Summer Class: Machine Learning and Data Science

Shuaifeng Li

Department of Physics, University of Michigan

June & July 2025

Table of Contents

Table of Contents

Overview of Machine Learning – History

Overview of Machine Learning – History

- 1700s–1800s: Optimization in Mathematics

Overview of Machine Learning – History

- 1700s–1800s: Optimization in Mathematics
  - Calculus of Variations – Euler, Lagrange, essentially early optimization: finding the path, shape, or function that minimizes (or maximizes) a quantity.
  - Least squares – Carl Friedrich Gauss introduced the least squares method to predict the orbit of Ceres from noisy astronomical data.

Overview of Machine Learning – History

- 1700s–1800s: Optimization in Mathematics
  - Calculus of Variations – Euler, Lagrange, essentially early optimization: finding the path, shape, or function that minimizes (or maximizes) a quantity.
  - Least squares – Carl Friedrich Gauss introduced the least squares method to predict the orbit of Ceres from noisy astronomical data.
- 1943–1960s: Early Neural Models

Overview of Machine Learning – History

- 1700s–1800s: Optimization in Mathematics
  - Calculus of Variations – Euler, Lagrange, essentially early optimization: finding the path, shape, or function that minimizes (or maximizes) a quantity.
  - Least squares – Carl Friedrich Gauss introduced the least squares method to predict the orbit of Ceres from noisy astronomical data.
- 1943–1960s: Early Neural Models
  - Logical circuits as brain model – McCulloch & Pitts neuron: binary inputs, weighted summation, and thresholding.
  - Perceptron (1958) – Frank Rosenblatt introduces linear classifier trained via weight updates.

Overview of Machine Learning – History

Unfortunately... In 1969, Minsky and Papert mathematically proved that a single-layer perceptron cannot solve problems like XOR (Exclusive or, non-linearly separable). The community was disappointed on neural networks for decades, focusing instead on symbolic AI.

Overview of Machine Learning – History

- 1969–1990s: Statistical Learning Takes Over

Overview of Machine Learning – History

- 1969–1990s: Statistical Learning Takes Over
  - Support Vector Machine (SVM) – Vapnik solved convex quadratic programming problems for robust classification.
  - Regularization – Optimization + generalization (bias-variance tradeoff).

Overview of Machine Learning – History

- 1969–1990s: Statistical Learning Takes Over
  - Support Vector Machine (SVM) – Vapnik solved convex quadratic programming problems for robust classification.
  - Regularization – Optimization + generalization (bias-variance tradeoff).
- Backpropagation & Neural Revival
  - Backpropagation (1986) – Gradient descent through a network using chain rule, rediscovered by Rumelhart, Hinton, and Williams.

Overview of Machine Learning – History

- 1969–1990s: Statistical Learning Takes Over
  - Support Vector Machine (SVM) – Vapnik solved convex quadratic programming problems for robust classification.
  - Regularization – Optimization + generalization (bias-variance tradeoff).
- Backpropagation & Neural Revival
  - Backpropagation (1986) – Gradient descent through a network using chain rule, rediscovered by Rumelhart, Hinton, and Williams.
- 2000s–Today: Optimization Powers Everything
  - Convex Optimization – Well-understood optimization algorithms like LASSO, Ridge Regression, and Logistic Regression train simple, interpretable models.
  - Deep Learning – Backpropagation trains deep nets with millions of parameters.
  - Reinforcement Learning – Optimizing long-term rewards (policy gradients, Q-learning).

Modern AI Achievements

- **AI for Classification & Prediction:** AI that understands and categorizes existing data.

Modern AI Achievements

- **AI for Classification & Prediction:** AI that understands and categorizes existing data.
  - IBM's Watson: Advanced Question-Answering & Information Retrieval. Understanding human language to find the single correct answer in a massive dataset.

Modern AI Achievements

- **AI for Classification & Prediction:** AI that understands and categorizes existing data.
  - IBM's Watson: Advanced Question-Answering & Information Retrieval. Understanding human language to find the single correct answer in a massive dataset.
  - AlexNet: Deep Convolutional Neural Networks. Its dominant victory in the ImageNet competition was the "big bang" for deep learning.

Modern AI Achievements

- **AI for Strategic Reasoning:** AI that plans and strategizes in complex, rule-based systems.

Modern AI Achievements

- **AI for Strategic Reasoning:** AI that plans and strategizes in complex, rule-based systems.
  - DeepMind's AlphaGo: Deep Neural Networks combined with Monte Carlo Tree Search and Reinforcement Learning. Defeated the world's best Go player. It learns by playing against itself.

Modern AI Achievements

- **AI for Generation:** AI that creates entirely new, original content.

Modern AI Achievements

- **AI for Generation:** AI that creates entirely new, original content.
  - Generative Adversarial Networks (GANs): Introduced the concept of two AIs (a "generator" and a "discriminator") competing to produce novel, lifelike images/videos, unlocking the potential for AI-driven creativity.

Modern AI Achievements

- **AI for Generation:** AI that creates entirely new, original content.
  - Generative Adversarial Networks (GANs): Introduced the concept of two AIs (a "generator" and a "discriminator") competing to produce novel, lifelike images/videos, unlocking the potential for AI-driven creativity.
  - 2021: DALL-E: Text-to-Image Generation. Bridged the gap between language and vision. It made generative AI accessible by allowing users to create complex, original images from simple text prompts.

Modern AI Achievements

- **Large Language Models (LLMs):** AI that understands, generates, and converses using human language.

Modern AI Achievements

- **Large Language Models (LLMs):** AI that understands, generates, and converses using human language.
  - The Transformer Architecture: Advanced Language Understanding. This paper ("Attention Is All You Need") introduced the architecture that underpins all modern LLMs, allowing AI to process language with a deep awareness of context.

Modern AI Achievements

- **Large Language Models (LLMs):** AI that understands, generates, and converses using human language.
  - The Transformer Architecture: Advanced Language Understanding. This paper ("Attention Is All You Need") introduced the architecture that underpins all modern LLMs, allowing AI to process language with a deep awareness of context.
  - GPT-2→GPT-3: Coherent Text Generation. Produced text that is not only grammatically correct but also logically consistent and easy to understand.

Modern AI Achievements

- **Large Language Models (LLMs):** AI that understands, generates, and converses using human language.
  - The Transformer Architecture: Advanced Language Understanding. This paper ("Attention Is All You Need") introduced the architecture that underpins all modern LLMs, allowing AI to process language with a deep awareness of context.
  - GPT-2→GPT-3: Coherent Text Generation. Produced text that is not only grammatically correct but also logically consistent and easy to understand.
  - GPT-4 and Many Others: Multimodal Reasoning. These models blur the lines between AI classes. They are fundamentally LLMs but also possess generative and classification capabilities across different data types (text, images, audio, video, code). This convergence is the current state-of-the-art.

Modern AI Achievements

- **Hugging Face: Chatbot Arena LLM Leaderboard**

Modern AI Achievements

- **Hugging Face: Chatbot Arena LLM Leaderboard**

Model
Gemini-2.5-Pro-Preview-06-05
Gemini-2.5-Pro-Preview-05-06
o3-2025-04-16
ChatGPT-4o-latest (2025-03-26)
DeepSeek-R1-0528
Gemini-2.5-Flash-Preview-05-20
Grok-3-Preview-02-24
GPT-4.5-Preview
Gemini-2.5-Flash-Preview-04-17
Qwen3-235B-A22B-no-thinking

(a) Overall

Model
Gemini-2.5-Pro-Preview-06-05
Gemini-2.5-Pro-Preview-05-06
Gemini-2.5-Flash-Preview-05-20
o3-2025-04-16
Qwen3-235B-A22B
Qwen3-235B-A22B-no-thinking
GPT-4.5-Preview
o4-mini-2025-04-16
DeepSeek-R1-0528
Gemini-2.5-Flash-Preview-04-17

(b) Math

Model
Gemini-2.5-Pro-Preview-06-05
Gemini-2.5-Pro-Preview-05-06
o3-2025-04-16
ChatGPT-4o-latest (2025-03-26)
Grok-3-Preview-02-24
Gemini-2.5-Flash-Preview-05-20
DeepSeek-R1-0528
GPT-4.5-Preview
Claude Opus 4 (20250514)
Qwen3-235B-A22B-no-thinking

(c) Coding

Machine Learning (ML)

Machine Learning (ML)

- Not a magic.

Machine Learning (ML)

- Not a magic.
- Not just neural networks.

Machine Learning (ML)

- Not a magic.
- Not just neural networks.
- Learn from data, identify patterns, and make decisions with minimal human intervention.

Machine Learning (ML)

- Not a magic.
- Not just neural networks.
- Learn from data, identify patterns, and make decisions with minimal human intervention.

Supervised Learning

Learning with a “supervisor” (labeled data).

Unsupervised Learning

Discovering patterns in unlabeled data.

Semi-supervised Learning (Reinforcement Learning)

Learning through trial and error with rewards/penalties.

Supervised Learning

Algorithm learns from a dataset where each data point is labeled with a known outcome.

Supervised Learning

Algorithm learns from a dataset where each data point is labeled with a known outcome.
Goal: Learn a mapping function to predict outputs for new, unseen inputs.

Supervised Learning

Algorithm learns from a dataset where each data point is labeled with a known outcome.

Goal: Learn a mapping function to predict outputs for new, unseen inputs.

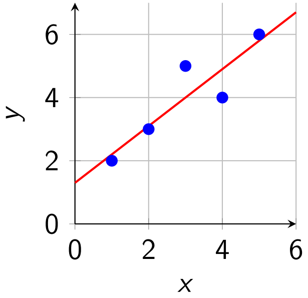
Common Tasks:

- **Classification:** Output is a category (e.g.,"spam" or "not spam", "cat" or "dog").
- **Regression:** Output is a continuous value (e.g., predicting house prices, stock values).

Supervised Learning – Regression and Classification

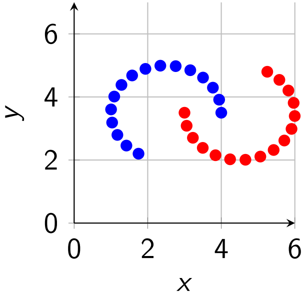
Regression

Goal: Model the relationship between a dependent variable (response) and one or more independent variables (predictors) by fitting an equation.



Classification

Goal: Model decision boundaries that separate input data points to discrete categories or classes in the feature space.



Unsupervised Learning

Algorithm learns from unlabeled dataset.
Goal: Discover hidden patterns, structures, or relationships within the data itself.

Unsupervised Learning

Algorithm learns from unlabeled dataset.
Goal: Discover hidden patterns, structures, or relationships within the data itself.
Common Tasks:

- **Clustering:** Grouping similar data points together.
- **Dimensionality Reduction:** Reducing the number of variables while preserving important information (e.g., Principal Component Analysis).
- **Association Rule Mining:** Discovering potential relationships (e.g., "People who watch *Iron Man 1* **probably** also watch *Iron Man 2*.").

Unsupervised Learning – Clustering

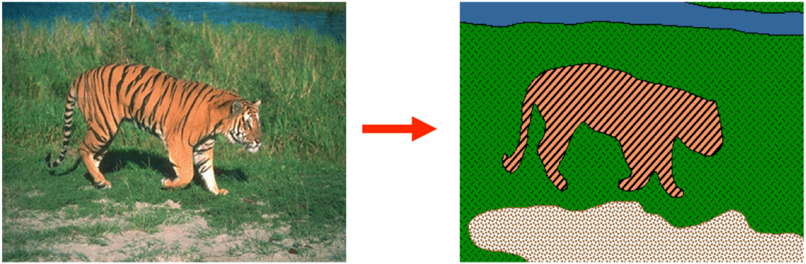


Figure: Image segmentation using K-means algorithm.

Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.

Reinforcement Learning

An “agent” learns by interacting with an “environment”. Receives “rewards” or “penalties” for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

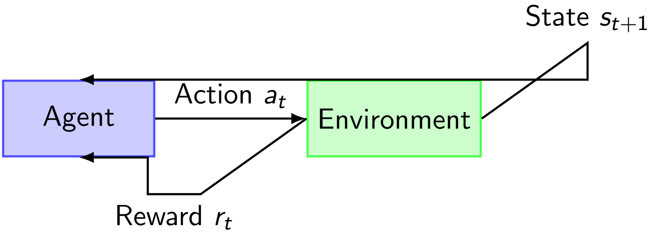
Common Tasks: Training robots (Locomotion, grasping), game playing, self-driving cars.

Reinforcement Learning

An “agent” learns by interacting with an “environment”. Receives “rewards” or “penalties” for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

Common Tasks: Training robots (Locomotion, grasping), game playing, self-driving cars.

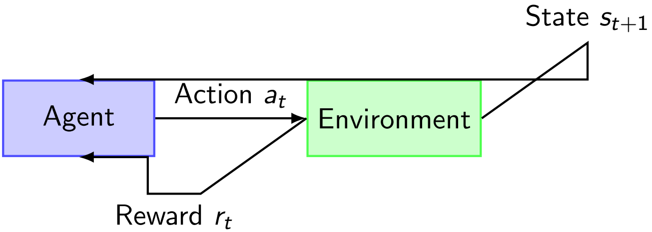


Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

Common Tasks: Training robots (Locomotion, grasping), game playing, self-driving cars.



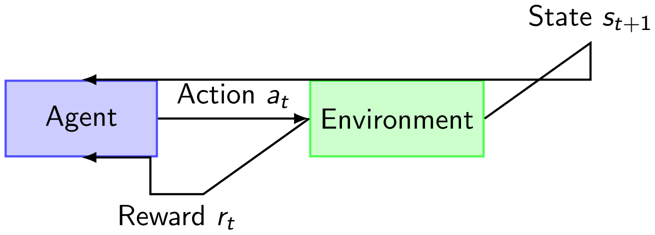
- Agent observes a state, takes an action.

Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

Common Tasks: Training robots (Locomotion, grasping), game playing, self-driving cars.



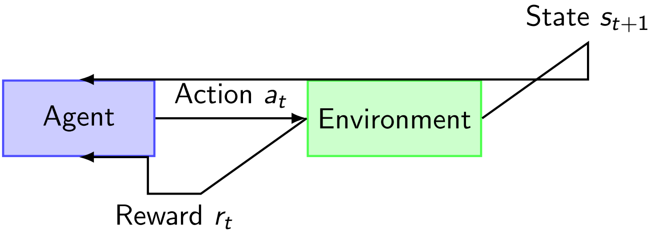
- Agent observes a state, takes an action.
- Environment returns a new state and a reward.

Reinforcement Learning

An "agent" learns by interacting with an "environment". Receives "rewards" or "penalties" for actions. Learning by trial and error.
Goal: Maximize cumulative reward.

- No labeled input/output pairs (unsupervised learning).
- Feedback is given in terms of rewards after actions (supervised learning).

Common Tasks: Training robots (Locomotion, grasping), game playing, self-driving cars.



- Agent observes a state, takes an action.
- Environment returns a new state and a reward.
- The agent updates its strategy (policy).

Classical Machine Learning vs Deep Learning

Aspect	**Classical Machine Learning:** Mathematically predefined models.	**Deep Learning:** Artificial neural networks with multiple layers ("deep"). Inspired by human brain.
Feature Extraction	Hand-crafted features (e.g., SIFT, PCA, statistical measures)	Automatically learned from raw data (via neural network layers)
Algorithm Examples	Decision Trees, SVM, k-Nearest Neighbors (KNN), Logistic Regression	Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Transformers
Data Requirements	Works well with smaller, clean datasets	Requires large volumes of labeled data to perform well
Model Complexity	Shallow, interpretable models with fewer parameters	Deep, multi-layered models with millions of parameters
Computational Power	Low to moderate, runs on CPU	High, typically requires GPUs/TPUs for training

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.
- **Loss Function (Cost/Objective Function)**
  - Measures how well the model is performing.
  - Quantifies the "error" or difference between model predictions and actual (ground truth) labels.

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.
- **Loss Function (Cost/Objective Function)**
  - Measures how well the model is performing.
  - Quantifies the "error" or difference between model predictions and actual (ground truth) labels.
  - Example
    - Regression: Mean Squared Error, $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
    - Classification: Cross-Entropy Loss, $\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$.

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.
- **Loss Function (Cost/Objective Function)**
  - Measures how well the model is performing.
  - Quantifies the "error" or difference between model predictions and actual (ground truth) labels.
  - Example
    - Regression: Mean Squared Error, $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
    - Classification: Cross-Entropy Loss, $\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$.
- **Goal of Optimization:** Find model parameters (weights, biases) that minimize the loss function.

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.
- **Loss Function (Cost/Objective Function)**
  - Measures how well the model is performing.
  - Quantifies the "error" or difference between model predictions and actual (ground truth) labels.
  - Example
    - Regression: Mean Squared Error, $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
    - Classification: Cross-Entropy Loss, $\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$.
- **Goal of Optimization:** Find model parameters (weights, biases) that minimize the loss function.
- **Optimization Algorithms:**
  - Used to adjust parameters to reduce loss.
  - Most common: Gradient Descent and its variants.

Training as Optimization

- **Core Idea:** Training a machine learning model is an optimization problem.
- **Loss Function (Cost/Objective Function)**
  - Measures how well the model is performing.
  - Quantifies the "error" or difference between model predictions and actual (ground truth) labels.
  - Example
    - Regression: Mean Squared Error, $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.
    - Classification: Cross-Entropy Loss, $\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$.
- **Goal of Optimization:** Find model parameters (weights, biases) that minimize the loss function.
- **Optimization Algorithms:**
  - Used to adjust parameters to reduce loss.
  - Most common: Gradient Descent and its variants.
  - This is where **convex optimization** becomes highly relevant, as many ML loss functions are designed to be convex.

Table of Contents

What is Optimization?

- **Broad Definition:** Making the best possible decisions under given circumstances.

What is Optimization?

- **Broad Definition:** Making the best possible decisions under given circumstances.
- **Mathematical Formulation:**

$$\begin{array}{rll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \ldots, m \\ & h_j(x) = 0, & j = 1, \ldots, p \end{array}$$

  - $x \in \mathbb{R}^n$: vector of optimization variables
  - $f_0(x)$: objective function to minimize
  - $f_i(x) \leq 0$: inequality constraints
  - $h_j(x) = 0$: equality constraints

What is Optimization?

- **Broad Definition:** Making the best possible decisions under given circumstances.
- **Mathematical Formulation:**

$$\begin{array}{rll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0, & i = 1, \ldots, m \\ & h_j(x) = 0, & j = 1, \ldots, p \end{array}$$

  - $x \in \mathbb{R}^n$: vector of optimization variables
  - $f_0(x)$: objective function to minimize
  - $f_i(x) \leq 0$: inequality constraints
  - $h_j(x) = 0$: equality constraints
- Unfortunately, most are **intractable**. The exception: convex optimization.

Convex Set

- A shape with no “holes” or “indentations”. A line segment between any two points in the set lies entirely within the set.

Convex Set

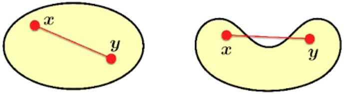
- A shape with no “holes” or “indentations”. A line segment between any two points in the set lies entirely within the set.
- **Definition:**

$$C \text{ is convex if } \forall x, y \in C,\ \forall \theta \in [0,1],\ \theta x + (1-\theta) y \in C$$

Convex Set

- A shape with no "holes" or "indentations". A line segment between any two points in the set lies entirely within the set.
- **Definition:**

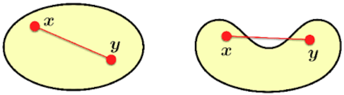
$$C \text{ is convex if } \forall x, y \in C,\ \forall \theta \in [0,1],\ \theta x + (1-\theta) y \in C$$



Convex Set

- A shape with no "holes" or "indentations". A line segment between any two points in the set lies entirely within the set.
- **Definition:**

$$C \text{ is convex if } \forall x, y \in C,\ \forall \theta \in [0,1],\ \theta x + (1-\theta)y \in C$$



- **Examples:**
  - Euclidean space: $\mathbb{R}^n$
  - Affine subspace: $\{x \in \mathbb{R}^n \mid Ax = b\}$
  - Half-space: $\{x \in \mathbb{R}^n \mid a^T x \leq b\}$
  - Polyhedron: $\{x \in \mathbb{R}^n \mid Ax \leq b\}$
  - Norm ball (general): $\{x \in \mathbb{R}^n \mid \|x - x_c\| \leq r\}$, for any norm $\|\cdot\|$
  - Intersection of Convex Sets

Convex Function

- Graph has a “cup-like” shape. Line segment between any two points on the graph lies on or above the graph.

Convex Function

- Graph has a "cup-like" shape. Line segment between any two points on the graph lies on or above the graph.
- **Definition:** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex if

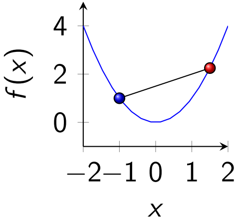
$$\forall x, y \in \mathrm{dom}(f),\ \theta \in [0,1], f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

Convex Function

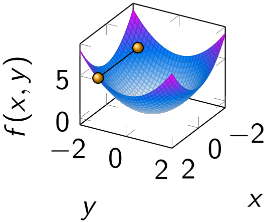
- Graph has a "cup-like" shape. Line segment between any two points on the graph lies on or above the graph.
- **Definition:** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex if

$$\forall x, y \in \mathrm{dom}(f),\ \theta \in [0,1], f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

$f(x) = x^2$



$f(x, y) = x^2 + y^2$

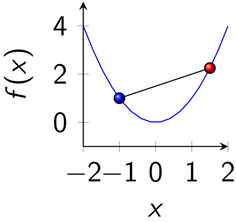


Convex Function

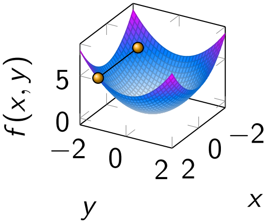
- Graph has a "cup-like" shape. Line segment between any two points on the graph lies on or above the graph.
- **Definition:** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex if

$$\forall x, y \in \mathrm{dom}(f),\ \theta \in [0,1], f(\theta x + (1-\theta)y) \le \theta f(x) + (1-\theta) f(y)$$

$f(x) = x^2$



$f(x,y) = x^2 + y^2$



- **Strictly Convex:** Inequality is strict ($<$) for $x \neq y$, $\theta \in [0,1]$. Has at most one global minimum.

Convex Function – Properties

Epigraph Condition

f is convex $\iff$ its epigraph is a convex set.

First-Order Condition if f is continuously differentiable

f is convex $\iff$ $\mathrm{dom}(f)$ is convex and
$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y \in \mathrm{dom}(f)$.

Second-Order Condition if f is twice continuously differentiable

f is convex $\iff$ $\mathrm{dom}(f)$ is convex and Hessian $\nabla^2 f(x) \succeq 0$.

Convex Function – Examples

- Common Convex Functions:
  - Linear/Affine: $a^T x + b$
  - Quadratic: $x^T Q x + b^T x + c, Q \succeq 0$
  - Norms: $\|x\|_p$
  - Exponential: e^{ax}
  - Negative Logarithm: $-\log(x)$ for $x > 0$

Convex Function – Examples

- Common Convex Functions:
  - Linear/Affine: $a^T x + b$
  - Quadratic: $x^T Qx + b^T x + c, Q \succeq 0$
  - Norms: $\|x\|_p$
  - Exponential: e^{ax}
  - Negative Logarithm: $-\log(x)$ for $x > 0$
- Operation Preserving Convexity:
  - Sum of convex functions.
  - Non-negative weighted sum of convex functions.
  - Composition with affine mapping: $f(Ax + b)$ if f is convex.

Convex Function – Optimization

$$
\begin{aligned}
\text{minimize} \quad & f_0(x) \\
\text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& h_j(x) = 0, \quad j = 1, \ldots, p
\end{aligned}
$$

- $x \in \mathbb{R}^n$: optimization variables
- $f_0(x)$: convex objective function
- $f_i(x) \leq 0$: convex inequality constraint functions
- $h_j(x) = 0$: affine equality constraint functions

Convex Function – Optimization

$$
\begin{aligned}
\text{minimize} \quad & f_0(x) \\
\text{subject to} \quad & f_i(x) \le 0, \quad i = 1, \ldots, m \\
& h_j(x) = 0, \quad j = 1, \ldots, p
\end{aligned}
$$

- $x \in \mathbb{R}^n$: optimization variables
- $f_0(x)$: convex objective function
- $f_i(x) \le 0$: convex inequality constraint functions
- $h_j(x) = 0$: affine equality constraint functions

Special classes:

- Linear Programming (LP): $f_0(x)$ and $f_i(x)$ are linear (affine).
- Quadratic Programming (QP): $f_0(x)$ is convex quadratic and $f_i(x)$ are linear (affine). Least Squares: $\min \frac{1}{2}\|Ax - b\|_2^2$, s.t., $Rx \preceq d$.

Optimization Algorithms

- Most algorithms are iterative: Start with initial guess x_0, generate $x_1, x_2, \cdots \to x^*$.

Optimization Algorithms

- Most algorithms are iterative: Start with initial guess x_0, generate $x_1, x_2, \cdots \to x^*$.
- Common Update Rule: $x_{k+1} = x_k + \alpha_k d_k$
  - d_k: search direction
  - $\alpha_k > 0$: step size (learning rate)

Optimization Algorithms

- Most algorithms are iterative: Start with initial guess x_0, generate $x_1, x_2, \cdots \to x^*$.
- Common Update Rule: $x_{k+1} = x_k + \alpha_k d_k$
  - d_k: search direction
  - $\alpha_k > 0$: step size (learning rate)
- Challenge: Choose effective d_k and α_k.

Gradient Descent

- Core Idea: Iteratively move in the direction opposite to the gradient (steepest descent).

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

Gradient Descent

- Core Idea: Iteratively move in the direction opposite to the gradient (steepest descent).

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

- Suitability: Differentiable convex and non-convex function. Converges to global minimum for convex f if α_k is chosen well.

Gradient Descent

- Core Idea: Iteratively move in the direction opposite to the gradient (steepest descent).

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

- Suitability: Differentiable convex and non-convex function. Converges to global minimum for convex f if α_k is chosen well.
- Choice of α_k is critical.
  - Too small α_k: small convergence.
  - Too large α_k: overshoot, divergence.

  No convergence is guaranteed. For convergence, an additional *line search* is required.

Gradient Descent

- Core Idea: Iteratively move in the direction opposite to the gradient (steepest descent).

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

- Suitability: Differentiable convex and non-convex function. Converges to global minimum for convex f if α_k is chosen well.
- Choice of α_k is critical.
  - Too small α_k: small convergence.
  - Too large α_k: overshoot, divergence.

  No convergence is guaranteed. For convergence, an additional *line search* is required.
- MATLAB Demonstration: $f(x) = \frac{1}{2} x^T \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} x$.

Accelerated Gradient Descent

- Core Idea: Improves gradient descent convergence rate for smooth convex functions by incorporating "momentum" from previous steps.

$$y_k = x_k + \beta_k(x_k - x_{k-1})$$

Accelerated Gradient Descent

- Core Idea: Improves gradient descent convergence rate for smooth convex functions by incorporating "momentum" from previous steps.

$$y_k = x_k + \beta_k(x_k - x_{k-1})$$

$$x_{k+1} = y_k - \alpha_k \nabla f(y_k)$$

Gradient evaluated at "lookahead" point y_k.

Accelerated Gradient Descent

- Core Idea: Improves gradient descent convergence rate for smooth convex functions by incorporating "momentum" from previous steps.

$$y_k = x_k + \beta_k(x_k - x_{k-1})$$

$$x_{k+1} = y_k - \alpha_k \nabla f(y_k)$$

Gradient evaluated at "lookahead" point y_k.

- Suitability: Smooth, convex problems where faster convergence than gradient descent is desired.

Accelerated Gradient Descent

- Core Idea: Improves gradient descent convergence rate for smooth convex functions by incorporating "momentum" from previous steps.

$$y_k = x_k + \beta_k(x_k - x_{k-1})$$

$$x_{k+1} = y_k - \alpha_k \nabla f(y_k)$$

  Gradient evaluated at "lookahead" point y_k.
- Suitability: Smooth, convex problems where faster convergence than gradient descent is desired.
- MATLAB Demonstration: $f(x) = \frac{1}{2}x^T \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} x$.

Newton's Method

- Core Idea: Uses a second-order approximation of the function f at the current point x_k and takes a step towards the minimum of this quadratic.

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x_k) \Delta x$$

Newton's Method

- Core Idea: Uses a second-order approximation of the function f at the current point x_k and takes a step towards the minimum of this quadratic.

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x_k) \Delta x$$

$$\nabla f(x_k) + H_f(x_k)\Delta x = 0 \Rightarrow \Delta x = -[H_f(x_k)]^{-1} \nabla f(x_k)$$

Newton's Method

- Core Idea: Uses a second-order approximation of the function f at the current point x_k and takes a step towards the minimum of this quadratic.

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x_k) \Delta x$$

$$\nabla f(x_k) + H_f(x_k) \Delta x = 0 \Rightarrow \Delta x = -[H_f(x_k)]^{-1} \nabla f(x_k)$$

$$x_{k+1} = x_k - [H_f(x_k)]^{-1} \nabla f(x_k)$$

Newton's Method

- Core Idea: Uses a second-order approximation of the function f at the current point x_k and takes a step towards the minimum of this quadratic.

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x_k) \Delta x$$

$$\nabla f(x_k) + H_f(x_k)\Delta x = 0 \Rightarrow \Delta x = -[H_f(x_k)]^{-1} \nabla f(x_k)$$

$$x_{k+1} = x_k - [H_f(x_k)]^{-1} \nabla f(x_k)$$

- Suitability: Twice differentiable convex function. Hessian is available, positive definite and not too costly to compute/invert.

Newton's Method

- Core Idea: Uses a second-order approximation of the function f at the current point x_k and takes a step towards the minimum of this quadratic.

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x_k)\Delta x$$

$$\nabla f(x_k) + H_f(x_k)\Delta x = 0 \Rightarrow \Delta x = -[H_f(x_k)]^{-1}\nabla f(x_k)$$

$$x_{k+1} = x_k - [H_f(x_k)]^{-1}\nabla f(x_k)$$

- Suitability: Twice differentiable convex function. Hessian is available, positive definite and not too costly to compute/invert.
- MATLAB Demonstration: $f(x) = \frac{1}{2}x^T \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} x$.

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

$$\nabla f(x_k) \approx \nabla f(x_{k+1}) + H_f(x_{k+1})(x_k - x_{k+1})$$

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

$$\nabla f(x_k) \approx \nabla f(x_{k+1}) + H_f(x_{k+1})(x_k - x_{k+1})$$

$B_{k+1} s_k = y_k$ with $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$.

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

$$\nabla f(x_k) \approx \nabla f(x_{k+1}) + H_f(x_{k+1})(x_k - x_{k+1})$$

$B_{k+1} s_k = y_k$ with $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_s} + \frac{y_k y_k^T}{y_k^T s_k}$$

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

$$\nabla f(x_k) \approx \nabla f(x_{k+1}) + H_f(x_{k+1})(x_k - x_{k+1})$$

$B_{k+1} s_k = y_k$ with $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_s} + \frac{y_k y_k^T}{y_k^T s_k}$$

- Suitability: Smooth. Faster convergence when Hessians are costly.

Quasi-Newton Method

- Core Idea: Newton's method is expensive (Hessian calculation and Hessian inversion). Quasi-Newton methods avoid these issues by approximating the Hessian matrix (or its inverse) using only first-order information (gradients).
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

$$x_{k+1} = x_k - B_k^{-1} \nabla f(x_k)$$

$$\nabla f(x_k) \approx \nabla f(x_{k+1}) + H_f(x_{k+1})(x_k - x_{k+1})$$

$B_{k+1} s_k = y_k$ with $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = x_{k+1} - x_k$.

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_s} + \frac{y_k y_k^T}{y_k^T s_k}$$

- Suitability: Smooth. Faster convergence when Hessians are costly.
- MATLAB Demonstration: $f(x) = \frac{1}{2} x^T \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} x$.

Line Search

- Core Idea: Not standalone. Procedure to find the appropriate step size α_k within other algorithms.

Line Search

- Core Idea: Not standalone. Procedure to find the appropriate step size α_k within other algorithms.
- Backtracking: Starting with large α, reduce by factor $\rho \in (0, 1)$ until sufficient decrease (e.g., Armijo-Goldstein condition) is met.

$$f(x_k + \alpha d_k) \leq f(x_k) + \rho \cdot \alpha \cdot \nabla f(x_k)^T d_k$$

Line Search

- Core Idea: Not standalone. Procedure to find the appropriate step size α_k within other algorithms.
- Backtracking: Starting with large α, reduce by factor $\rho \in (0, 1)$ until sufficient decrease (e.g., Armijo-Goldstein condition) is met.

$$f(x_k + \alpha d_k) \leq f(x_k) + \rho \cdot \alpha \cdot \nabla f(x_k)^T d_k$$

- Enhances robustness and efficiency by dynamically adjusting step size.

Line Search

- Core Idea: Not standalone. Procedure to find the appropriate step size α_k within other algorithms.
- Backtracking: Starting with large α, reduce by factor $\rho \in (0, 1)$ until sufficient decrease (e.g., Armijo-Goldstein condition) is met.

$$f(x_k + \alpha d_k) \leq f(x_k) + \rho \cdot \alpha \cdot \nabla f(x_k)^T d_k$$

- Enhances robustness and efficiency by dynamically adjusting step size.
- MATLAB Demonstration: $f(x) = \frac{1}{2} x^T \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} x$.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.
  - A vector g is a subgradient of f at x if $f(y) \geq f(x) + g^T(y - x)$ for all y. The tangent line is a global underestimator.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.
    - A vector g is a subgradient of f at x if $f(y) \geq f(x) + g^T(y - x)$ for all y. The tangent line is a global underestimator.
- Subdifferential $\partial f(x)$: The set of all subgradients at x.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.
  - A vector g is a subgradient of f at x if $f(y) \geq f(x) + g^T(y - x)$ for all y. The tangent line is a global underestimator.
- Subdifferential $\partial f(x)$: The set of all subgradients at x.
  - Example: For $f(x) = |x|$, $\partial f(x) = \mathrm{sign}(x)$ if $x \neq 0$, and $\partial f(0) = [-1, 1]$.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.
  - A vector g is a subgradient of f at x if $f(y) \geq f(x) + g^T(y - x)$ for all y. The tangent line is a global underestimator.
- Subdifferential $\partial f(x)$: The set of all subgradients at x.
  - Example: For $f(x) = |x|$, $\partial f(x) = \mathrm{sign}(x)$ if $x \neq 0$, and $\partial f(0) = [-1, 1]$.
- Update Rule: $x_{k+1} = x_k - \alpha_k g_k$, where g_k is any subgradient from $\partial f(x_k)$.

Subgradient Method

- Problem: What if the function isn't differentiable everywhere?
- Subgradient: A generalization of the gradient for non-differentiable convex functions.
  - A vector g is a subgradient of f at x if $f(y) \geq f(x) + g^T(y - x)$ for all y. The tangent line is a global underestimator.
- Subdifferential $\partial f(x)$: The set of all subgradients at x.
  - Example: For $f(x) = |x|$, $\partial f(x) = \mathrm{sign}(x)$ if $x \neq 0$, and $\partial f(0) = [-1, 1]$.
- Update Rule: $x_{k+1} = x_k - \alpha_k g_k$, where g_k is any subgradient from $\partial f(x_k)$.
- MATLAB Demonstration: $F(x) = \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$.

Proximal Gradient Descent

- Core Idea: For composite problems

$$\min_x F(x) = f(x) + g(x)$$

  - $f(x)$: Convex, differentiable.
  - $g(x)$: Convex, possibly non-differentiable.
  - Generalize gradient descent to handle non-smooth $g(x)$.

Proximal Gradient Descent

- Core Idea: For composite problems

$$\min_x F(x) = f(x) + g(x)$$

  - $f(x)$: Convex, differentiable.
  - $g(x)$: Convex, possibly non-differentiable.
  - Generalize gradient descent to handle non-smooth $g(x)$.
- Update Rule:

$$x_{k+1} = \mathbf{prox}_{\alpha_k g}(x_k - \alpha_k \nabla f(x_k))$$

Proximal Gradient Descent

- Core Idea: For composite problems

$$\min_x F(x) = f(x) + g(x)$$

  - $f(x)$: Convex, differentiable.
  - $g(x)$: Convex, possibly non-differentiable.
  - Generalize gradient descent to handle non-smooth $g(x)$.

- Update Rule:

$$x_{k+1} = \mathbf{prox}_{\alpha_k g}(x_k - \alpha_k \nabla f(x_k))$$

- Proximal Operator:

$$\mathbf{prox}_{\alpha g}(v) = \operatorname*{argmin}_x \left(g(x) + \frac{1}{2\alpha}\|x - v\|_2^2 \right)$$

.

Proximal Gradient Descent

- Core Idea: For composite problems

$$\min_x F(x) = f(x) + g(x)$$

  - $f(x)$: Convex, differentiable.
  - $g(x)$: Convex, possibly non-differentiable.
  - Generalize gradient descent to handle non-smooth $g(x)$.
- Update Rule:

$$x_{k+1} = \mathbf{prox}_{\alpha_k g}(x_k - \alpha_k \nabla f(x_k))$$

- Proximal Operator:

$$\mathbf{prox}_{\alpha g}(v) = \underset{x}{\operatorname{argmin}} \left(g(x) + \frac{1}{2\alpha} \|x - v\|_2^2 \right)$$

.

  - Small α: Stay close to the "gradient-updated" point v.
  - Large α: Minimize $g(x)$ (enforcing sparsity, constraints, etc.).

Proximal Gradient Descent

We approximate $F(x) = f(x) + g(x)$ at x_k by:

$$Q(x; x_k) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2\alpha} \|x - x_k\|_2^2 + g(x).$$

Proximal Gradient Descent

We approximate $F(x) = f(x) + g(x)$ at x_k by:

$$Q(x; x_k) = f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2\alpha}\|x - x_k\|_2^2 + g(x).$$

Surrogate function:

$$x_{k+1} = \underset{x}{\operatorname{argmin}} Q(x; x_k).$$

Proximal Gradient Descent

We approximate $F(x) = f(x) + g(x)$ at x_k by:

$$Q(x; x_k) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2\alpha}\|x - x_k\|_2^2 + g(x).$$

Surrogate function:

$$x_{k+1} = \underset{x}{\operatorname{argmin}}\, Q(x; x_k).$$

$$\begin{aligned}
=&\underset{x}{\operatorname{argmin}} \left[f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2\alpha}\|x - x_k\|_2^2 + g(x) \right] \\
=&\underset{x}{\operatorname{argmin}} \left[f(x_k) + \frac{1}{2\alpha}\|x - (x_k - \alpha\nabla f(x_k))\|_2^2 - \frac{\alpha}{2}\|\nabla f(x_k)\|_2^2 + g(x) \right] \\
=&\underset{x}{\operatorname{argmin}} \left[\frac{1}{2\alpha}\|x - (x_k - \alpha\nabla f(x_k))\|_2^2 + g(x) \right] \\
=&\operatorname{prox}_{\alpha g}\left(x_k - \alpha\nabla f(x_k)\right)
\end{aligned}$$

Proximal Gradient Descent

- For $g(x) = \lambda\|x\|_1$, **prox** is soft-thresholding:

$$\mathbf{prox}_{\alpha_k \lambda \|\cdot\|}(v_i) = \operatorname{sign}(v_i) \max\left(0, |v_i| - \lambda\alpha_k\right).$$

Proximal Gradient Descent

- For $g(x) = \lambda \|x\|_1$, **prox** is soft-thresholding:

$$\mathbf{prox}_{\alpha_k \lambda \|\cdot\|}(v_i) = \mathrm{sign}(v_i) \max\left(0, |v_i| - \lambda \alpha_k\right).$$

- MATLAB Demonstration: $F(x) = \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1$.

Summary of What You've Learned Today

- Overview of Machine Learning: History, Classical Machine Learning, Deep Learning.

Summary of What You've Learned Today

- Overview of Machine Learning: History, Classical Machine Learning, Deep Learning.
- Core Concepts: Convex Sets, Convex Functions, Convex Problem Formulation.

Summary of What You've Learned Today

- Overview of Machine Learning: History, Classical Machine Learning, Deep Learning.
- Core Concepts: Convex Sets, Convex Functions, Convex Problem Formulation.
- A Tour of Optimization Algorithms:
    - First-order Method: Gradient Descent, Accelerated Gradient Descent, Sub-gradient Method, Proximal Gradient Descent.
    - Second-order Method: Newton's Method, Quasi-Newton Method.

Summary of What You've Learned Today

- Overview of Machine Learning: History, Classical Machine Learning, Deep Learning.
- Core Concepts: Convex Sets, Convex Functions, Convex Problem Formulation.
- A Tour of Optimization Algorithms:
  - First-order Method: Gradient Descent, Accelerated Gradient Descent, Sub-gradient Method, Proximal Gradient Descent.
  - Second-order Method: Newton's Method, Quasi-Newton Method.
- Boyd, Stephen P., and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Summary of What You've Learned Today

- Overview of Machine Learning: History, Classical Machine Learning, Deep Learning.
- Core Concepts: Convex Sets, Convex Functions, Convex Problem Formulation.
- A Tour of Optimization Algorithms:
    - First-order Method: Gradient Descent, Accelerated Gradient Descent, Sub-gradient Method, Proximal Gradient Descent.
    - Second-order Method: Newton's Method, Quasi-Newton Method.
- Boyd, Stephen P., and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

See you next week (6/30) for Lecture 2 on
Singular Value Decomposition (1).