D}\left[(h(\mathbf{x}|D)-\bar{h}(\mathbf{x}))^2\right]}_{\text{Variance}} + \underbrace{\mathbb{E}_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x})-y)^2\right]}_{\text{Noise}} + \underbrace{\mathbb{E}_{\mathbf{x}}\left[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))^2\right]}_{\text{Bias}^2}$$

Bias–Variance: Where the Square Lives

- **Important Note:** Bias^2 is a mean of squares (not a square of a mean),

$$\mathbb{E}_{\mathbf{x}}\big[(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2\big] \neq \big(\mathbb{E}_{\mathbf{x}}[\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x})]\big)^2.$$

- **Why is the square inside?** After taking $\mathbb{E}_D$ and $\mathbb{E}_{y|\mathbf{x}}$, the remainder is a function of $\mathbf{x}$ only; we then average over $\mathbb{P}(\mathbf{x})$. Hence we square pointwise and integrate globally over all the points, and the power 2 stays inside the expectation.
- Pointwise MSE for a fix $\mathbf{x}$ (the proof is similar, do it yourself !):

$$\mathrm{MSE}(\mathbf{x}) = \mathbb{E}_{D,y|\mathbf{x}}\big[(h(\mathbf{x} \mid D) - y)^2\big] = \underbrace{\mathrm{Var}_D[h(\mathbf{x} \mid D)]}_{\text{Variance}} + \underbrace{\mathrm{Var}(y \mid \mathbf{x})}_{\text{Noise}} + \underbrace{(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2}_{\text{Bias}^2}.$$

High Bias in Simple Models

Simple models, such as linear regression, often underfit.

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x$$

- The model is too simple to capture the true underlying pattern $f(x)$.
- Even with infinite data, the average hypothesis $\bar{h}(\mathbf{x}) = \mathbb{E}_D[h(\mathbf{x}|D)]$ will be systematically wrong.
- The model has **low variance** (changing the dataset D doesn't change $h(\mathbf{x}|D)$ much) but **high bias**.

$$\text{Bias}^2 \gg \text{Variance}$$

- **Result**: Leads to high generalization error (poor performance) on both the training and test sets. This is **underfitting**.

High Variance in Complex Models

Complex models, such as high-degree polynomials, tend to overfit.

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m \quad \text{(for large } m\text{)}$$

- The model is too flexible and starts to fit the random noise in the training data D.
- The learned hypothesis $h(\mathbf{x}|D)$ is highly sensitive to the specific training data.
- The model has **low bias** (it's flexible enough to capture the true pattern) but **high variance**.

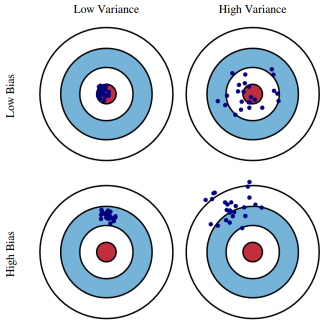
$$\text{Variance} \gg \text{Bias}^2$$

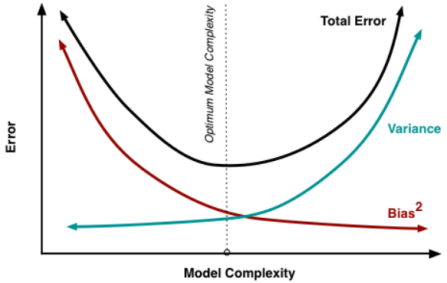
- **Result**: The model performs very well on the training set ($J_{\text{train}} \approx 0$) but fails to generalize to the test set ($J_{\text{test}} \gg 0$). This is **overfitting**.

Bias-Variance Tradeoff

Tradeoff: Balancing between bias and variance is key for optimal performance

- Low complexity: High bias, low variance
- High complexity: Low bias, high variance





Regularization

Prevent overfitting by penalizing large weights.

$$\min_{\mathbf{w}} J_\lambda(\mathbf{w}) = J(\mathbf{w}) + \lambda R(\mathbf{w})$$

- Common regularizers: L1 and L2 norms.
- The parameter λ controls the balance between fit and simplicity.

Equivalent constrained form: minimize the empirical objective under a budget on the regularizer

$$\min_{\mathbf{w}} J(\mathbf{w}) \quad \text{s.t.} \quad R(\mathbf{w}) \le t.$$

- Under mild conditions, for every $t \ge 0$ there exists a $\lambda \ge 0$ such that the solutions of the two problems coincide, and for every $\lambda \ge 0$ there exists a $t \ge 0$ yielding an equivalent solution (Lagrange multiplier correspondence).

L2 Regularization (Ridge Regression)

- **Regularizer**: The L2 norm (squared) of the weight vector.

$$R(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^{D} w_j^2$$

- **Cost Function**:

$$J_{\text{Ridge}}(\mathbf{w}) = \underbrace{\sum_{i=1}^{N} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2}_{\text{SSE (Data Fit)}} + \underbrace{\lambda \sum_{j=1}^{D} w_j^2}_{\text{L2 Penalty}}$$

- **Effect**:
  - Penalizes large weights, forcing them to be small.
  - Shrinks all coefficients towards zero, but **does not** make them exactly zero.
  - Reduces model complexity and prevents overfitting (decreases variance).

L1 Regularization (Lasso Regression)

- **Regularizer**: The L1 norm of the weight vector.

$$R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^{D} |w_j|$$

- **Cost Function**:

$$J_{\text{Lasso}}(\mathbf{w}) = \underbrace{\sum_{i=1}^{N} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2}_{\text{SSE (Data Fit)}} + \underbrace{\lambda \sum_{j=1}^{D} |w_j|}_{\text{L1 Penalty}}$$

- **Effect**:
  - Also penalizes large weights.
  - Has a key property: it can force some weight coefficients to be **exactly zero**.
  - This performs automatic **feature selection**, creating a sparse model.

Ridge vs. Lasso

Ridge Regression (L2)	**Lasso Regression (L1)**
Constraint: $\|\mathbf{w}\|_2^2 \leq t$ (L2 ball).	Constraint: $\|\mathbf{w}\|_1 \leq t$ (L1 ball).
Contours of $J(\mathbf{w})$ typically touch the constraint boundary at smooth points.	Contours often hit the sharp **corners** of the diamond.
Produces small but nonzero coefficients (no exact sparsity).	Produces **sparse** solutions (some coefficients exactly zero).
Differentiable everywhere; the regularizer $\|\mathbf{w}\|_2^2$ is smooth.	Not differentiable at $w_i = 0$; $\|\mathbf{w}\|_1$ has sharp corners.
Strongly convex $\Rightarrow$ guarantees a **unique optimum**.	Not strongly convex $\Rightarrow$ may yield **multiple optima**.

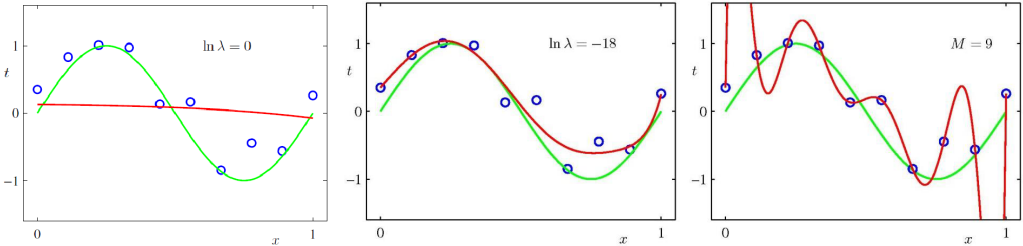
Effect of Regularization Parameter λ

Balancing Fit and Complexity:

$$J_\lambda(\mathbf{w}) = J(\mathbf{w}) + \lambda \sum_{j=1}^{m} w_j^2 = J(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Large λ: Forces smaller weights, reduces complexity, increases bias, decreases variance
- Small λ: Allows larger weights, increases complexity, reduces bias, increases variance

Effect of Regularization parameter λ



$$J_\lambda(\mathbf{w}) = \sum_{i=1}^{n} \left(t^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}) \right)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x + \cdots + w_9 x^9$$

Figures adapted from Machine Learning and Pattern Recognition, Bishop

Effect of regularization on weights

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Table adapted from Machine Learning and Pattern Recognition, Bishop

Introduction to Regression (Probabilistic Perspective)

- **Objective:** Model the relationship between input $\mathbf{x}$ and output y.
- **Uncertainty:** Output y has an associated uncertainty modeled by a probability distribution.
- **Example:**

$$y = h_{\mathbf{w}}(\mathbf{x}) + \varepsilon \quad , \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

- The goal is to learn $h_{\mathbf{w}}(\mathbf{x})$ to predict y.

Curve Fitting with Noise

- In real-world scenarios, observed output y is noisy.
- **Model: True output plus noise**

$$y = h_{\mathbf{w}}(\mathbf{x}) + \varepsilon$$

- Noise represents unknown or unmodeled factors.
- **Example:** Predicting house prices based on features with inherent unpredictability.

Expected Value of Output

- Best Estimate: The conditional expectation of y given $\mathbf{x}$.

$$\mathbb{E}[y|\mathbf{x}] = h_{\mathbf{w}}(\mathbf{x})$$

- Goal: Learn a function $h_{\mathbf{w}}(\mathbf{x})$ that represents the average behavior of the data.
- Key Point: The model captures the mean of the target variable given input $\mathbf{x}$.

Maximum Likelihood Estimation (MLE)

- **MLE:** A method to estimate parameters that maximize the likelihood of the data.
- Given data $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, MLE maximizes:

$$\mathscr{L}(D; \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} \mathbb{P}(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

- MLE finds parameters $\mathbf{w}$ and σ^2 that best explain the data.

Maximum Likelihood Estimation (cont.)

- Instead of maximizing the likelihood, it is often easier to maximize the log-likelihood:

$$\log \mathscr{L}(D; \mathbf{w}, \sigma^2) = \sum_{i=1}^{n} \log \mathbb{P}(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

- It is because $\log f(x)$ preserves the behaviour of $f(x)$.
- It is also easier to find derivative on summation of terms.

Univariate Linear Function Example

- Assuming Gaussian noise with parameters $(0, \sigma^2)$, probability of observing real output value y is:

$$\mathbb{P}(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - h_{\mathbf{w}}(\mathbf{x}))^2}{2\sigma^2}\right)$$

- For a simple linear model $h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x$ we have:

$$\mathbb{P}(y|x, \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - w_0 - w_1 x)^2}{2\sigma^2}\right)$$

- **Key Observation:** Points far from the fitted line will have a low likelihood value.

Log-Likelihood and Sum of Squares

- Using log-likelihood we have:

$$\log \mathscr{L}(D; \mathbf{w}, \sigma^2) = -n \log \sigma - \frac{n}{2} \log(2\pi) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}))^2$$

- Since the objective of MLE is to optimize with regards to random variables, we can rule out the constants:

$$\log \mathscr{L}(D; \mathbf{w}, \sigma^2) \sim - \sum_{i=1}^{n} (y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}))^2$$

- **Equivalence:** Maximizing the log-likelihood is equivalent to minimizing the Sum of Squared Errors (SSE):

$$\mathbf{w}_{\text{MLE}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)}))^2$$

Maximum A Posteriori (MAP) Estimation

- **MAP**: Instead of just maximizing the likelihood $\mathbb{P}(D \mid \mathbf{w})$, we incorporate a **prior** belief $\mathbb{P}(\mathbf{w})$ about what the weights should be.

$$\mathbb{P}(\mathbf{w} \mid D) = \frac{\mathbb{P}(D \mid \mathbf{w})\mathbb{P}(\mathbf{w})}{\mathbb{P}(D)} \propto \underbrace{\mathbb{P}(D \mid \mathbf{w})}_{\text{Likelihood}} \times \underbrace{\mathbb{P}(\mathbf{w})}_{\text{Prior}}$$

- **Objective**: Find the $\mathbf{w}$ that maximizes this posterior.

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} \mathbb{P}(\mathbf{w} \mid D) = \arg\max_{\mathbf{w}} \mathbb{P}(D \mid \mathbf{w})\mathbb{P}(\mathbf{w})$$

- In log-space (easier to optimize):

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} \big(\log \mathbb{P}(D \mid \mathbf{w}) + \log \mathbb{P}(\mathbf{w})\big)$$

MAP with a Gaussian Prior (Derivation)

Let's use the derivation using $\mathbf{w}$ instead of β.

- **Likelihood (from MLE)**: Assume Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

$$\mathbb{P}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2\right)$$

- **Prior**: Assume a **Gaussian prior** on weights, $\mathbf{w} \sim \mathcal{N}(0, \tau^2 I)$. This means we believe weights should be small and centered at zero.

$$\mathbb{P}(\mathbf{w}) \propto \exp\left(-\frac{1}{2\tau^2}\|\mathbf{w}\|_2^2\right)$$

- **Posterior**: Posterior $\propto$ Likelihood $\times$ Prior

$$\mathbb{P}(\mathbf{w} \mid \mathbf{y}, \mathbf{X}) \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 - \frac{1}{2\tau^2}\|\mathbf{w}\|_2^2\right)$$

MAP with a Gaussian Prior (Ridge)

- **Find MAP**: We want to maximize the posterior.

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{y}-\mathbf{X}\mathbf{w}\|_2^2 - \frac{1}{2\tau^2}\|\mathbf{w}\|_2^2\right)$$

- Maximizing $e^{-g(x)}$ is the same as minimizing $g(x)$.

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} \left\{\frac{1}{2\sigma^2}\|\mathbf{y}-\mathbf{X}\mathbf{w}\|_2^2 + \frac{1}{2\tau^2}\|\mathbf{w}\|_2^2\right\}$$

- Multiply by $2\sigma^2$ (doesn't change the minimum) and set $\lambda = \frac{\sigma^2}{\tau^2}$:

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} \left\{\|\mathbf{y}-\mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2\right\}$$

- **Conclusion**: MAP estimation with a Gaussian prior is **identical** to **Ridge (L2) Regression**.

MAP, Lasso, and Conclusion

What about Lasso (L1)?

- This is equivalent to MAP estimation using a **Laplace prior** on the weights.

$$\mathbb{P}(\mathbf{w}) \propto \exp\left(-\frac{\|\mathbf{w}\|_1}{b}\right)$$

- A Laplace prior is sharply peaked at zero, representing a stronger belief that many weights should be exactly zero (sparsity).

Key Takeaway:

- **MLE** $\implies$ Minimize SSE (can overfit).
- **MAP + Gaussian Prior** $\implies$ **Ridge (L2) Regression**.
- **MAP + Laplace Prior** $\implies$ **Lasso (L1) Regression**.

Contributions

- **This slide has been prepared thanks to:**
  - Alireza Mirrokni
  - Aida Jalali
  - Arshia Gharooni
  - Mahan Bayhaghi
  - Amirreza Vishteh

[1] C. M., *Pattern Recognition and Machine Learning*.
Information Science and Statistics, New York, NY: Springer, 1 ed., Aug. 2006.

[2] M. Soleymani Baghshah, "Machine learning." Lecture slides.

[3] A. Ng and T. Ma, *CS229 Lecture Notes*.

[4] T. Mitchell, *Machine Learning*.
McGraw-Hill series in computer science, New York, NY: McGraw-Hill Professional, Mar. 1997.

[5] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data: A Short Course*.
New York, NY: AMLBook, 2012.

[6] S. Goel, H. Bansal, S. Bhatia, R. A. Rossi, V. Vinay, and A. Grover, "CyCLIP: Cyclic Contrastive Language-Image Pretraining," *ArXiv*, vol. abs/2205.14459, May 2022.