Machine Learning (CE 40477)

Fall 2025

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

October 11, 2025

Teaching Team

Teaching Team

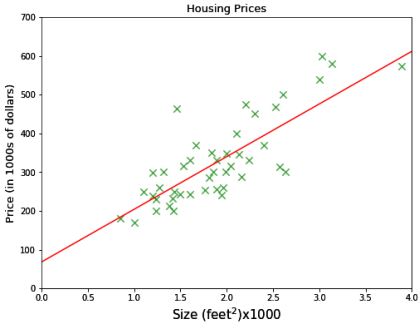
- **Instructor:** Dr. Ali Sharifi Zarchi
  asharifi@sharif.edu
  asharifiz@gmail.com
- **Team Captain:** Alireza Mirrokni
  alirezamirrokni28@gmail.com

Course Overview

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

Learn to predict outcomes using labeled data.

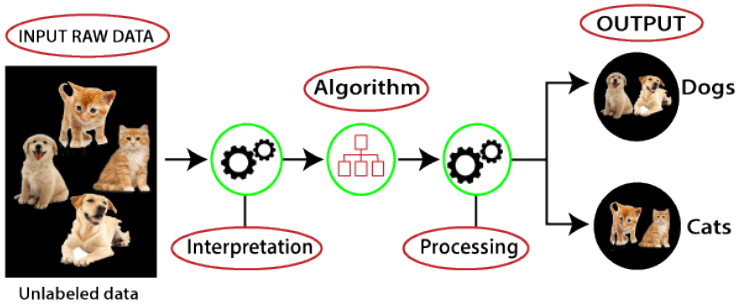
- Predict house prices from past sales data.
- Use features such as size, location, and room count.



Learn More: Predicting House Prices with Linear Regression

Discover hidden structures within unlabeled data.

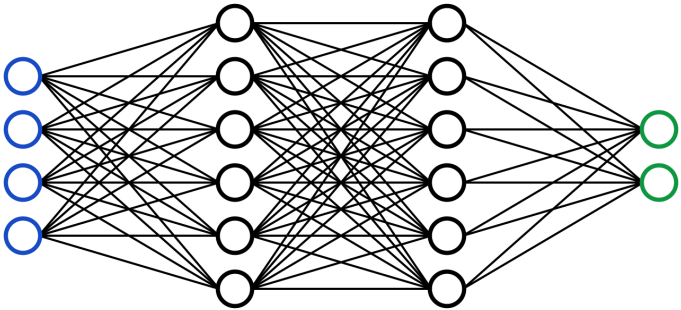
- Identify groups or patterns without predefined labels.



Example: Image clustering.

Neural Networks

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



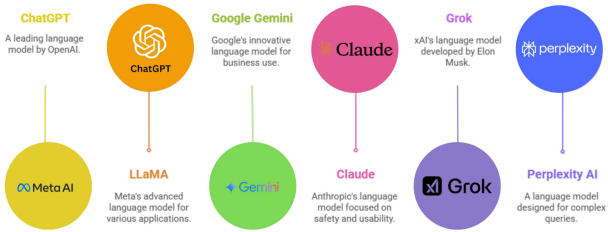
Examples: Facial recognition, voice assistants.

Computer Vision

- Enables machines to perceive and interpret visual information

Examples: Factory quality control, medical image analysis.

- 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."

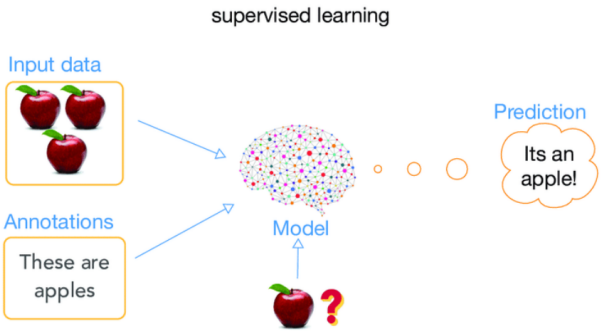
- Formally: Learning Problem = (T, P, E)
- Example: Predicting house prices using past data.

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 $y = f(x) + \epsilon$, 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:** $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
- **Pick a formula** $g : \mathcal{X} \to \mathcal{Y}$ **that approximates the target function** f
  - selected from a set of hypotheses $\mathcal{H}$

Components of (Supervised) Learning (cont.)

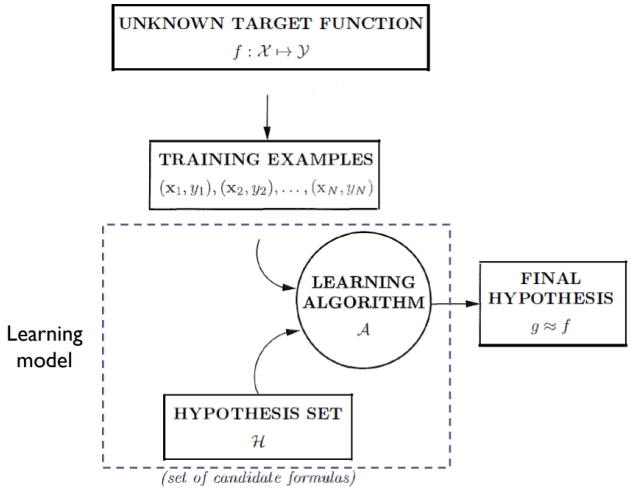
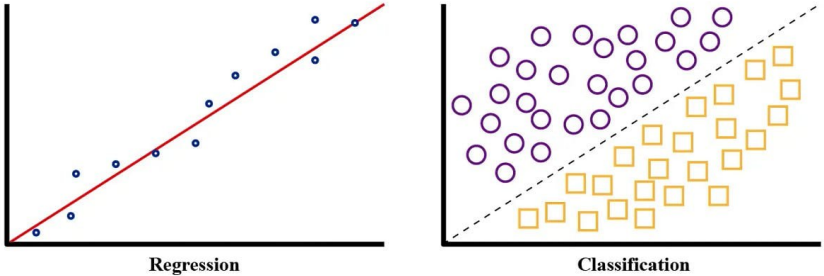


Figure adapted from Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. Learning from Data: A Short Course

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 $\mathscr{H} = \{h(x,\theta)|\theta \in \Theta\}$, where $h(x,\theta)$ represents candidate functions and θ is the learning parameters of problem.
  - **Learning Algorithm**: Find $\theta^* \in \Theta$ such that $h(x,\theta^*) \approx f(x)$.
- 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 (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 $h(x; \mathbf{w})$ 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 $h(x;\mathbf{w})$ fits the training data.
- In regression problems, the most common error function is the **Squared Error (SE)**:

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

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

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

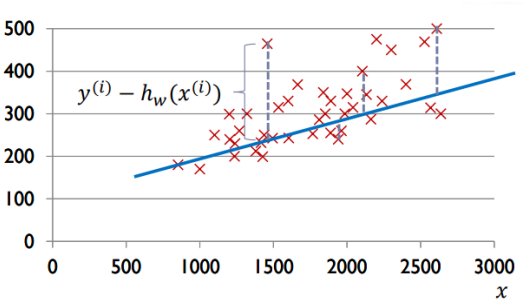
- **Objective:** Minimize the cost function to find the best parameters $\mathbf{w}$.

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



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

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

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

The learning algorithm

- **Objective:** Choose **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:

$$w^* = \arg\min_w J(w)$$

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

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

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

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

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

  - 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:

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

- Applying necessary conditions for the optimal parameters:

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

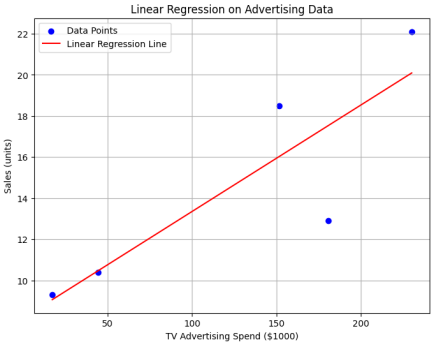
$$\frac{\partial J(w)}{\partial w_0} = \sum_{i=1}^{5} 2 \left(y^{(i)} - w_0 - w_1 x^{(i)} \right) (-1) = 0 \implies 624.1 w_1 + 5 w_0 - 73.2 = 0$$

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

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

- 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:

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

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

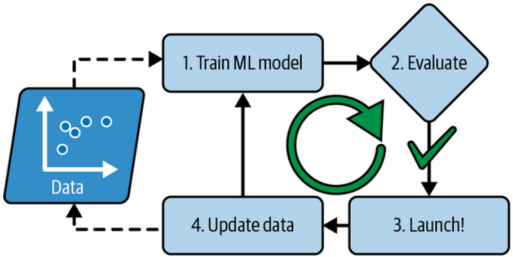
- $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$ is called pseudo-inverse of matrix $\mathbf{X}$
- The matrix $\mathbf{X}$ is often not square yet not invertible.
- The pseudo-inverse can be computed for any matrix, regardless of its shape.

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.

Practical limitations of analytical solution

- **Online learning:** Data observations are arriving in a continuous stream
  - Predictions must be made before seeing all of the data.
  - Analytical solutions require all data to be available upfront, which is not feasible in online learning scenarios.
  - Analytical methods are not adaptable to new data without re-computing the entire solution.



Cost function optimization

- **Another approach,**
  - Start from an initial guess and iteratively change w to minimize $J(w)$.
    - The gradient descent algorithm
- **Steps:**
  - Start from w^0
  - Repeat:
    - Update w^t to 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 w^t:

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

- $J(w)$ **decreases fastest** if one goes from w^t in the direction of $-\nabla J(w^t)$
- **Assumption**: $J(w)$ is defined and differentiable in a neighborhood of a point 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 $\ell(\mathbf{w}+\mathbf{s}) < \ell(\mathbf{w})$:

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

- **Minimize** $J(w)$

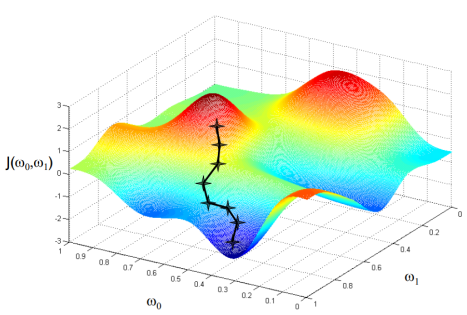
$$w^{t+1} = w^t - \eta \nabla_w J(w^t)$$

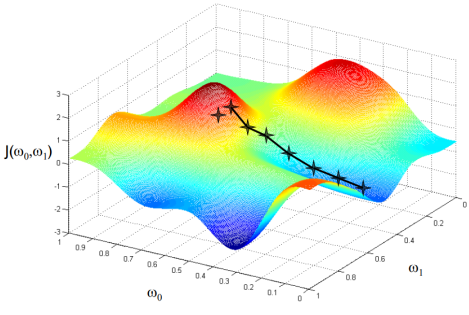
$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_d} \end{bmatrix}$$

- If η is small enough, then $J(w^{t+1}) \leq J(w^t)$.
- 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.

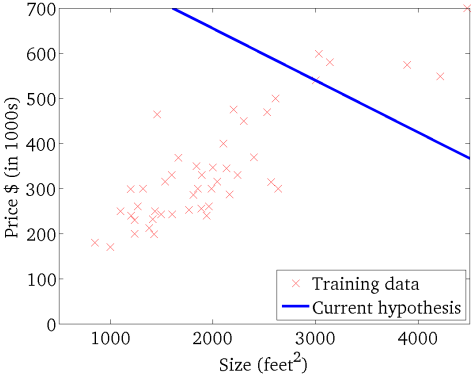
Cost function optimization

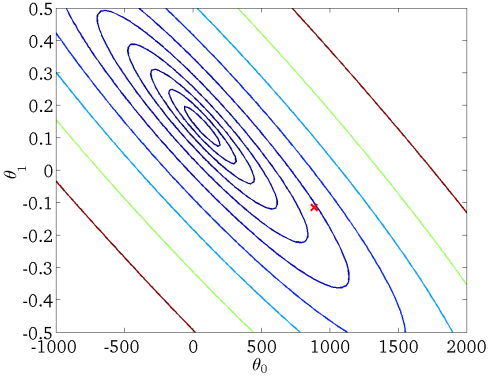
- Weight update rule having $h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is as follows:

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

- 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.

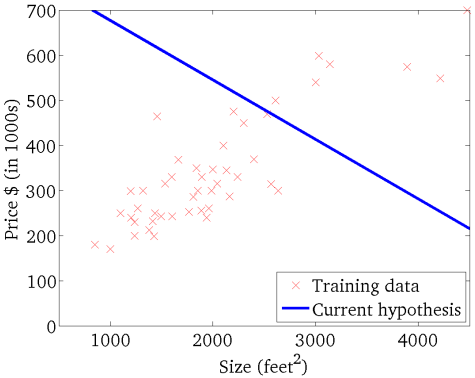
Gradient descent overview

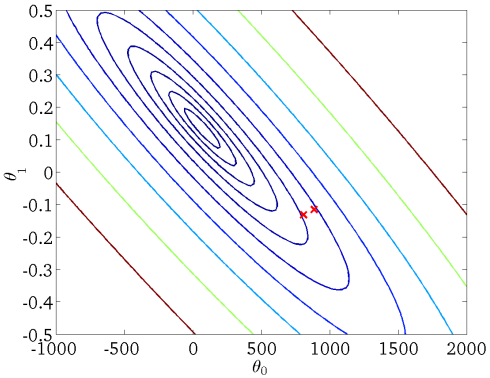




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

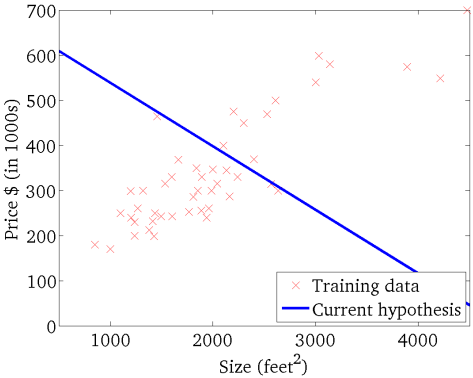
Gradient descent overview (cont.)

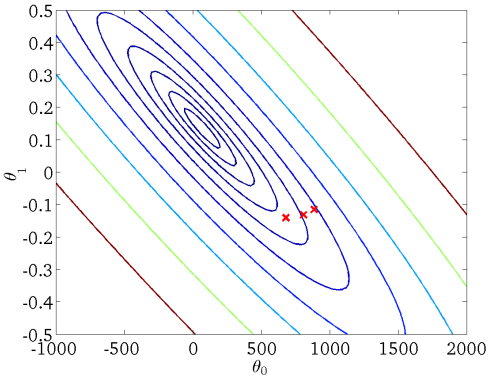




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

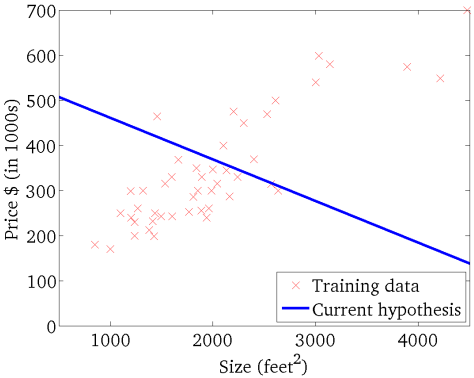
Gradient descent overview (cont.)

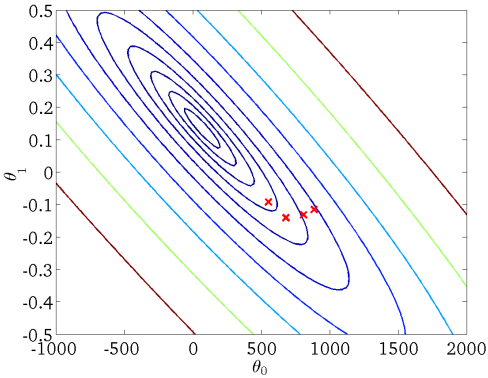




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

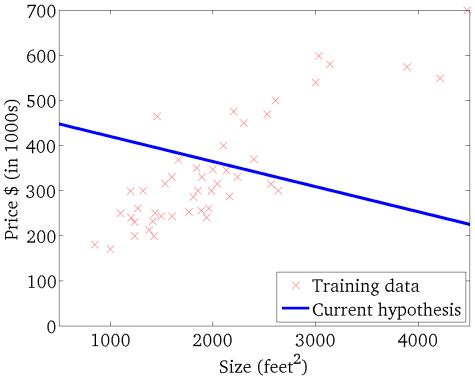
Gradient descent overview (cont.)

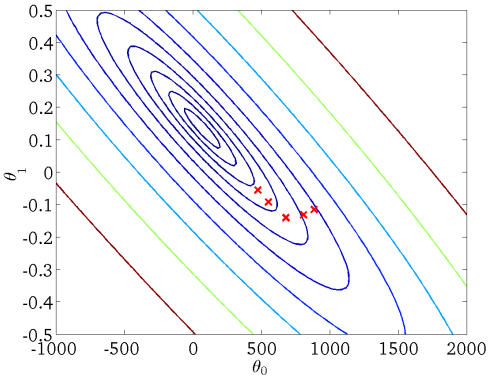




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

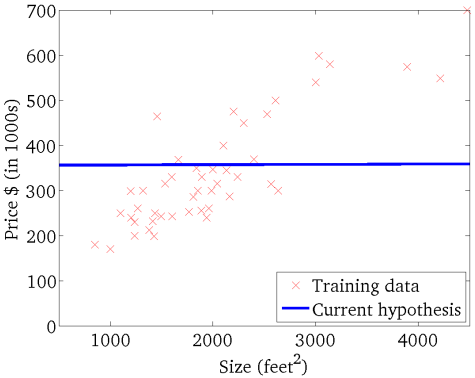
Gradient descent overview (cont.)

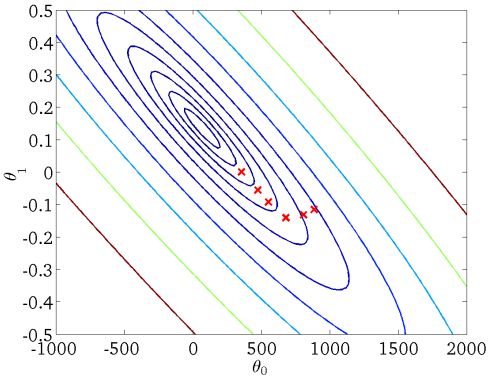




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

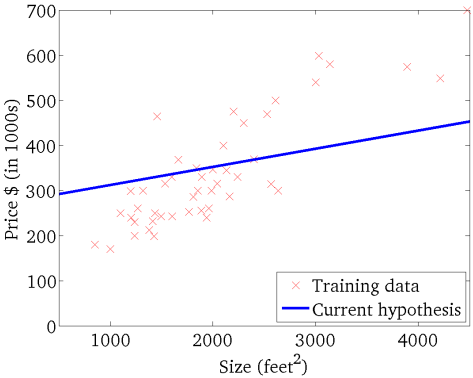
Gradient descent overview (cont.)

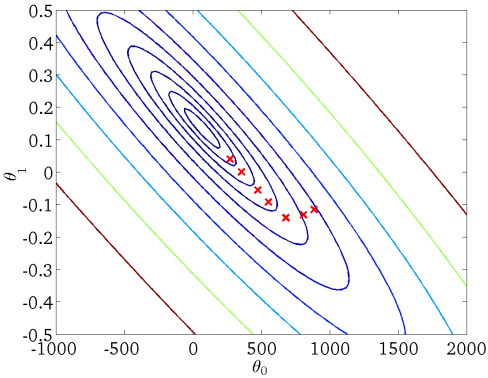




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

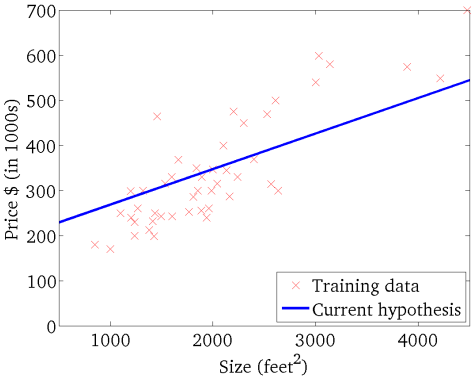
Gradient descent overview (cont.)

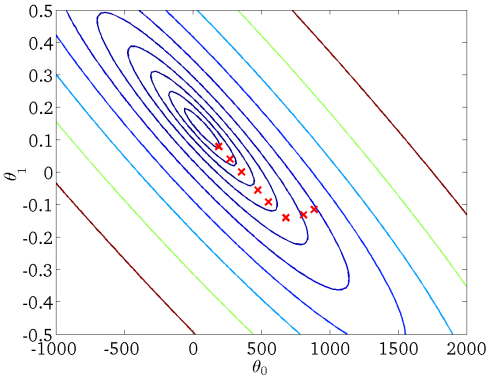




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

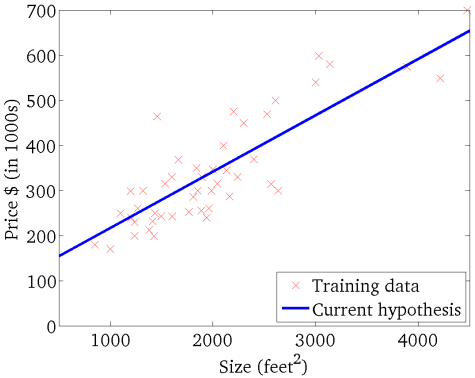
Gradient descent overview (cont.)

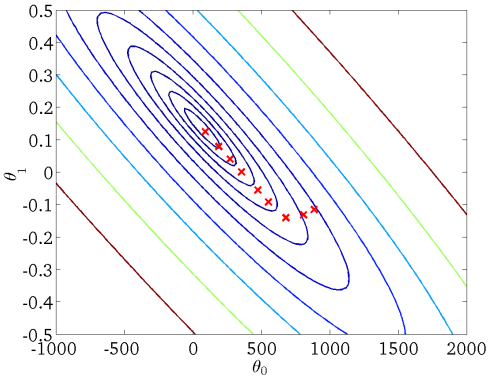




Figures adapted from slides of Andrew Ng, Machine Learning course, Stanford.

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.

- **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.

Stochastic gradient descent: online learning

- Often, stochastic gradient descent gets close to the minimum much faster than batch gradient descent.
- Note however that it may never converge to the minimum, and the parameters will keep oscillating around the minimum of the cost function;
  - In practice, most of the values near the minimum will be reasonably good approximations to the true minimum.

- Newton's method assumes that the loss ℓ is **twice differentiable** and uses the approximation with the Hessian (2nd order Taylor approximation).
- The **Hessian Matrix** contains all second-order partial derivatives and is defined as:

$$H(\mathbf{w}) = \begin{pmatrix} \frac{\partial^2 \ell}{\partial w_1^2} & \frac{\partial^2 \ell}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \ell}{\partial w_1 \partial w_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 \ell}{\partial w_n \partial w_1} & \cdots & \cdots & \frac{\partial^2 \ell}{\partial w_n^2} \end{pmatrix}.$$

- Because of the convexity of ℓ, it is a symmetric, positive semi-definite matrix.
- **Note:** A symmetric matrix M is positive semi-definite if it has only non-negative eigenvalues, or equivalently, for any vector $\mathbf{x}$ we have $\mathbf{x}^\top M\mathbf{x} \geq 0$.

- The second-order approximation of $\ell(\mathbf{w}+\mathbf{s})$ is:

$$\ell(\mathbf{w}+\mathbf{s}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}.$$

- This describes a convex parabola, and we can find its minimum by solving:

$$\arg\min_{\mathbf{s}} \ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H(\mathbf{w})\mathbf{s}.$$

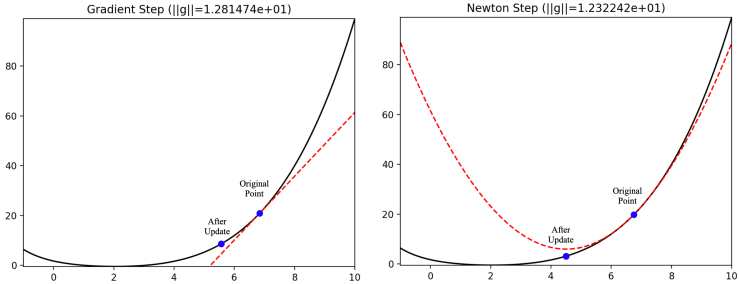
- Taking the derivative with respect to $\mathbf{s}$ and setting it to zero:

$$g(\mathbf{w}) + H(\mathbf{w})\mathbf{s} = 0 \quad \Rightarrow \quad \mathbf{s} = -[H(\mathbf{w})]^{-1} g(\mathbf{w}).$$

- Newton's method converges extremely fast if the approximation is accurate and the step size is sufficiently small.

Newton's Method (cont.)

- Approximate the Hessian $H(\mathbf{w})$ with its diagonal to combine Newton's method and gradient descent, scaling steps by the inverse Hessian.
- Use gradient descent first, then switch to Newton's method near the optimum for stability and accuracy.



Limitations of linear regression

- It is possible that the best fitted line is still far off the real pattern of samples:

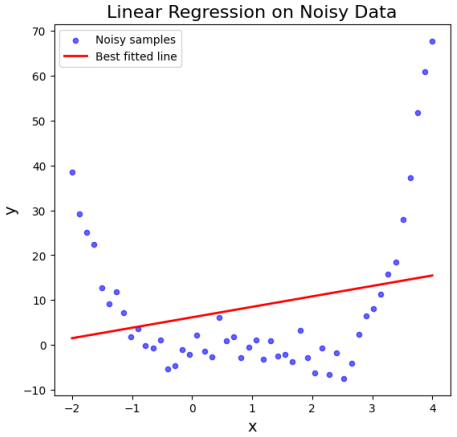


Figure 1: No line can be fitted to generalize on these samples

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

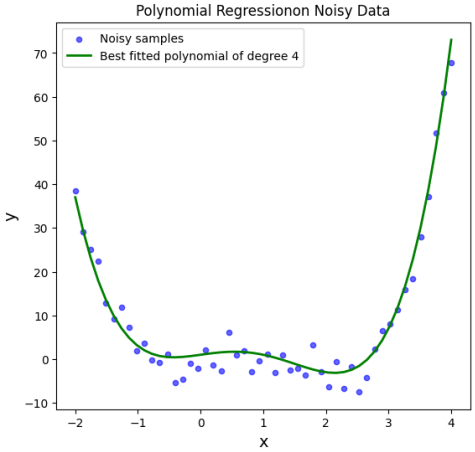


Figure 2: Best fitted polynomial of degree 4 can generalize well on samples

Beyond Linear Regression: change of basis

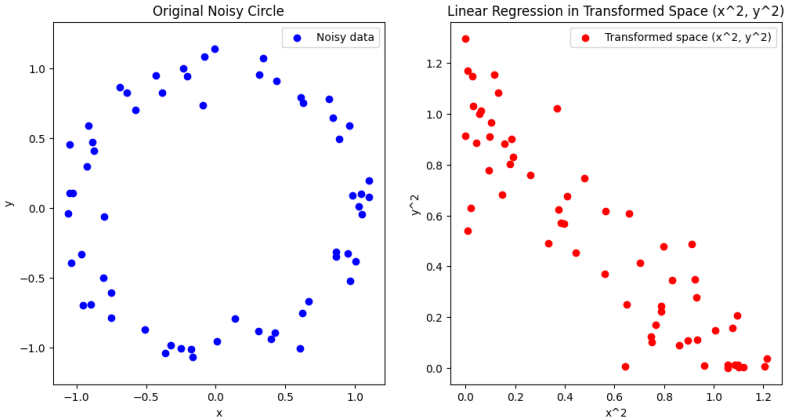


Figure 3: Changing the basis of $[1, x, y]$ to $[1, x^2, y^2]$, we can use linear regression

Polynomial regression: Univariate

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

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

- **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

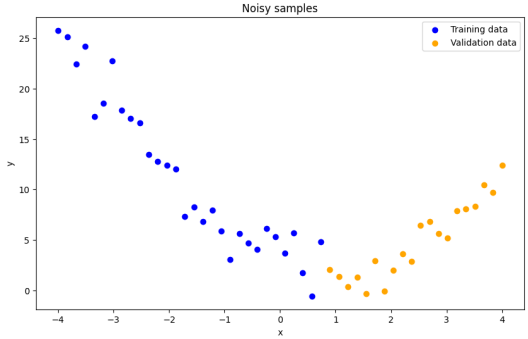
$$\hat{\mathbf{w}} = \left(\mathbf{X}'^T\mathbf{X}'\right)^{-1}\mathbf{X}'^T y$$

Training and Validation

- To better distinct between linear regression and polynomial regression, we will show that linear model can not generalize well.
- Assume the samples are split into two subsets. **Train dataset** which is used to train a regression model. As well as a **Validation dataset** to find the best regression model for an application.
- If a model can generalize well on validation set, it can be a good candidate.

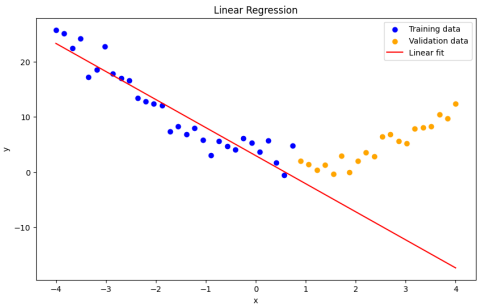
Polynomial regression: example

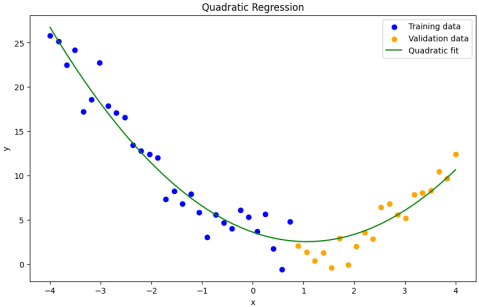
- Consider the following noisy samples



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.

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

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

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

- **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 $p(x, y)$

$$J(w) = \mathbb{E}_{p(x,y)}[(y - h_w(x))^2]$$

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

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

- **Training (empirical) error**:

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

- **Test error**:

$$J_{\text{test}}(w) = \frac{1}{m} \sum_{i=1}^{m} \left(y_{\text{test}}^{(i)} - h_w(x_{\text{test}}^{(i)}) \right)^2$$

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

Overfitting Definition

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

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

- 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

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

- 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 $< \epsilon$
- Test error $> \epsilon$

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.

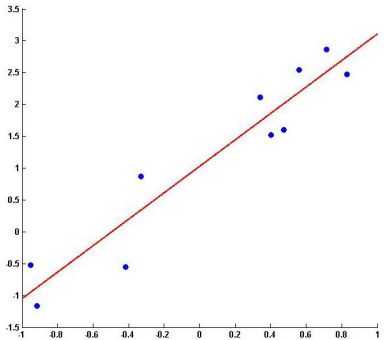
Symptoms:

- Training error $> \epsilon$

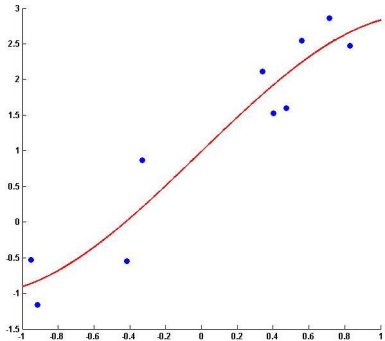
Remedies:

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

Generalization: polynomial regression



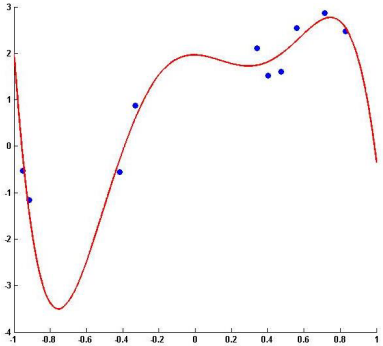
Degree of 1



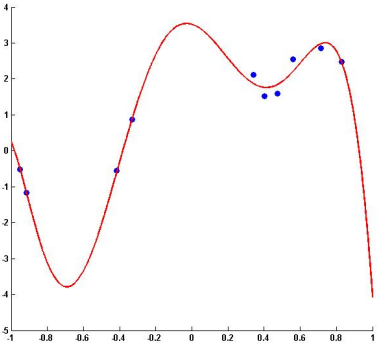
Degree of 3

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

Overfitting: polynomial regression



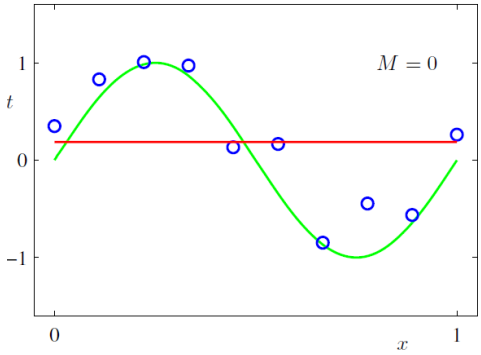
Degree of 5

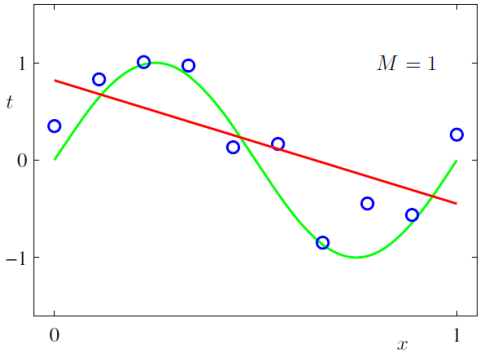


Degree of 7

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

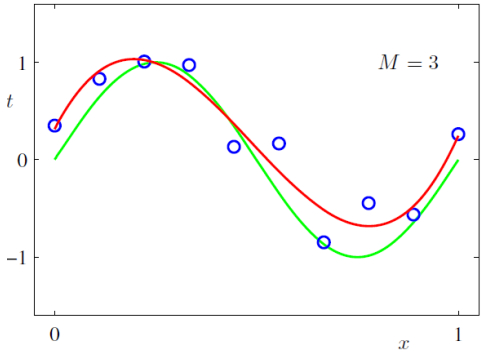
Polynomial regression with various degrees: example

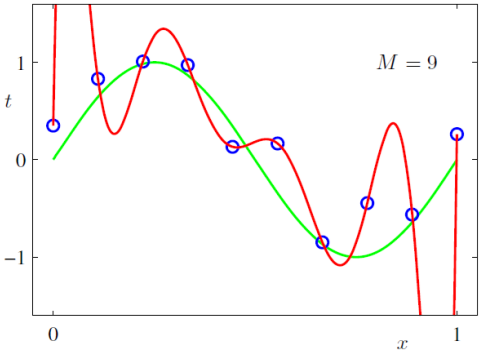




Figures adapted from Machine Learning and Pattern Recognition, Bishop

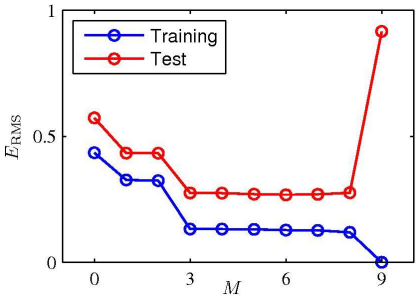
Polynomial regression with various degrees: example (cont.)





Figures adapted from Machine Learning and Pattern Recognition, Bishop

Root mean squared error



$$E_{RMS} = \sqrt{\frac{\sum_{i=1}^{n} \left(y^{(i)} - f(\mathbf{x}^{(i)}; \mathbf{w})\right)^2}{n}}$$

Figures adapted from Machine Learning and Pattern Recognition, Bishop

Generalization error decomposition:

$$\mathbb{E}[(y - h_w(x))^2] = (\text{Bias})^2 + \text{Variance} + \text{Noise}$$

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

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

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

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

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

High Bias in Simple Models

Explanation: Simple models, such as linear regression, often underfit

$$h_w(x) = w_0 + w_1 x$$

- Bias remains large even with infinite data

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

- Leads to large generalization error

High Variance in Complex Models

Explanation: Complex models tend to overfit

$$h_w(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m$$

- Variance dominates when the model is too complex

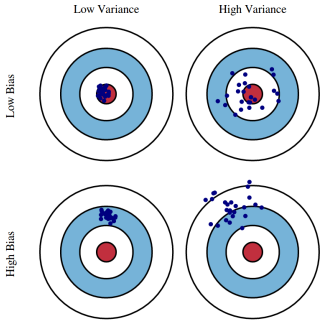
$$\text{Variance} \gg \text{Bias}$$

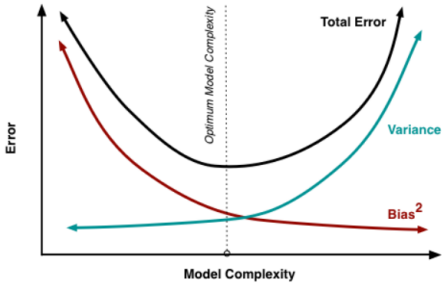
- Fits noise, leading to high test error

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





Purpose: Prevent overfitting by penalizing large weights

$$J_\lambda(w) = J(w) + \lambda R(w)$$

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

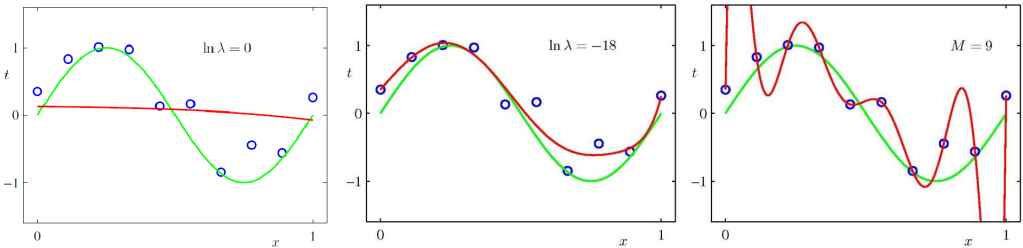
Effect of Regularization Parameter λ

Balancing Fit and Complexity:

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

- 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



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

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

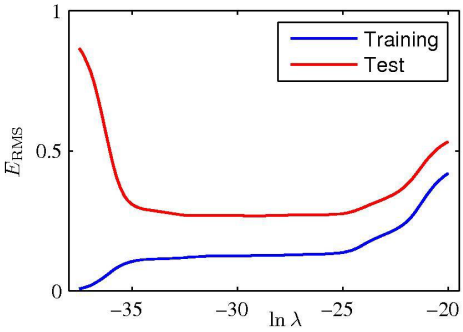
Figures adapted from Machine Learning and Pattern Recognition, Bishop

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

Regularization parameter



- λ controls the effective complexity of the model
- hence the degree of overfitting

Figures adapted from Machine Learning and Pattern Recognition, Bishop

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

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

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

Curve Fitting with Noise

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

$$y = f(\mathbf{x}; \mathbf{w}) + \epsilon$$

- 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 $\mathbf{x}$.

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

- Goal: Learn a function $f(\mathbf{x}; \mathbf{w})$ 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 $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, MLE maximizes:

$$L(\mathscr{D}; \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2)$$

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

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

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

- 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:

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

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

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

- **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:

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

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

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

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

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

Estimating σ^2

- The maximum likelihood estimate of the noise variance σ^2:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f(\mathbf{x}^{(i)};\hat{\mathbf{w}})\right)^2$$

- Interpretation: Mean squared error of the predictions.
- Note: σ^2 reflects the noise level in the observations.

Maximum A Posteriori Estimation (MAP)

- **MAP:** Estimate parameters by maximizing the posterior:

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathscr{D}) \propto \arg\max_{\mathbf{w}} \prod_{i=1}^{n} p(y_i \mid \mathbf{x}_i, \mathbf{w}, \sigma^2)\, p(\mathbf{w})$$

- Equivalently, maximize the log-posterior:

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i, \mathbf{w}, \sigma^2) + \log p(\mathbf{w})$$

- In MAP with a prior, large datasets wash out the prior, so the posterior effectively matches the data-only estimate ($\approx$ MLE).
- When data are scarce, the prior injects knowledge that shapes the posterior, making MAP meaningfully different from MLE.

MAP with a Gaussian Prior

- **Prior (additional assumption):** $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I})$

$$p(\mathbf{w}) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{\mathbf{w}^\top \mathbf{w}}{2\tau^2}\right)$$

- With Gaussian noise $y_i = \mathbf{x}_i^\top \mathbf{w} + \varepsilon_i, \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, maximizing the log-posterior is (up to constants) equivalent to:

$$\arg\min_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \frac{1}{2\tau^2} \|\mathbf{w}\|_2^2$$

- Defining $\lambda = \dfrac{\sigma^2}{n\tau^2}$, the MAP objective becomes

$$J_{\text{MAP}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \lambda \|\mathbf{w}\|_2^2$$

- **Equivalence:** This is Ridge (Regularized) Regression.

Contributions

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

[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.