




CEU

*Universidad
San Pablo*

Lesson 1. Introduction

Medicine School

Contents

-  **Evaluation**
- **Continuous evaluation (ordinary call):**
 - Class tests & participation (SE2, 80%).
 - Programming projects (SE3, 20%).
 - **Extraordinary call:**
 - Final exam (SE1, 80%).
 - Continuous activities (20%).
 - **Requirement:** 75% attendance in theory, 100% in practicals.

Block I. Knowledge-driven AI

- Representation of knowledge, propositional & first-order logic.
- Expert systems: rules, facts, ontologies.
- Heuristic search, planning (STRIPS, GraphPlan).
- Adversarial games, alpha-beta pruning, Monte Carlo Tree Search.

Block II. Classical Machine Learning

- Fundamentals: supervised learning, overfitting, cross-validation, evaluation metrics.
- Regression (linear, logistic), k-NN, decision trees, Random Forest.
- Support Vector Machines (SVM), ensemble methods (boosting).
- Dimensionality reduction (PCA, t-SNE).

Block III. Deep Learning

- Neural networks: MLPs, backpropagation, optimization, regularization.
- CNNs for biomedical images.
- RNNs, LSTMs, GRUs for biological sequences & time-series.
- Transformers (BERT, GPT, ProtBERT, ESM) for text, genomics, proteomics.
- Generative models: Autoencoders, VAEs, GANs, diffusion models.
→ Applications: protein design, synthetic biological data.

Block IV. Reinforcement Learning







- Agents, environments, rewards, policies.
- Q-learning, SARSA, Policy Gradient, DQN.
- Biomedical applications: molecule design, experimental planning.



































Block I. Knowledge-driven AI

- Early AI focused on **explicit knowledge**: facts, rules, ontologies.
- Computers reason by **logic** (propositional, first-order).
- **Expert systems** use “if–then” rules and inference engines.
- Applications in **biomedicine**:
 - Clinical decision support (diagnosis from symptoms).
 - Ontologies (Gene Ontology, SNOMED CT) for structuring biological knowledge.
 - Rule-based planning in experimental workflows.
- **Key idea**: Instead of learning from data, the system reasons from encoded human knowledge.

Block I. Knowledge-driven AI

<https://geneontology.org/>

-   biological_process 1203818
-   cellular_component 1238284
-   molecular_function 1214874

-   biological_process 1203818
 -   biological phase ...
 -   biological process involved in interspecies interaction between organisms ...
 -   biological process involved in intraspecies interaction between organisms ...
 -   biological regulation ...
 -   negative regulation of HRI-mediated signaling ...
 -   negative regulation of brood size ...
 -   positive regulation of brood size ...
 -   regulation of biological process ...
 -   regulation of biological quality ...
 -   regulation of brood size ...
 -   regulation of buoyancy ...
 -   regulation of molecular function ...
 -   regulation of pH ...
 -   cellular process ...
 -   detoxification ...
 -   developmental process ...

Block I. Knowledge-driven AI

Knowledge graph <https://copenmed.org/>

Diabetes



Tipo Entidad: GroupOfDiseases



+ DETALLES



| Entidad ▼ | Idioma | Nivel |
|---|---------|-------|
|  CIE-11: Bloque L2-5A1 (Diabetes mellitus) | Español | 2 |
|  Diabetes | Español | 0 |
|  Diabetes | English | 0 |
|  ICD-11: Block L2-5A1 (Diabetes mellitus) | English | 2 |

+ DESCRIPCIONES EXTENDIDAS




| Descripción ▼ | Idioma |
|---|---------|
|  Diabetes is a chronic (long-lasting) health condition that affects how your body turns food into energy. Your body breaks down most of the food you eat into sugar (glucose) and releases it into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin. Insulin acts like a key to let the blood sugar into your body's cells for use as energy. With diabetes, your body doesn't make enough insulin or can't use it as well as it should. When there isn't enough insulin or cells stop responding to insulin, too much blood sugar stays in your bloodstream. Over time, that can cause serious health problems, such as heart disease, vision loss, and kidney disease. | English |
|  La diabetes es una enfermedad en la que los niveles de glucosa (azúcar) de la sangre están muy altos. La insulina es una hormona que ayuda a que la glucosa entre a las células para suministrarles energía. En la diabetes tipo 1, el cuerpo no produce insulina. En la diabetes tipo 2, la más común, el cuerpo no produce o no usa la insulina de manera adecuada. Sin suficiente insulina, la glucosa permanece en la sangre. Con el tiempo, el exceso de glucosa en la sangre puede causar problemas serios. Puede dañar los ojos, los riñones y los nervios. La diabetes también puede causar enfermedades cardíacas, derrames cerebrales y la necesidad de amputar un miembro. Las mujeres embarazadas también pueden desarrollar diabetes. Llamada diabetes gestacional. Un análisis de sangre puede mostrar si tiene diabetes. Los síntomas más frecuentes son: aumento de sed y de la micción, fatiga, visión borrosa, pérdida de peso inesperada, aumento del hambre. El exceso de | Español |

Block I. Knowledge-driven AI

Knowledge graph

+ ASOCIACIONES: Diabetes →...

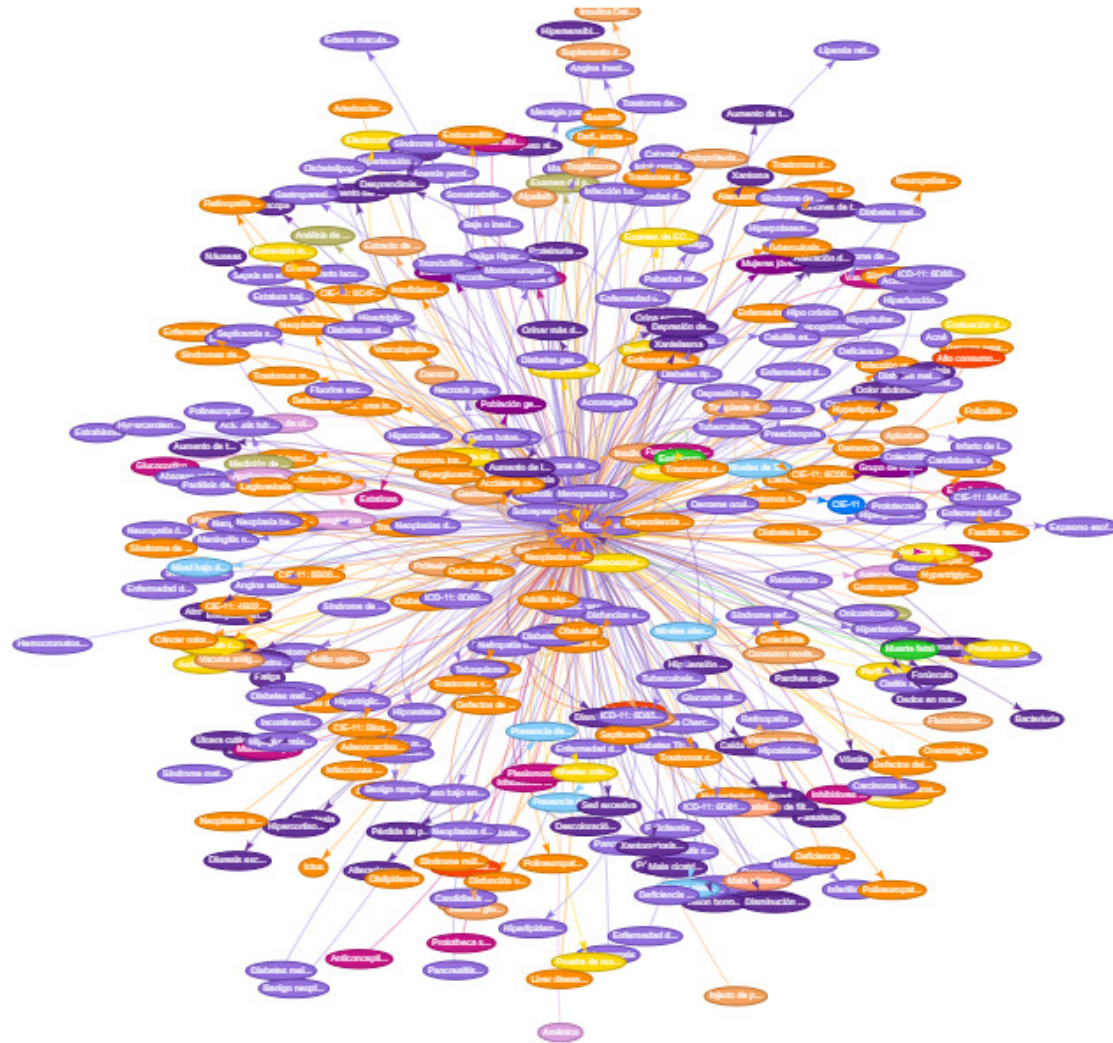
| Entidad | Tipo de Asociación ▼ | Fuerza |
|---|--|--------|
|  <i>Endocrinología</i> | Group belongs to the domain of Specialty | 1 |
|  <i>Análisis de sangre (glucosa)</i> | Group can be diagnosed with Test | 0.9 |

ASOCIACIONES: ...→Diabetes

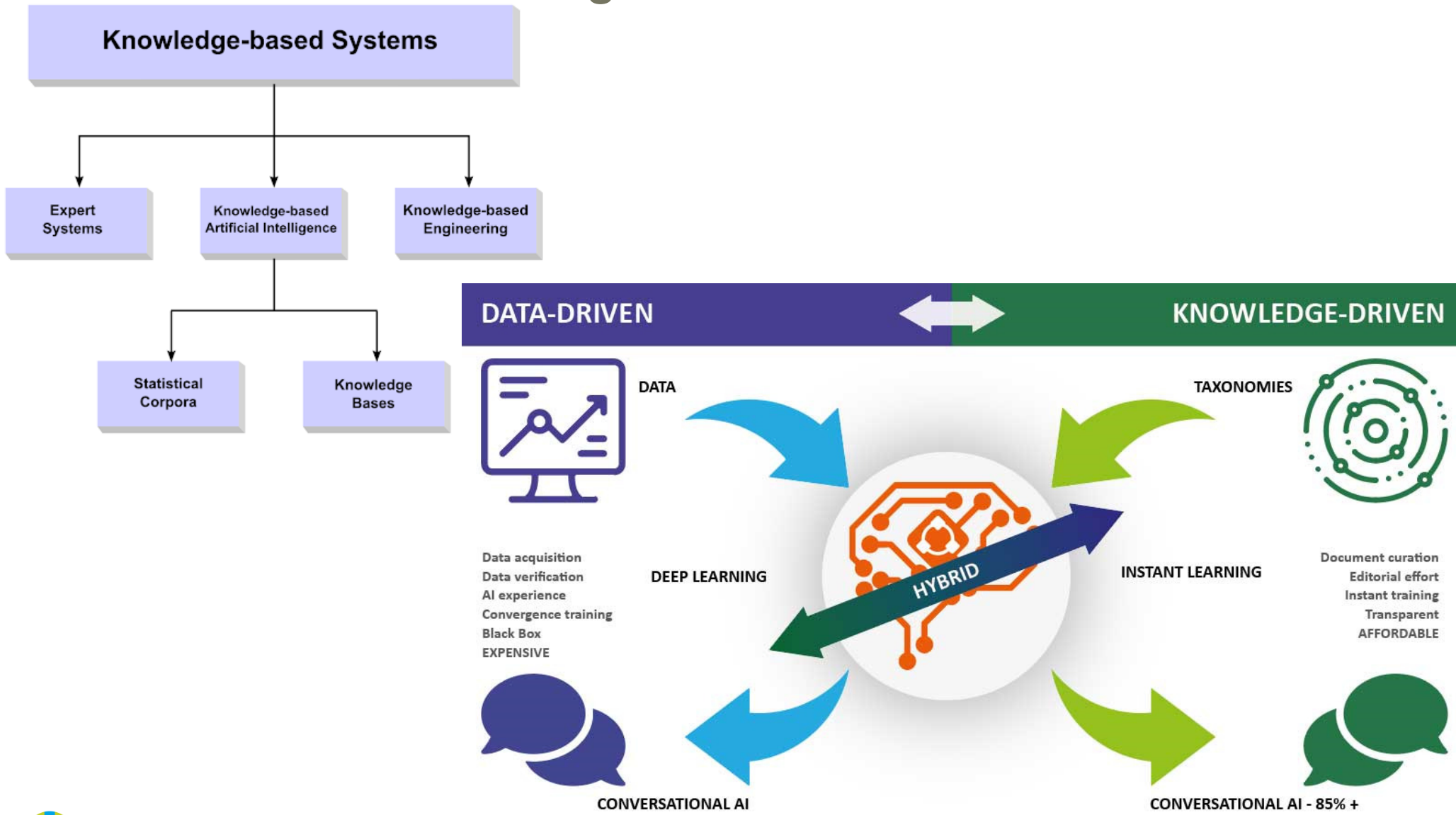
| Entidad | Tipo de Asociación ▼ | Fuerza |
|--|--------------------------|--------|
|  <i>Estrés crónico</i> | Activity may cause Group | 0.5 |
|  <i>Mala alimentación</i> | Activity may cause Group | 0.5 |

Block I. Knowledge-driven AI

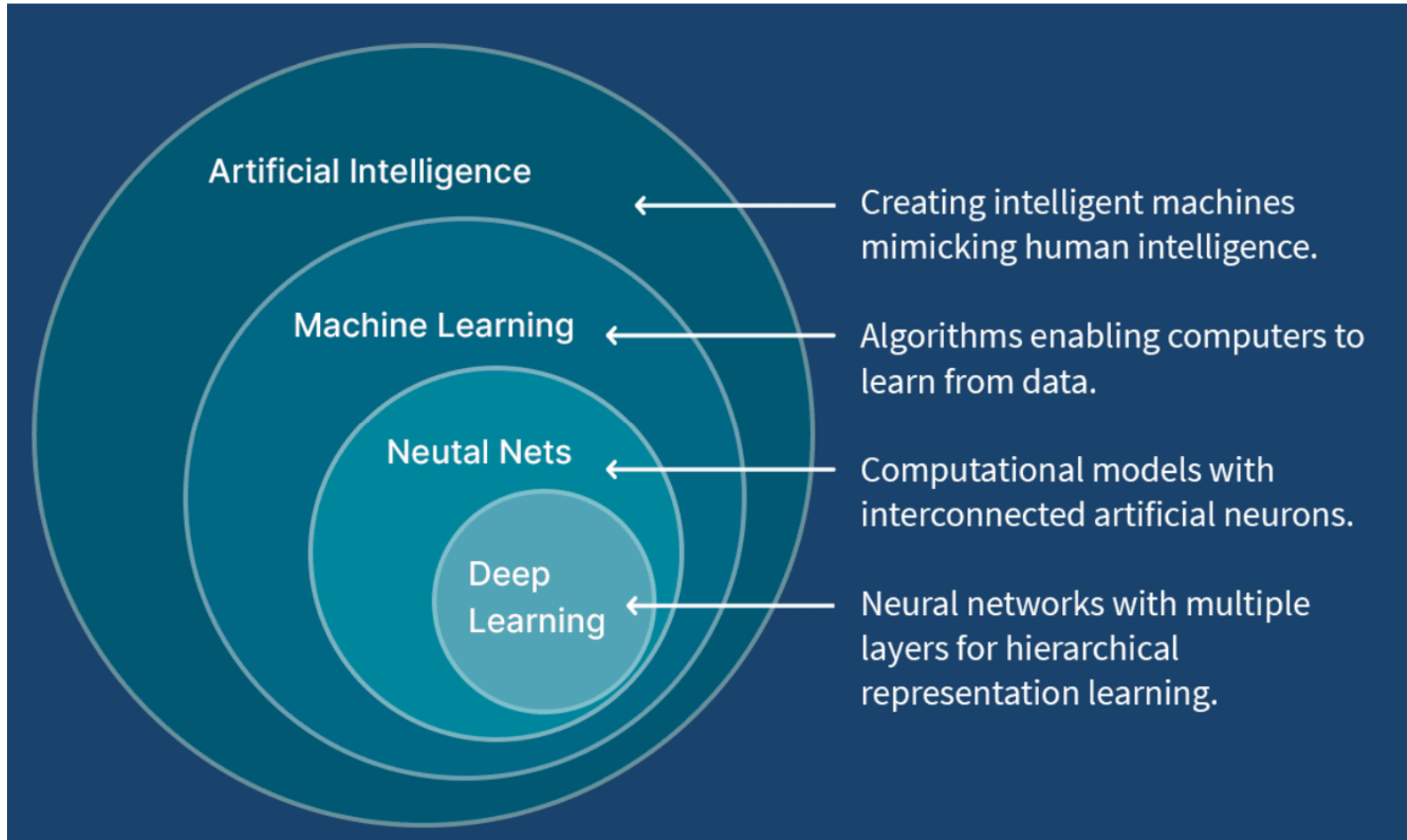
Knowledge graph



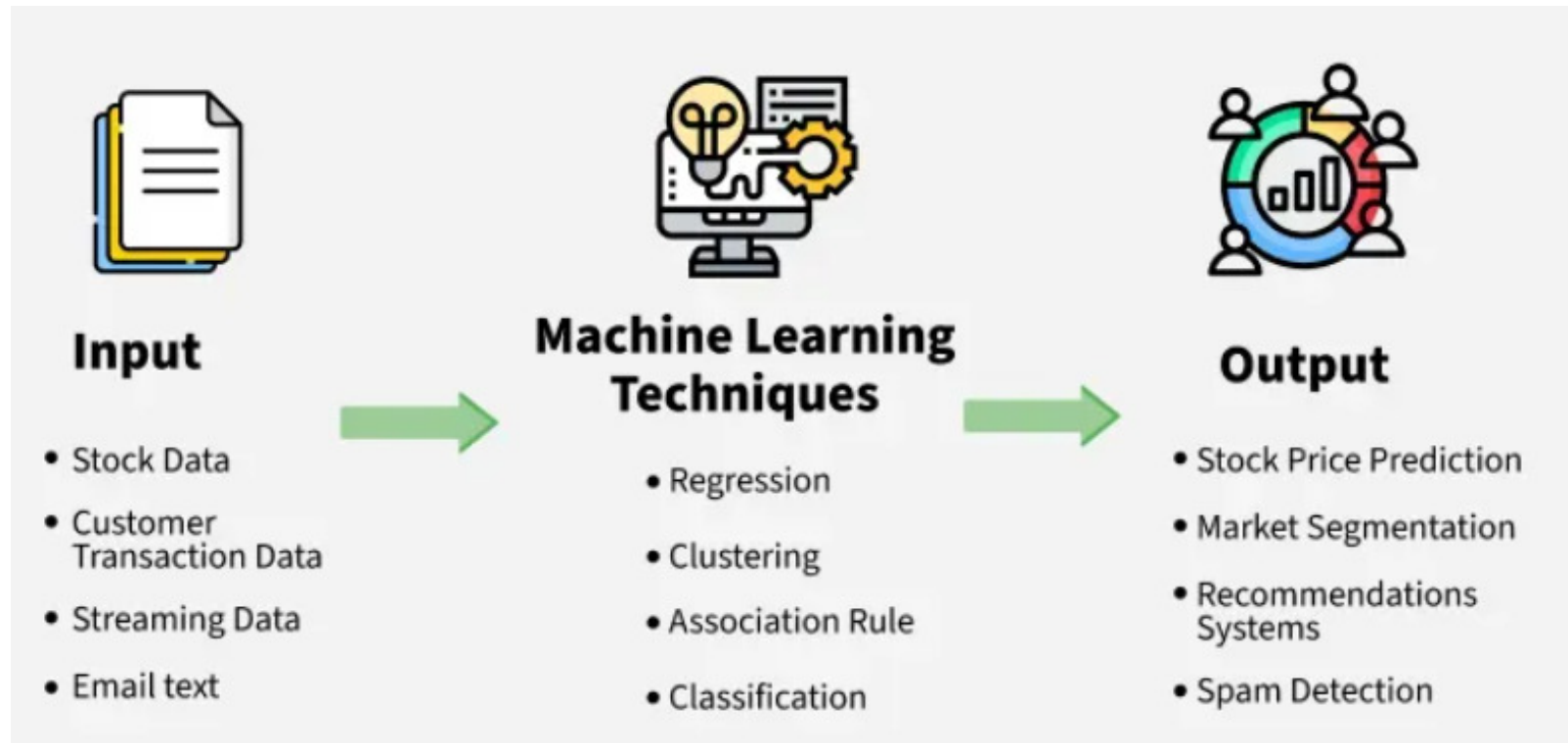
Block I. Knowledge-driven AI



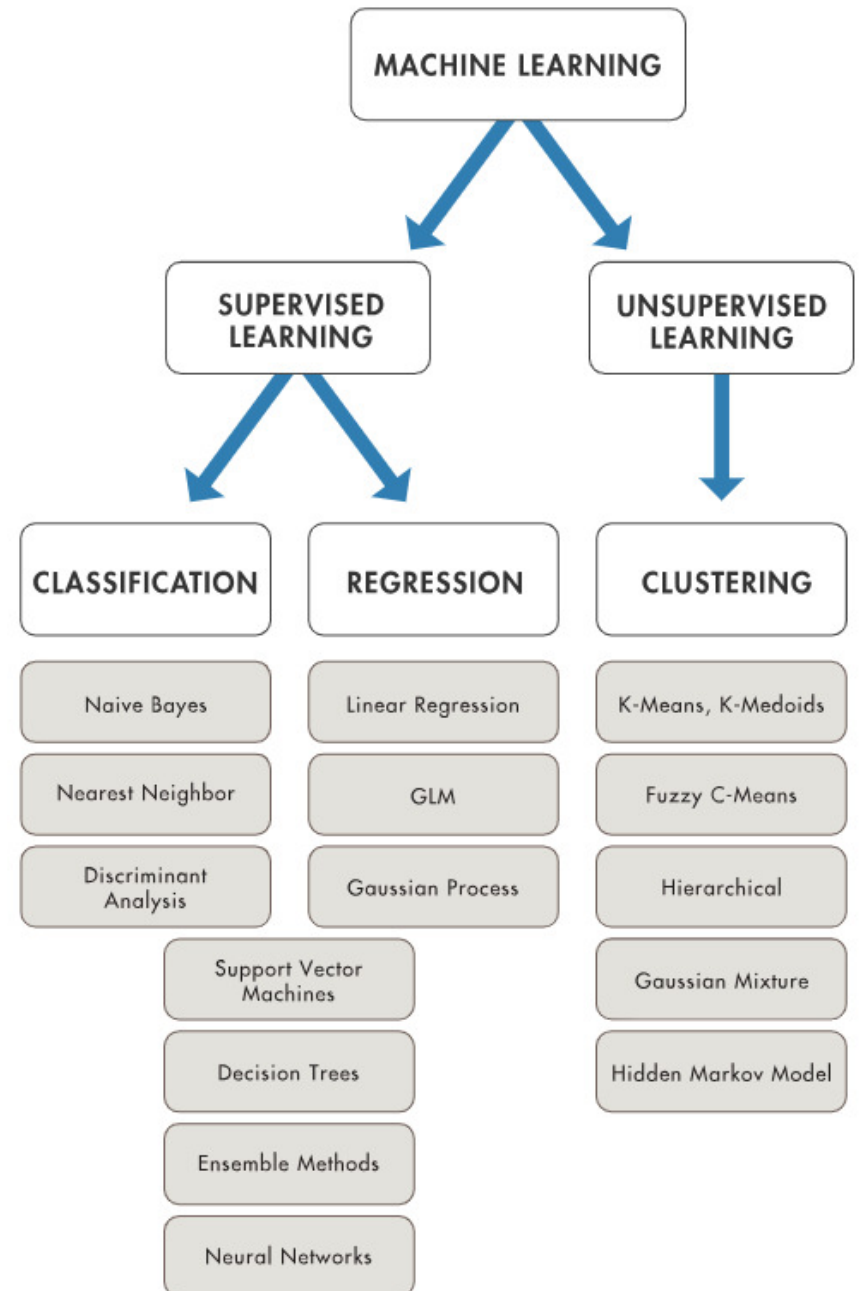
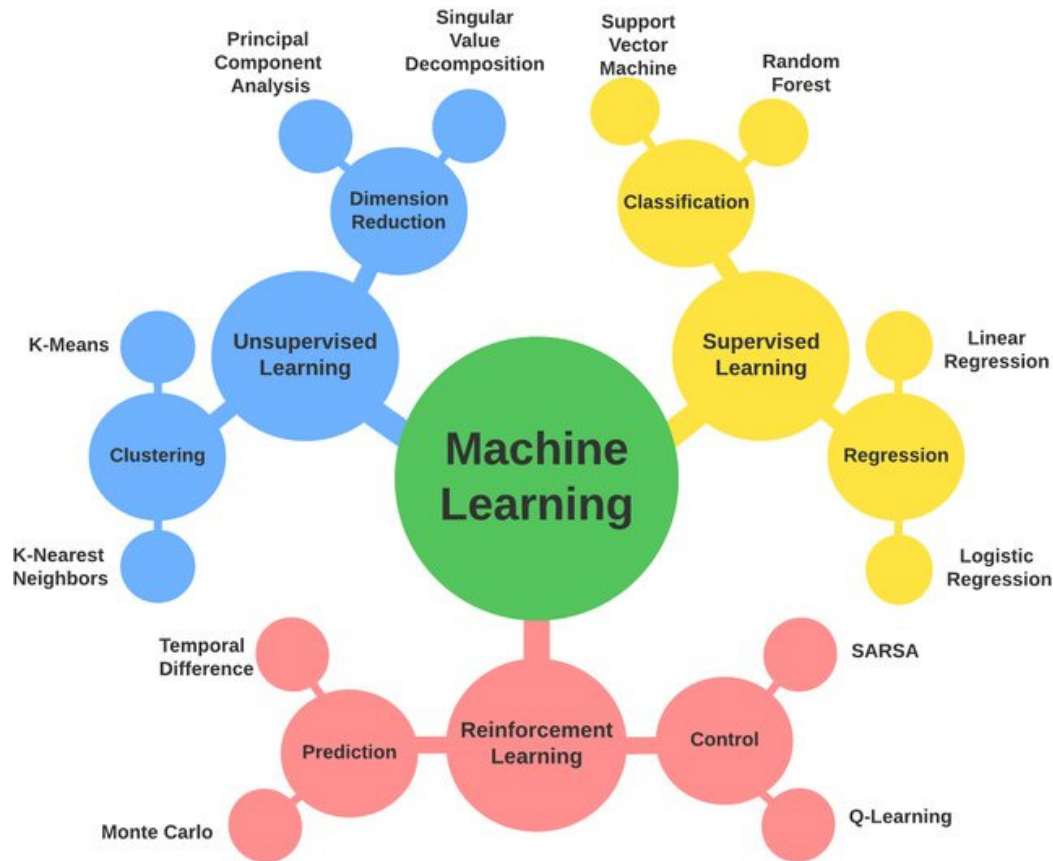
Block II. Machine learning



Block II. Machine learning

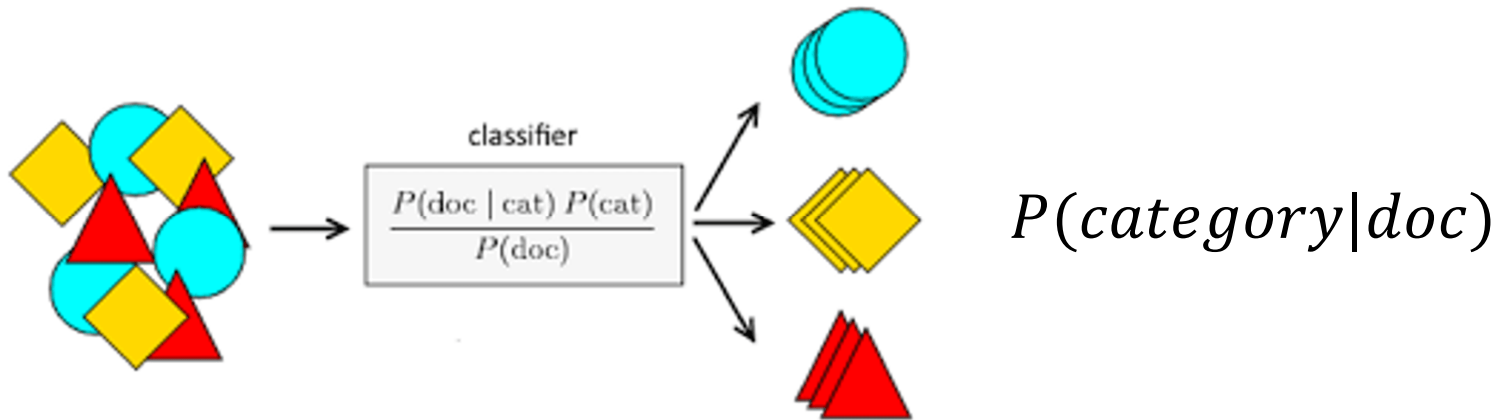


Block II. Machine learning



Block II. Machine learning

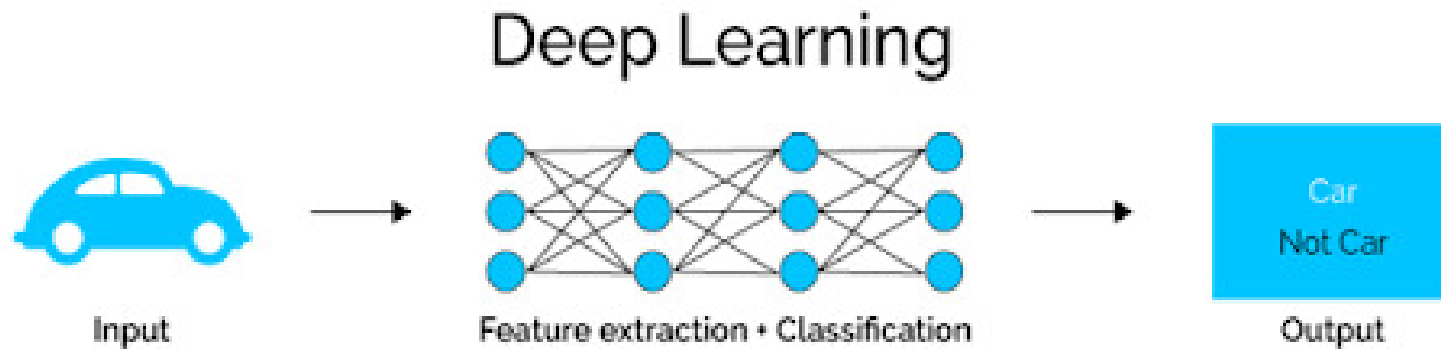
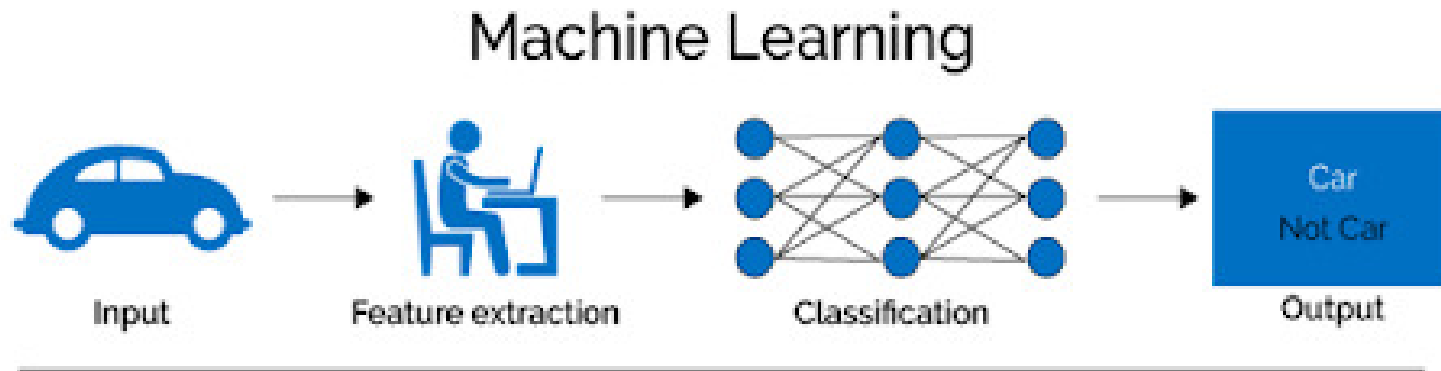
Naïve Bayes classifier



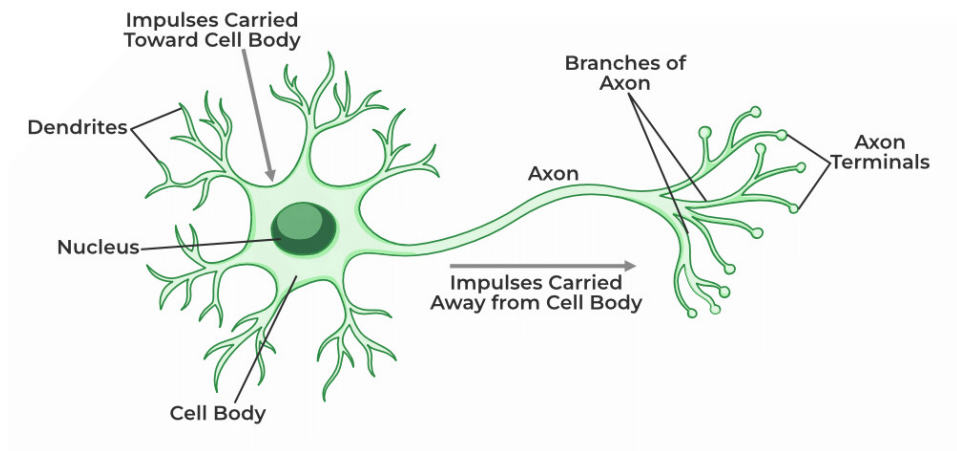
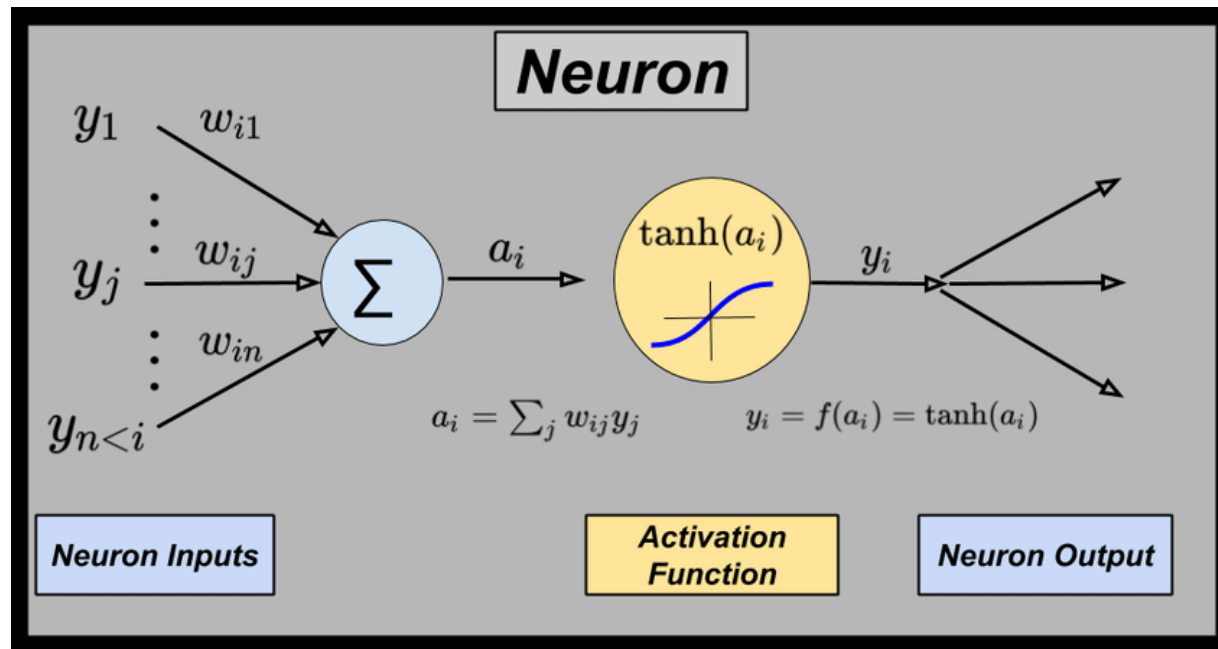
$$p(C_k \mid x_1, \dots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^n p(x_i \mid C_k)$$

where the evidence $Z = p(\mathbf{x}) = \sum_k p(C_k) p(\mathbf{x} \mid C_k)$

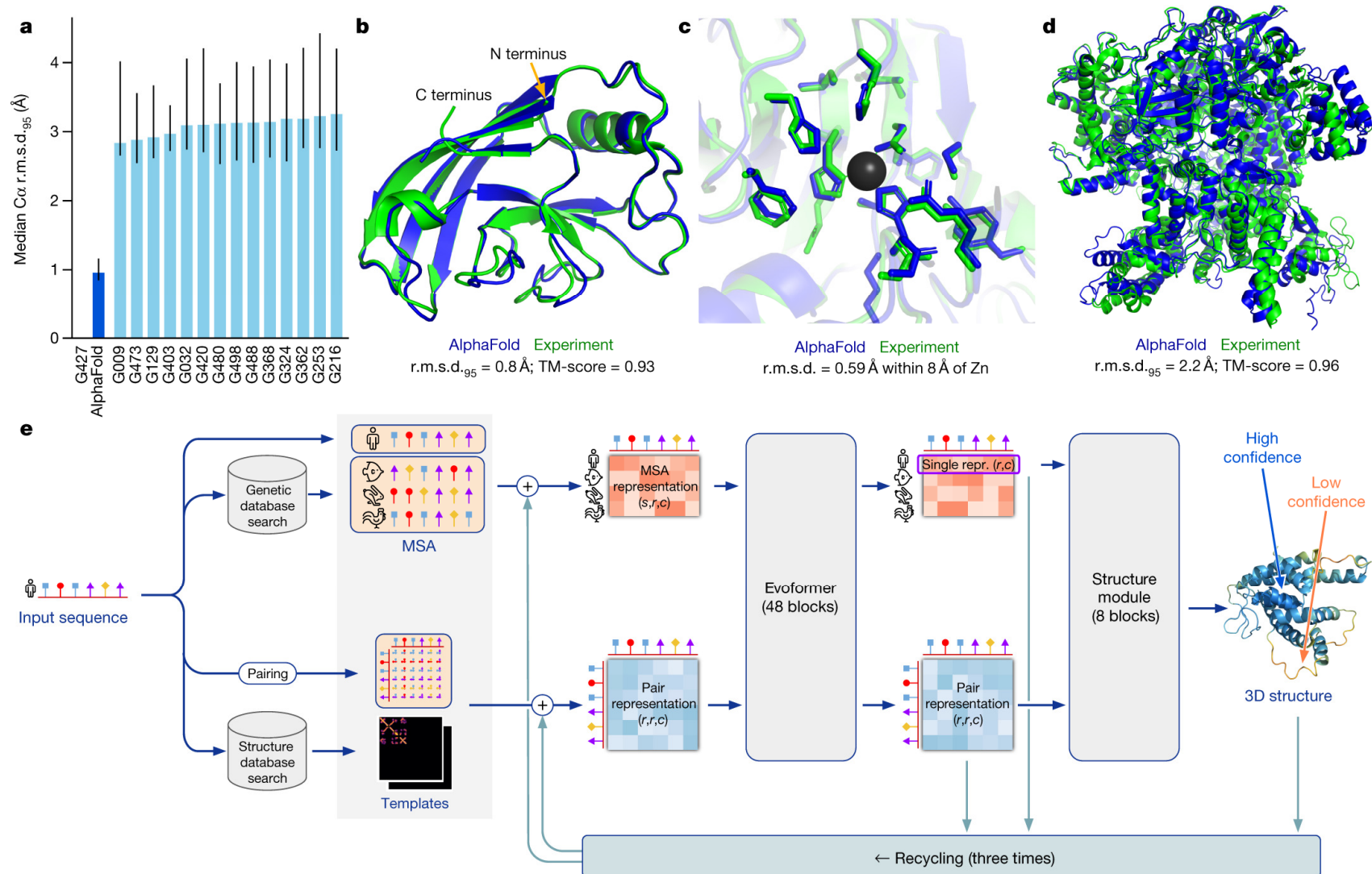
Block III. Deep learning



Block III. Deep learning



Block III. Deep learning



Block III. Deep learning

The Nobel Prize in Physics 2024

John J. Hopfield

“for foundational discoveries and inventions that enable machine learning with artificial neural networks”



© Nobel Prize Outreach. Photo: Nanaka Adachi

Geoffrey Hinton

“for foundational discoveries and inventions that enable machine learning with artificial neural networks”



© Nobel Prize Outreach. Photo: Clément Morin

Block III. Deep learning

The Nobel Prize in Chemistry 2024

David Baker

“for computational protein design”



© Nobel Prize Outreach. Photo: Clément Morin

Demis Hassabis

“for protein structure prediction”



© Nobel Prize Outreach. Photo: Clément Morin

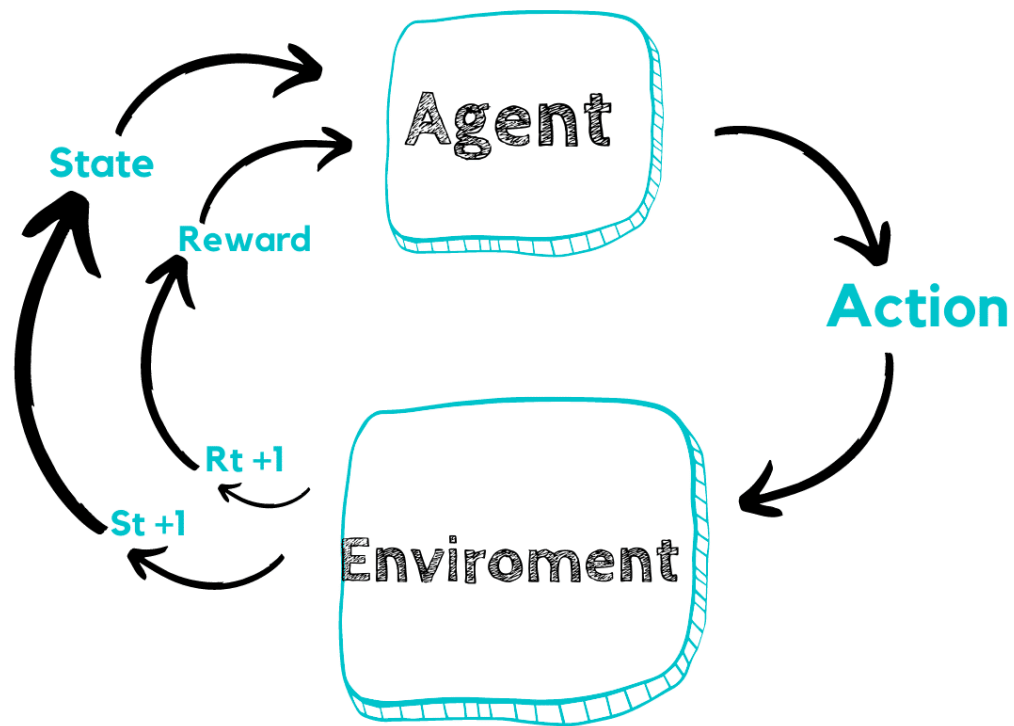
John Jumper

“for protein structure prediction”



© Nobel Prize Outreach. Photo: Clément Morin

Block IV. Reinforcement learning



Block IV. Reinforcement learning

naturemedicine

Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [nature medicine](#) > [comment](#) > article

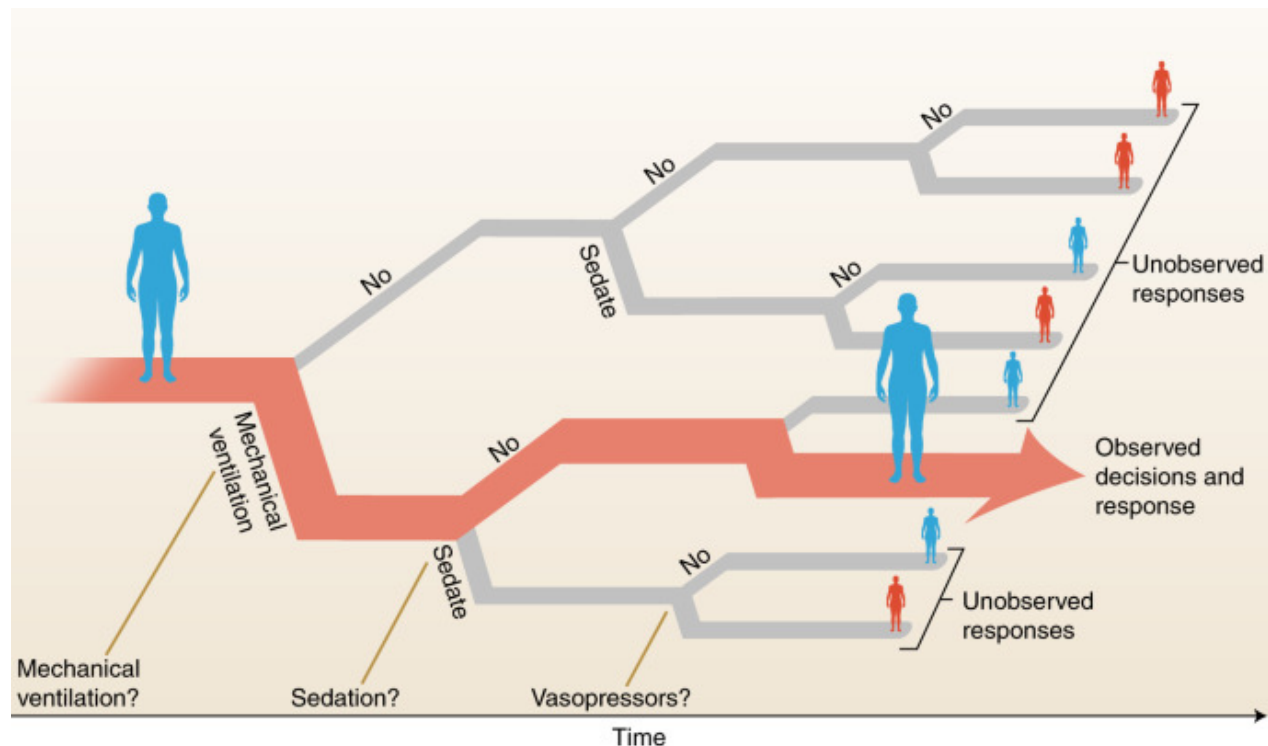
Comment | Published: 07 January 2019

Guidelines for reinforcement learning in healthcare

[Omer Gottesman](#), [Fredrik Johansson](#), [Matthieu Komorowski](#), [Aldo Faisal](#), [David Sontag](#), [Finale Doshi-Velez](#)

& [Leo Anthony Celi](#) 

[Nature Medicine](#) 25, 16–18 (2019) | [Cite this article](#)



Block IV. Reinforcement learning



Thinking the unthinkable

Learning from chess engines: how reinforcement learning could redefine clinical decision-making in rheumatology

Thomas Hügle

1. Department of Rheumatology, Lausanne University Hospital and University of Lausanne, Lausanne, Switzerland

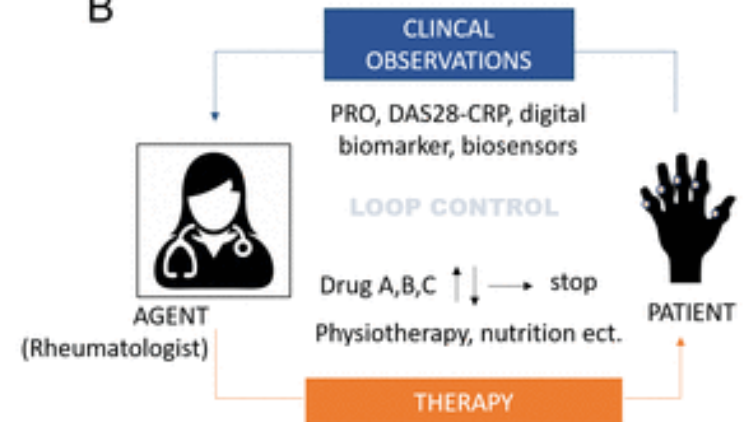
Correspondence to Professor Thomas Hügle, Department of Rheumatology, University of Lausanne, Lausanne 1011, Switzerland; thomas.hugle@chuv.ch

REINFORCEMENT LEARNING

A



B



Contents



Evaluation

- **Continuous evaluation (ordinary call):**
 - Class tests & participation (SE2, 80%).
 - Programming projects (SE3, 20%).
- **Extraordinary call:**
 - Final exam (SE1, 80%).
 - Continuous activities (20%).
- **Requirement:** 75% attendance in theory, 100% in practicals.



Block I. Knowledge-driven AI

- Representation of knowledge, propositional & first-order logic.
- Expert systems: rules, facts, ontologies.
- Heuristic search, planning (STRIPS, GraphPlan).
- Adversarial games, alpha-beta pruning, Monte Carlo Tree Search.

Block II. Classical Machine Learning

- Fundamentals: supervised learning, overfitting, cross-validation, evaluation metrics.
- Regression (linear, logistic), k-NN, decision trees, Random Forest.
- Support Vector Machines (SVM), ensemble methods (boosting).
- Dimensionality reduction (PCA, t-SNE).

Block III. Deep Learning

- Neural networks: MLPs, backpropagation, optimization, regularization.
- CNNs for biomedical images.
- RNNs, LSTMs, GRUs for biological sequences & time-series.
- Transformers (BERT, GPT, ProtBERT, ESM) for text, genomics, proteomics.
- Generative models: Autoencoders, VAEs, GANs, diffusion models.
→ Applications: protein design, synthetic biological data.

Block IV. Reinforcement Learning

- Agents, environments, rewards, policies.
- Q-learning, SARSA, Policy Gradient, DQN.
- Biomedical applications: molecule design, experimental planning.



CEU

*Universidad
San Pablo*

Lesson 2. Representation of knowledge

Medicine School

Contents

- Introduction
- Tabular representations
- Graph-based representations
- Ontologies and taxonomies
- Vector-space representations
- Probabilistic representations
- Causal representations
- Large Language Models as implicit representations

Introduction: Why Knowledge Representation (KR)?

1. From Data to Knowledge

- **Data:** raw measurements (e.g., gene expression values, protein structures, patient records).
- **Information:** structured data with context (e.g., gene annotations, protein–protein interactions).
- **Knowledge:** integrated, interpretable representation that supports reasoning and decision-making.

| Tissue | Gene expression levels | | | | | Diagnosis |
|--------|------------------------|--------|--------|--------|--------|-----------|
| | Gene 1 | Gene 2 | Gene 3 | Gene 4 | Gene 5 | |
| 1 | 0.405 | 0.326 | 0.234 | 0.348 | 0.748 | normal |
| 2 | 0.089 | 0.293 | 0.192 | 0.123 | 0.385 | normal |
| 3 | 0.459 | 0.125 | 0.543 | 0.334 | 0.218 | tumor |
| 4 | 0.123 | 0.389 | 0.238 | 0.651 | 0.972 | normal |
| 5 | 0.951 | 0.040 | 0.490 | 0.283 | 0.321 | normal |
| 6 | 0.297 | 0.859 | 0.219 | 0.783 | 0.984 | tumor |

Atomic Coordinates: PDB Format

| | | Amino Acid | | Chain name | | Sequence Number | | -----Coordinates----- | | | | (etc.) |
|------|---|------------|-----|------------|---|-----------------|--|-----------------------|-------|-------|-----|--------|
| | | Element | | | | | | X | Y | Z | | |
| ATOM | 1 | N | ASP | L | 1 | | | 4.060 | 7.307 | 5.186 | ... | |
| ATOM | 2 | CA | ASP | L | 1 | | | 4.042 | 7.776 | 6.553 | ... | |
| ATOM | 3 | C | ASP | L | 1 | | | 2.668 | 8.426 | 6.644 | ... | |
| ATOM | 4 | O | ASP | L | 1 | | | 1.987 | 8.438 | 5.606 | ... | |
| ATOM | 5 | CB | ASP | L | 1 | | | 5.090 | 8.827 | 6.797 | ... | |
| ATOM | 6 | CG | ASP | L | 1 | | | 6.338 | 8.761 | 5.929 | ... | |
| ATOM | 7 | OD1 | ASP | L | 1 | | | 6.576 | 9.758 | 5.241 | ... | |
| ATOM | 8 | OD2 | ASP | L | 1 | | | 7.065 | 7.759 | 5.948 | ... | |

\\
Element position within amino acid

Introduction: Why Knowledge Representation (KR)?

2. Why Knowledge Representation Matters

A central question in AI: *how should an intelligent system represent what it knows about the world?*

- Different tasks require different representations:
 - Storing and querying biological datasets (tables).
 - Modeling relationships (graphs).
 - Using domain vocabularies (ontologies).
 - Capturing similarity (embeddings).
 - Handling uncertainty (probabilistic models).
 - Leveraging implicit knowledge (LLMs).

3. The Bioinformatics Perspective

Biological systems are **complex, multi-layered, and noisy**. KR helps:

- Integrate heterogeneous data (omics, literature, clinical).
- Enable **semantic interoperability** across databases.
- Support reasoning about biological function, disease mechanisms, and treatment options.

Tabular representations

Tabular Data

columns = attributes for those observations

| Player | Minutes | Points | Rebounds | Assists |
|--------|---------|--------|----------|---------|
| A | 41 | 20 | 6 | 5 |
| B | 30 | 29 | 7 | 6 |
| C | 22 | 7 | 7 | 2 |
| D | 26 | 3 | 3 | 9 |
| E | 20 | 19 | 8 | 0 |
| F | 9 | 6 | 14 | 14 |
| G | 14 | 22 | 8 | 3 |
| I | 22 | 36 | 0 | 9 |
| J | 34 | 8 | 1 | 3 |

Rows = observations

Strengths:

- **Simplicity:** intuitive and widely understood.
- **Standardization:** CSV, Excel are universal.
- **Efficiency:** great for storage, querying, and statistics.
- **First step** for most bioinformatics pipelines.

Limitations:

- Poor at capturing **relationships** between entities (e.g., protein A interacts with protein B).
- Rigid structure — difficult to represent **hierarchies** (species taxonomy) or **uncertain information**.
- Not well-suited for **temporal or causal dependencies**.

Gene expression matrix:

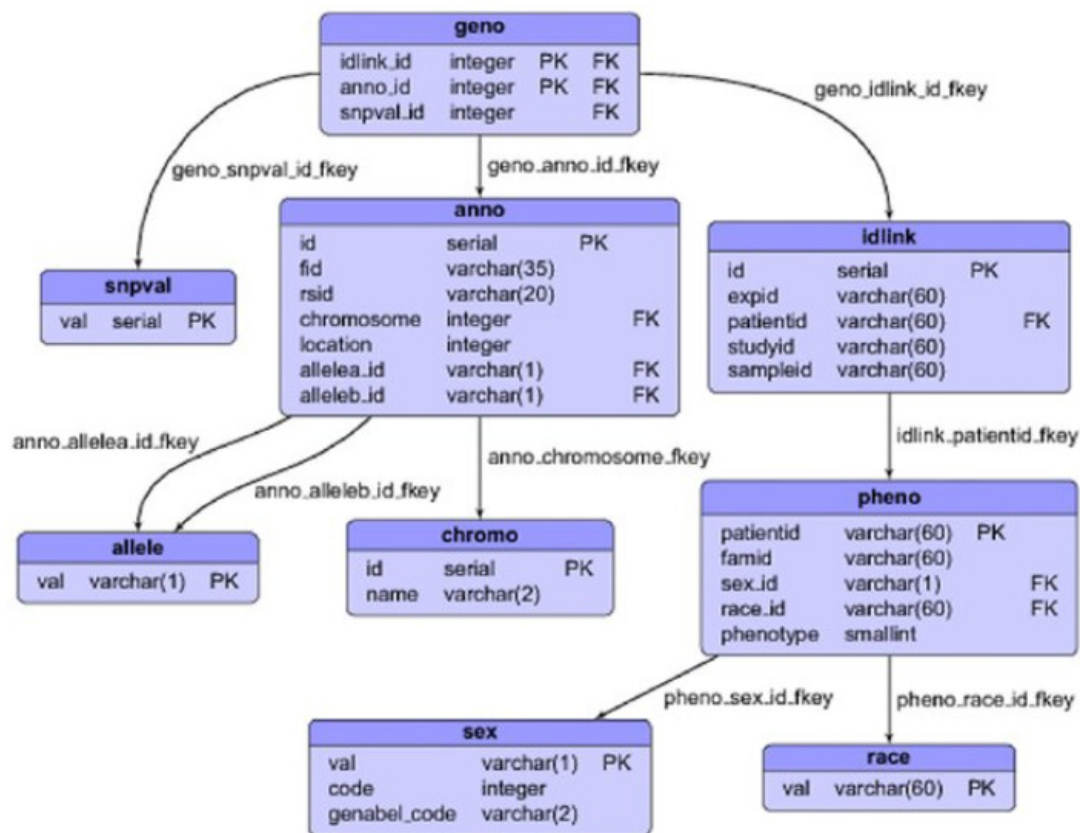
- Rows = genes, Columns = samples, Values = expression levels.

Clinical trial data:

- Rows = patients, Columns = age, treatment, response, side effects.

Tabular representations: Database

Structured Query Language (SQL)



Figure

Caption

Figure 2. Database Schema. Geno Single database schema for the Affymetrix platform. In this diagram, the rectangles correspond to database tables, and the rows in each rectangle correspond to database table columns. The four columns in a row correspond to, from left to right, database name (column 1), data type (column 2 ... [Read more](#)

Available via license: [CC BY 4.0](#)

Content may be subject to copyright.

SNPpy - Database Management for SNP Data from Genome Wide Association Studies

Tabular representations: Database

◆ General Sequence Databases

- **GenBank** (NCBI, USA) – Comprehensive DNA sequence database.
 - **EMBL-EBI/ENA** (Europe) – European Nucleotide Archive.
 - **DDBJ** (Japan) – DNA Data Bank of Japan.
- Together, these form the **INSDC** (International Nucleotide Sequence Database Collaboration).

◆ Protein Sequence & Functional Databases

- **UniProt** – The gold standard for protein sequences & annotations.
 - **UniProtKB/Swiss-Prot** (curated, manually reviewed).
 - **UniProtKB/TrEMBL** (automatically annotated).
- **RefSeq** (NCBI Reference Sequences) – curated reference proteins and transcripts.

◆ Protein Structure Databases

- **PDB** (Protein Data Bank) – 3D structures of proteins, nucleic acids, complexes.
- **AlphaFold DB** (EMBL-EBI & DeepMind) – predicted protein structures at proteome scale.

Tabular representations: Database

◆ Gene & Genome Databases

- **Ensembl** (EMBL-EBI) – genome browser & annotations for many species.
- **NCBI Gene** – gene-specific information (function, sequence, references).
- **UCSC Genome Browser** – visualization of genomic data.

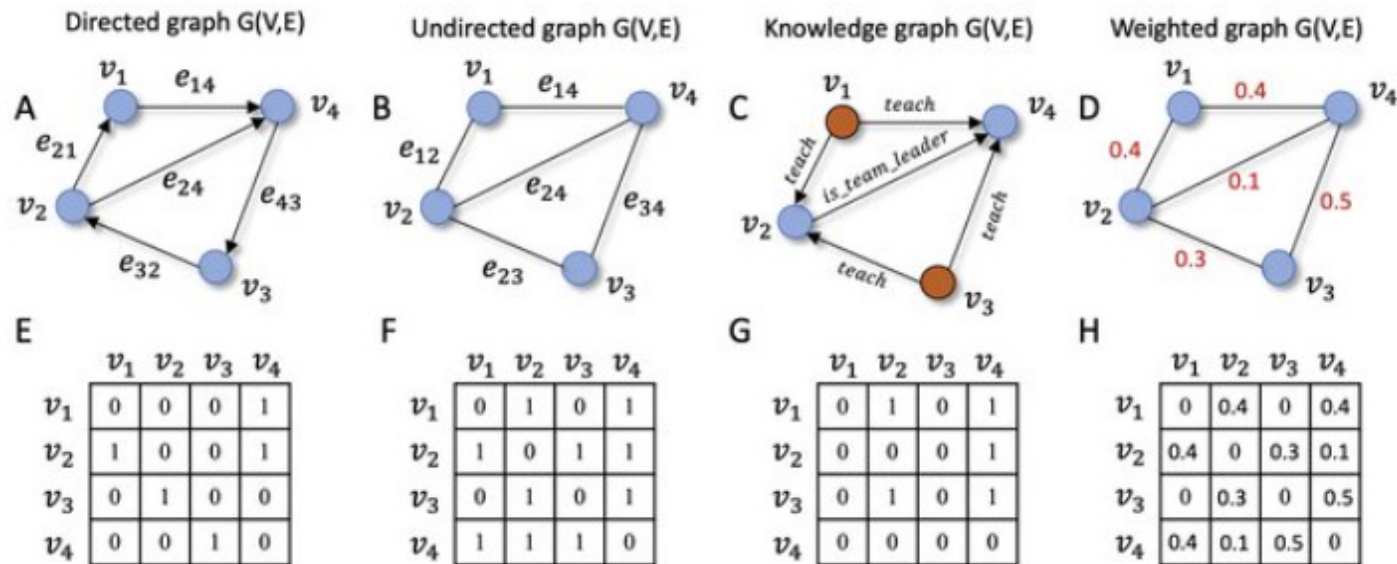
◆ Functional Annotation & Pathways

- **Gene Ontology (GO)** – standardized vocabulary for gene/protein function.
- **KEGG** (Kyoto Encyclopedia of Genes and Genomes) – pathways, interactions, and molecular networks.
- **Reactome** – curated human pathways and reactions.

◆ Expression & Variation Databases

- **GEO** (Gene Expression Omnibus, NCBI) – transcriptomics datasets (microarray, RNA-seq).
- **ArrayExpress** (EBI) – gene expression and functional genomics data.
- **dbSNP** – single nucleotide polymorphisms.
- **gnomAD** – large-scale human genome variation database.

Graph-based representations



Strengths:

- Natural for representing **complex biological systems**.
- Flexible: can represent **directed/undirected, weighted/unweighted, temporal** edges.
- Enables **graph algorithms**: shortest path, clustering, community detection.
- Foundation for **knowledge graphs** and **graph neural networks (GNNs)**.

Limitations:

- Can be **computationally expensive** for very large graphs.
- Requires careful design of ontology/edge semantics.
- Not always ideal for **numeric, tabular-style queries**.

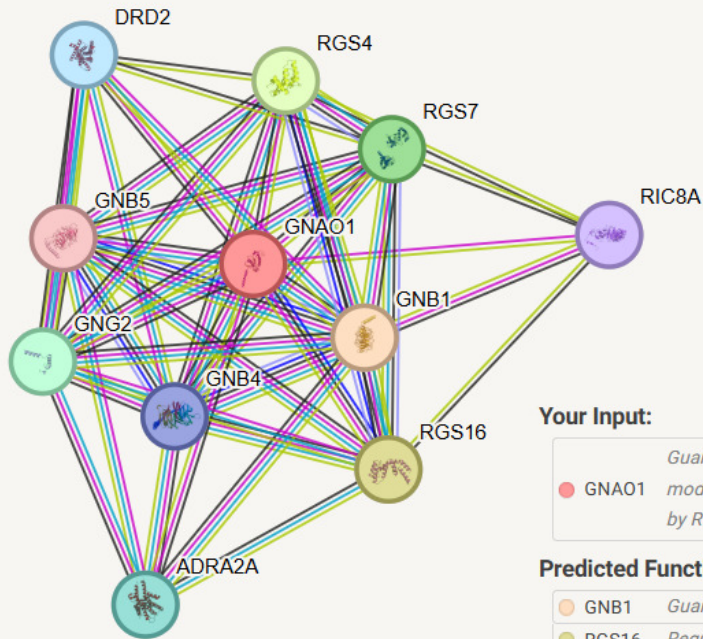
Graph-based representations

- **Protein–Protein Interaction (PPI) networks:**
Nodes = proteins, Edges = physical/functional interactions.
- **Metabolic and signaling pathways:**
Nodes = metabolites/enzymes, Edges = reactions.
- **Gene co-expression networks:**
Nodes = genes, Edges = correlation links.
- **Disease–gene networks:**
Nodes = diseases and genes, Edges = associations.
- **Knowledge Graphs** (modern extension):
Bio2RDF, Hetionet, Wikidata (biomedical subset).

Graph-based representations



Search



Your Input:

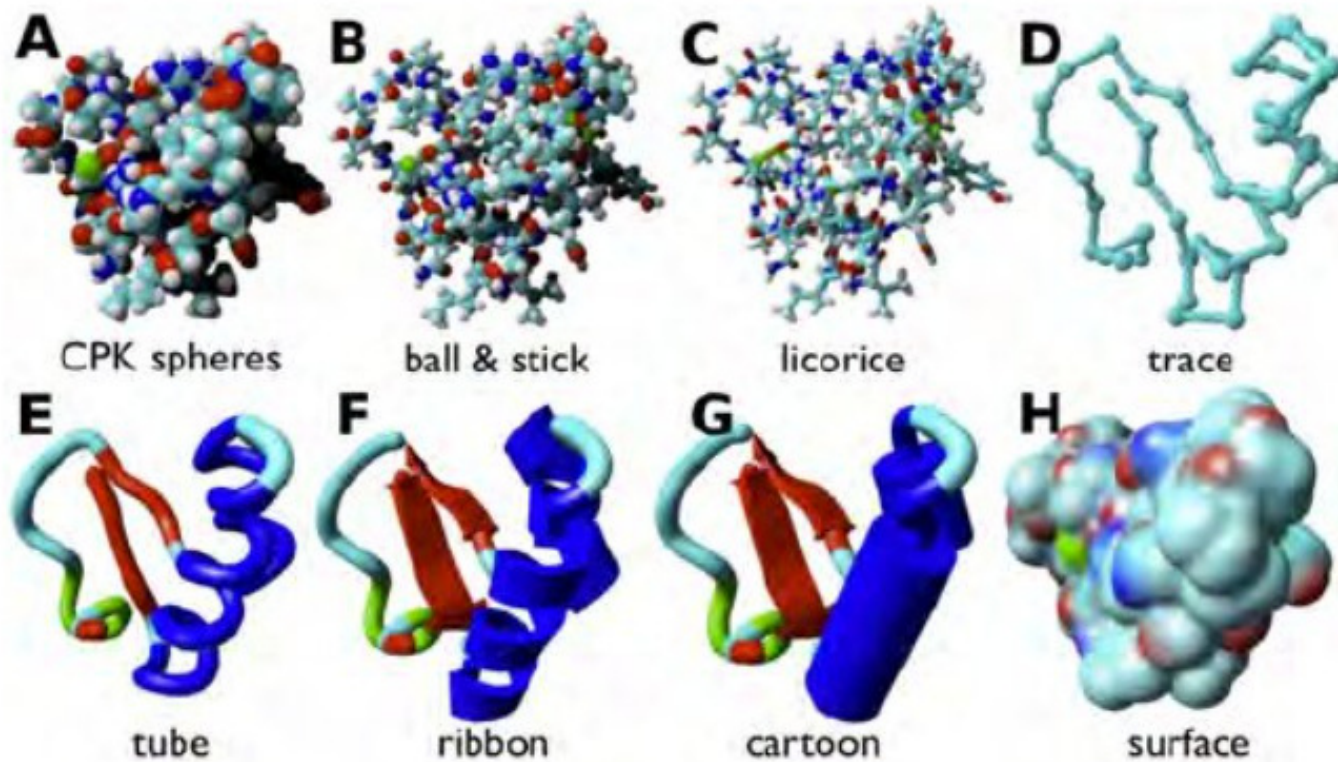
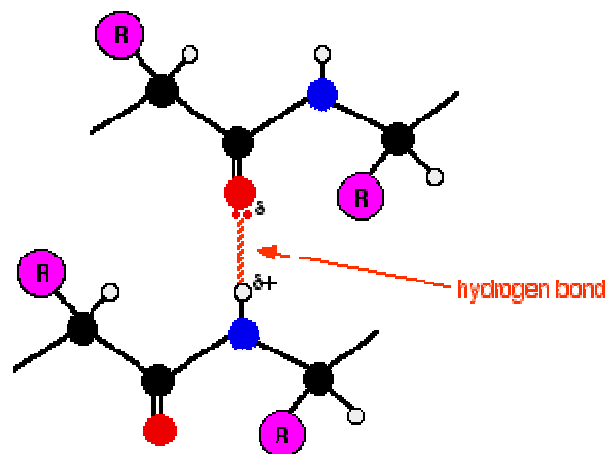


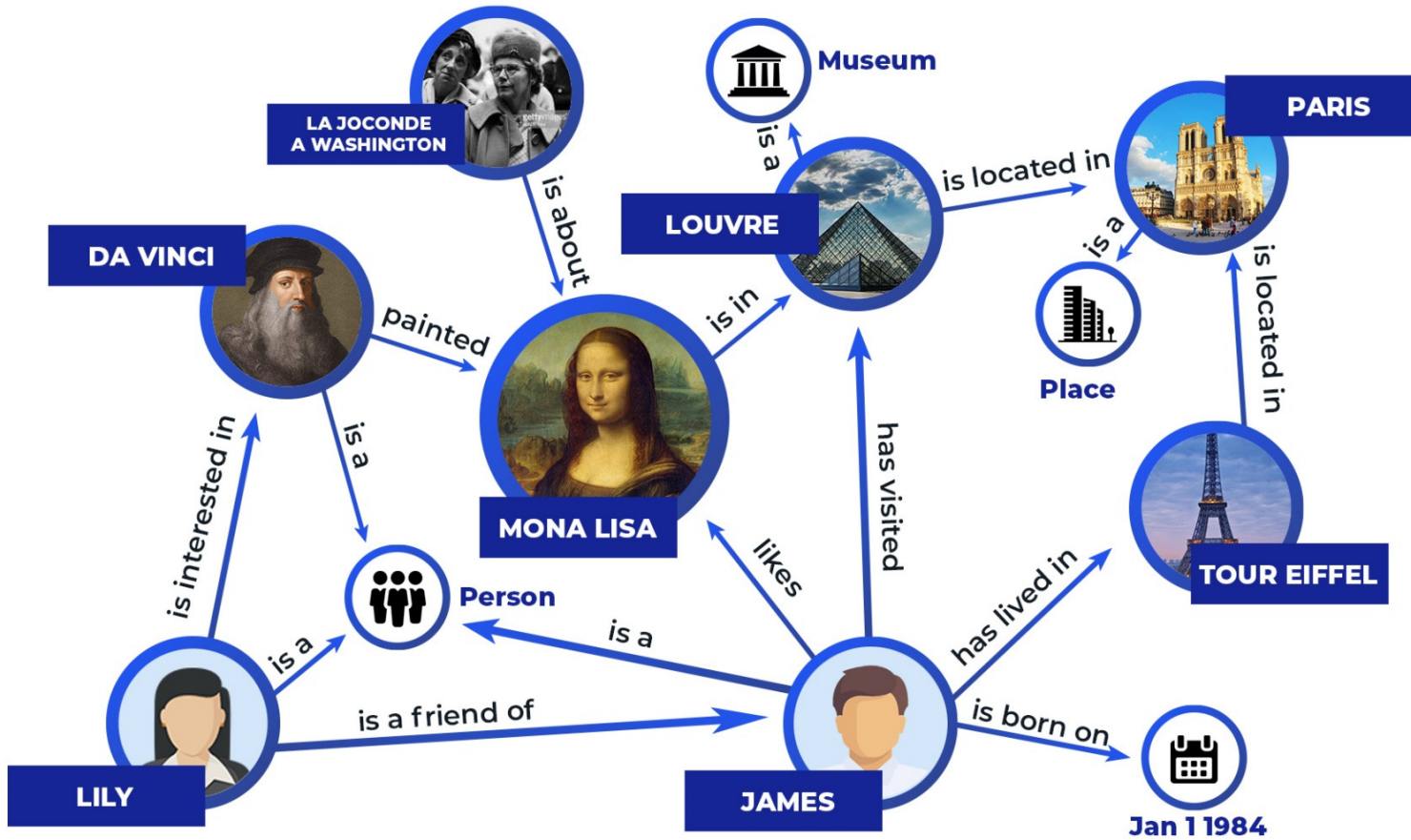
Guanine nucleotide-binding protein G(o) subunit alpha; Guanine nucleotide-binding proteins (G proteins) are involved as modulators or transducers in various transmembrane signaling systems. The G(o) protein function is not clear. Stimulated by RGS14; Belongs to the G-alpha family. G(i/o/t/z) subfamily. (354 aa)

Predicted Functional Partners:


| | | Neighborhood | Gene Fusion | Cooccurrence | Coexpression | Experiments | Databases | Textmining | [Homology] | Score |
|--------|--|--------------|-------------|--------------|--------------|-------------|-----------|------------|------------|-------|
| GNB1 | Guanine nucleotide-binding protein G(I)/G(S)/G(T) subunit beta-1; Guanine nucleotide-binding proteins (G proteins) are involve... | | | | | | | | | 0.996 |
| RGS16 | Regulator of G-protein signaling 16; Regulates G protein-coupled receptor signaling cascades. Inhibits signal transduction by i... | | | | | | | | | 0.993 |
| RGS4 | Regulator of G-protein signaling 4; Inhibits signal transduction by increasing the GTPase activity of G protein alpha subunits th... | | | | | | | | | 0.989 |
| RGS7 | Regulator of G-protein signaling 7; Regulates G protein-coupled receptor signaling cascades. Inhibits signal transduction by in... | | | | | | | | | 0.981 |
| GNG2 | Guanine nucleotide-binding protein G(I)/G(S)/G(O) subunit gamma-2; Guanine nucleotide-binding proteins (G proteins) are invo... | | | | | | | | | 0.980 |
| ADRA2A | Alpha-2A adrenergic receptor; Alpha-2 adrenergic receptors mediate the catecholamine- induced inhibition of adenylyl cyclas... | | | | | | | | | 0.956 |
| DRD2 | D(2) dopamine receptor; Dopamine receptor whose activity is mediated by G proteins which inhibit adenylyl cyclase; Belongs t... | | | | | | | | | 0.950 |
| GNB4 | Guanine nucleotide-binding protein subunit beta-4; Guanine nucleotide-binding proteins (G proteins) are involved as a modulat... | | | | | | | | | 0.947 |
| RIC8A | Synembryn-A; Guanine nucleotide exchange factor (GEF), which can activate some, but not all, G-alpha proteins. Able to activa... | | | | | | | | | 0.946 |
| GNB5 | Guanine nucleotide-binding protein subunit beta-5; Enhances GTPase-activating protein (GAP) activity of regulator of G protein... | | | | | | | | | 0.935 |

Graph-based representations



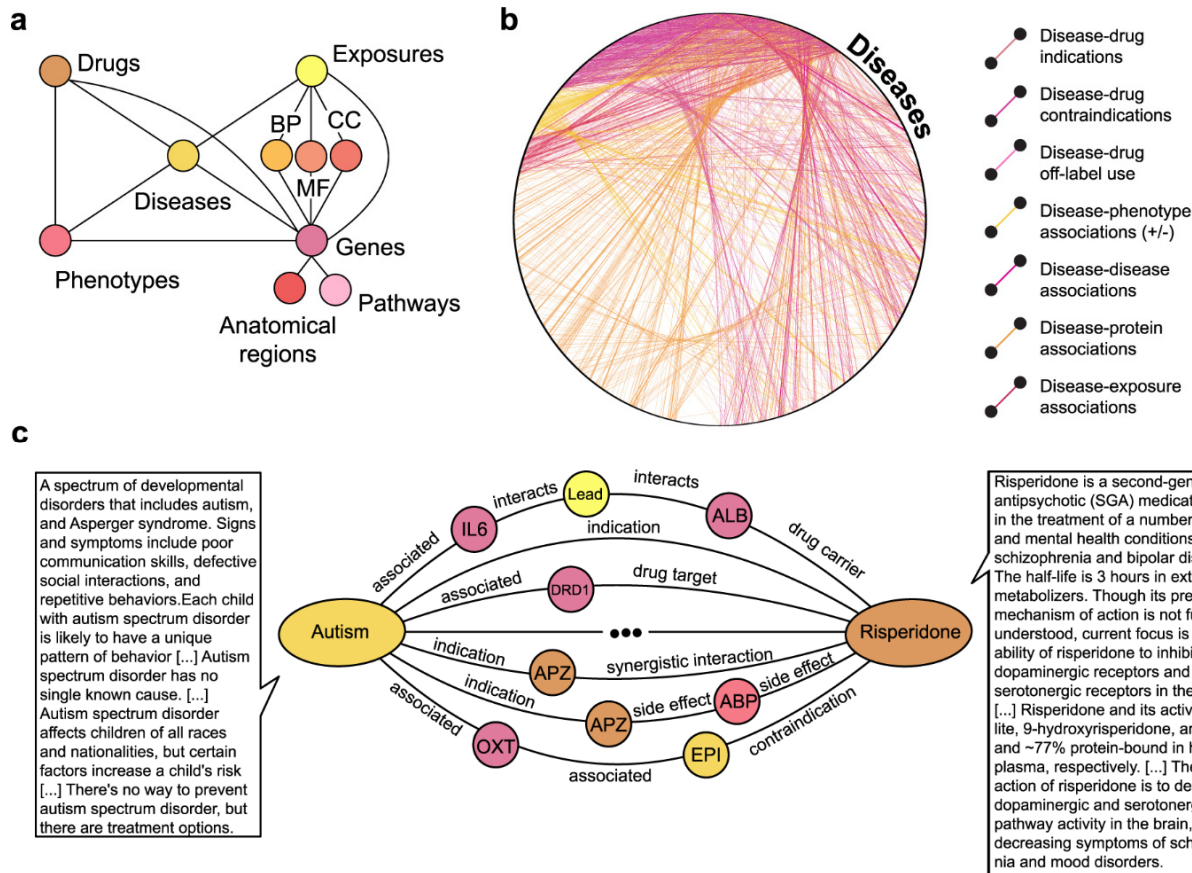


Building a knowledge graph to enable precision medicine

[Payal Chandak](#), [Kexin Huang](#) & [Marinka Zitnik](#) 

[Scientific Data](#) **10**, Article number: 67 (2023) | [Cite this article](#)

Graph-based representations



Graph-based representations

◆ Protein–Protein Interaction (PPI) Databases

- **STRING** – protein–protein interaction networks (experimental + predicted).
- **BioGRID** – curated database of protein and genetic interactions.
- **IntAct** (EMBL-EBI) – molecular interaction database.
- **DIP** (Database of Interacting Proteins) – experimentally determined interactions.
- **MINT** – molecular interactions curated from literature.

◆ Pathways & Biological Networks

- **KEGG** (Kyoto Encyclopedia of Genes and Genomes) – pathway maps (metabolic, signaling, disease).
- **Reactome** – curated biological pathways, with graph export.
- **WikiPathways** – community-curated pathways in graph form.
- **MetaCyc** – metabolic pathways across organisms.

Graph-based representations

◆ Disease–Gene & Heterogeneous Graphs

- DisGeNET – gene–disease associations.
- Hetionet – heterogeneous biomedical knowledge graph (compounds, diseases, genes, side effects).
- PharmGKB – pharmacogenomic networks linking genes, drugs, and diseases.

◆ Chemical & Drug Networks

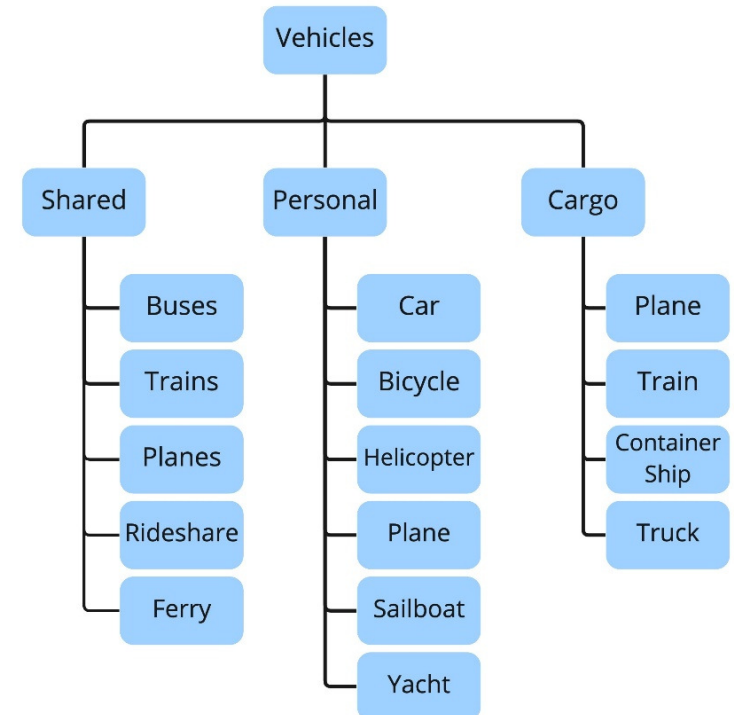
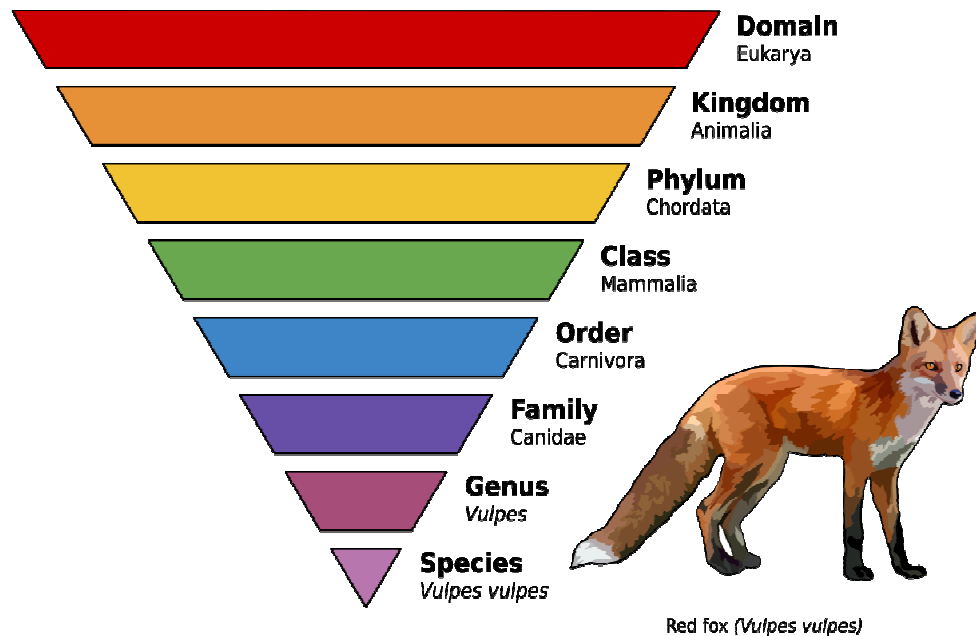
- DrugBank – drug–target and drug–drug interactions.
- STITCH – chemical–protein interaction networks (extension of STRING).
- ChEMBL – bioactive molecules and targets (graph queries possible).

◆ Integrated Knowledge Graphs

- Bio2RDF – converts multiple biomedical databases into RDF linked data.
- OpenBioLink – benchmark KG for drug–target/disease prediction.
- Wikidata (biomedical subset) – large open KG including genes, proteins, pathways, drugs.

Ontology and Taxonomy representations

Taxonomy

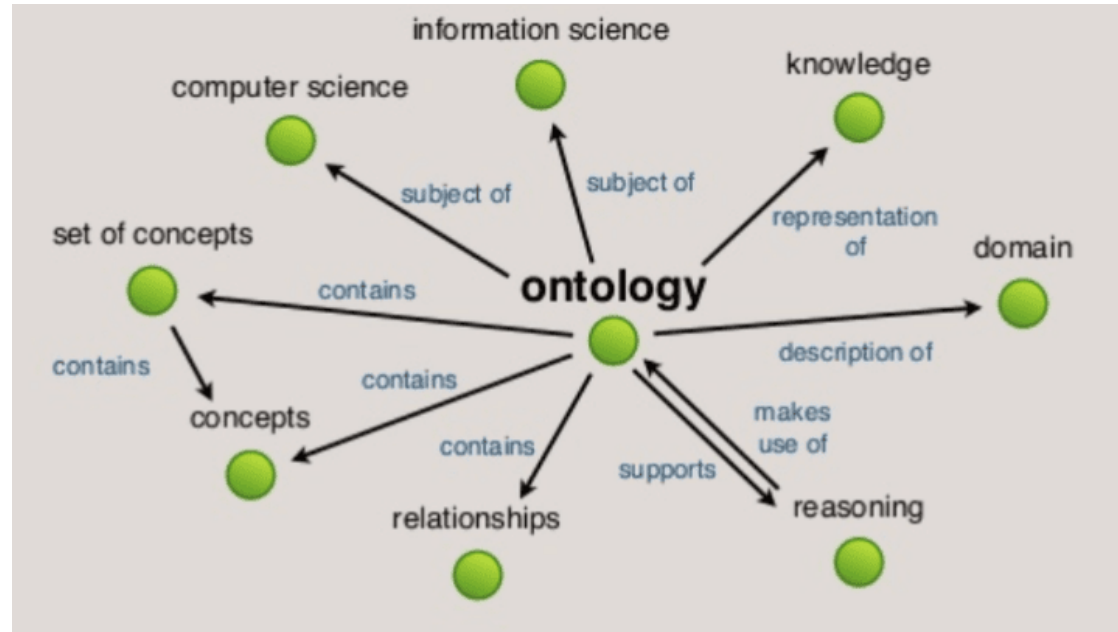


hierarchical classification of entities

Ontology and Taxonomy representations

Ontology

richer web of concepts and relations



Strengths:

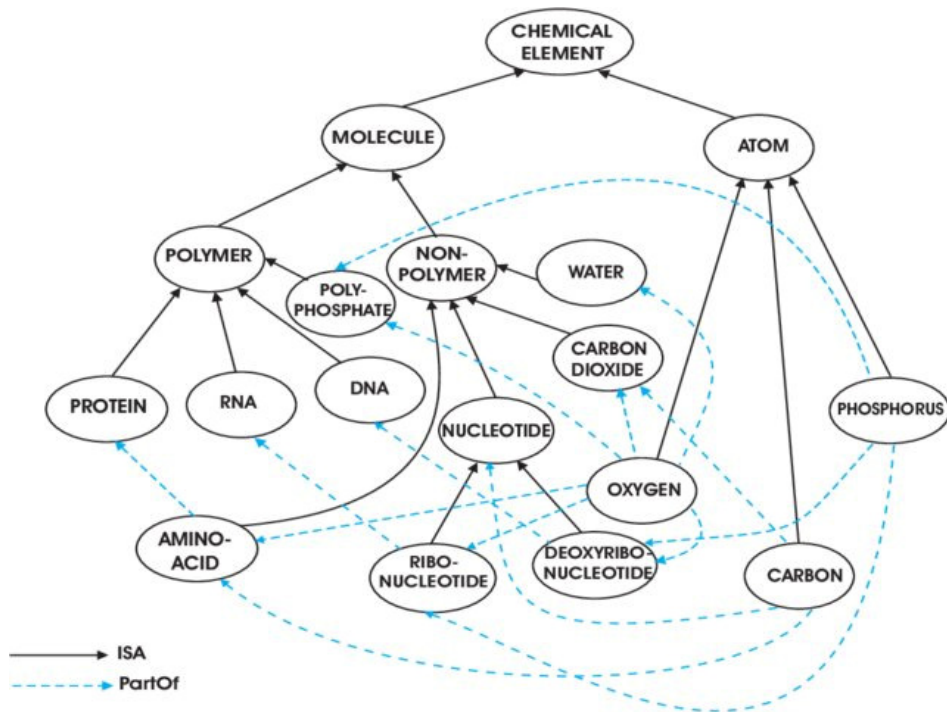
- Provides **shared vocabulary** for a community.
- Supports **reasoning and inference** (e.g., if “all kinases phosphorylate proteins” and “X is a kinase”, then infer X phosphorylates proteins).
- Enables **data integration** across heterogeneous databases.
- Foundation of the **Semantic Web and FAIR principles**.

Limitations

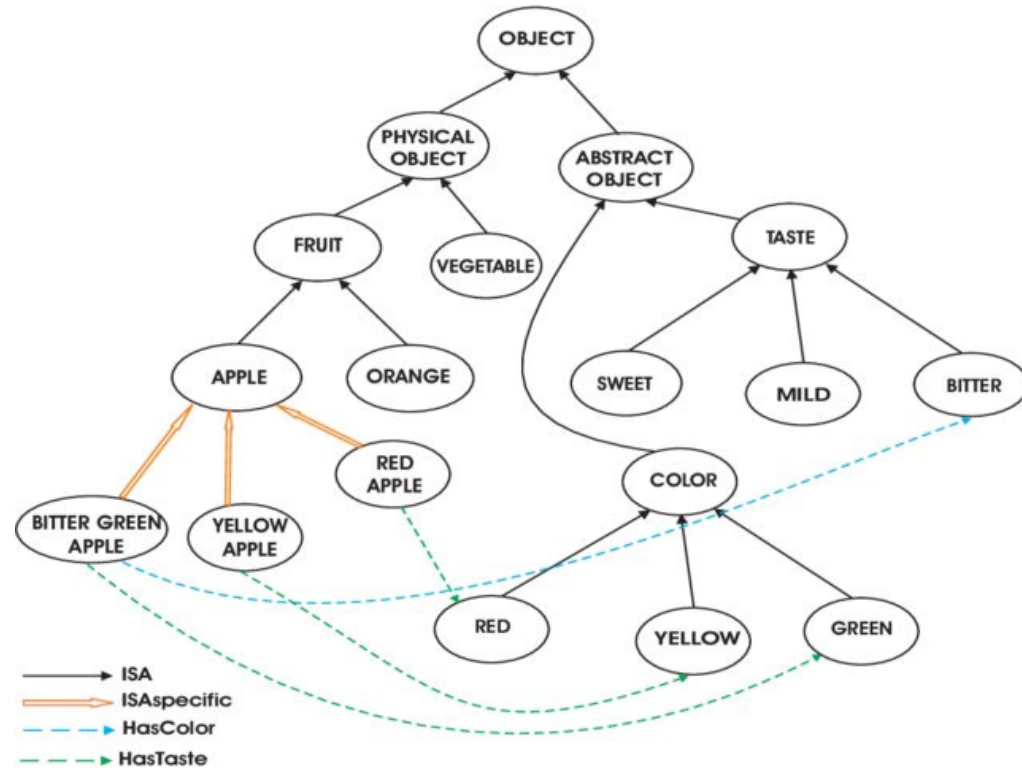
- Building and maintaining ontologies is **labor-intensive**.
- Ambiguity, synonyms, and evolving biology pose challenges.
- Reasoning can be **computationally expensive** on large ontologies.

Ontology and Taxonomy representations

Chemical ontology



Fruit ontology



Ontology and Taxonomy representations

◆ Core Biological Ontologies

- Gene Ontology (GO) – functions, processes, and cellular components of genes/proteins.
- Sequence Ontology (SO) – terms for genomic features (exon, intron, promoter, variant types).
- Protein Ontology (PRO) – structured vocabulary of protein forms, complexes, and modifications.
- NCBI Taxonomy (strictly a taxonomy, but used as an ontology backbone for species classification).

◆ Chemical & Molecular Ontologies

- ChEBI (Chemical Entities of Biological Interest) – small molecules and roles.
- Rhea – curated reactions ontology (linked with ChEBI and UniProt).
- PSI-MOD – controlled vocabulary of protein modifications.
- OBI (Ontology for Biomedical Investigations) – experimental methods, assays, instruments.

◆ Disease & Phenotype Ontologies

- Disease Ontology (DO) – human diseases with hierarchical classification.
- Human Phenotype Ontology (HPO) – standardized vocabulary of phenotypic abnormalities.
- MONDO Disease Ontology – unified disease ontology integrating DO, Orphanet, OMIM.
- Orphanet Rare Disease Ontology (ORDO) – rare disease classification.

Ontology and Taxonomy representations

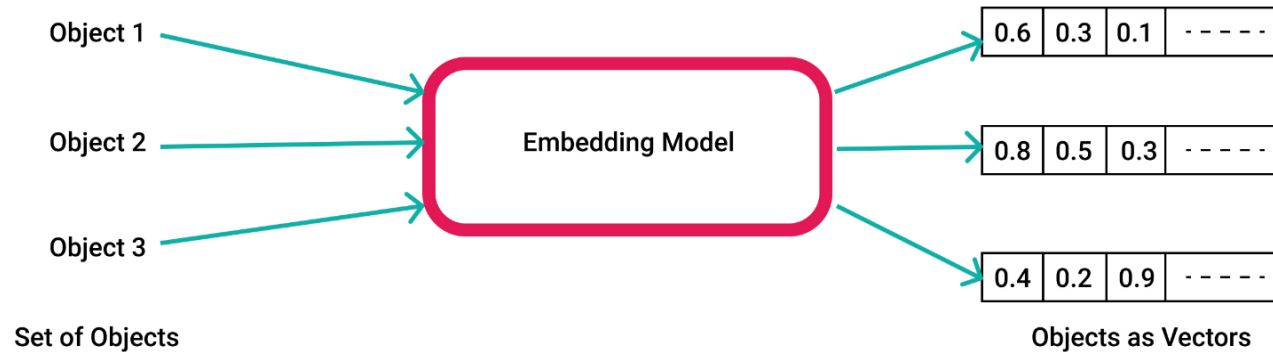
◆ Clinical & Medical Ontologies

- **SNOMED CT** – large-scale clinical terminology (signs, symptoms, diagnoses).
- **ICD-10 / ICD-11** – WHO's classification of diseases.
- **LOINC** – standard for laboratory tests and measurements.
- **UMLS (Unified Medical Language System)** – meta-ontology integrating multiple vocabularies.

◆ Systems Biology & Pathways

- **Reactome Ontology** – structured vocabulary for biological pathways.
- **BioPAX (Biological Pathway Exchange ontology)** – exchange format for pathways.
- **Cell Ontology (CL)** – controlled vocabulary of cell types across species.
- **Uberon** – multi-species anatomy ontology.

Vector-space representations (Embeddings)

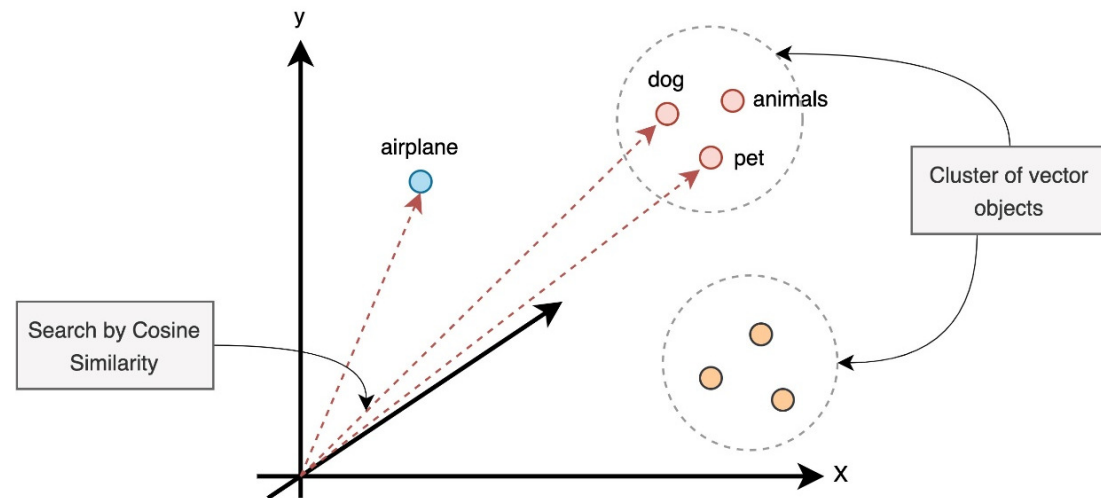


Strengths

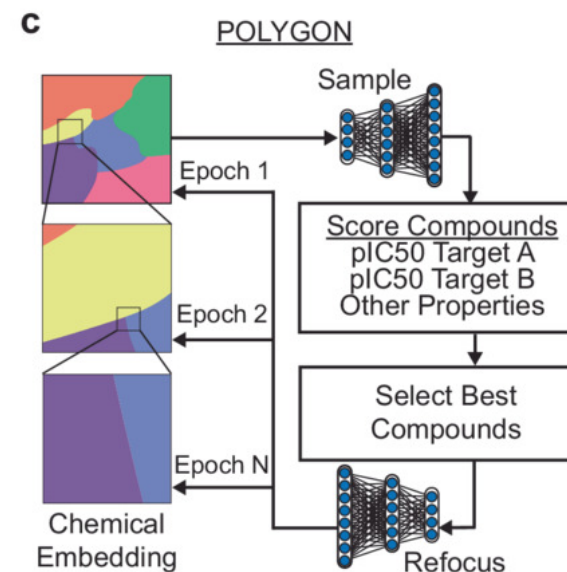
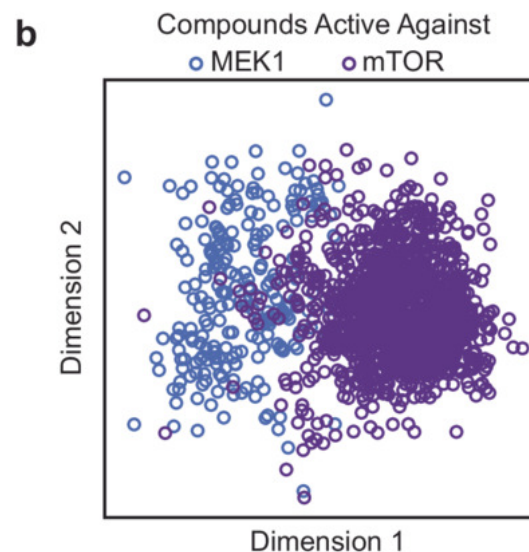
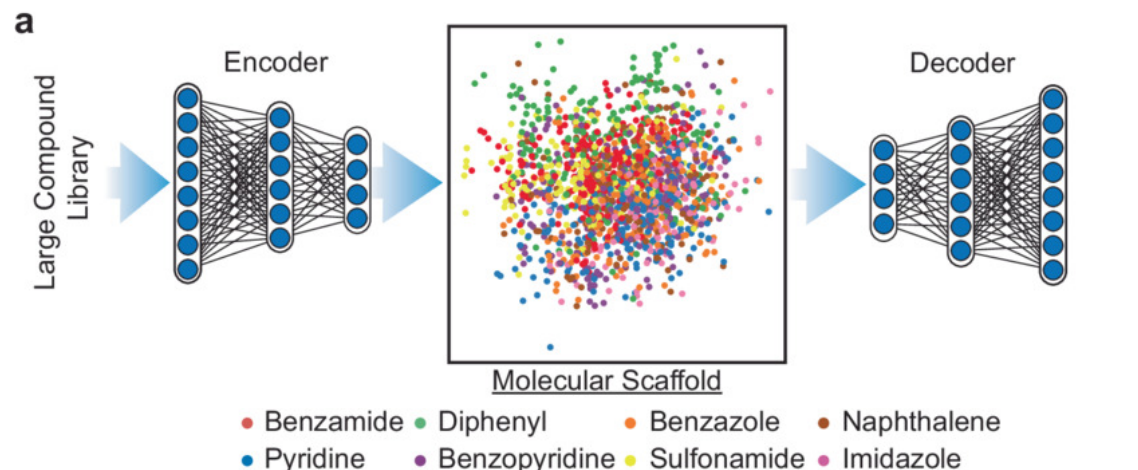
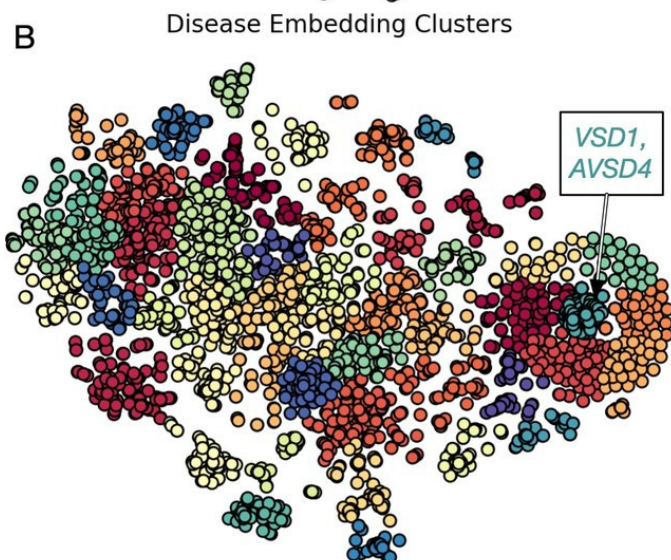
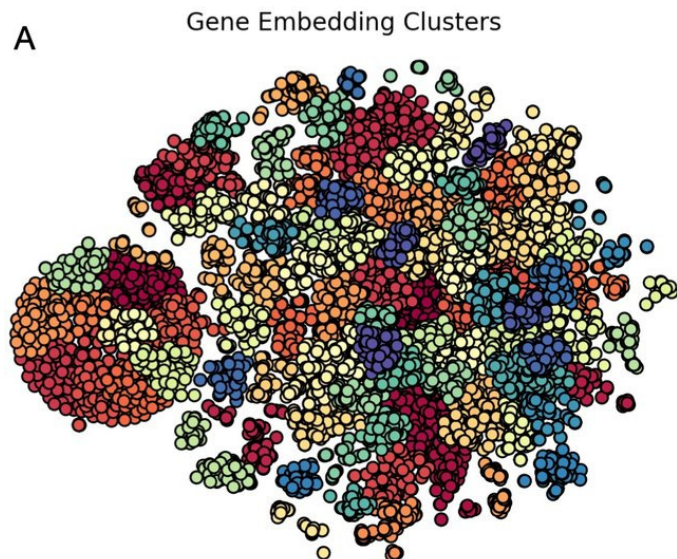
- Captures **semantic similarity** even without explicit links.
- Continuous, compact representation → good for ML and deep learning.
- Supports **clustering, classification, search, and integration** across modalities (e.g., genes + text + images).

Limitations:

- Often **opaque / hard to interpret** (“black box”).
- Doesn’t directly capture **hierarchical or causal structure**.
- Requires large, high-quality training data.



Vector-space representations (Embeddings)



Vector-space representations (Embeddings)

◆ Protein Sequence Embeddings

- **ESM (Evolutionary Scale Models)** – large-scale transformer models trained on UniProt; strong in structure/function prediction.
- **ProtBERT / ProtT5 / ProtTrans** – protein language models trained on millions of sequences, inspired by NLP transformers.
- **ProtVec** – early word2vec-style embeddings of protein k-mers.
- **UniRep** – LSTM-based embeddings of proteins from UniRef50.
- **SeqVec** – protein embeddings using ELMo-like architectures.

◆ Molecular & Drug Embeddings

- **Mol2Vec** – word2vec-like embeddings of molecular substructures (SMILES-based).
- **ChemBERTa** – transformer-based embeddings for small molecules.
- **DeepChem / Chemprop** – GNN-based embeddings for molecules.
- **STITCH embeddings** – drug–chemical–protein interaction networks converted to vectors.

Vector-space representations (Embeddings)

◆ Biomedical Text Embeddings

- **BioBERT** – BERT pretrained on PubMed abstracts and PMC articles.
- **SciBERT** – BERT trained on a large corpus of scientific text (biomedical-heavy).
- **PubMedBERT** – BERT trained *from scratch* on PubMed abstracts.
- **ClinicalBERT** – domain-specific embeddings for clinical notes.
- **BlueBERT** – trained on PubMed + MIMIC clinical notes.

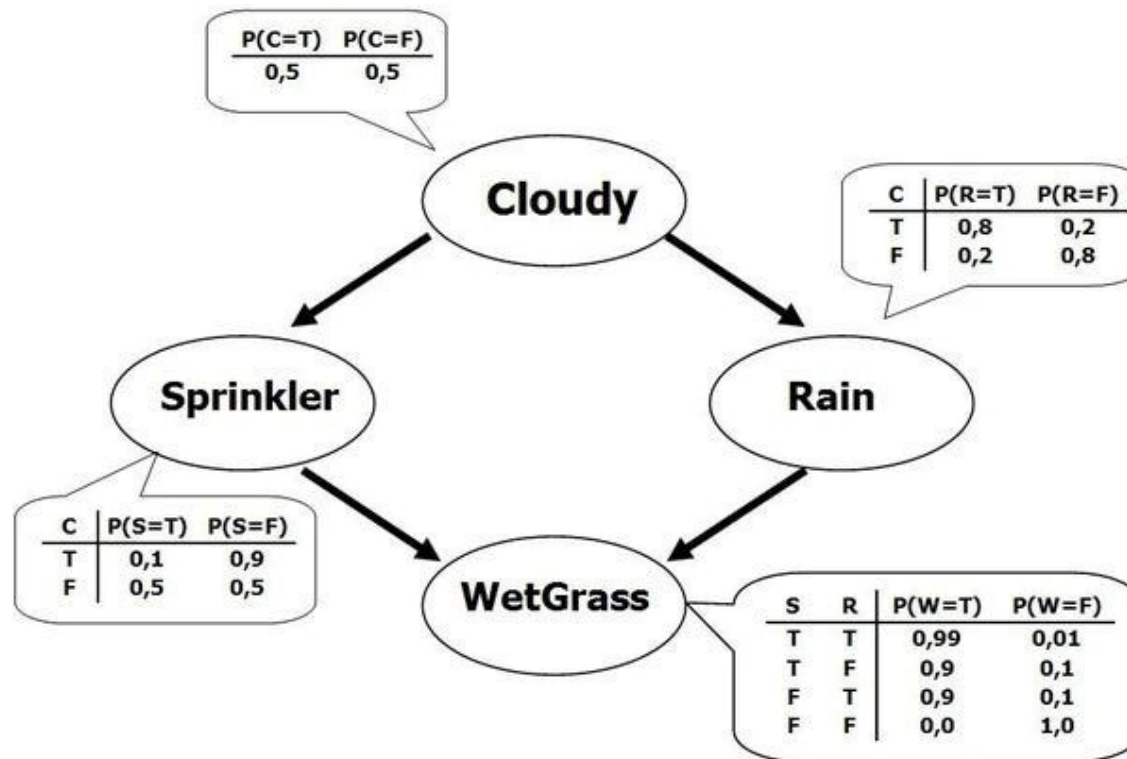
◆ Graph & Network Embeddings

- **node2vec / DeepWalk** – unsupervised embeddings of biological networks (e.g., PPI networks, gene–disease graphs).
- **GraphSAGE** – inductive embeddings for large graphs.
- **HetGNN / Heterogeneous GNNs** – embedding multimodal biomedical networks (e.g., Hetionet).
- **LINE** – scalable embeddings for large biological knowledge graphs.

Probabilistic representations

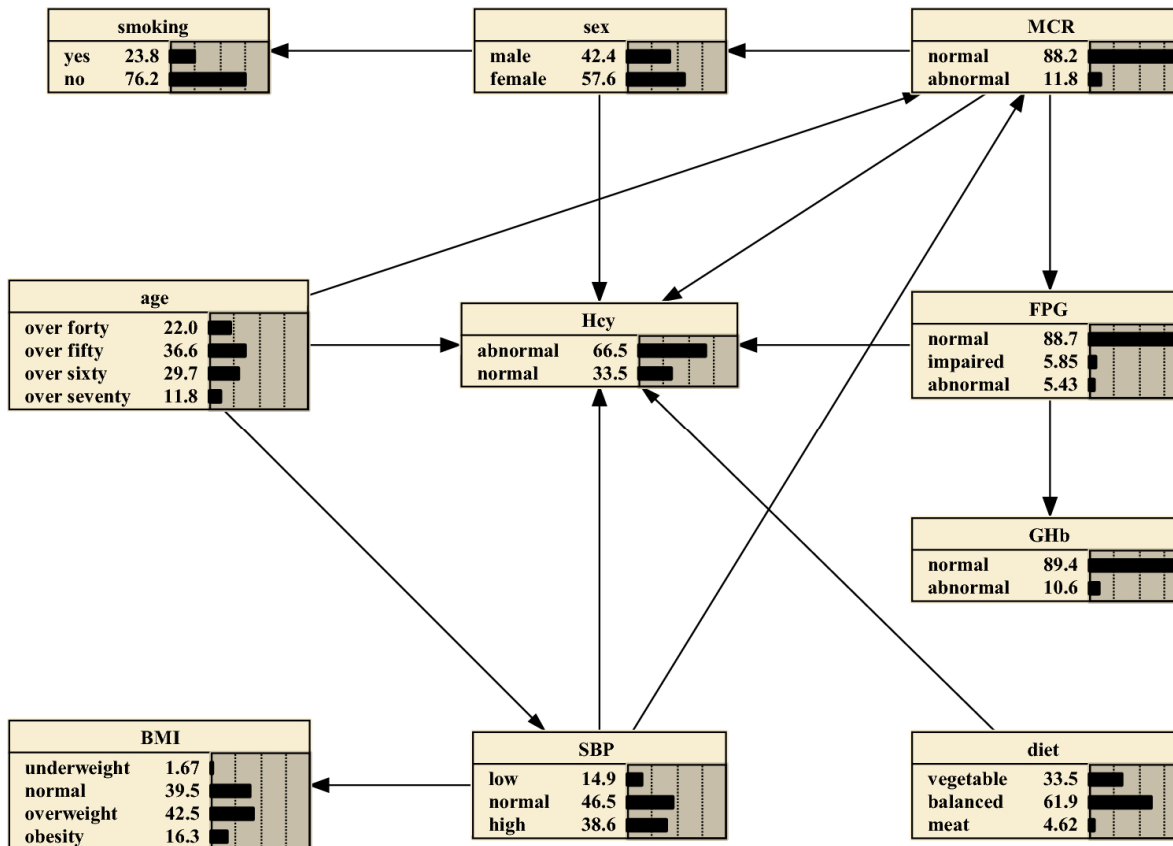
- Real-world biological data is **noisy, incomplete, and uncertain**.
- Probabilistic models represent **uncertainty** about facts and relationships.

Bayesian networks



Probabilistic representations

Bayesian networks



scientific reports

Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [scientific reports](#) > [articles](#) > article

Article | [Open access](#) | Published: 28 January 2023

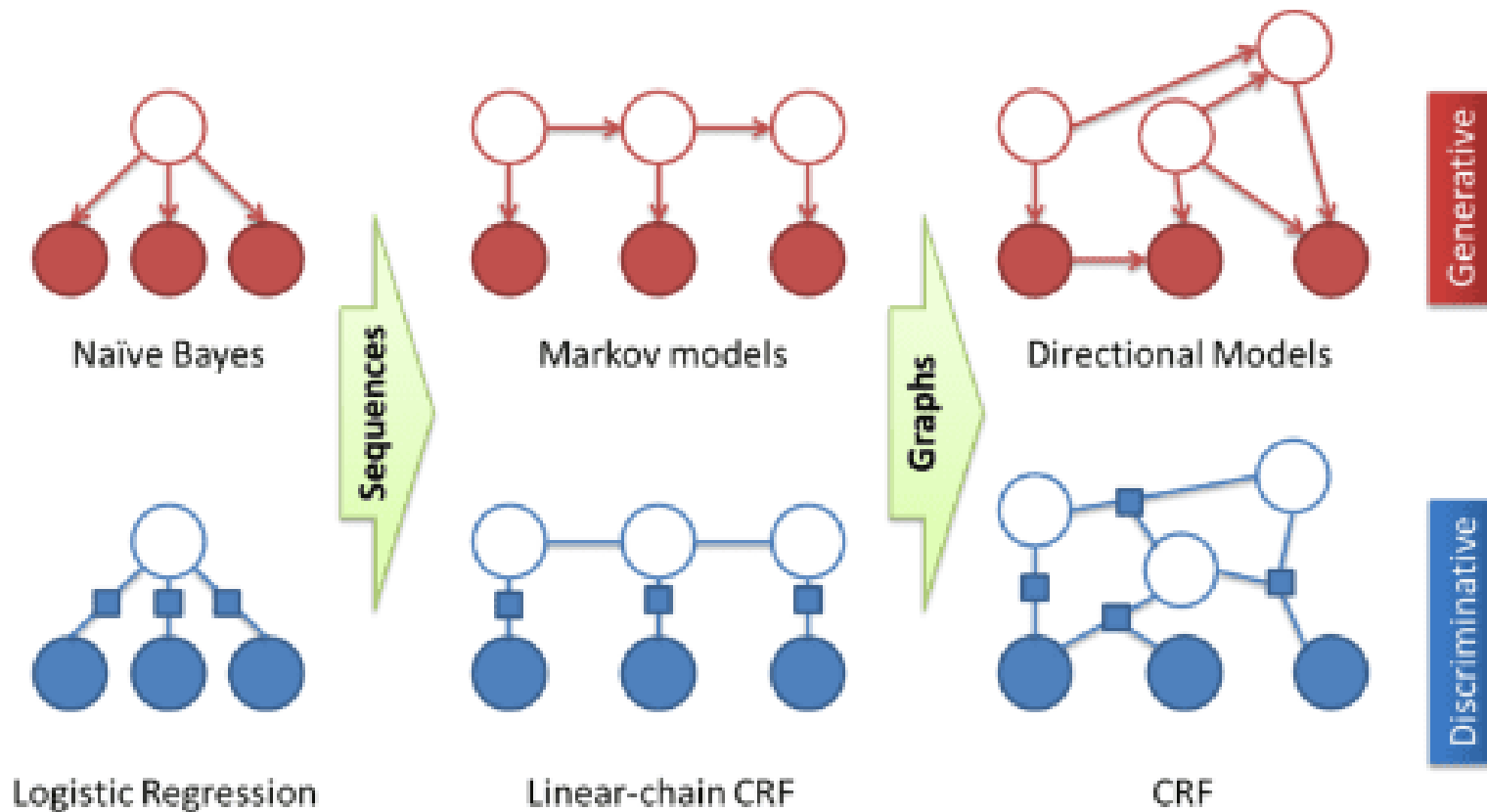
Using Bayesian networks with Tabu-search algorithm to explore risk factors for hyperhomocysteinemia

[Wenzhu Song](#), [Zhiqi Qin](#), [Xueli Hu](#), [Huimin Han](#), [Aizhong Li](#), [Xiaoshaung Zhou](#), [Yafeng Li](#) & [Rongshan Li](#)

[Scientific Reports](#) **13**, Article number: 1610 (2023) | [Cite this article](#)

Probabilistic representations

Markov Random Fields and Conditional Random Fields



Probabilistic representations

Markov Random Fields and Conditional Random Fields

BMC Bioinformatics

Home About Articles Submission Guidelines Collections Join The Board

Submit manuscript

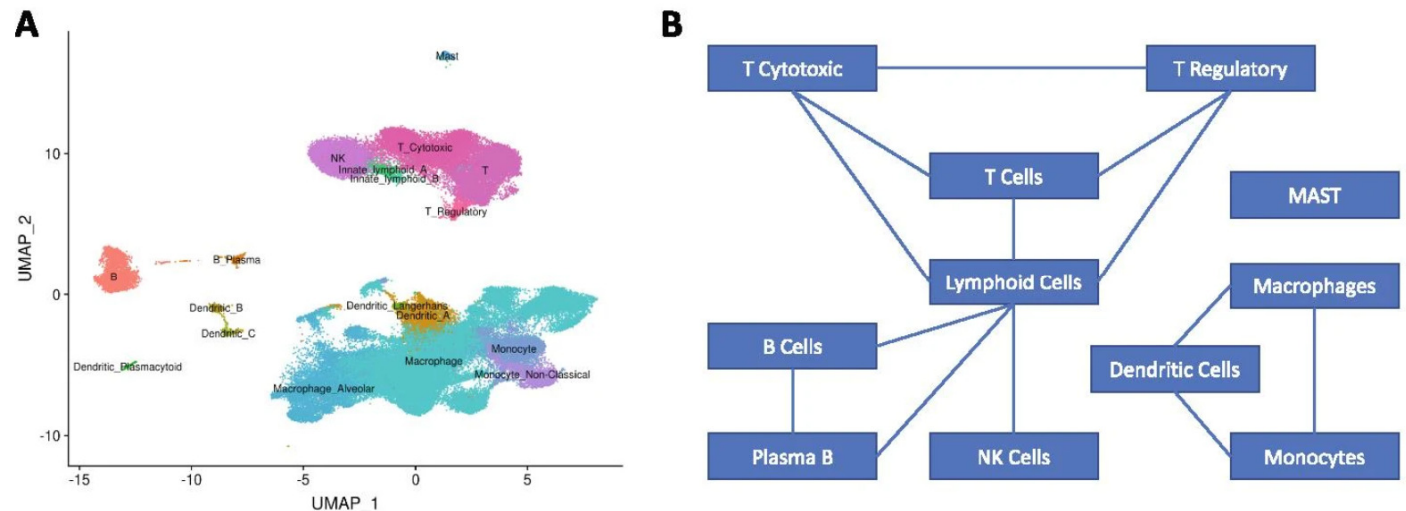
Research | [Open access](#) | Published: 26 October 2021

A Markov random field model for network-based differential expression analysis of single-cell RNA-seq data

Hongyu Li, Biqing Zhu, Zhichao Xu, Taylor Adams, Naftali Kaminski & Hongyu Zhao

BMC Bioinformatics 22, Article number: 524 (2021) | [Cite this article](#)

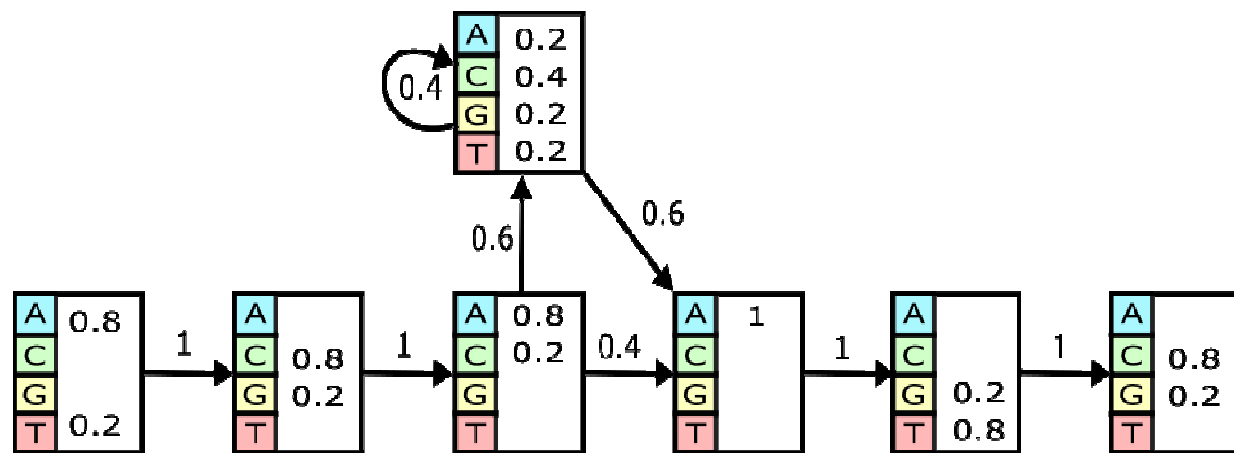
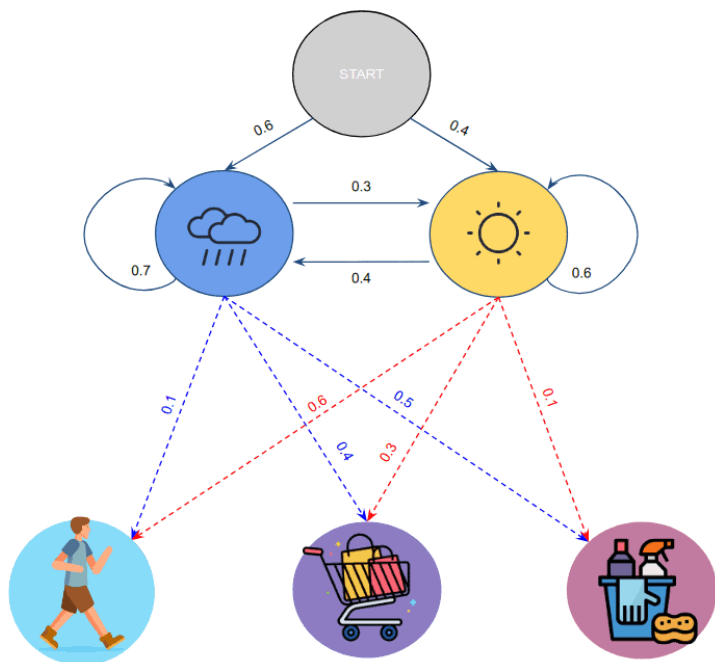
From: [A Markov random field model for network-based differential expression analysis of single-cell RNA-seq data](#)



Cell Type Clusters. **A** shows the UMAP of eighteen immune cell types, and **B** shows the network among these immune cell types that are determined by domain knowledge

Probabilistic representations

Hidden Markov Models



Probabilistic representations

Hidden Markov Models

Start with a multiple sequence alignment

↓

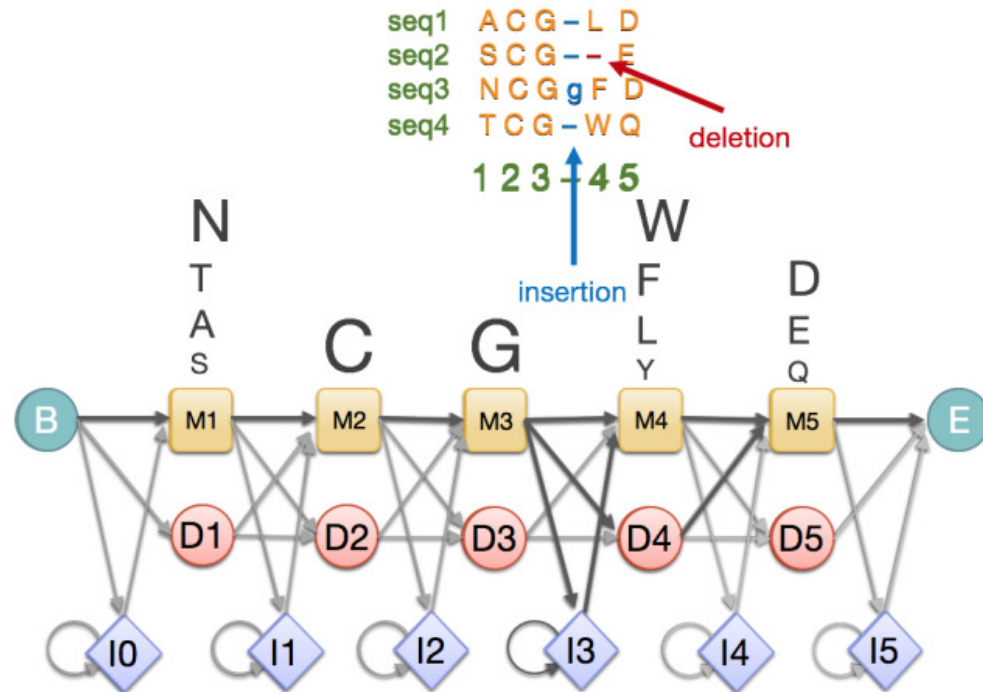
Insertions / deletions can be modelled

↓

Occupancy and amino acid frequency at each position in the alignment are encoded

↓

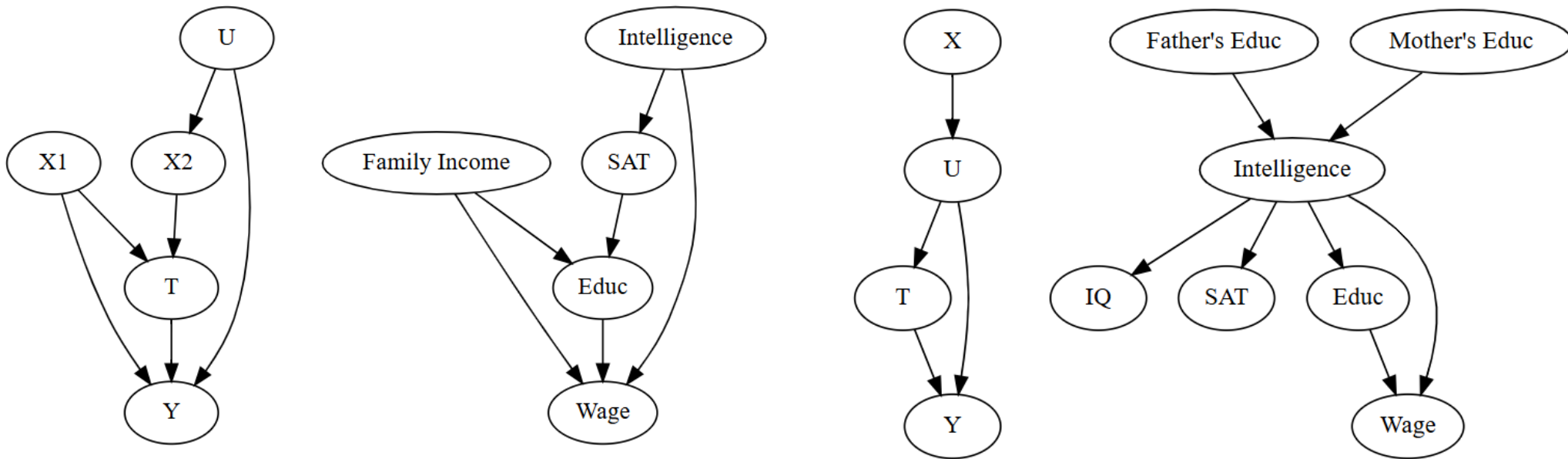
Profile created



<https://www.ebi.ac.uk/training/online/courses/pfam-creating-protein-families/what-are-profile-hidden-markov-models-hmms/>

Causal representations

Causal graphs



<https://matheusfacure.github.io/python-causality-handbook/04-Graphical-Causal-Models.html>

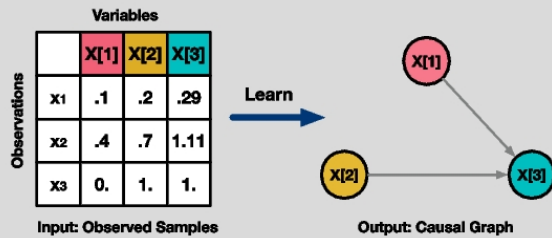
Causal representations

Developing a novel causal inference algorithm for personalized biomedical causal graph learning using meta machine learning

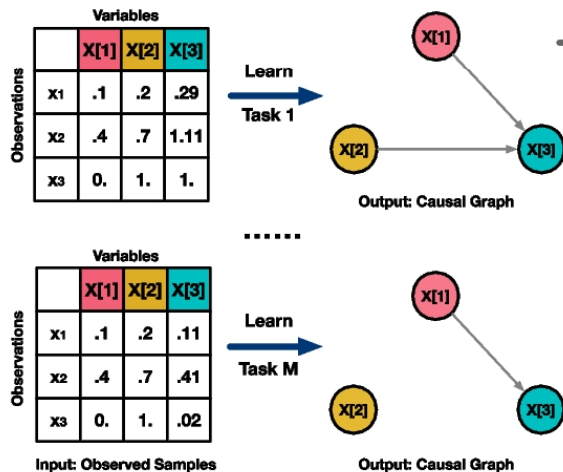
Hang Wu, Wenqi Shi & May D. Wang

BMC Medical Informatics and Decision Making 24, Article number: 137 (2024) | Cite this article

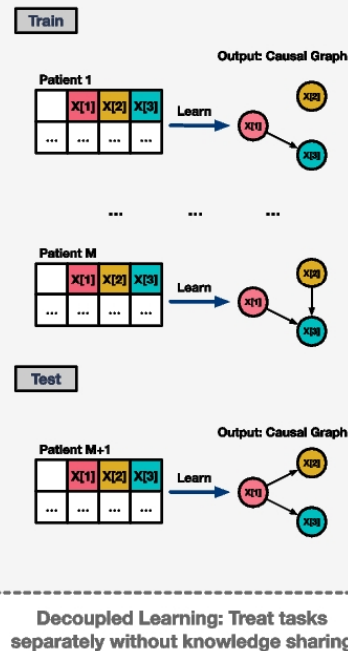
A Single Dataset -> A Single Graph



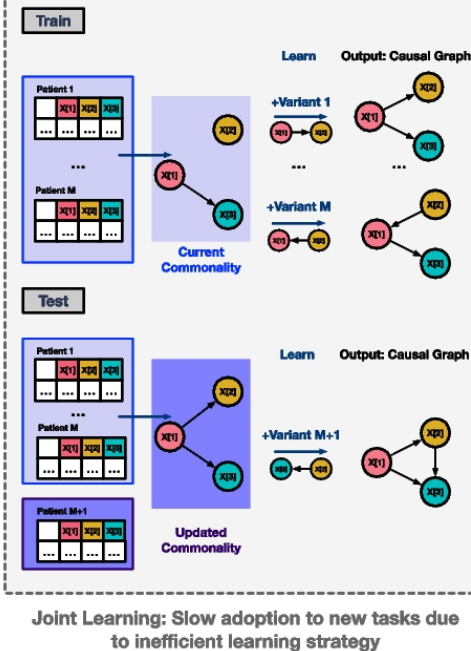
Multiple Datasets -> Multiple Graphs



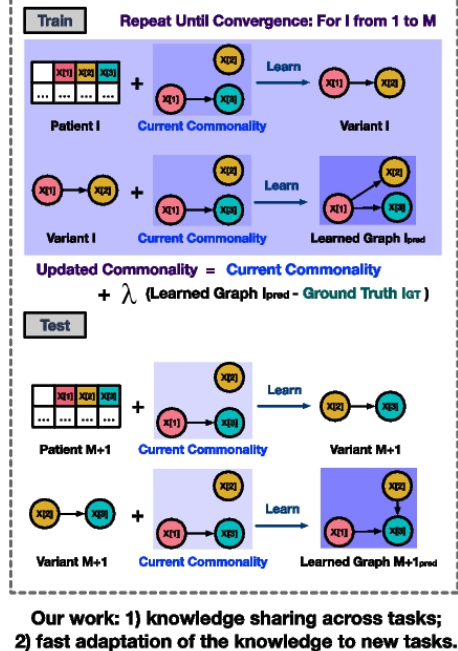
(a) Baseline 1: Decoupled Learning



(b) Baseline 2: Joint Learning



(c) Proposed Method: Meta Learning



LLMs

Core idea:

- **LLMs (e.g., GPT, BioBERT, ESM, ProtGPT2)** store knowledge in their **parameters** after training on massive corpora.
- Represent knowledge **implicitly** rather than through explicit symbols, graphs, or equations.
- Can answer questions, generate hypotheses, and integrate across domains.

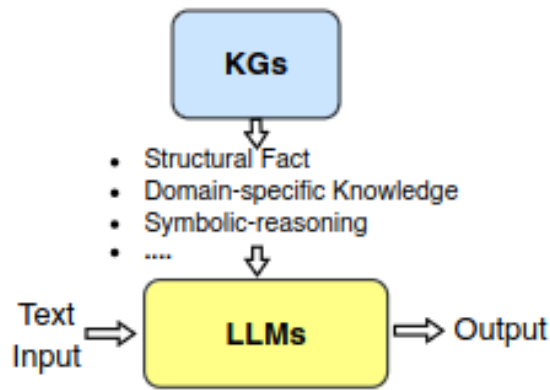
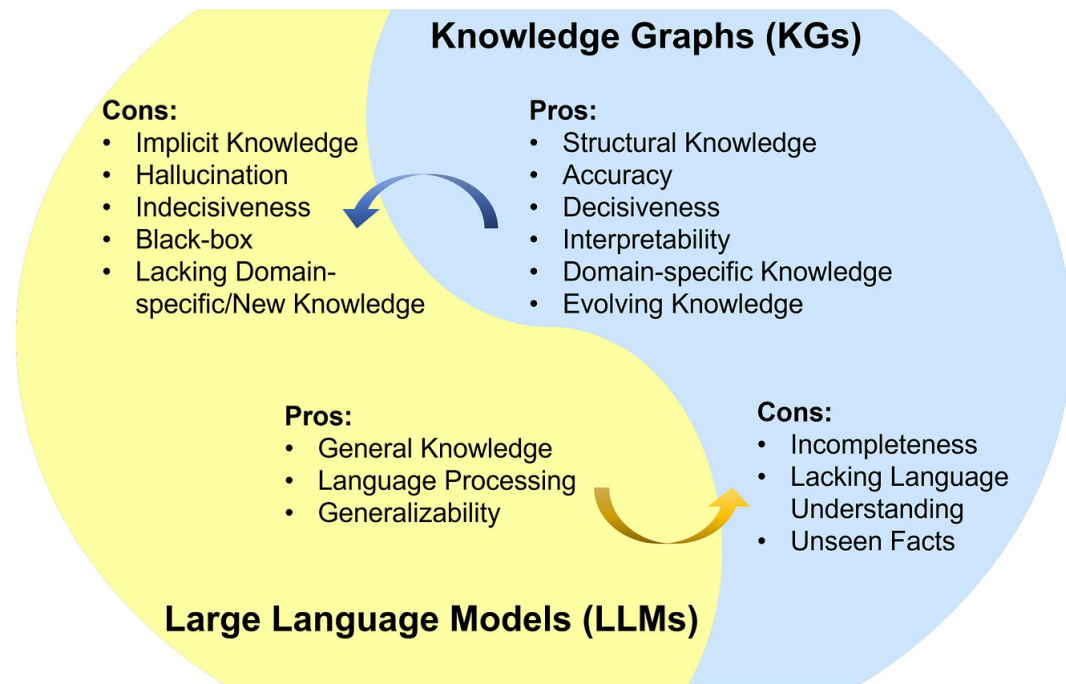
Strengths:

- **Scalable knowledge capture:** billions of facts encoded in weights.
- **Natural language interface:** can interact using plain text, bridging expert and machine.
- **Cross-domain reasoning:** integrate biological, clinical, and chemical knowledge.
- **Few-shot/zero-shot learning:** apply to new tasks without retraining.

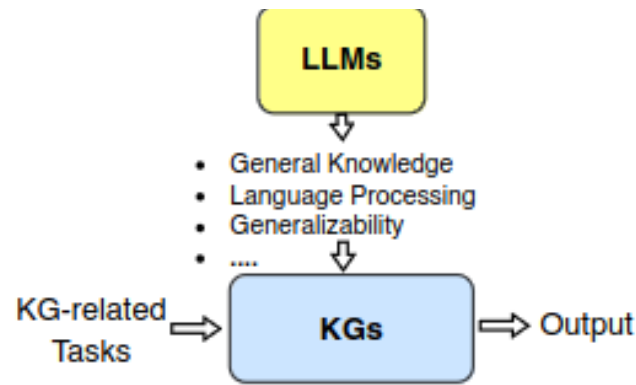
Limitations:

- **Hallucinations:** may generate plausible but false statements.
- **Lack of explicit semantics:** hard to guarantee correctness or trace reasoning.
- **Updating knowledge** requires retraining or fine-tuning.
- **Interpretability challenge:** we don't know exactly "where" a fact is stored.

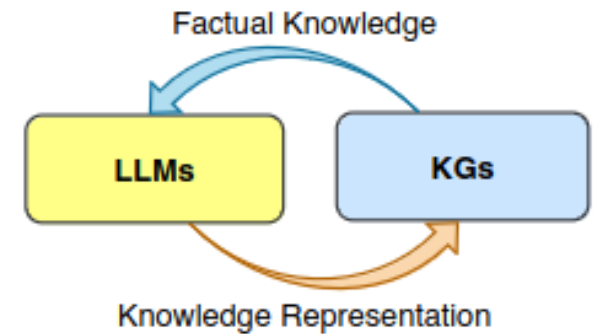
LLMs



a. KG-enhanced LLMs



b. LLM-augmented KGs



c. Synergized LLMs + KGs

<https://www.wisecube.ai/blog/combining-large-language-models-and-knowledge-graphs/>

LLMs

◆ Biomedical & Clinical Text LLMs

- **BioBERT** – BERT pretrained on PubMed abstracts + PMC full-text.
- **PubMedBERT** – trained *from scratch* only on PubMed abstracts (better biomedical domain fit).
- **SciBERT** – trained on a large corpus of scientific publications (biomedical-heavy).
- **ClinicalBERT** – fine-tuned on MIMIC-III clinical notes for clinical NLP.
- **BlueBERT** – pretrained on PubMed + MIMIC-III clinical data.
- **BioClinicalBERT** – variant of BioBERT adapted to clinical text.
- **Med-BERT** – trained on structured EHR data for patient-level prediction.

◆ Multimodal / Knowledge-Integrated Biomedical LLMs

- **BioMegatron** – large transformer trained on PubMed and clinical text (NVIDIA).
- **MedGPT / BioGPT** – GPT-style biomedical text generators trained on PubMed.
- **GALEN** – LLM integrating biomedical text with UMLS ontology.
- **GatorTron** – very large clinical LM trained on de-identified EHRs (UF Health + NVIDIA).
- **DRAGON** – integrates protein sequences + drug data for drug repurposing tasks.

Contents

- Introduction
- Tabular representations
- Graph-based representations
- Ontologies and taxonomies
- Vector-space representations
- Probabilistic representations
- Causal representations
- Large Language Models as implicit representations



CEU

*Universidad
San Pablo*

Lesson 3. Logical agents and expert systems

Medicin School

Contents

- Propositional
- 1st order logic
- Knowledge base
- Reasoning in propositional logic
- Horn clauses
- Expert systems
- AI Planning
- Adversarial games

Propositional logic

Introduction to Propositional Logic

Propositional logic (also called **Boolean logic**) is the simplest form of logic used to represent knowledge. It deals with **propositions** — statements about the world that can be either **true** or **false**, but not both.

1. Basic Elements

- **Propositions:** atomic statements, usually written as capital letters:
 - `P = "BRCA1 is a gene"`
 - `Q = "BRCA1 is associated with breast cancer"`
- **Truth values:** each proposition is either **true** (T) or **false** (F).

2. Logical Connectives

We can build more complex statements using connectives:

| Symbol | Name | Example | Meaning |
|-------------------|---------|-----------------------|---|
| \neg | NOT | $\neg P$ | "not P" (true if P is false) |
| \wedge | AND | $P \wedge Q$ | "P and Q" (true if both are true) |
| \vee | OR | $P \vee Q$ | "P or Q" (true if at least one is true) |
| \rightarrow | IMPLIES | $P \rightarrow Q$ | "If P then Q" |
| \leftrightarrow | IFF | $P \leftrightarrow Q$ | "P if and only if Q" |

Propositional logic

3. Semantics (Truth Tables)

Each connective has a **truth table**. Example: implication ($P \rightarrow Q$):

| P | Q | $P \rightarrow Q$ |
|---|---|-------------------|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

4. Knowledge Bases

- A **knowledge base (KB)** in propositional logic is a set of sentences.
- Example (bioinformatics KB):
 - `Gene(BRCA1)` \rightarrow represented as proposition `G`.
 - `Associated(BRCA1, BreastCancer)` \rightarrow proposition `A`.
 - Rule: `G \wedge A \rightarrow CandidateBiomarker`.

Propositional logic

5. Inference

- **Entailment** (\models): $KB \models \alpha$ means that α must be true in every situation (model) where KB is true.
- **Inference algorithms** (model checking, resolution, forward/backward chaining) try to derive whether α follows from KB .

Example:

If $P = \text{"BRCA1 encodes a DNA repair protein"}$ and $Q = \text{"BRCA1 is cancer-relevant"}$,
and KB contains the rule $P \rightarrow Q$, then knowing P allows us to **infer** Q .

First-order Logic

Introduction to First-Order Logic (FOL)

First-Order Logic (FOL) extends propositional logic by introducing **objects**, **relations**, and **quantifiers**, allowing us to express much richer statements about the world.

1. Basic Elements

- **Constants:** represent specific objects (e.g., `BRCA1`, `BreastCancer`).
- **Predicates:** describe properties of objects or relations between them (e.g., `Gene(x)`, `Associated(x,y)`).
- **Variables:** placeholders for objects (e.g., `x`, `y`).
- **Functions:** map objects to other objects (less common in bioinformatics, but possible: `Encodes(BRCA1) = Protein_BRCA1`).

Example:

- `Gene(BRCA1)` means "BRCA1 is a gene."
- `Associated(BRCA1, BreastCancer)` means "BRCA1 is associated with Breast Cancer."

2. Quantifiers

Two quantifiers extend expressiveness beyond propositional logic:

- **Universal quantifier (\forall):** "for all"
 - Example: $\forall x: \text{Gene}(x) \rightarrow \text{HasDNA}(x)$
("All genes have DNA").
- **Existential quantifier (\exists):** "there exists"
 - Example: $\exists x: \text{Associated}(x, \text{BreastCancer})$
("There exists a gene associated with breast cancer").

First-order Logic

3. Syntax of FOL

- Atomic sentences: `Predicate(term1, ..., termn)`
- Complex sentences: built with logical connectives (\neg , \wedge , \vee , \rightarrow , \leftrightarrow) and quantifiers.

Example:

$\forall g, d: (\text{Gene}(g) \wedge \text{Disease}(d) \wedge \text{Associated}(g, d)) \rightarrow \text{CandidateBiomarker}(g, d)$

"Any gene associated with a disease is a candidate biomarker for that disease."

4. Semantics

- A **model** in FOL specifies:
 - a set of **objects** (the domain),
 - the meaning of **constants**,
 - the truth of each predicate over objects.
- Sentences are **true or false** with respect to a model.

Example model:

- Domain = {BRCA1, TP53, BreastCancer, LungCancer}
- Interpretation: `Gene(BRCA1)=true`, `Associated(TP53, LungCancer)=true`, etc.

First-order Logic

5. Why FOL is More Expressive than Propositional Logic

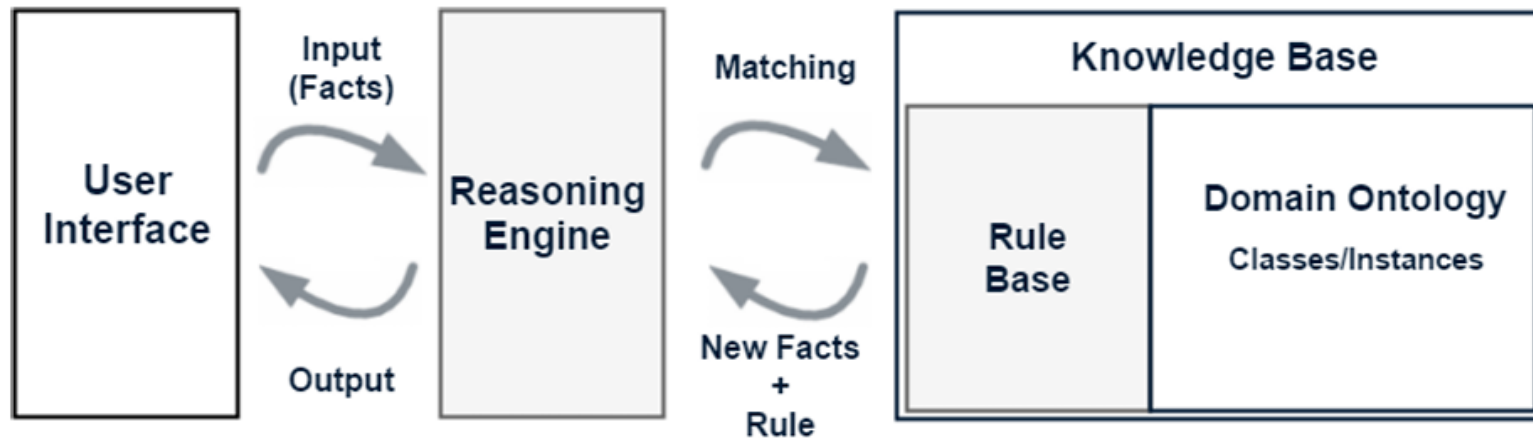
- Propositional logic: $P = \text{"BRCA1 is a gene"} , Q = \text{"TP53 is a gene"}$ — each fact is separate.
- FOL: $\forall x: \text{Gene}(x) \rightarrow \text{HasDNA}(x)$ — one general rule covers all genes.
- This compactness and generality make FOL essential for domains like bioinformatics, where rules apply to large sets of entities.

Knowledge base

Structure of a Knowledge Base

A KB typically includes:

- **Facts:** atomic assertions known to be true (e.g., "BRCA1 is a human gene").
- **Rules:** implications or constraints (e.g., "If a gene is related to DNA repair, then it is relevant for cancer").
- **Ontology / Vocabulary:** definitions of the terms used (e.g., what counts as a gene, protein, disease).



Knowledge base

Facts:

- `Gene(BRCA1)`
- `Gene(TP53)`
- `Disease(BreastCancer)`
- `Disease(LungCancer)`
- `Encodes(BRCA1, Protein_BRCA1)`
- `Encodes(TP53, Protein_p53)`
- `Associated(BRCA1, BreastCancer)`
- `Associated(TP53, LungCancer)`
- `DNARepair(Protein_BRCA1)`
- `TumorSuppressor(Protein_p53)`

Rules:

1. $\forall g, p : \text{Encodes}(g, p) \wedge \text{DNARepair}(p) \rightarrow \text{CancerRelevant}(g)$
2. $\forall g, d : \text{Associated}(g, d) \rightarrow \text{CandidateBiomarker}(g, d)$
3. $\forall p : \text{TumorSuppressor}(p) \rightarrow \text{CancerRelevantGeneOf}(p)$

Possible inference:

- From `Encodes(BRCA1, Protein_BRCA1)` and `DNARepair(Protein_BRCA1)`, we infer `CancerRelevant(BRCA1)`.
- From `Associated(BRCA1, BreastCancer)`, we infer `CandidateBiomarker(BRCA1, BreastCancer)`.

Knowledge base

Facts: Triples in RDF model (Resource Description Framework)

```
:BRCA1    rdf:type    :Gene .
:TP53     rdf:type    :Gene .
:BreastCancer rdf:type :Disease .
:LungCancer rdf:type :Disease .

:BRCA1    :encodes    :Protein_BRCA1 .
:TP53     :encodes    :Protein_p53 .

:Protein_BRCA1 :hasFunction :DNARepair .
:Protein_p53   :hasFunction :TumorSuppressor .

:BRCA1    :associatedWith :BreastCancer .
:TP53     :associatedWith :LungCancer .
```

Vocabulary: OWL

```
:Gene      rdf:type owl:Class .
:Disease   rdf:type owl:Class .
:Protein   rdf:type owl:Class .
```

Rules: Semantic Web Rule Language (SWRL) style (used with OWL ontologies)

```
Gene(?g) ^ encodes(?g, ?p) ^ hasFunction(?p, DNARepair) → CancerRelevant(?g)
```

Inference: SPARQL

Which genes are candidate
biomarkers for BreastCancer?

```
SELECT ?gene WHERE {
  ?gene :associatedWith :BreastCancer .
}
```

Knowledge base

Procedural vs Declarative Knowledge


| Aspect | Procedural Knowledge | Declarative Knowledge |
|------------------------|---|---|
| Definition | "Know-how": instructions for performing tasks, encoded as procedures or algorithms. | "Know-that": explicit statements of facts, rules, and relationships about the world. |
| Representation | Programs, code, procedures (imperative). | Sentences in a logical or ontology-based knowledge representation language (e.g., RDF, OWL, rules). |
| Execution | Directly executed by the system (algorithm follows the steps). | Requires an inference engine to derive new facts or answer queries. |
| Flexibility | Hard to modify or reuse; knowledge is embedded in the procedure. | Easy to update; new facts/rules can be added without rewriting the whole system. |
| Example (general AI) | A function <code>sort(list)</code> implementing bubble sort. | A logical rule: $\forall x,y: \text{Greater}(x,y) \rightarrow \neg \text{Greater}(y,x)$. |
| Bioinformatics Example | A Python script that runs BLAST and parses results step by step. | RDF triples: <code>:BRCA1 :associatedWith :BreastCancer .</code> + rule: <code>Gene(?g) ^ associatedWith(?g, ?d) →</code> <code>CandidateBiomarker(?g, ?d) .</code> |


Knowledge base

Procedural vs Declarative Knowledge

| Aspect | Procedural Knowledge | Declarative Knowledge |
|---------------|--|---|
| Advantages | Efficient; optimized for specific tasks. | Transparent, explainable; reusable across tasks; supports reasoning. |
| Disadvantages | Brittle; hard to adapt; not explainable. | Inference can be computationally expensive; needs good knowledge engineering. |

Knowledge engineering process

1. Identify the task.
2. Assemble relevant knowledge.
3. Choose a vocabulary (ontology).
4. Encode axioms (rules).
5. Encode problem instances (data).
6. Pose queries.
7. Debug the KB .

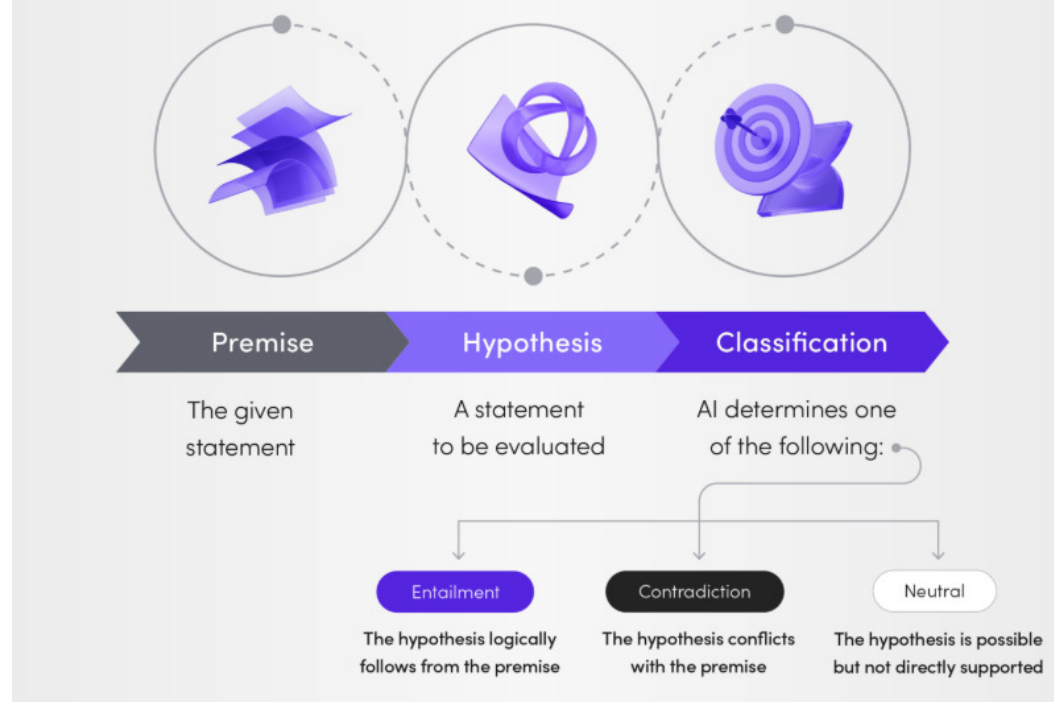
 This is *exactly* what bioinformatics projects like the **Gene Ontology** or **Reactome** have done: careful ontology design, curated axioms, and then queries/inference over large datasets.

Reasoning in propositional logic

1. Entailment vs. Inference

- **Entailment (\models):** A sentence α is *entailed* by a KB if α is true in *every model* where KB is true.
 - Example: KB = {"P \rightarrow Q", "P"} entails Q.
- **Inference (\vdash):** The mechanical process of deriving α from KB using rules.
 - A sound inference algorithm only produces sentences that are actually entailed.

How Natural Language Inference (NLI) Works



Reasoning in propositional logic

2. Key Inference Rules

Some reasoning steps are so standard they are given names:

- **Modus Ponens (Implication Elimination):**

From $(P \rightarrow Q)$ and P , infer Q .

- Example: "If BRCA1 repairs DNA, then BRCA1 is cancer-relevant. BRCA1 repairs DNA. \Rightarrow BRCA1 is cancer-relevant."

- **And-Elimination:**

From $(P \wedge Q)$, infer P (and also Q).

- **And-Introduction:**

From P and Q , infer $(P \wedge Q)$.

- **Double-Negation Elimination:**

From $\neg(\neg P)$, infer P .

- **Contrapositive:**

From $(P \rightarrow Q)$, infer $(\neg Q \rightarrow \neg P)$.

- **De Morgan's Laws:**

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

These rules allow chaining reasoning steps into **proofs**.

Reasoning in propositional logic

3. Proofs and Search

- A **proof** is a sequence of applications of inference rules leading from KB to the goal sentence.
- Finding proofs can be seen as a **search problem**, where states are sentences and operators are inference rules.
- This connects inference to earlier search algorithms (breadth-first, depth-first, etc.).

4. Resolution Rule

- A single, powerful inference rule:
From $(P \vee Q)$ and $(\neg Q \vee R)$, infer $(P \vee R)$.
- Resolution is **sound and complete**: together with search, it can derive any entailed conclusion in propositional logic.
- This is the basis of **SAT solvers** used in many computational tasks today. SAT=Boolean SATisfiability Problem

Reasoning in propositional logic

5. Forward and Backward Chaining

- **Forward chaining (data-driven):**

Start from known facts, apply rules until the query appears.

- Efficient with Horn clauses.
- Example in bioinformatics: "If Gene(g) and Associated(g, d) then Biomarker(g, d)." Start from facts about BRCA1 and derive biomarkers.

- **Backward chaining (goal-driven):**

Start from the query and work backward through rules until reaching known facts.

- Used in logic programming (Prolog).
- Example: "Is BRCA1 a biomarker?" → Check whether the rule conditions can be satisfied.

6. Soundness, Completeness, Monotonicity

- **Soundness:** inference never produces false conclusions.
- **Completeness:** inference can derive all true conclusions.
- **Monotonicity:** once something is inferred, it remains true even if more knowledge is added (contrast with non-monotonic reasoning in later AI).

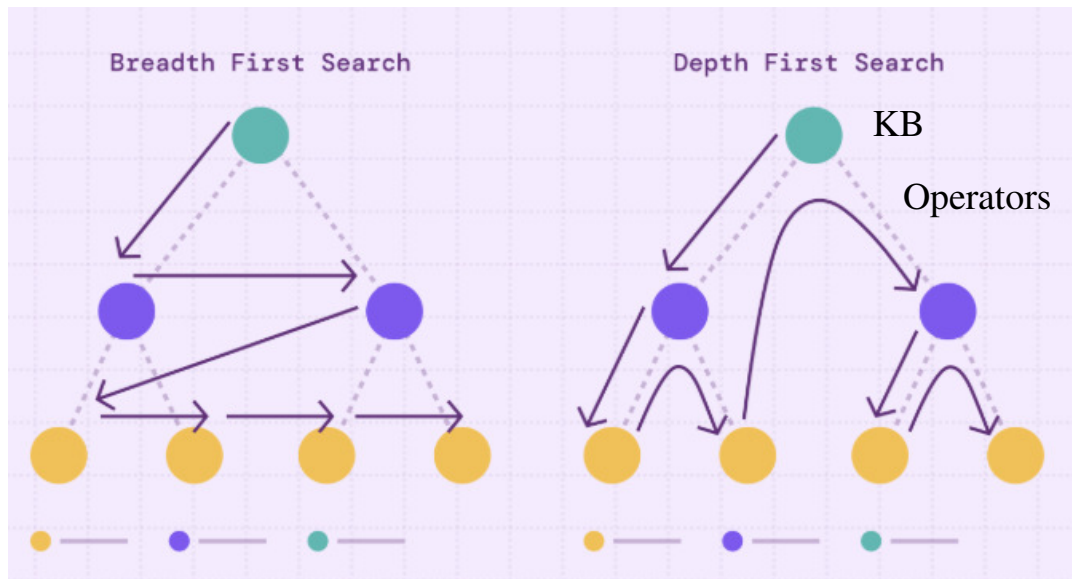
Reasoning in propositional logic

Search Strategies in Propositional Inference

1. Inference as Search

- A **proof** is a sequence of applications of inference rules leading from the knowledge base (KB) to the query (α).
- This can be modeled as a **state space search problem**:
 - **States** = sets of sentences derived so far.
 - **Operators** = inference rules (e.g., Modus Ponens, And-Elimination, Resolution).
 - **Start state** = initial KB.
 - **Goal test** = does the query α appear in the current set of sentences?

Thus, proving α is like finding a path from KB to α in the space of logical consequences.



Reasoning in propositional logic

Search Complexity and Strategies

- **Naïve proof search:** can be very inefficient, since the space of possible sentences is huge.
- **Systematic search strategies:**
 - **Breadth-First Proof Search:** guarantees shortest proof (fewest steps), but memory-heavy.
 - **Depth-First Proof Search:** memory-light, but may get lost in irrelevant inference chains.
 - **Heuristic Search:** prioritizes inference steps that seem closer to the query (e.g., focusing on symbols that appear in the goal).

Diagnostic vs. Causal Reasoning

- The chapter distinguishes **diagnostic rules** (from effect to cause) and **causal rules** (from cause to effect).
- In bioinformatics:
 - Diagnostic reasoning → "If tumor shows microsatellite instability, then mismatch repair genes may be mutated."
 - Causal reasoning → "If BRCA1 is mutated, then DNA repair is impaired, leading to increased cancer risk."
- Modern systems biology integrates both reasoning styles.

Reasoning in propositional logic

1. Heuristic Search

- **Search** = systematic exploration of possible states to find a solution (already seen in pathfinding, game search, and logical inference).
- In **logical inference**, proof search can be framed as a search problem: states = partial proofs, operators = inference rules, goal = prove a query.
- **Heuristics** = guidance functions that estimate "how close" a state is to a goal, to avoid brute-force search.
 - Example in expert systems: choose which rule to apply next based on relevance to the query.
 - In bioinformatics: heuristics guide sequence alignment (e.g., BLAST uses heuristics to prune search), or pathway reconstruction (focus on biologically plausible steps).

👉 So in this context: **heuristic search** = **goal-directed inference**, where rules are not applied blindly but **guided by measures of relevance or probability**.

Horn clauses

1. What is a Horn clause?

A **Horn clause** is a special kind of logical sentence in first-order logic with **at most one positive literal**.

- In propositional terms, a clause is a disjunction of literals (e.g. $\neg P \vee \neg Q \vee R$).
- A **Horn clause** has ≤ 1 **positive literal**.

Examples:

- $\neg P \vee \neg Q \vee R$ (which is equivalent to $P \wedge Q \rightarrow R$)
- $\neg P$ (equivalent to $P \rightarrow \text{False}$)
- R (a fact: equivalent to $\text{True} \rightarrow R$)

2. Why Horn clauses are important

- They form the **logical core of rule-based systems**.
- **Inference is efficient**: forward and backward chaining with Horn clauses can be done in **linear time** relative to the size of the KB, unlike general FOL inference which is semi-decidable and potentially exponential.
- They are the foundation of **logic programming languages** such as **Prolog**.

Horn clauses

3. Forms of Horn clauses

- **Definite clause:** exactly one positive literal.

$$(\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n \vee Q) \equiv (P_1 \wedge P_2 \wedge \dots \wedge P_n) \rightarrow Q$$

→ Used as **production rules** in expert systems.

- **Fact:** no negative literals, only one positive literal.

Q

Example: `Gene(BRCA1)` .

- **Goal clause (query):** no positive literals (only negated terms).

$$\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n$$

Equivalent to asking whether $P_1 \wedge P_2 \wedge \dots \wedge P_n$ can be proven.

Horn clauses

Facts (Horn clauses with no premises):

- `Gene(BRCA1).`
- `Gene(TP53).`
- `Encodes(BRCA1, Protein_BRCA1).`
- `DNARepair(Protein_BRCA1).`

Rules

```
CancerRelevant(G) :- Gene(G), Encodes(G, P), DNARepair(P).
```

Query

```
?- CancerRelevant(BRCA1).
```

→ The inference engine matches the rule, finds that BRCA1 encodes Protein_BRCA1, and Protein_BRCA1 has DNARepair, so it concludes `CancerRelevant(BRCA1)` is true.

Expert systems

1. Definition

An **expert system** is a computer program designed to **emulate the decision-making ability of a human expert**.

- It contains a **knowledge base** (facts + rules) and an **inference engine** (reasoning mechanism).
- Famous examples: **MYCIN** (medical diagnosis), **DENDRAL** (chemistry).

2. Foundations

- Expert systems are **rooted in propositional and first-order logic** reasoning.
- But they usually **do not implement full logical inference** (too expensive).
- Instead, they use **restricted subsets**:
 - **Production rules**: IF <conditions> THEN <conclusion/action>.
 - **Horn clauses**: efficient fragment of FOL.
 - **Forward chaining (data-driven)** or **backward chaining (goal-driven)**.


So, expert systems are **practical implementations** of the logical agents we discussed.

Expert systems

3. Key Differences vs. General Logical Agents

| Aspect | Logical Agents (R&N Ch.7–8) | Expert Systems |
|-----------------------|---|--|
| Representation | Sentences in propositional or FOL; general-purpose logic. | Production rules (IF–THEN), frames, or Horn clauses; domain-specific. |
| Inference | Sound & complete inference possible (but expensive). | Heuristic or restricted inference (forward/backward chaining); often incomplete. |
| Scope | General-purpose reasoning about any domain. | Narrow, specialized domain (medicine, chemistry, etc.). |
| Knowledge Acquisition | Abstractly: define ontology, axioms, facts. | Pragmatically: extract rules from experts (knowledge engineers interviewing doctors, scientists). |
| Explanations | Logic entails proofs; proofs can be shown as chains. | Expert systems emphasized <i>explainability</i> : “The system concluded X because rules R1, R3 fired.” |
| Uncertainty | Classical logic = crisp true/false. | Many expert systems added certainty factors (MYCIN used probabilities / confidence scores). |

Expert systems

 [BLAST](#) [Align](#) [Peptide search](#) [ID mapping](#) [SPARQL](#) [ARBA](#) Advanced

ARBA - ARBA00004015

[Download](#) [View proteins](#)


| IF | THEN |
|---|--|
| <div>InterPro signature IPR001279</div> <div>InterPro signature IPR017782</div> <div>PANTHER signature PTHR11935:SF80</div> | <div>function Thiolesterase that catalyzes the hydrolysis of S-D-lactoyl-glutathione to form glutathione and D-lactic acid</div> |

OR

| IF | THEN |
|--|------|
| <div>FunFam signature 3.60.15.10:FF:000019</div> <div>taxon Mammalia</div> | |

Annotated UniProtKB entries

[Browse all 1,125 entries](#)

 **B4DT01 · B4DT01_HUMAN**
Hydroxyacylglutathione hydrolase, mitochondrial · [Homo sapiens \(Human\)](#) · EC:3.1.2.6 · 305 amino acids · Evidence at transcript level · **Annotation score:** 3/5
[#Hydrolase](#)
1 domain · 1 publication

Expert systems

1. Sequence Analysis & Annotation

- **GENAID** (1980s) – an expert system for automatic DNA sequence interpretation.
- **EMBL-EBI RuleBase / HAMAP (UniProtKB)** – rule-based annotation pipelines; although modernized, they still embody expert system principles (IF–THEN rules for annotating protein function).
- **Prolog-based gene analysis systems** – early use of Horn clause inference to detect motifs and gene structures.

2. Structural Biology

- **DENDRAL** (chemistry, precursor of expert systems in molecular analysis) – though not bioinformatics in the modern sense, it inspired later systems for protein mass spectrometry interpretation.
- **ESPRIT** – expert system for protein structure prediction, combining rules about secondary structure with experimental constraints.
- **PROSPECTOR** – expert system for predicting protein tertiary structures, used rules derived from known motifs.

Expert systems

3. Clinical Bioinformatics / Medical Expert Systems

- **MYCIN** (Stanford, 1970s) – medical diagnosis of bacterial infections; not bioinformatics per se, but a **prototype** that influenced many biomedical expert systems.
- **PUFF** – rule-based system for interpreting pulmonary function tests.
- **ONCOCIN** – cancer chemotherapy treatment planning.
- **DXplain** – a medical decision support system still used in some clinical environments.
- These clinical expert systems influenced bioinformatics approaches for linking molecular markers to diagnoses.

4. Pathway & Metabolism Modeling

- **PathFinder** – rule-based system for metabolic pathway reconstruction.
- **MetaCyc/Pathway Tools** (SRI International) – combines expert rules with databases to predict metabolic pathways in newly sequenced organisms.
- **EcoCyc** – originally contained expert-system components for metabolic annotation in *E. coli*.

AI Planning

While inference is about **what is true**, **planning** is about **what to do** (sequence of actions to achieve a goal). Both rely on logical representations.

- **STRIPS (Stanford Research Institute Problem Solver, 1970s):**
 - A classical planning language and algorithm.
 - States are sets of facts (propositions).
 - Actions are defined by **preconditions** (what must be true to apply the action) and **effects** (how the world changes).
 - Planning = search in the space of states, applying actions until a goal is satisfied.
 - Example: In the Wumpus world → action "Grab" has precondition `At(Agent, Gold)` and effect `Has(Agent, Gold)`.
 - In bioinformatics: could be used to model workflows (e.g., precondition = "sequence assembled", action = "annotate with GO term").
- **GraphPlan (1995):**
 - Builds a **planning graph** that alternates levels of actions and propositions.
 - Compactly represents which actions can occur and which facts can hold at each step.
 - Uses graph structure to extract a valid plan (if one exists) or prove impossibility.
 - More efficient than naïve STRIPS search.
 - In bioinformatics: analogous to workflow systems (like Galaxy, Scipion, or Taverna), where tasks must follow dependencies.

AI Planning

Example: Variant Annotation Pipeline as a Planning Problem

Context

Goal: annotate genetic variants (e.g., from a VCF file) with functional consequences.

This requires sequencing data, a reference genome, and annotation databases.

1. States

Represented as sets of propositions.

- `Has(RawReads)`
- `Has(AlignedReads)`
- `Has(Variants)`
- `Has(AnnotatedVariants)`

2. Actions (with Preconditions → Effects)

- **AlignReads**
 - Preconditions: `Has(RawReads) ∧ Has(ReferenceGenome)`
 - Effects: add `Has(AlignedReads)`
- **CallVariants**
 - Preconditions: `Has(AlignedReads)`
 - Effects: add `Has(Variants)`
- **AnnotateVariants**
 - Preconditions: `Has(Variants) ∧ Has(AnnotationDB)`
 - Effects: add `Has(AnnotatedVariants)`

AI Planning

3. Initial State

SCSS

```
{ Has(RawReads), Has(ReferenceGenome), Has(AnnotationDB) }
```

4. Goal

SCSS

```
{ Has(AnnotatedVariants) }
```

5. Plan Found (sequence of actions)

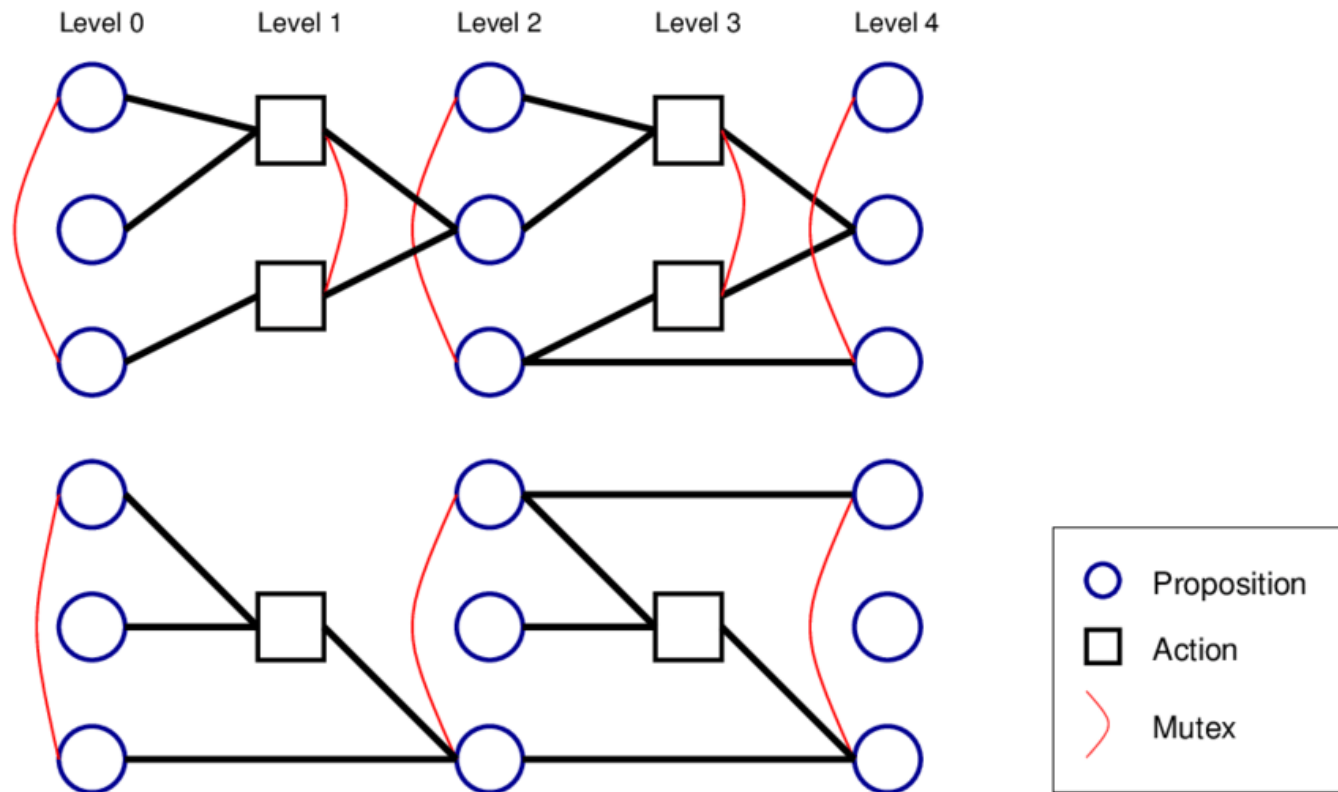
1. **AlignReads** → produces aligned reads.
2. **CallVariants** → produces list of variants.
3. **AnnotateVariants** → produces annotated variants (goal achieved).

Why This Is Planning

- Each step has **preconditions and effects**, exactly like STRIPS.
- The system must **search** through possible action sequences to reach the goal.
- If something is missing (e.g., no annotation database), the plan cannot succeed → planning detects infeasibility.

AI Planning

GraphPlan example



Adversarial games

2. How Adversarial Search differs from Logical Agents / Planning

| Dimension | Logical Agents (Ch.7–8) | Planning (STRIPS/GraphPlan) | Adversarial Games (Ch.5–6) |
|------------------|--|--|--|
| Focus | Reason about truth (facts, rules, queries). | Plan sequences of <i>your own</i> actions to reach a goal. | Plan under competition : actions alternate with an opponent's moves. |
| Uncertainty | Can be deterministic (FOL) or probabilistic. | Typically deterministic state transitions (though later extended to stochastic). | Opponent's moves create uncertainty; modeled via minimax search (adversarial uncertainty). |
| Search structure | Proof trees, entailment, resolution. | Planning graphs, forward/backward search in state space. | Game trees with MAX and MIN nodes; search for strategies, not just solutions. |
| Domain use | Knowledge bases (ontology reasoning, diagnosis). | Automated lab workflows, pathway assembly, experiment planning. | Competitive or adversarial scenarios (games, negotiations, host–pathogen arms races). |

Adversarial games

Adversarial Games: Old-Style AI vs. Modern Reinforcement Learning

| Aspect | Traditional AI (1970s–1990s) | Modern RL (2000s–today) |
|------------------|--|--|
| Representation | Explicit game tree : states and legal moves encoded symbolically. | Environment modeled as a Markov Decision Process (MDP) ; often implicit via simulation, not an explicit tree. |
| Search Method | Minimax search with depth-limited lookahead; alpha-beta pruning for efficiency. | Trial-and-error learning : agents learn by interacting with the environment, optimizing long-term reward. |
| Evaluation | Static evaluation functions (heuristics hand-crafted by experts). | Value functions or policies learned from data (Q-learning, policy gradient, actor-critic). Often approximated with neural networks (Deep RL). |
| Knowledge Source | Human knowledge encoded in rules/heuristics (e.g., “control the center in chess”). | Self-play and large-scale experience; agent discovers strategies automatically (AlphaGo, AlphaZero). |
| Strengths | Transparent reasoning; explainable (can show which branch of the tree led to a move). | Handles very large or continuous state spaces; adapts by learning. Scales to Go, StarCraft, protein folding, drug design. |
| Limitations | Explodes combinatorially; limited depth and coverage; requires strong human-designed heuristics. | Requires enormous data/simulation; learned policies can be opaque (black box); weaker guarantees on optimality. |

Adversarial games

Examples

- **Old AI:**
 - *IBM Deep Blue (1997)* beat Kasparov at chess using alpha–beta search with expert-crafted heuristics and huge computational power.
- **Modern RL:**
 - *AlphaGo / AlphaZero (2016–2018)* beat world champions in Go and chess by combining reinforcement learning, self-play, and deep neural networks for policy/value approximation.
 - Similar principles now applied in **drug design** (reinforcement learning for molecule generation) and **protein folding** (AlphaFold2's learning framework has RL components).

Adversarial games

Alpha–Beta Pruning

1. Context

- In **adversarial games** (like chess), the traditional algorithm is **minimax search**:
 - MAX nodes = our moves (we want to maximize utility).
 - MIN nodes = opponent's moves (they want to minimize our utility).
 - Search proceeds down the game tree to a fixed depth, then evaluates leaf states with a heuristic.
- Problem: The game tree is huge. Even for chess, there are more possible states than atoms in the universe.
- **Alpha–beta pruning** is an optimization that **cuts off branches** of the tree that cannot influence the final decision.

2. Core Idea

- While doing minimax search:
 - α (alpha): the **best value that MAX can guarantee so far** (lower bound for MAX).
 - β (beta): the **best value that MIN can guarantee so far** (upper bound for MAX).
- If at some point we find that the current node's value is **worse than an already known alternative**, we can **stop exploring that branch** — it will never affect the outcome.

Adversarial games

3. Simple Example

Suppose MAX must choose between moves **A** and **B**:

- While exploring **A**, we find that its best outcome is at least **+5** ($\alpha = 5$).
- Now we start exploring **B**, where MIN will try to minimize. If at some point we find that **B** cannot possibly do better than 5 ($\beta \leq 5$), we **prune** the rest of B's branches — no need to explore further, since MAX will never prefer B over A.

4. Effect

- Alpha-beta does **not change the result** of minimax — it just avoids wasting time on irrelevant branches.
- With perfect move ordering, alpha-beta can reduce the effective branching factor from **b** to \sqrt{b} , roughly doubling the search depth for the same computational cost.
- This was crucial for systems like **Deep Blue**, which could search 10–12 moves deep in chess instead of just 6–7.

Adversarial games

5. Analogy in Bioinformatics

While alpha–beta pruning comes from games, the idea of **pruning the search space by bounding outcomes** is very relevant:

- In **sequence alignment**, heuristic pruning (BLAST's word filter) avoids exploring alignments that can't beat the best score.
- In **protein docking**, pruning rules discard conformations that already exceed an energy threshold.
- In **phylogenetic tree search**, heuristics prune branches of evolutionary trees that can't lead to better likelihood scores.

So you can present alpha–beta pruning as an **early example of search-space reduction**, a principle used widely in bioinformatics algorithms.

Adversarial games

Monte Carlo Search

1. Motivation

- In many domains (Go, protein folding, drug design), the **search space is astronomically large**.
- Even alpha-beta pruning cannot explore deep enough.
- Solution: Instead of exploring the **entire tree**, use **random sampling (Monte Carlo)** to approximate the value of states.

2. Basic Monte Carlo Search

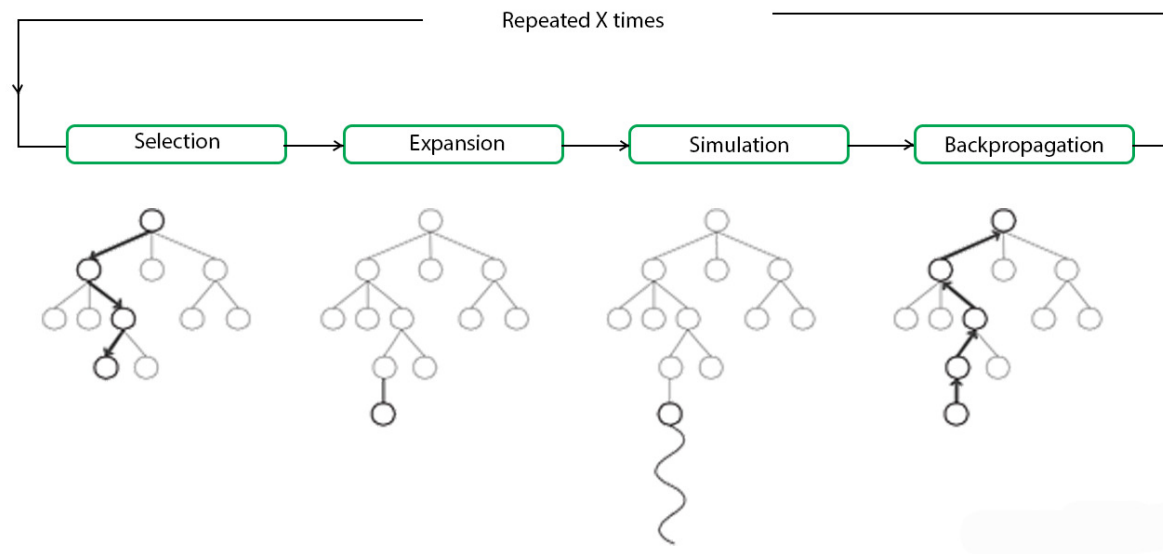
- To evaluate a move:
 1. From the current state, **simulate many random games** (rollouts) until the end.
 2. Record the outcome (win/loss, or a score).
 3. Estimate the value of the move as the **average outcome** of these simulations.
- The more simulations, the better the estimate.

Adversarial games

3. Monte Carlo Tree Search (MCTS)

A powerful extension that builds a **partial search tree** guided by random exploration. It has four phases:

1. **Selection:** Navigate the tree from the root, choosing child nodes using a heuristic like **UCT (Upper Confidence Bound for Trees)** that balances **exploration vs exploitation**.
 2. **Expansion:** If a leaf node is not fully explored, add a new child (new possible move).
 3. **Simulation (Rollout):** Play randomly (or with a simple policy) from that child to the end.
 4. **Backpropagation:** Update the values (win/loss averages) along the path back to the root.
- Over many iterations, the search tree **grows where it matters most**.
 - Used famously by **AlphaGo** and **AlphaZero** to master Go and Chess.



Adversarial games

4. Comparison with Minimax/Alpha-Beta

| Aspect | Minimax + Alpha-Beta | Monte Carlo Search / MCTS |
|------------------|---|---|
| Tree exploration | Systematic, deterministic | Random sampling of promising branches |
| Evaluation | Requires heuristic function at cut-off | Uses simulated rollouts (no hand-coded heuristics needed) |
| Completeness | Exact (with enough depth) | Approximate, improves with more simulations |
| Strength | Strong in shallow, tactical domains (e.g. chess with strong heuristics) | Strong in deep, complex domains (e.g. Go, large biological search spaces) |

Adversarial games

5. Bioinformatics Analogies

Monte Carlo methods map very naturally to bioinformatics:

- **Protein folding / docking:**
 - Search space of conformations is enormous.
 - Random sampling (Monte Carlo simulations) + scoring functions approximate good solutions.
- **Molecular dynamics simulations:**
 - Monte Carlo sampling used to explore possible trajectories.
- **Drug discovery:**
 - Reinforcement learning + MCTS used to generate molecules, simulating possible modifications to optimize activity.
- **Pathway modeling:**
 - Explore alternative regulatory interactions by sampling possible network states.

Practice

https://github.com/cossorzano/COSS_DataAnalysis_notebooks/blob/main/ArtificialIntelligence/Host_Pathogen.ipynb

✓ Host–Pathogen Arms Race as an Adversarial Game

In this notebook we will explore a **toy model of the evolutionary arms race** between a **host (or clinician)** and a **pathogen**.
The aim is to illustrate how concepts from **adversarial AI and decision-making** can be applied to **bioinformatics and systems biology**.

Problem Setup

- **Pathogen genotype:**

Represented as an 8-bit binary string (`00000000`).

- Each bit corresponds to the presence (1) or absence (0) of a resistance mutation.
- Mutations can make the pathogen resistant to certain drugs, but each resistance bit carries a **fitness cost**.

- **Host (Leader):**

Chooses a **treatment policy** at the beginning and sticks with it for the whole simulation.

Available policies:

- `DrugA`
- `DrugB`
- `Combo` (A + B together)
- `Holiday` (no treatment)



CEU

*Universidad
San Pablo*

Lesson 4. Classical machine learning

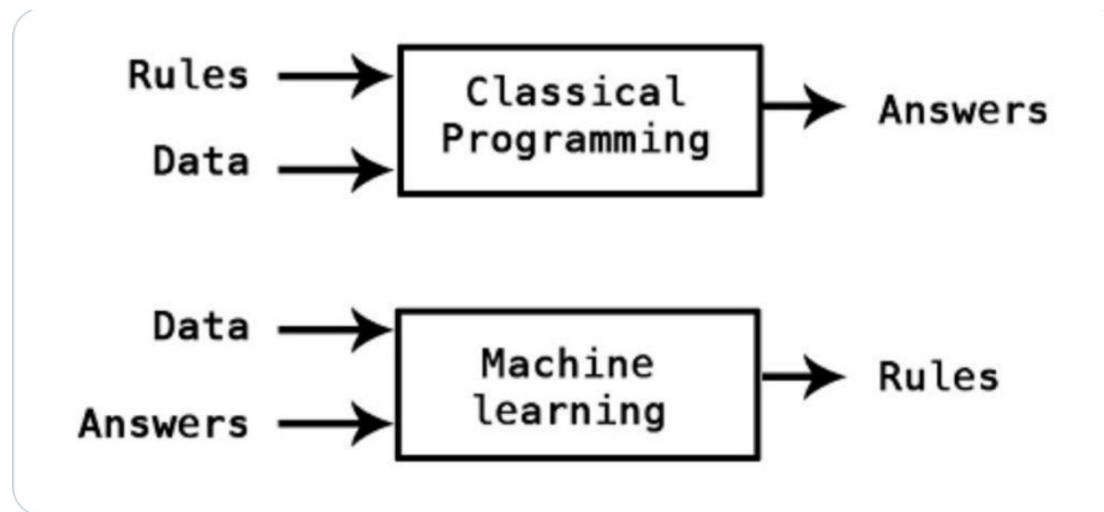
Medicin School

Contents

- Overview
- Overfitting and underfitting
- Evaluation metrics
- Cross validation
- Bias and variance tradeoff
- No free lunch theorem

Overview

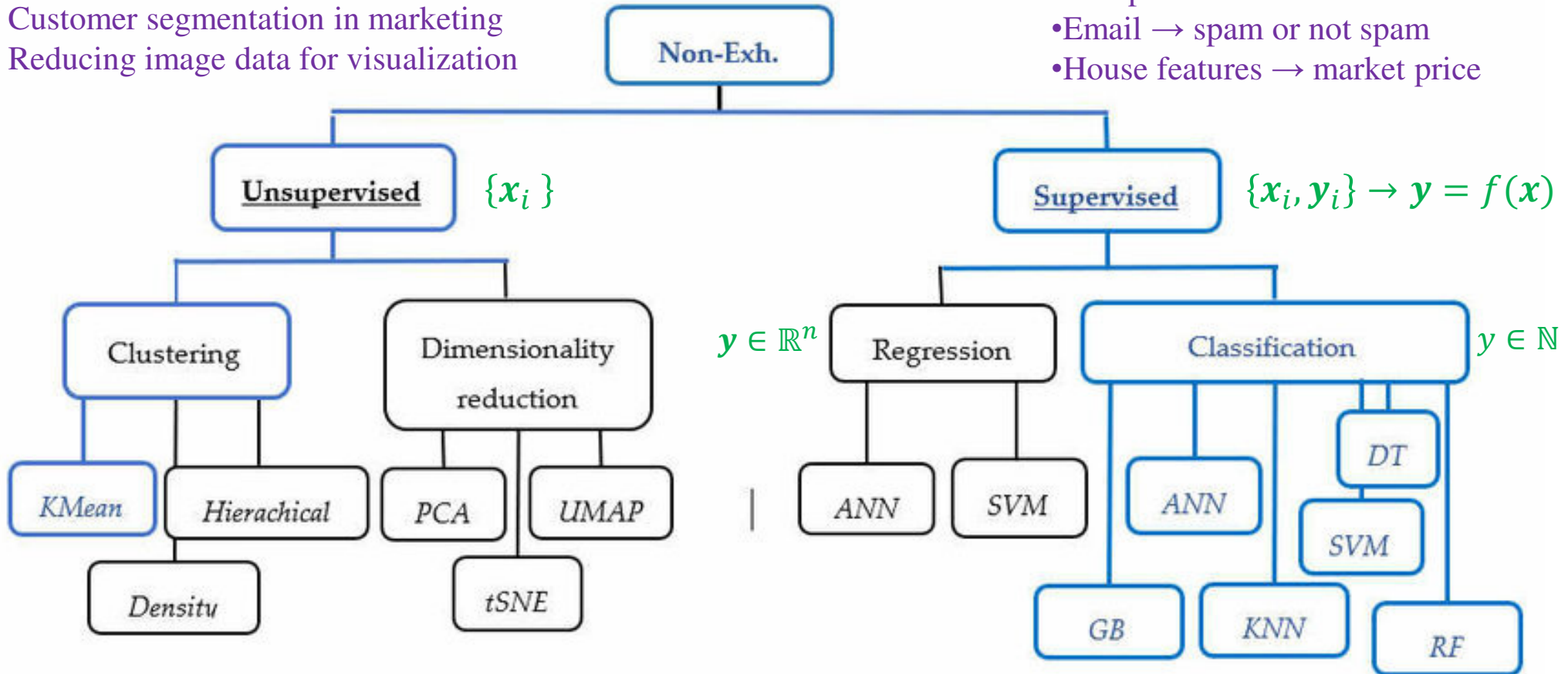
- Machine Learning = algorithms that **learn patterns from data**
- Not explicitly programmed with rules
- Goal: make **predictions or decisions** on new data
- Two main paradigms: **Supervised** and **Unsupervised** learning



Overview

Examples:

Customer segmentation in marketing
Reducing image data for visualization

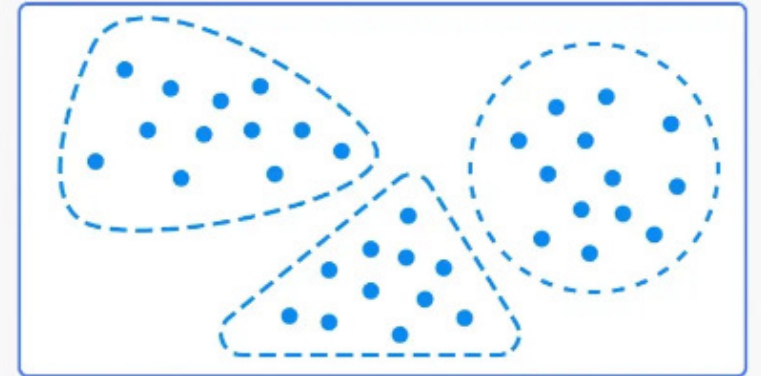
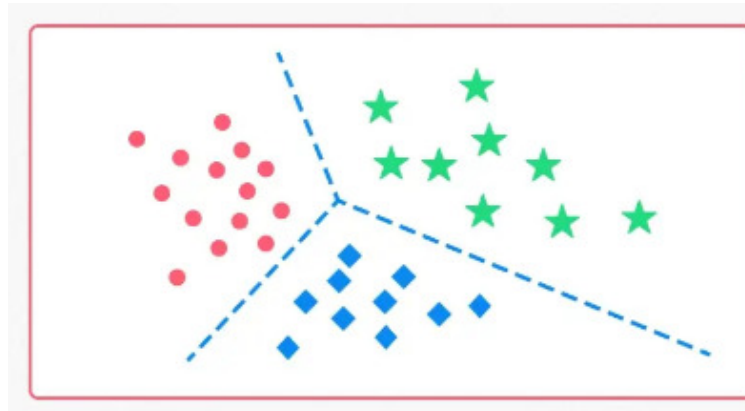
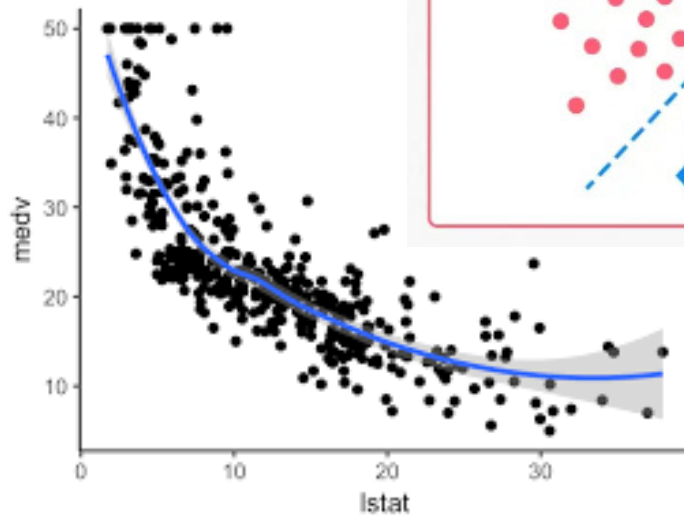


Overview

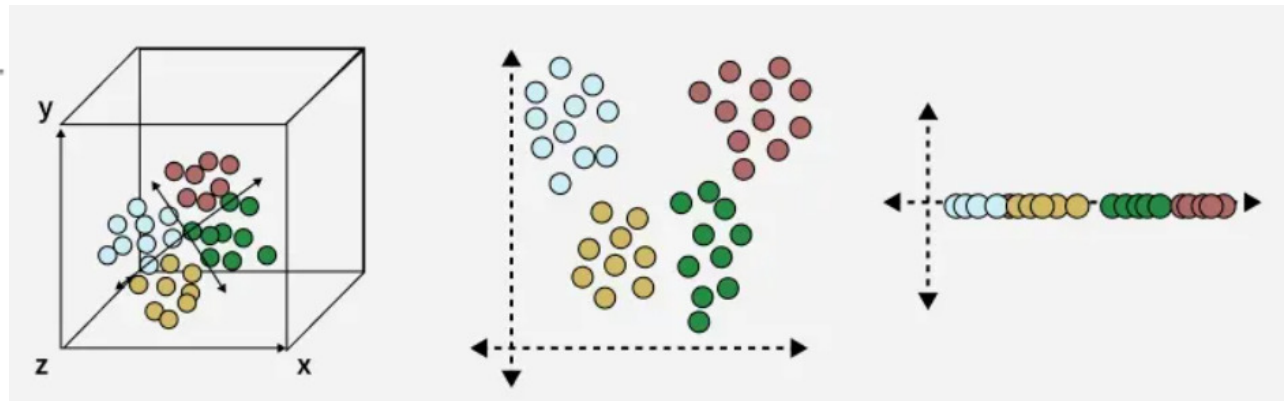
Classification

Clustering

Regression



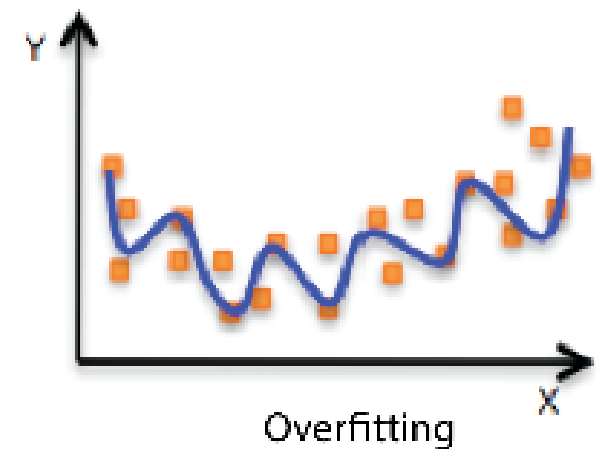
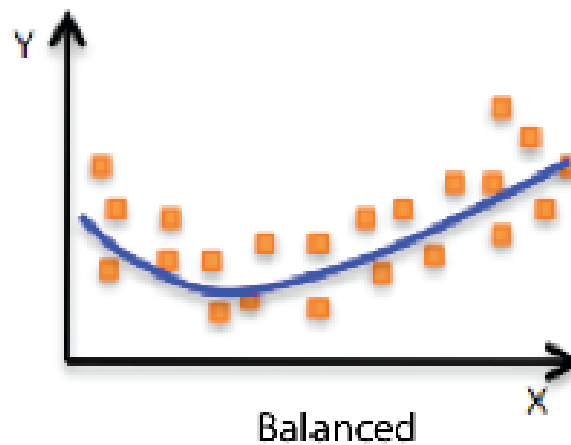
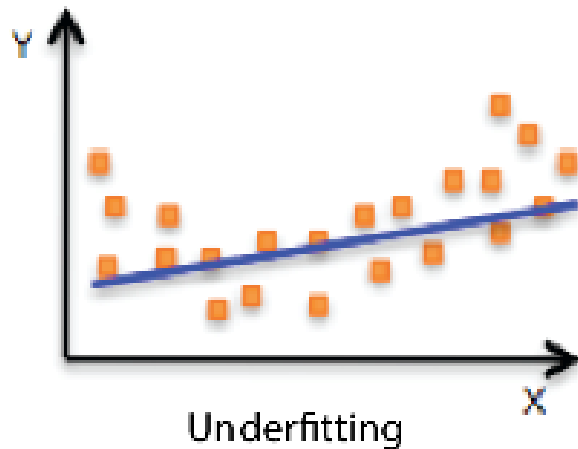
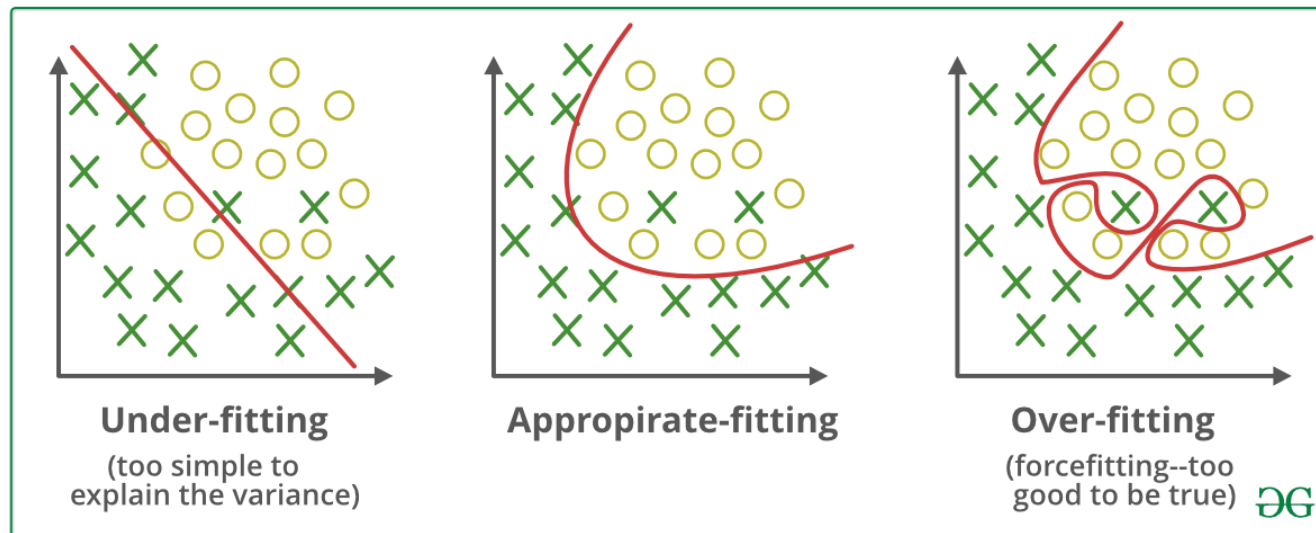
Dimensionality reduction



Overfitting and underfitting

- **Underfitting**
 - Model is **too simple**
 - Fails to capture patterns in training data
 - Both training and test error are high
- **Overfitting**
 - Model is **too complex**
 - Learns noise and idiosyncrasies in training data
 - Training error very low, but test error high
- **Good fit**
 - Balance between complexity and generalization
 - Low training error + low test error

Overfitting and underfitting



Evaluation metrics

The performance of a machine learning model must be measured quantitatively. The choice of metric depends on the task.

For **regression problems**, common metrics include:

- **Mean Squared Error (MSE)**: the average of squared differences between predicted and true values. It penalizes large errors heavily.
- **Root Mean Squared Error (RMSE)**: the square root of MSE, expressed in the same units as the data, which makes interpretation easier.
- **Mean Absolute Error (MAE)**: the average of absolute differences between predictions and true values. It is less sensitive to outliers.
- **R^2 (Coefficient of Determination)**: measures the proportion of variance in the data explained by the model, with values closer to 1 indicating a better fit.

Evaluation metrics

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

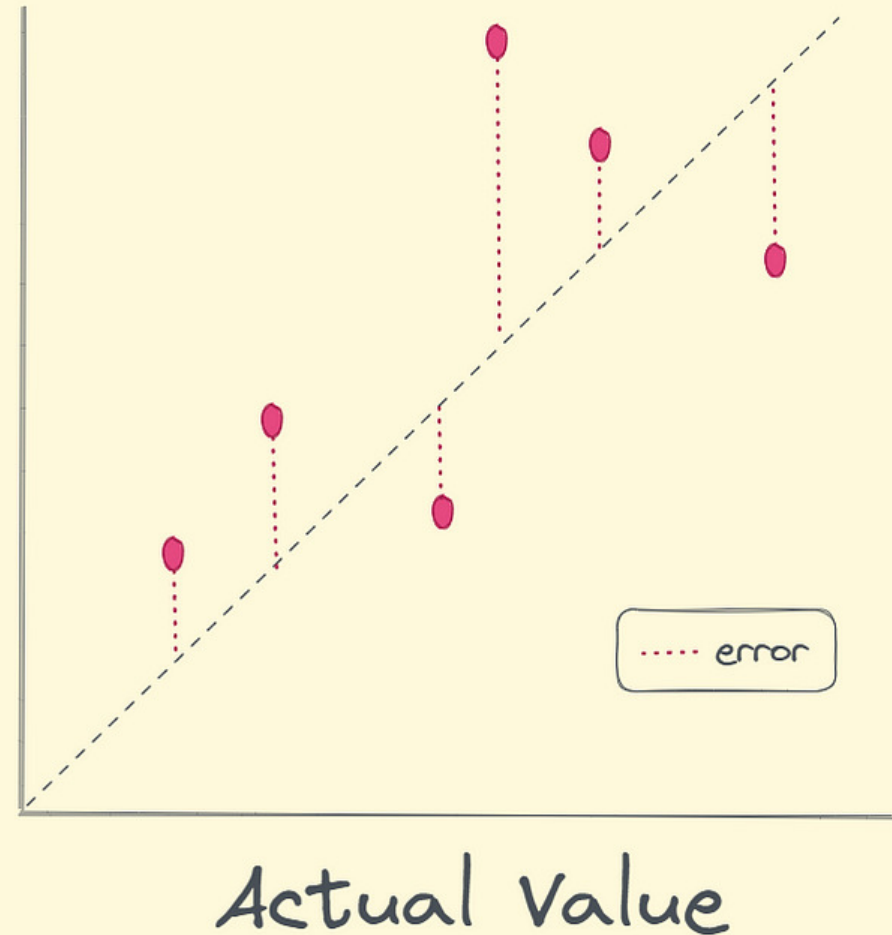
$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Predicted Value



Evaluation metrics

Pearson's correlation coefficient

Perfect
Positive

Strong
Positive

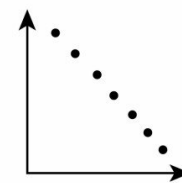
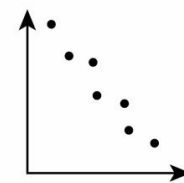
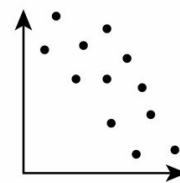
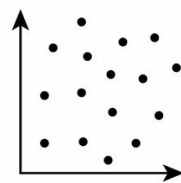
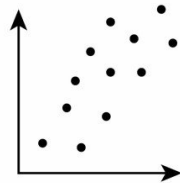
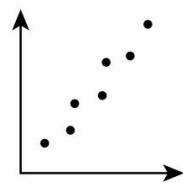
Weak
Positive

No
Correlation

Weak
Negative

strong
Negative

Perfect
Negative



$$R^2 = \rho^2$$

1

0.9

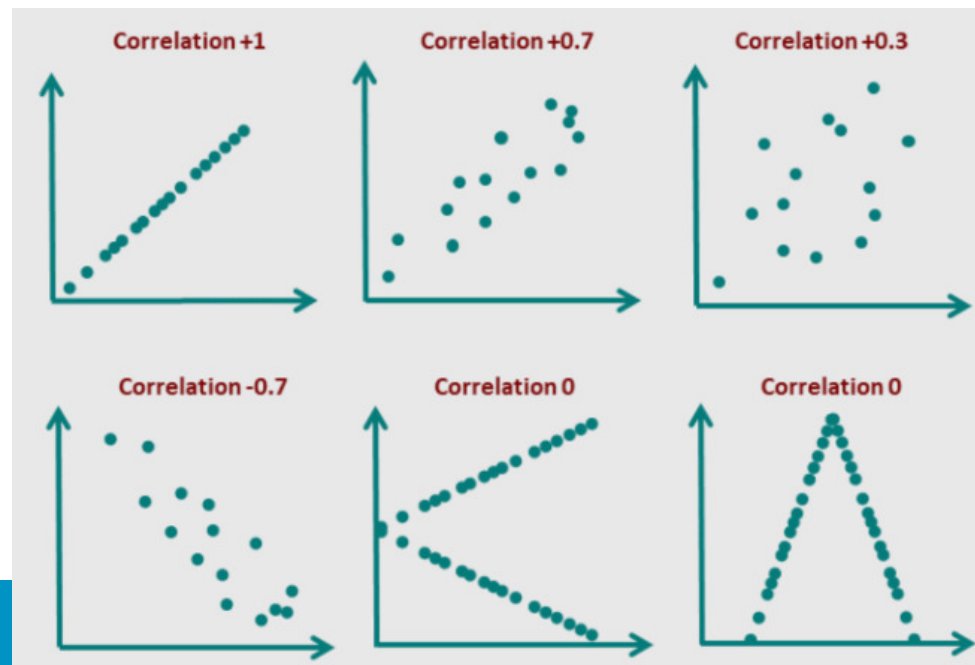
0.5

0

-0.5

-0.9

-1



Evaluation metrics

For **classification problems**, metrics are based on comparing predicted labels to true labels.

- **Accuracy:** the fraction of correctly classified examples. It is easy to understand but may be misleading in imbalanced datasets.
- **Precision:** the fraction of predicted positives that are true positives, measuring reliability of positive predictions.
- **Recall (Sensitivity):** the fraction of actual positives correctly identified, measuring how many relevant cases are found.
- **F1 Score:** the harmonic mean of precision and recall, useful when a balance between the two is needed.
- **ROC curve and AUC:** graphical and numerical summaries of the trade-off between true positives and false positives.

Evaluation metrics







| | | POSITIVE | NEGATIVE |
|---------------|----------|----------|----------|
| ACTUAL VALUES | POSITIVE | TP | FN |
| | NEGATIVE | FP | TN |

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

| | | Predicted | |
|--------|------------|---|---|
| | | Animal | Not animal |
| Actual | Animal |    | |
| | Not animal |  |   |

| | |
|-----------------|---|
| True Positives | 2 |
| True Negatives | 3 |
| False Positives | 0 |
| False Negatives | 1 |

| | | |
|-----------|------|---|
| Accuracy | 83% | $\frac{3+2}{3+2+0+1}$ |
| Precision | 75% | $\frac{3}{3+1}$ |
| Recall | 100% | $\frac{3}{3+0}$ |
| F1 score | 86% | $2 \cdot \frac{0.75 \cdot 1}{0.75 + 1}$ |

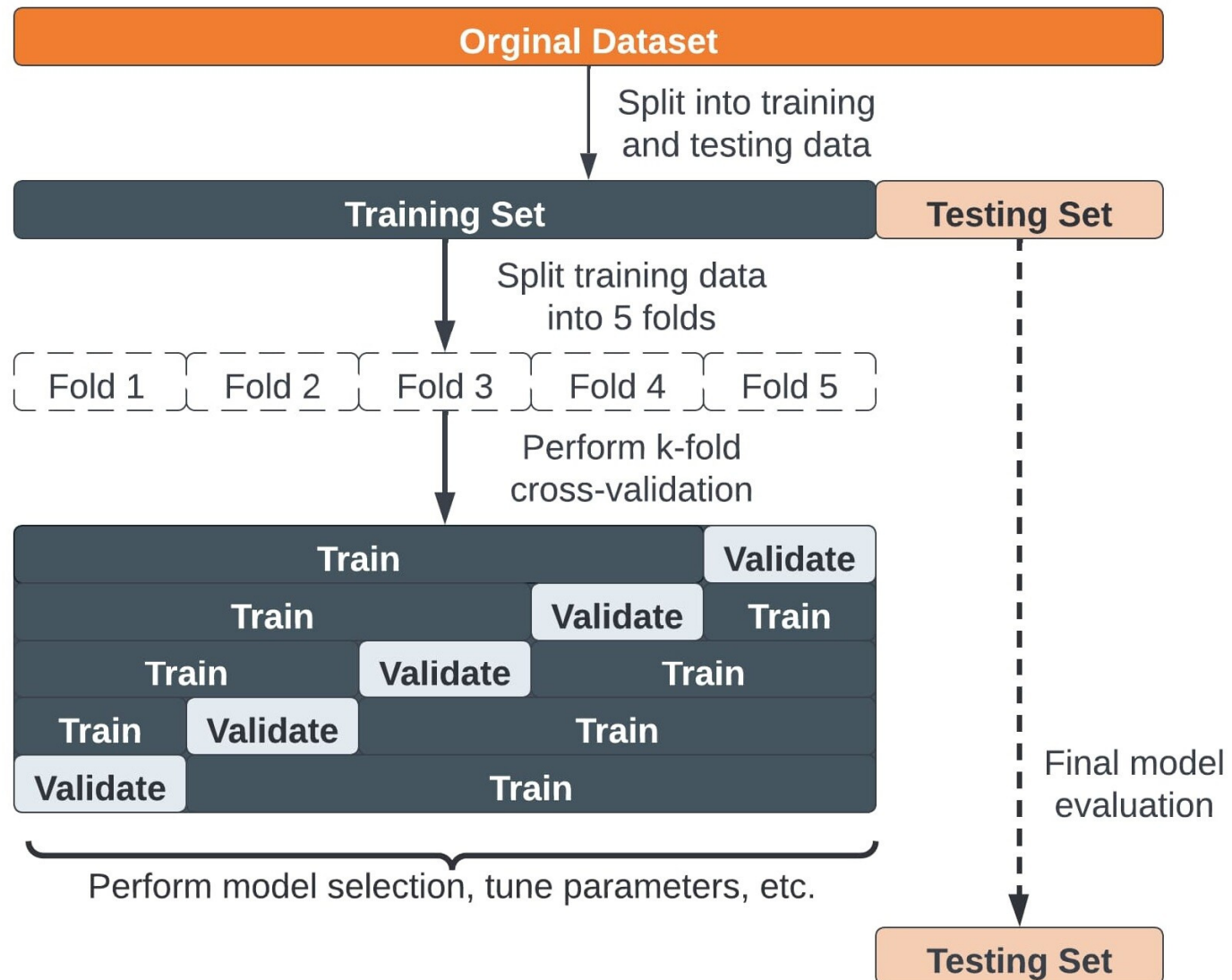
Cross validation

To evaluate how well a model generalizes, we cannot rely only on training error. We need to test the model on data it has not seen.

- **Hold-out validation** is the simplest approach: the dataset is split into a training set and a test set. The model is trained on one part and evaluated on the other. The drawback is that the result may depend too much on how the data was split.
- **k-fold cross-validation** addresses this. The dataset is divided into k equal parts. The model is trained k times, each time leaving out one part for testing and using the remaining $k-1$ parts for training. The final score is the average of all k tests. This reduces the dependence on a particular split.
- **Leave-one-out cross-validation (LOOCV)** is the extreme case where k equals the number of data points. Each observation is used once as the test set. It uses all data for training but can be computationally expensive.

Cross-validation provides a more reliable estimate of model performance and helps in selecting models and hyperparameters.

Cross validation



Bias and variance

A model's prediction error has two main sources: **bias** and **variance**.

Bias is the error due to simplifying assumptions.

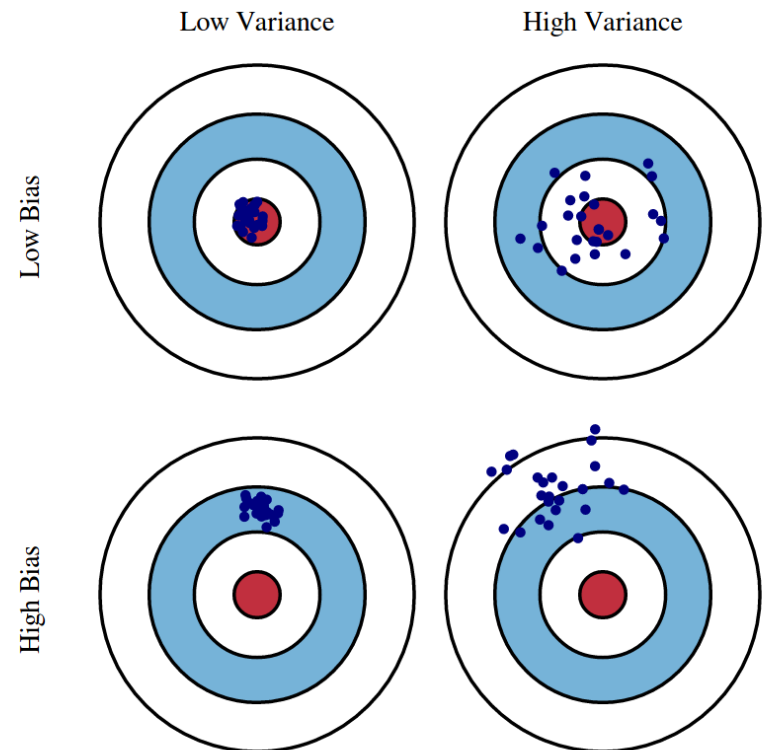
- A model with high bias is too rigid.
- It fails to capture the true patterns in the data.
- Leads to **systematic errors** (underfitting).

Variance is the error due to sensitivity to training data.

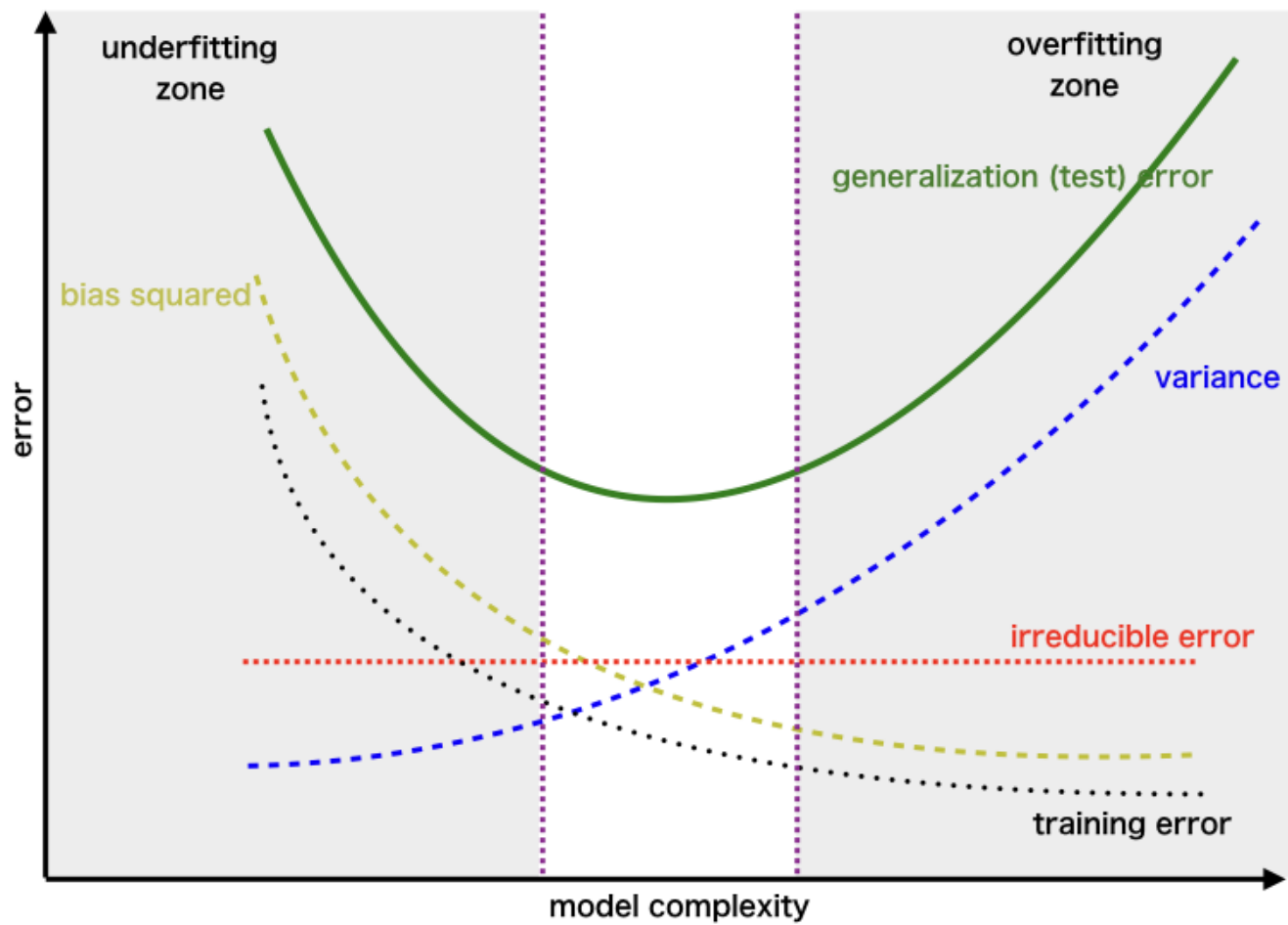
- A model with high variance adapts too much to noise.
- Predictions change strongly with small changes in the data.
- Leads to **poor generalization** (overfitting).

The **trade-off**:

- Increasing complexity reduces bias but increases variance.
- Decreasing complexity reduces variance but increases bias.
- The goal is to find a model with low total error, balancing both.



Bias and variance



Bias and variance

Let the true relationship be

$$Y = f(X) + \varepsilon,$$

where ε is random noise with mean zero and variance σ^2 .

Let $\hat{f}(X)$ be the prediction of a learning algorithm trained on a dataset.

The **expected prediction error** at a point x can be decomposed as

$$\mathbb{E}[(Y - \hat{f}(x))^2] = \underbrace{(\mathbb{E}[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible error}}.$$

- **Bias:** the squared difference between the average prediction and the true function, $(\mathbb{E}[\hat{f}(x)] - f(x))^2$.
- **Variance:** the variability of predictions across different training sets, $\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$.
- **Irreducible error:** the variance of the noise, σ^2 , which cannot be eliminated.

No free lunch theorem

In machine learning there is **no universally best algorithm**.

Performance always depends on the problem and the assumptions we make about the data.

Intuition:

- Suppose two algorithms, A and B, are compared across all possible datasets.
- On some datasets, A will perform better; on others, B will perform better.
- When averaged over all possible problems, **they perform the same**.
- This means: an algorithm's advantage comes only from how well its **inductive bias** matches the data.

Formal idea:

Let \mathcal{F} be the set of all possible functions mapping inputs to outputs.

If we assume a uniform prior over \mathcal{F} , then for any two learning algorithms A and B :

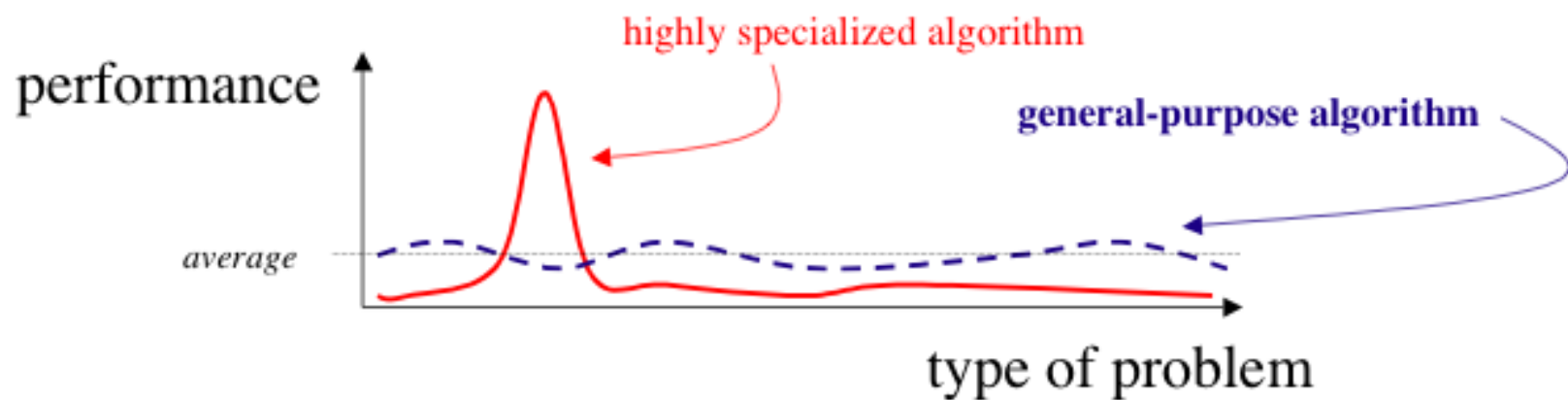
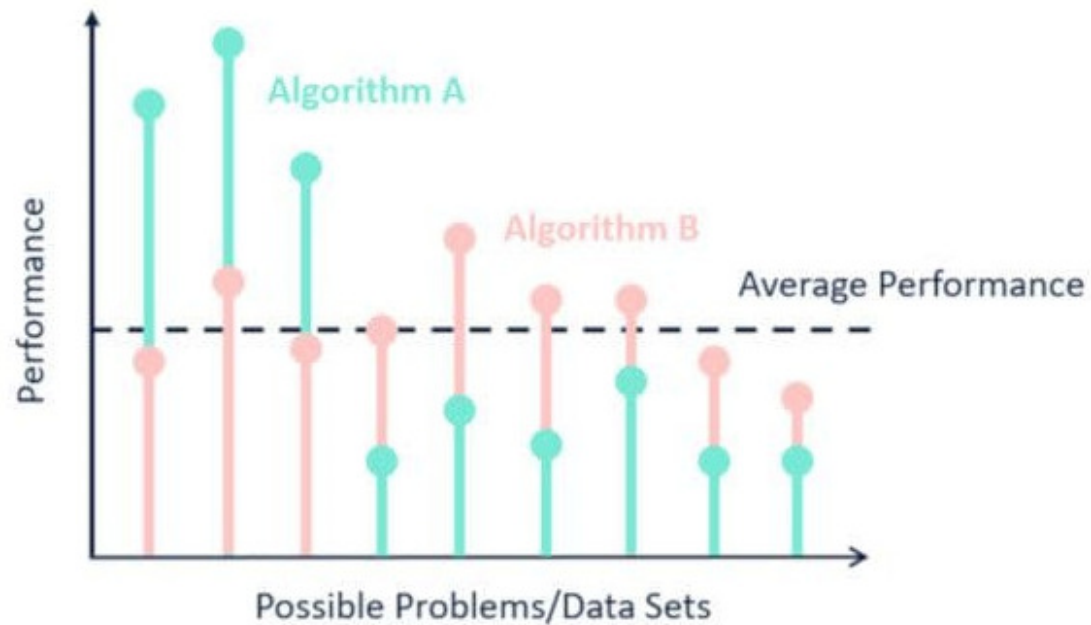
$$\sum_{f \in \mathcal{F}} \text{Error}_f(A) = \sum_{f \in \mathcal{F}} \text{Error}_f(B).$$

Thus, **no algorithm has lower error across all possible functions**.

Implication:

- We cannot pick "the best" algorithm in general.
- Success comes from **choosing the right model and assumptions** for the domain and data at hand.

No free lunch theorem



Contents

- Overview
- Overfitting and underfitting
- Evaluation metrics
- Cross validation
- Bias and variance tradeoff
- No free lunch theorem



CEU

*Universidad
San Pablo*

Lesson 5. Regression

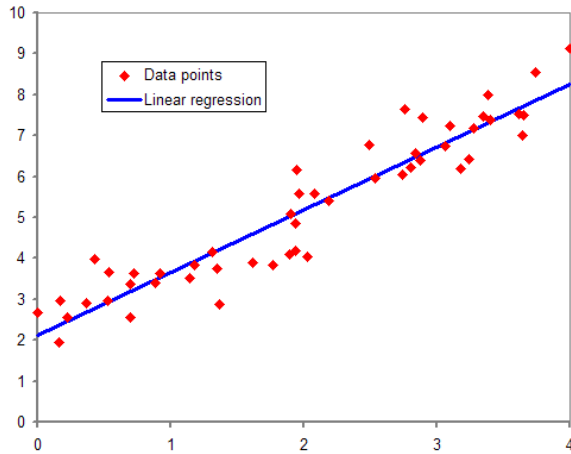
Medicin School

Contents

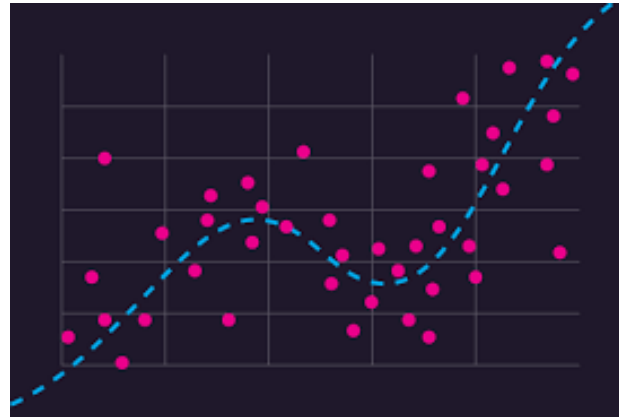
- Overview
- Linear regression
 - Simple, Multiple, Residual analysis, RBF
- Regularization
 - Ridge, Lasso
- Generalized Linear Models
- Non-linear regression
- Non-parametric regression
- Challenges in Bioinformatics

Overview

Linear

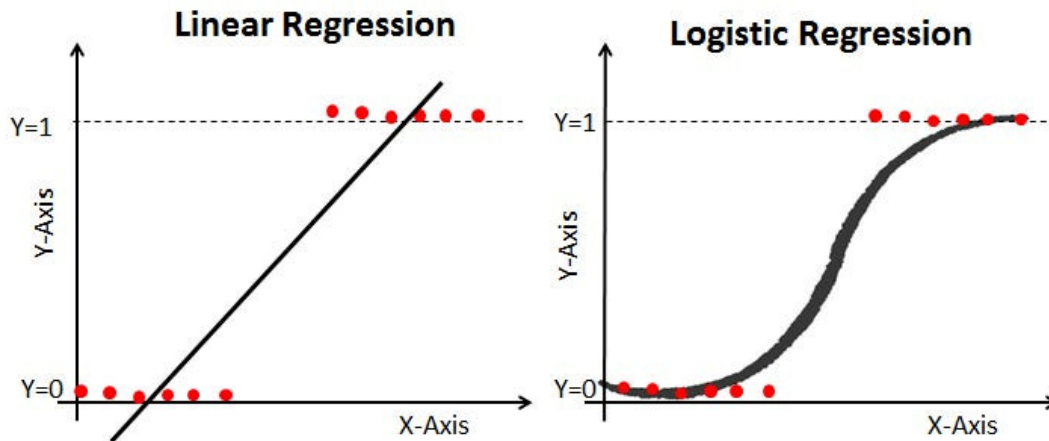


Non-linear

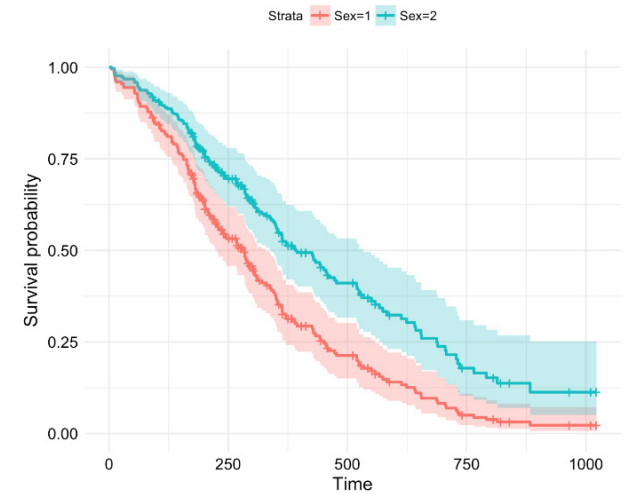


Logistic

Logistic Regression

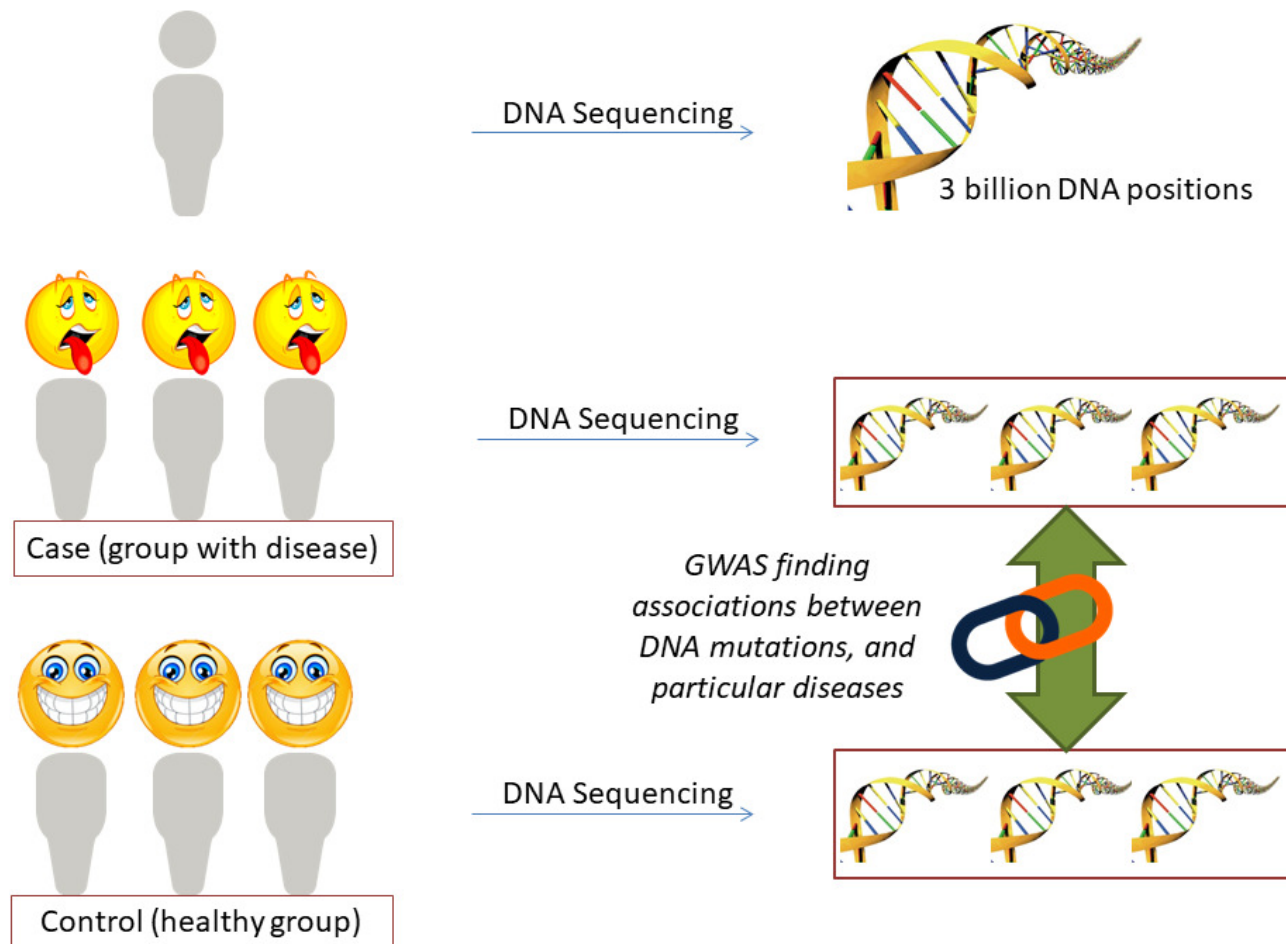


Hazard



Overview

GWAS (Genome Wide Association Study)



Overview

1. The data structure

We typically have:

- n individuals (samples)
- p genetic variants (usually SNPs)

Data can be arranged in:

- **Genotype matrix** $X \in \mathbb{R}^{n \times p}$, where $X_{ij} \in \{0, 1, 2\}$ encodes the number of minor alleles of SNP j in individual i (after QC and imputation, values may be fractional). QC=Quality Control
- **Phenotype vector** $y \in \mathbb{R}^n$, which can be:
 - continuous (e.g. height, blood pressure),
 - binary (e.g. disease status),
 - categorical or time-to-event in some extended models.
- **Covariates matrix** $C \in \mathbb{R}^{n \times q}$ for age, sex, ancestry principal components, etc.

At a biallelic SNP, each individual carries two alleles:

- 0 = homozygous major allele (e.g., AA)
- 1 = heterozygous (e.g., Aa)
- 2 = homozygous minor allele (e.g., aa)

So in the simplest case:

$$X_{ij} \in \{0, 1, 2\}$$

Overview

2. Single-variant association model

GWAS is usually performed SNP by SNP, testing for association between phenotype y and each genotype X_j while controlling for C .

(a) Linear regression (quantitative traits):

$$y = \alpha + X_j\beta_j + C\gamma + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I)$$

- Null hypothesis: $H_0 : \beta_j = 0$ (no effect of SNP j)
- Test statistic: $t = \hat{\beta}_j / \text{SE}(\hat{\beta}_j)$, compared to standard normal.

(b) Logistic regression (case-control traits):

$$\Pr(y_i = 1 \mid X_{ij}, C_i) = \frac{1}{1 + \exp(-(\alpha + \beta_j X_{ij} + C_i \gamma))}$$

- Null hypothesis: $H_0 : \beta_j = 0$
- Test statistic: likelihood ratio test, Wald test, or score test.

Simple linear regression

Problem setup

- Input variable: x (e.g., gene length).
- Output variable: y (e.g., expression level).
- Goal: predict y as a linear function of x .

Model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \dots, n$$

- β_0 : intercept
- β_1 : slope (effect of x on y)
- ε_i : error term

Ordinary Least Squares (OLS)

Idea

- Find β_0, β_1 that minimize squared errors:

$$\min_{\beta_0, \beta_1} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Solutions

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Multiple linear regression

Setup

- n : number of samples (e.g., patients, genes).
- p : number of input variables (features).
- q : number of outputs (responses).

Model

$$Y = XB + E$$

- Y : $n \times q$ response matrix

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1q} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nq} \end{bmatrix}$$

- X : $n \times (p + 1)$ design matrix (includes intercept column of 1's)

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}$$

- B : $(p + 1) \times q$ matrix of coefficients
- E : $n \times q$ error matrix

Ordinary Least Squares (OLS)

Goal

$$\hat{B} = \arg \min_B \|Y - XB\|_F^2$$

(where $\|\cdot\|_F$ is the Frobenius norm = sum of squared residuals).

Solution (Normal Equations)

$$\hat{B} = (X^\top X)^{-1} X^\top Y$$

Multiple linear regression

Example

| Obs | x_1 | x_2 | x_3 | y_1 | y_2 |
|-----|-------|-------|-------|-------|-------|
| 1 | 4 | 5 | 3 | 5.75 | 1.43 |
| 2 | 5 | 5 | 2 | 5.09 | 4.30 |
| 3 | 3 | 3 | 3 | 7.48 | 1.30 |
| 4 | 5 | 4 | 3 | 6.96 | 3.04 |
| 5 | 5 | 2 | 4 | 8.40 | 2.87 |
| 6 | 2 | 4 | 5 | 7.95 | -1.02 |
| 7 | 1 | 4 | 2 | 3.19 | -0.28 |
| 8 | 5 | 4 | 1 | 4.23 | 4.30 |
| 9 | 1 | 3 | 3 | 5.09 | -0.71 |
| 10 | 2 | 4 | 4 | 5.97 | 0.35 |

Multiple linear regression

We model:

$$Y = XB + E$$

- X (10×4): design matrix with intercept + 3 predictors
- B (4×2): coefficients (for each predictor and each output)
- Y (10×2): outputs

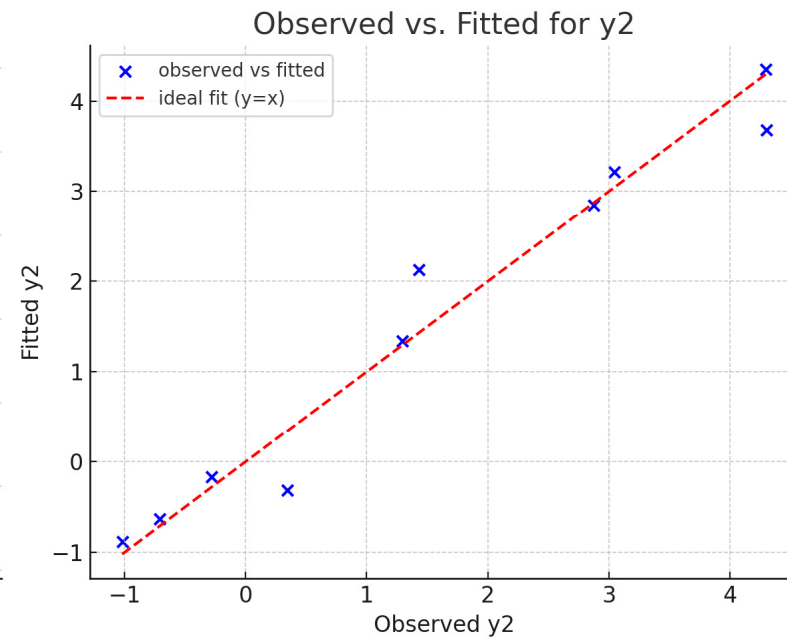
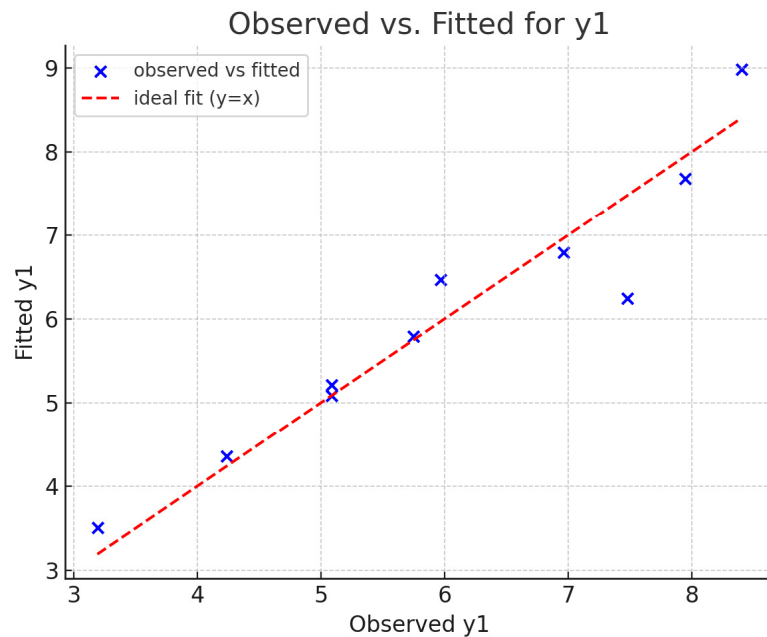
$$Y = \begin{bmatrix} 5.75 & 1.43 \\ 5.09 & 4.30 \\ 7.48 & 1.30 \\ 6.96 & 3.04 \\ 8.40 & 2.87 \\ 7.95 & -1.02 \\ 3.19 & -0.28 \\ 4.23 & 4.30 \\ 5.09 & -0.71 \\ 5.97 & 0.35 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 4 & 5 & 3 \\ 1 & 5 & 5 & 2 \\ 1 & 3 & 3 & 3 \\ 1 & 5 & 4 & 3 \\ 1 & 5 & 2 & 4 \\ 1 & 2 & 4 & 5 \\ 1 & 1 & 4 & 2 \\ 1 & 5 & 4 & 1 \\ 1 & 1 & 3 & 3 \\ 1 & 2 & 4 & 4 \end{bmatrix} \quad \hat{B} \approx \begin{bmatrix} 2.479 & 0.388 \\ 0.517 & 0.987 \\ -0.484 & -0.102 \\ 1.221 & -0.567 \end{bmatrix}$$

$$Y_1 = 2.479 + 0.517X_1 - 0.484X_2 + 1.221X_3$$

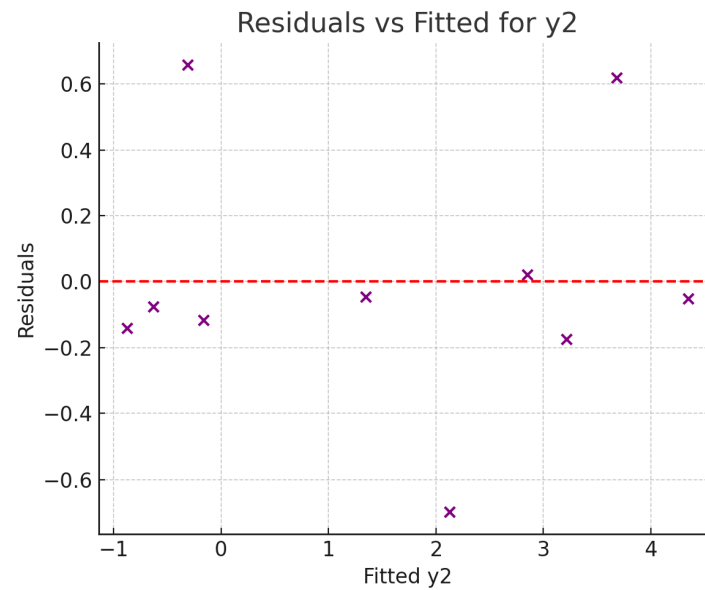
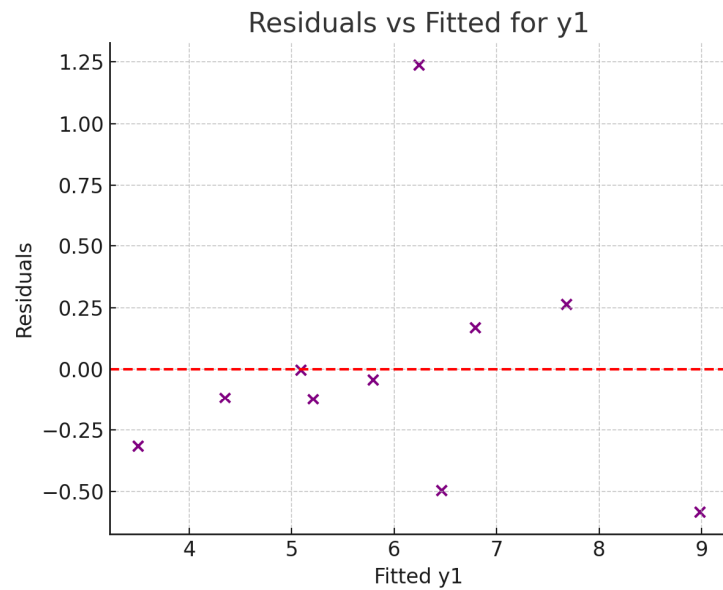
$$Y_2 = 0.388 + 0.987X_1 - 0.102X_2 - 0.567X_3$$

Multiple linear regression

| term | y1 Coef. | y1 Std.Err. | y1 t | y1 p-value | y2 Coef. | y2 Std.Err. | y2 t | y2 p-val |
|-----------|----------|-------------|--------|--------------|----------|-------------|--------|--------------|
| Intercept | 2.479 | 1.347 | 1.840 | <u>0.115</u> | 0.388 | 1.032 | 0.376 | <u>0.720</u> |
| x1 | 0.517 | 0.128 | 4.055 | <u>0.007</u> | 0.987 | 0.098 | 10.101 | <u>0.000</u> |
| x2 | -0.484 | 0.239 | -2.026 | <u>0.089</u> | -0.102 | 0.183 | -0.556 | <u>0.598</u> |
| x3 | 1.221 | 0.197 | 6.205 | <u>0.001</u> | -0.567 | 0.151 | -3.758 | <u>0.009</u> |



Multiple linear regression



| Output | R^2 | Adjusted R^2 |
|--------|-------|----------------|
| y_1 | 0.908 | 0.862 |
| y_2 | 0.961 | 0.942 |

Multiple linear regression

Assumptions

1. Linearity

- The relationship between predictors X and response Y is linear:

$$Y = XB + E$$

- No strong nonlinear patterns remain in residuals.

2. Independence

- Observations are independent of each other.
- Residuals are uncorrelated (no autocorrelation, important in time-series or spatial data).

3. Homoscedasticity (Constant Variance)

- The variance of errors ε_i is the same for all levels of the predictors.
- Residuals should show equal scatter across fitted values.

4. Normality of Errors

- The error terms ε_i are normally distributed.
- Important for hypothesis testing and confidence intervals (less critical for prediction).

5. No Perfect Multicollinearity

- Predictors are not exact linear combinations of each other.
- High multicollinearity inflates standard errors of coefficients.

Multiple linear regression

Residual analysis

1. Linearity

- Plot residuals vs. fitted values.
- Look for patterns: a curved shape suggests nonlinearity.
- Remedy: add polynomial terms, splines, or use nonlinear models.

2. Independence

- Examine study design (e.g., repeated measures?).
- For time-series: Durbin–Watson test for autocorrelation.
- For spatial data: Moran's I or variogram analysis.

3. Homoscedasticity (constant variance)

- Plot residuals vs. fitted values.
- If the spread increases with fitted values → heteroscedasticity.
- Remedies: transform response (log, sqrt), or use robust standard errors.

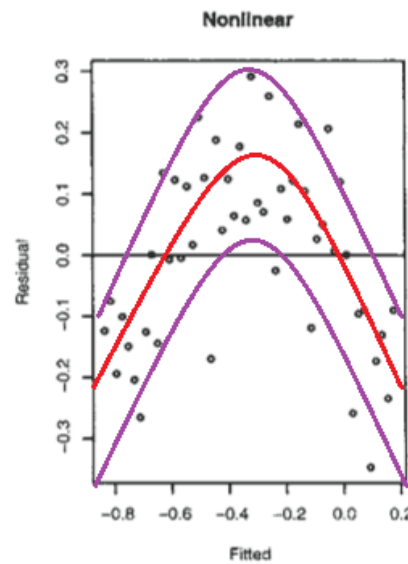
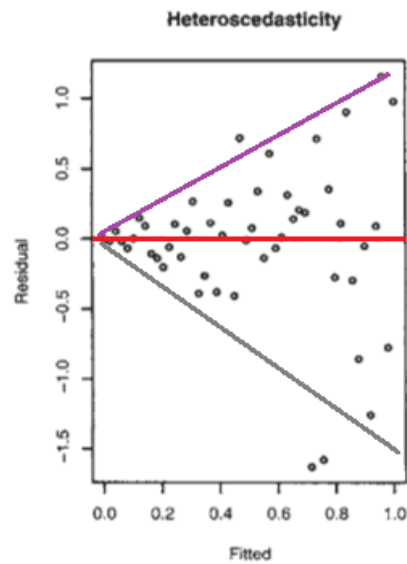
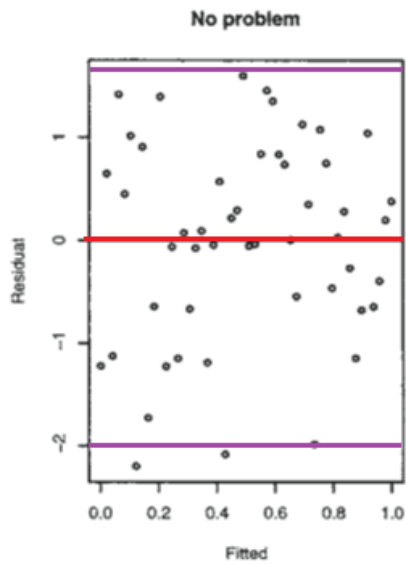
4. Normality of Errors

- Q–Q plot of residuals against the normal distribution.
- Histogram of residuals.
- Shapiro–Wilk or Kolmogorov–Smirnov test (caution: sensitive to n).

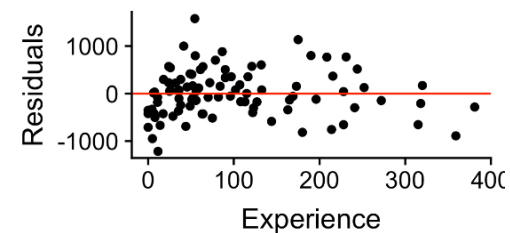
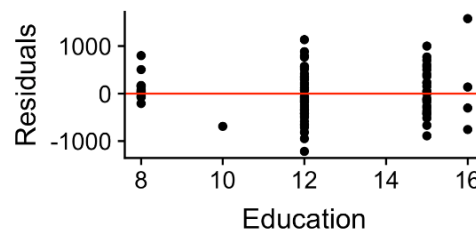
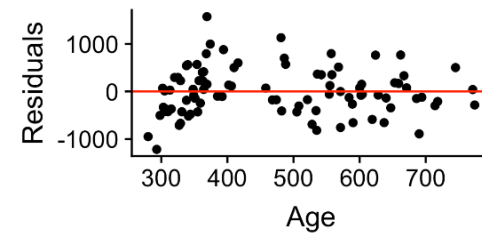
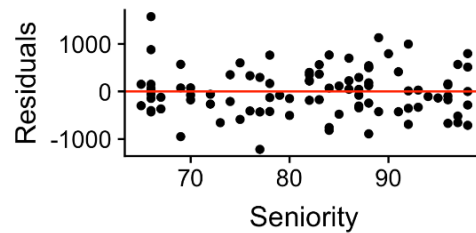
5. Multicollinearity

- Compute correlation matrix of predictors.
- Variance Inflation Factor (VIF):
 - Rule of thumb: $VIF > 10$ = problematic.
- Remedies: remove redundant predictors, use PCA, or apply Ridge regression.

Multiple linear regression

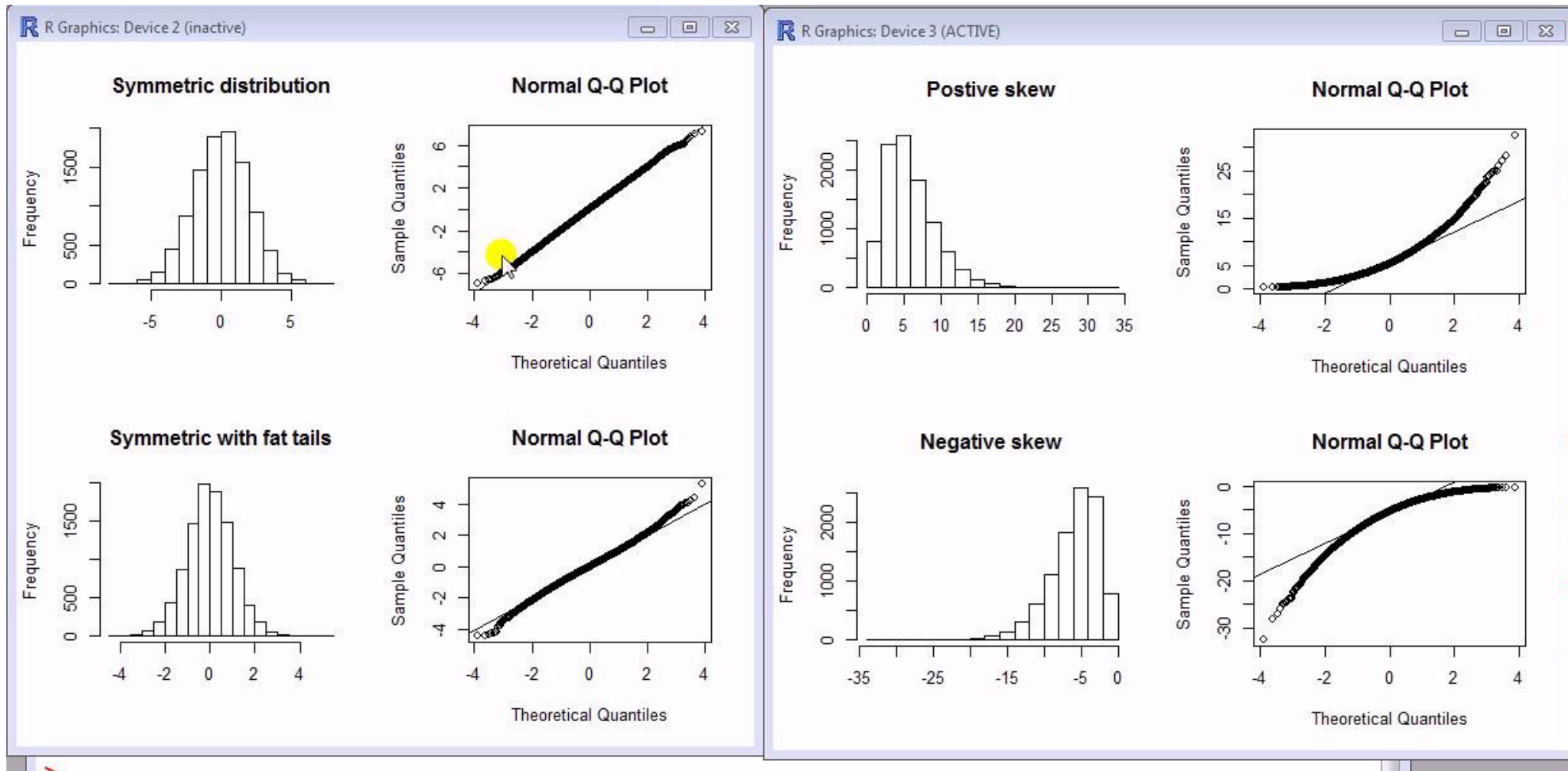


$$\epsilon \sim y$$



$$\epsilon \sim x$$

Multiple linear regression



Multiple linear regression

Polynomial fitting

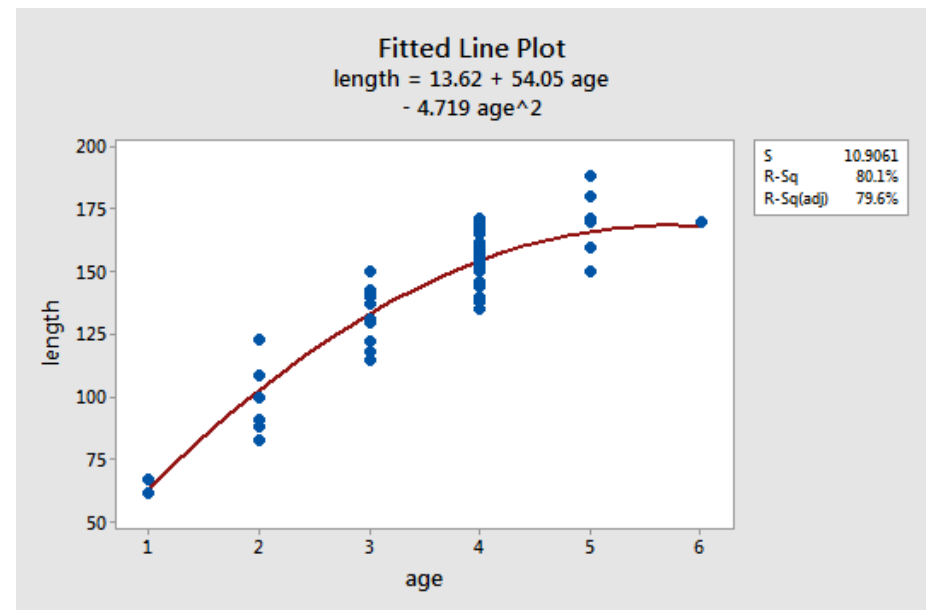
Motivation

- Many biological relationships are nonlinear (e.g., enzyme kinetics, dose-response).
- But polynomial models can capture curvature while remaining linear in the parameters.

Model (Quadratic Example)

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

- Still linear in coefficients $\beta_0, \beta_1, \beta_2$.
- Nonlinear only in the predictor x .



Multiple linear regression

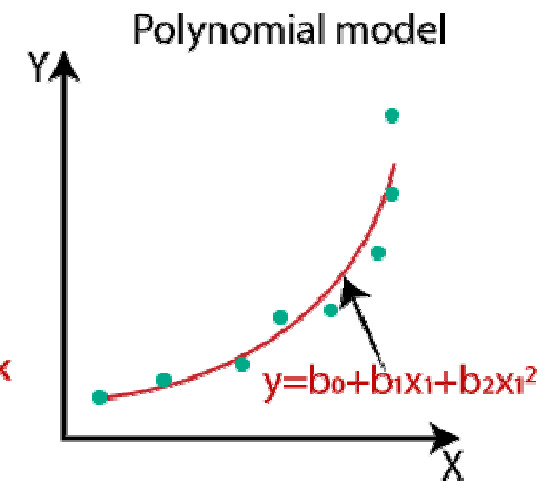
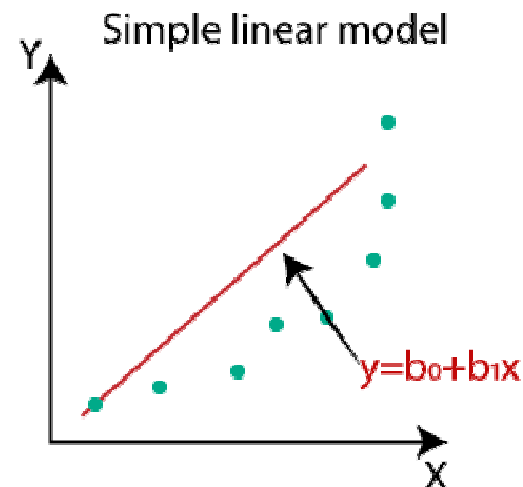
Matrix Formulation

For n observations and a degree-2 polynomial:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}, \quad y = X\beta + \varepsilon$$

- X contains polynomially expanded features.
- Then apply OLS as usual:

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



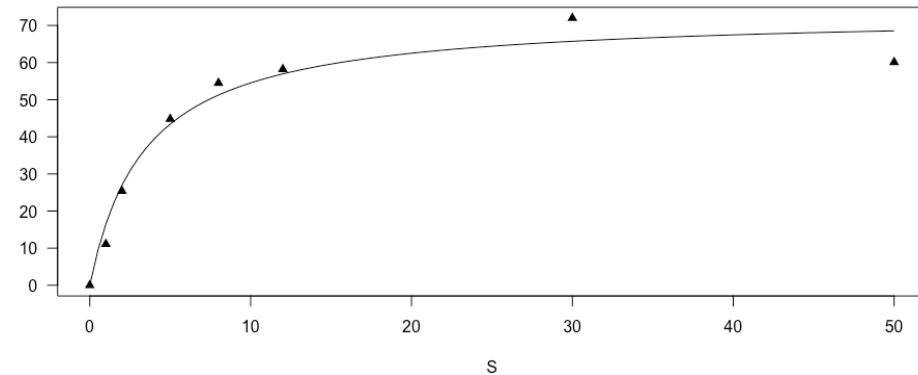
Multiple linear regression

Converting a problem into a linear one

Michaelis–Menten model

$$v = \frac{V_{\max}[S]}{K_m + [S]}$$

- v : reaction velocity
- $[S]$: substrate concentration (predictor)
- Parameters to estimate: V_{\max} , K_m



Why it is not linear regression

- In polynomial regression: model is linear in **coefficients** (β_0, β_1, \dots).
- In Michaelis–Menten:

$$v = \frac{V_{\max}}{1 + K_m/[S]}$$

→ **nonlinear in parameters** (V_{\max} , K_m).

- No transformation makes it linear in both parameters simultaneously.
- Estimation requires **nonlinear least squares** (iterative numerical optimization).

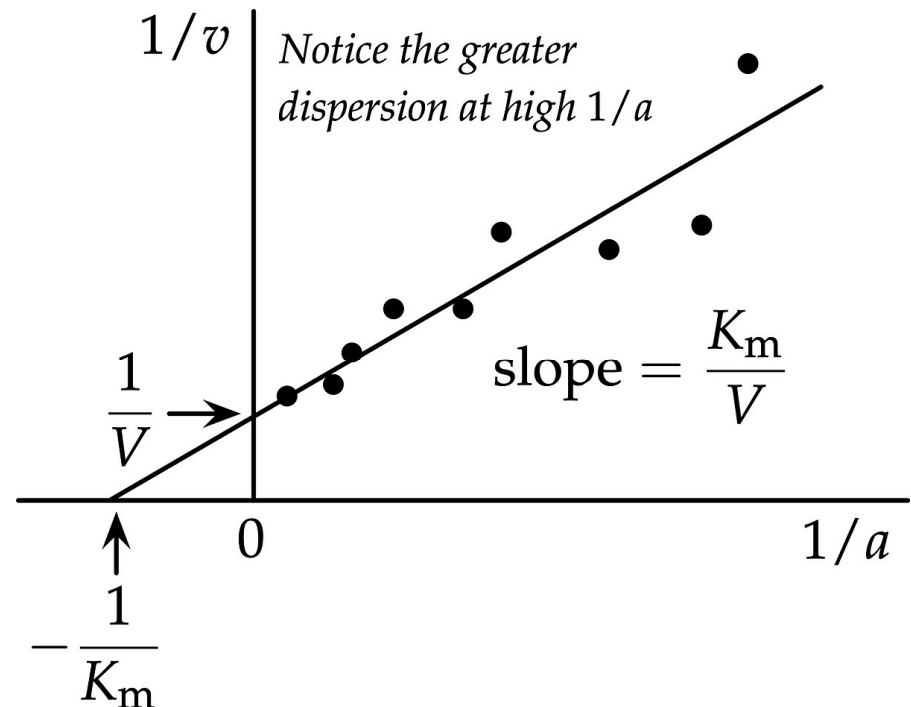
Multiple linear regression

- Lineweaver–Burk transformation:

$$\frac{1}{v} = \frac{K_m}{V_{\max}} \cdot \frac{1}{[S]} + \frac{1}{V_{\max}}$$

→ looks like a linear regression in $\frac{1}{v}$ vs. $\frac{1}{[S]}$.

- Problem: transformation distorts error structure (no longer homoscedastic).
- Modern practice: fit directly with nonlinear regression.



Multiple linear regression

Regression with Radial Basis Functions (RBFs)

Idea

- Approximate nonlinear functions by combining “bumps” centered at different points.
- Each bump is an RBF, typically Gaussian:

$$\phi_j(x) = \exp\left(-\frac{(x - c_j)^2}{2\sigma^2}\right)$$

where c_j = center, σ = width.

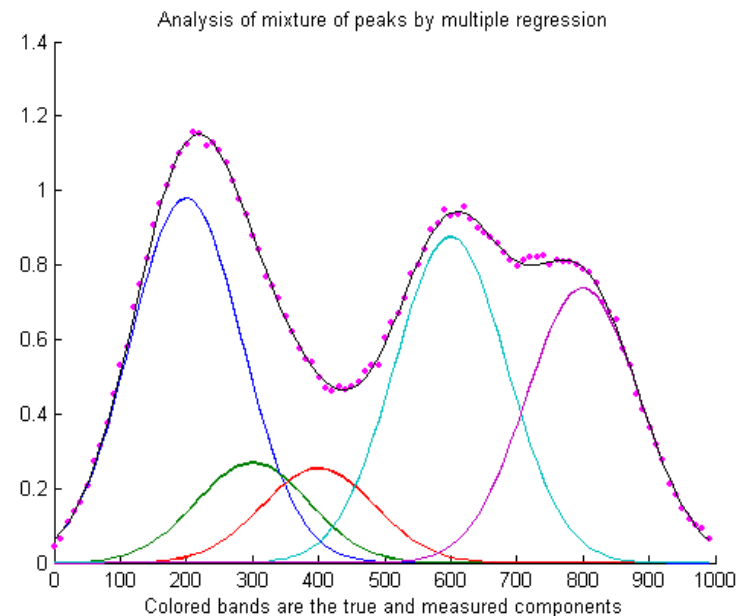
Model

$$y(x) = \beta_0 + \sum_{j=1}^m \beta_j \phi_j(x) + \varepsilon$$

- Nonlinear in x , but **linear in coefficients** β_j .
- Design matrix:

$$X_{ij} = \phi_j(x_i)$$

- Fit by OLS or Ridge/LASSO just like before.



Multiple linear regression

Multicollinearity

Motivation

- When predictors are correlated, regression coefficients become unstable.
- Standard errors inflate → harder to judge significance.
- VIF quantifies **how much multicollinearity inflates variance** of a coefficient.

Definition

For predictor x_j :

1. Regress x_j on all the other predictors.
2. Compute R_j^2 (coefficient of determination).
3. Variance Inflation Factor:

$$\text{VIF}_j = \frac{1}{1 - R_j^2}$$

Interpretation

- $\text{VIF}_j = 1$: no correlation with other predictors.
- $1 < \text{VIF}_j < 5$: moderate correlation, usually acceptable.
- $\text{VIF}_j > 10$: high multicollinearity → problematic.

Regularization

Motivation

- In high-dimensional settings ($p \gg n$), OLS estimates are unstable.
- Multicollinearity inflates variances of coefficients.
- Need regularization: shrink coefficients to stabilize the model.

Ridge regression

Model

$$\hat{B}^{\text{ridge}} = \arg \min_B \left(\|Y - XB\|_F^2 + \lambda \|B\|_2^2 \right)$$

- First term: residual sum of squares (fit).
- Second term: penalty on size of coefficients (L^2 norm).
- $\lambda \geq 0$: regularization parameter (controls shrinkage).

Closed-form solution

$$\hat{B}^{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top Y$$

- Adds λI to $X^\top X$, which removes singularity and stabilizes inversion.
- As $\lambda \rightarrow 0$, Ridge \rightarrow OLS.
- As $\lambda \rightarrow \infty$, coefficients shrink towards 0.

Regularization

Lasso regression

Motivation

- In high-dimensional data, many predictors may be irrelevant.
- We want a model that not only stabilizes coefficients (like Ridge) but also **selects features automatically**.

Model

$$\hat{B}^{\text{lasso}} = \arg \min_B \left(\|Y - XB\|_F^2 + \lambda \|B\|_1 \right)$$

- First term: fit (residual sum of squares).
- Second term: penalty on the absolute size of coefficients (L^1 norm).
- $\lambda \geq 0$: controls strength of penalty.

Key Property

- Ridge shrinks coefficients continuously but never exactly to zero.
- LASSO can set some coefficients **exactly to 0** → performs **variable selection**.
- The larger λ , the more coefficients are zeroed out.

Regularization

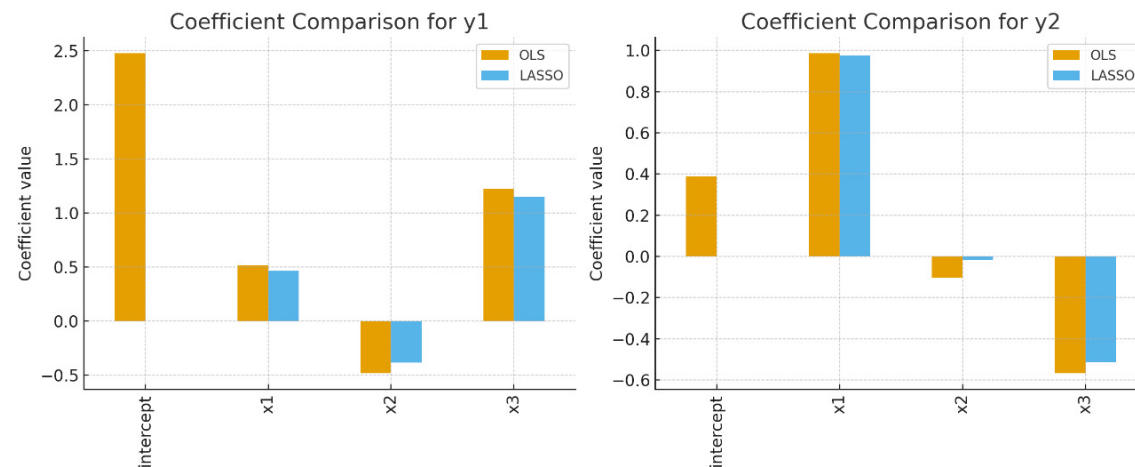
Example (OLS)

$$Y_1 = 2.479 + 0.517X_1 - 0.484X_2 + 1.221X_3$$

$$Y_2 = 0.388 + 0.987X_1 - 0.102X_2 - 0.567X_3$$

| term | y1 Coef. | y1 Std.Err. | y1 t | y1 p-value | y2 Coef. | y2 Std.Err. | y2 t | y2 p-val |
|-----------|----------|-------------|--------|--------------|----------|-------------|--------|--------------|
| Intercept | 2.479 | 1.347 | 1.840 | <u>0.115</u> | 0.388 | 1.032 | 0.376 | <u>0.720</u> |
| x1 | 0.517 | 0.128 | 4.055 | <u>0.007</u> | 0.987 | 0.098 | 10.101 | <u>0.000</u> |
| x2 | -0.484 | 0.239 | -2.026 | <u>0.089</u> | -0.102 | 0.183 | -0.556 | <u>0.598</u> |
| x3 | 1.221 | 0.197 | 6.205 | <u>0.001</u> | -0.567 | 0.151 | -3.758 | <u>0.009</u> |

Example (Lasso)



Generalized Linear Models

Motivation

- Linear regression assumes:
 - Errors are Gaussian
 - Response Y is continuous, unbounded
- But in bioinformatics:
 - Counts (RNA-seq, mutations)
 - Binary outcomes (disease vs. healthy)
 - Rates, survival times
- GLMs extend linear regression to other data types.

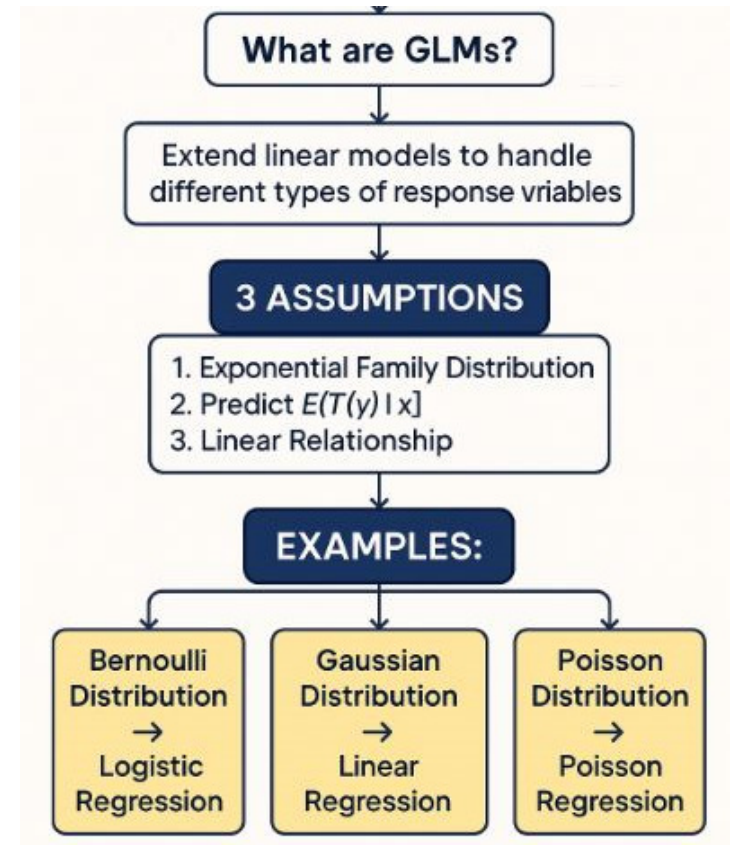
GLM Framework

- Random component:** distribution of Y (from exponential family: Normal, Binomial, Poisson, Gamma ...).
- Systematic component:** linear predictor

$$\eta = X\beta$$

- Link function:** connects mean of Y to η

$$g(\mu) = \eta, \quad \mu = \mathbb{E}[Y|X]$$



Generalized Linear Models

Examples of GLMs in Bioinformatics

- **Logistic Regression** (Binary outcome)

$$g(\mu) = \log \frac{\mu}{1 - \mu}, \quad Y \sim \text{Binomial}$$

Predict disease status from SNPs, gene expression, or imaging features.

- **Poisson Regression** (Count data)

$$g(\mu) = \log(\mu), \quad Y \sim \text{Poisson}$$

Model read counts in RNA-seq or number of mutations per gene.

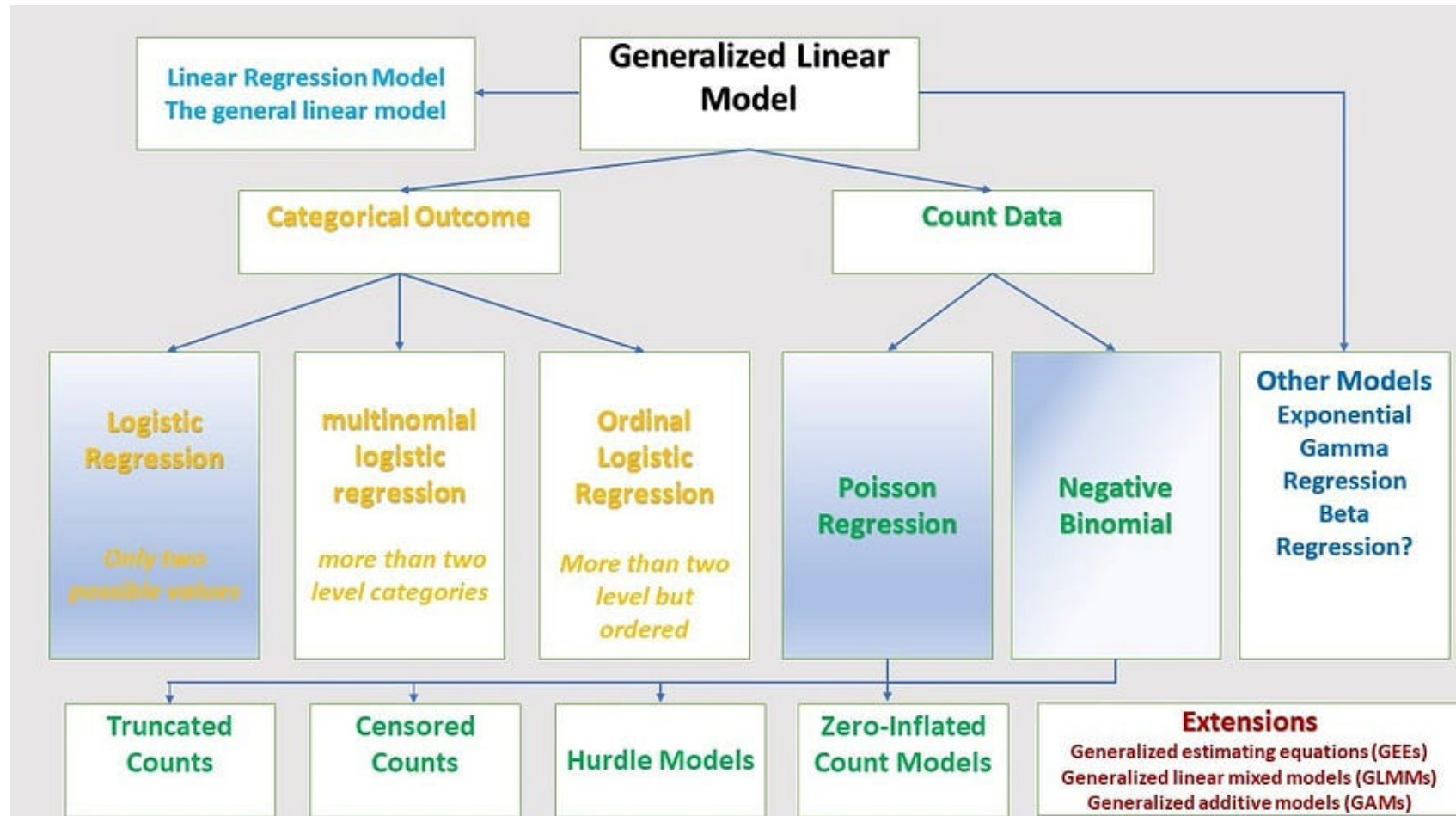
- **Gamma Regression** (Positive, skewed outcomes)

Useful for waiting times, reaction rates.

Key Idea

- Still linear in the predictors ($X\beta$).
- Nonlinear transformation (link function) ensures predictions respect the nature of data:
 - Probabilities between 0–1
 - Counts ≥ 0
 - Variance linked to mean

Generalized Linear Models



<https://medium.com/@sahin.samia/a-comprehensive-introduction-to-generalized-linear-models-fd773d460c1d>

Generalized Linear Models

Example: Predict disease status (e.g., cancer vs. control) from gene expression.

Logistic Regression (Binary GLM)

Problem

- Inputs $x_i \in \mathbb{R}^p$ (e.g., gene expression vector).
- Output $y_i \in \{0, 1\}$ (e.g., disease vs. healthy).
- Goal: model $P(y_i = 1 \mid x_i)$.

Model

- Linear predictor: $\eta_i = x_i^\top \beta$ (include intercept in x_i).
- Link (logit): $g(\mu_i) = \log \frac{\mu_i}{1-\mu_i} = \eta_i$.
- Mean: $\mu_i = \sigma(\eta_i) = \frac{1}{1+e^{-\eta_i}}$.

Likelihood (Bernoulli)

- $y_i \sim \text{Bernoulli}(\mu_i)$.
- Log-likelihood:

$$\ell(\beta) = \sum_{i=1}^n \left[y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right] = \sum_{i=1}^n \left[y_i x_i^\top \beta - \log(1 + e^{x_i^\top \beta}) \right]$$

Generalized Linear Models

Estimation

- Maximize $\ell(\beta)$ (no closed form).
- Gradient (score):

$$\nabla \ell(\beta) = X^\top (y - \mu), \quad \mu = \sigma(X\beta).$$

- Hessian (negative definite):

$$\nabla^2 \ell(\beta) = -X^\top W X, \quad W = \text{diag}\{\mu_i(1 - \mu_i)\}.$$

- Algorithms: Newton–Raphson / IRLS; or first-order methods (SGD, LBFGS).
- Regularization (common in omics):
 - **Ridge (L2)**: maximize $\ell(\beta) - \frac{\lambda}{2} \|\beta\|_2^2$.
 - **LASSO (L1)**: maximize $\ell(\beta) - \lambda \|\beta\|_1$ (feature selection).

Interpretation

- Odds: $\frac{\mu_i}{1-\mu_i} = e^{x_i^\top \beta}$.
- Coefficient β_j : a 1-unit increase in x_j multiplies the odds by e^{β_j} .
- Decision boundary: $x^\top \beta = 0$ (i.e., $\mu = 0.5$).

Non-linear regression

Regression models where the mean function is **nonlinear in parameters**:

$$y_i = f(x_i; \theta) + \varepsilon_i$$

Estimation

- No closed form for coefficients.
- Parameters estimated by **Nonlinear Least Squares (NLS)**:

$$\min_{\theta} \sum_{i=1}^n (y_i - f(x_i; \theta))^2$$

- Requires iterative numerical optimization (Gauss–Newton, Levenberg–Marquardt, gradient descent).
- Convergence depends on good initial guesses.

Key Differences vs. Linear Regression

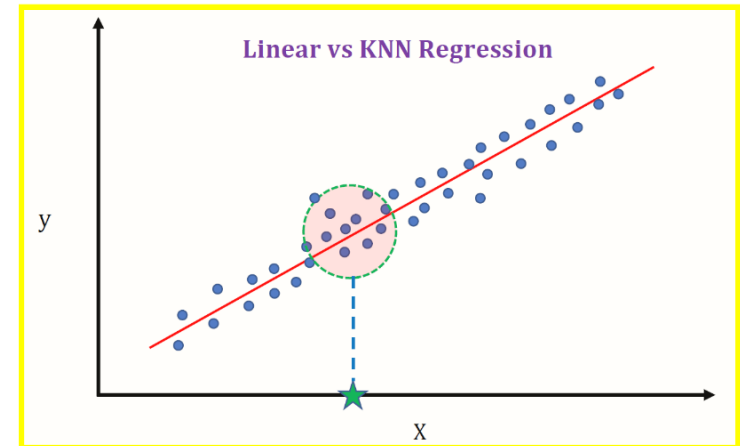
- Linear regression: closed-form OLS solution.
- Nonlinear regression: iterative, no guarantee of global optimum.
- Standard errors and confidence intervals obtained via approximations (e.g. observed Fisher information).

Non-parametric regression

- Goal: estimate $f(x)$ in $y = f(x) + \varepsilon$ **without** fixing a finite parametric form.
- Flexibility: lets the data shape f .
- Trade-off: higher variance, needs more data; careful smoothing is key.

k-Nearest Neighbors (kNN) Regression

- Rule: $\hat{f}(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$
- Local averaging over the k closest points to x .
- Hyperparameter: k (smoothness \uparrow with larger k).
- Pros: simple, no training; Cons: poor in high-D, discontinuous, sensitive to scaling.



Kernel (Nadaraya–Watson) Regression

- Estimator: $\hat{f}(x) = \frac{\sum_{i=1}^n K_h(x - x_i) y_i}{\sum_{i=1}^n K_h(x - x_i)}$
- Kernel $K_h(u) = \frac{1}{h} K(u/h)$ (e.g., Gaussian).
- Key knob: bandwidth h (small $h \rightarrow$ wiggly; large $h \rightarrow$ smooth).
- Pros: smooth fits; Cons: boundary bias, bandwidth selection critical.

Non-parametric regression

Local Linear Regression (LLR)

- Fit a **line** near x with kernel weights; predict at x .
- Reduces boundary bias vs. Nadaraya–Watson; better MSE generally.
- **Estimator:** $\hat{f}(x) = \hat{\beta}_0(x)$ from weighted least squares.

Splines & Smoothing Splines

- **Basis view:** $f(x) = \sum_j \beta_j B_j(x)$ (B-splines) with many knots.
- **Smoothing penalty:** $\min_f \sum_i (y_i - f(x_i))^2 + \lambda \int (f''(t))^2 dt$
- **Controls roughness** via λ ; solved in a spline basis.
- **Pros:** fast, interpretable smoothness; **Cons:** knot placement (if not smoothing splines).

LOESS / LOWESS (Locally Weighted Scatterplot Smoothing)

- Locally weighted **polynomial** (usually degree 1) in a window around x .
- **Span** controls smoothness (fraction of neighbors).
- Robust versions down-weight outliers.

Gaussian Process (GP) Regression (Bayesian NP)

- Put a GP prior on f : $f \sim \mathcal{GP}(0, k_\theta(\cdot, \cdot))$
- **Posterior mean** = predictor, **posterior variance** = uncertainty.
- Kernel k_θ (e.g., RBF/Matern) encodes smoothness; θ by ML or CV.
- **Pros:** uncertainty quantification; **Cons:** $\mathcal{O}(n^3)$ scaling (use sparse/inducing methods).

Challenges in Bioinformatics

1. High dimensionality, low sample size

- Typical omics data: $p \gg n$ (e.g., 20,000 genes vs. 200 patients).
- OLS unstable or undefined (cannot invert $X^T X$).
- Remedies:
 - Regularization (Ridge, LASSO, Elastic Net).
 - Dimension reduction (PCA, autoencoders, feature filtering).
 - Careful cross-validation to avoid overfitting.

2. Multicollinearity

- Biological variables are highly correlated (e.g., co-expressed genes, pathways).
- Effects:
 - Large standard errors for coefficients.
 - Difficult to interpret predictor importance.
- Remedies:
 - Use Ridge or Elastic Net (stabilizes estimates).
 - Group predictors (pathway-level features, latent factors).
 - Variance Inflation Factor (VIF) to diagnose redundancy.

Challenges in Bioinformatics

3. Missing data

- Common in clinical and omics datasets (failed assays, dropouts).
- Naïve deletion → information loss, bias.
- Strategies:
 - Simple: mean/mode imputation (weak).
 - Advanced: kNN imputation, matrix factorization, multiple imputation, deep generative models.
 - Consider *mechanism* of missingness (MCAR, MAR, MNAR).

Contents

- Overview
- Linear regression
 - Simple, Multiple, Residual analysis, RBF
- Regularization
 - Ridge, Lasso
- Generalized Linear Models
- Non-linear regression
- Non-parametric regression
- Challenges in Bioinformatics



CEU

*Universidad
San Pablo*

Lesson 6. Classification

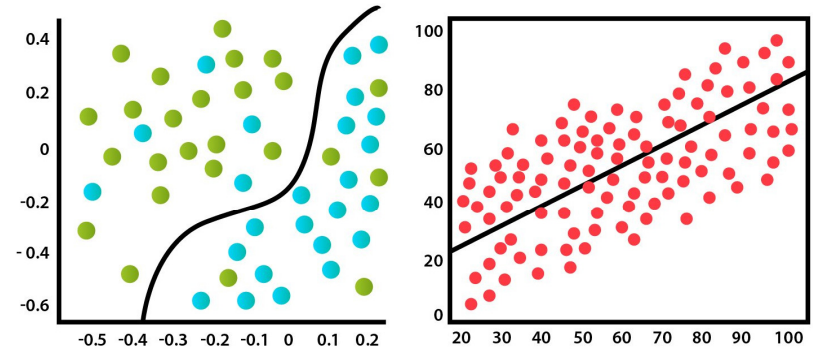
Medicin School

Contents

- Overview
- Data preparation
- Linear classifiers
 - LDA, QDA, Logistic regression
- Non-linear
 - SVM, kNN, Tree, Random Forest, Naïve Bayes
- Ensemble methods
- Challenges in Bioinformatics

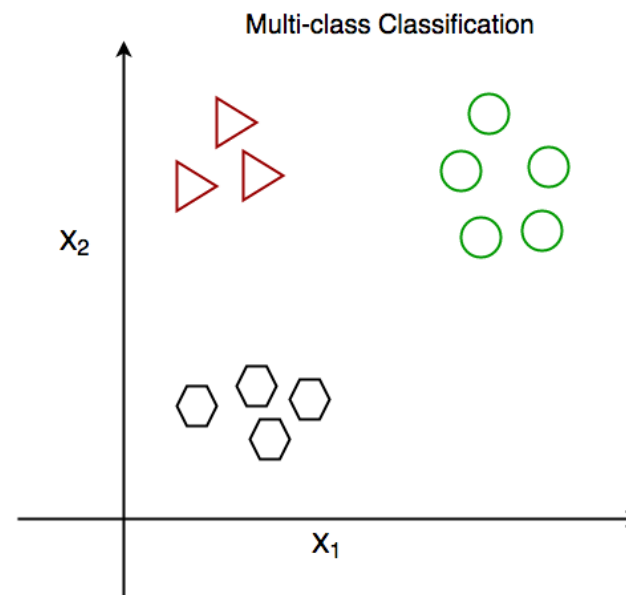
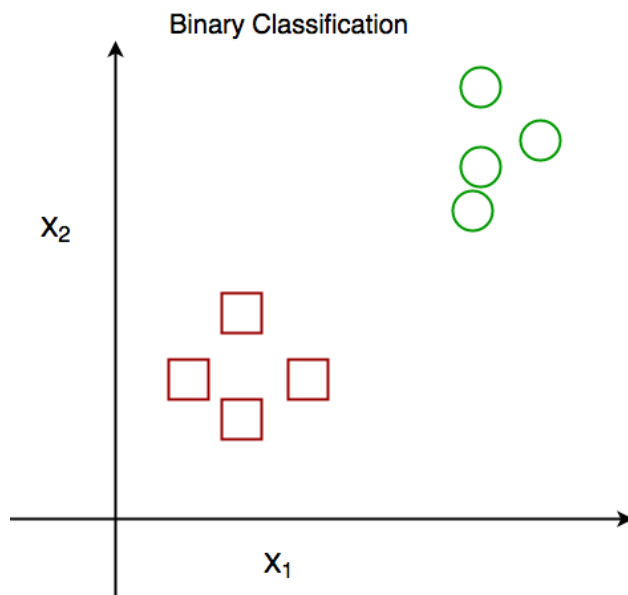
Overview

- Input space $X \in \mathbb{R}^p$, output labels $Y \in \{1, 2, \dots, K\}$.
- Classifier: function $f : X \rightarrow Y$.
- Goal: minimize classification error / maximize predictive accuracy.

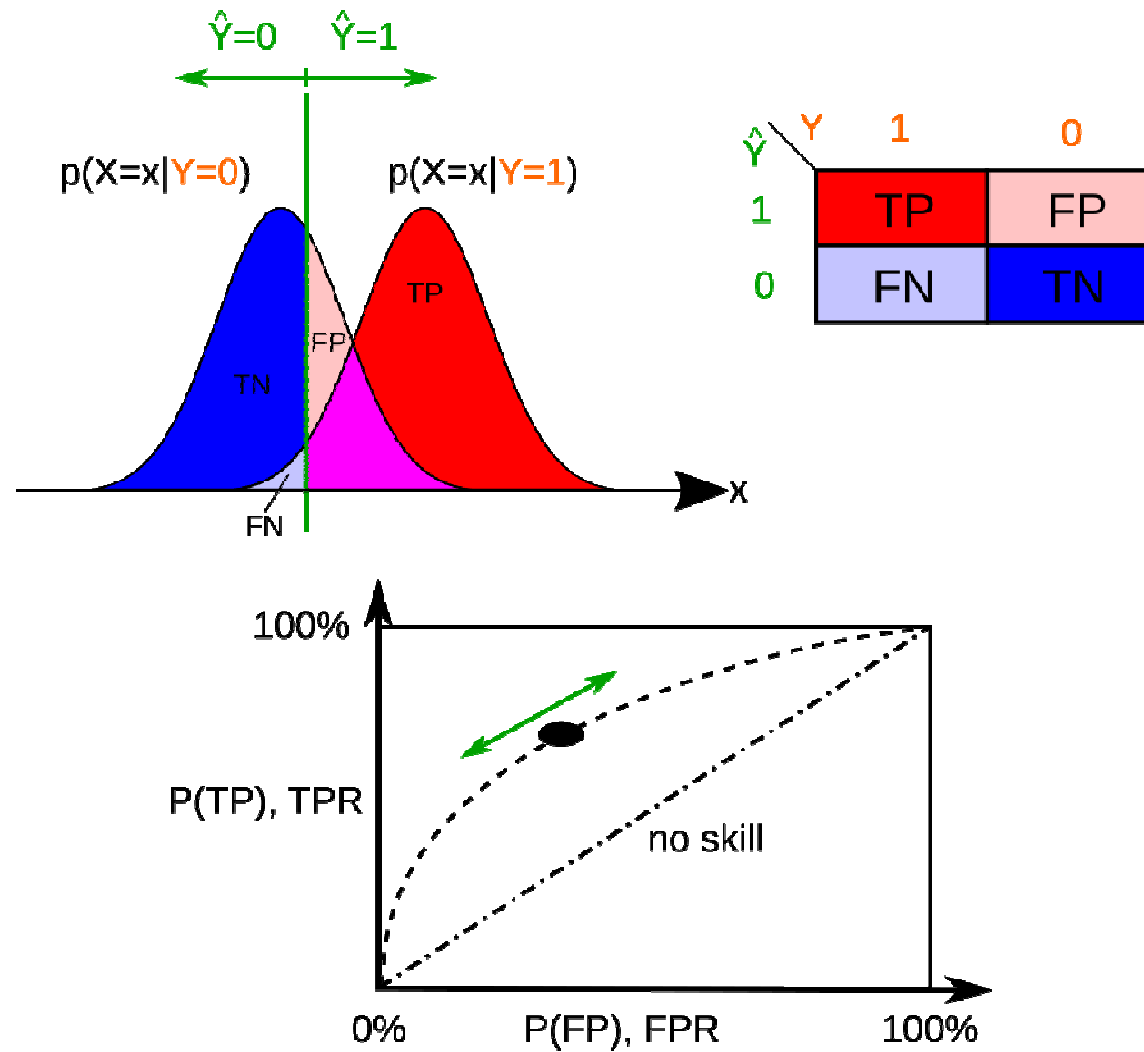


CLASSIFICATION

REGRESSION



Overview



Overview

ROC Curve and AUC

1. Why Not Accuracy?

- In imbalanced datasets (e.g., rare disease detection), accuracy can be misleading.
 - Example: 95% healthy, 5% diseased.
 - A classifier that always predicts “healthy” has 95% accuracy, but is useless.
- Instead, we need metrics that consider **true positives** and **false positives**.

2. ROC Curve (Receiver Operating Characteristic)

- Plot that shows **trade-off between sensitivity and specificity** at different thresholds.
- Definitions:
 - **True Positive Rate (TPR / Recall / Sensitivity):**

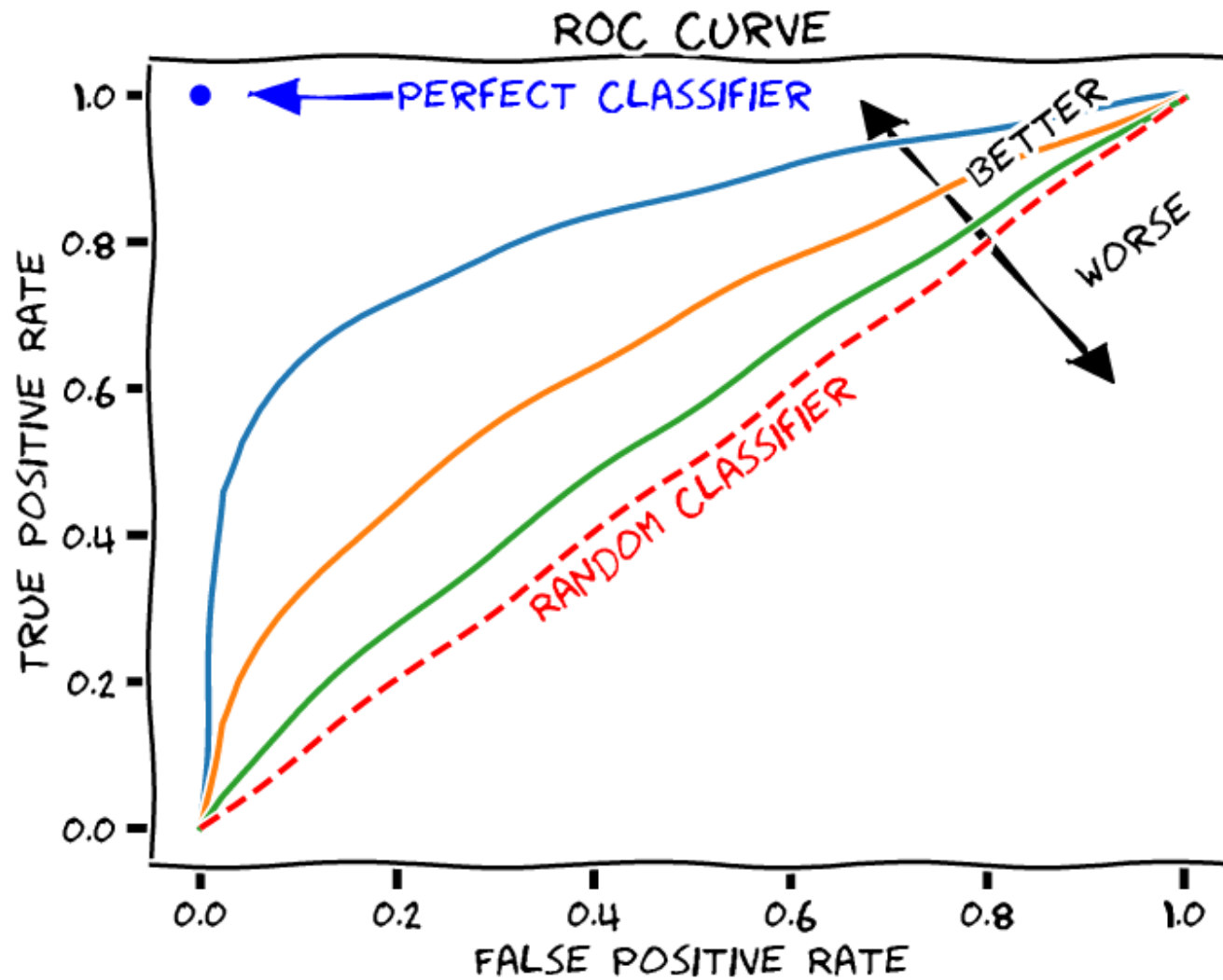
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **False Positive Rate (FPR):**

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- Procedure:
 - Classifier outputs probabilities or scores.
 - Vary the decision threshold (e.g., 0.1, 0.2, ..., 0.9).
 - Compute TPR and FPR at each threshold.
 - Plot **TPR (y-axis)** vs **FPR (x-axis)**.

Overview



Overview

3. AUC (Area Under the Curve)

- The area under the ROC curve:

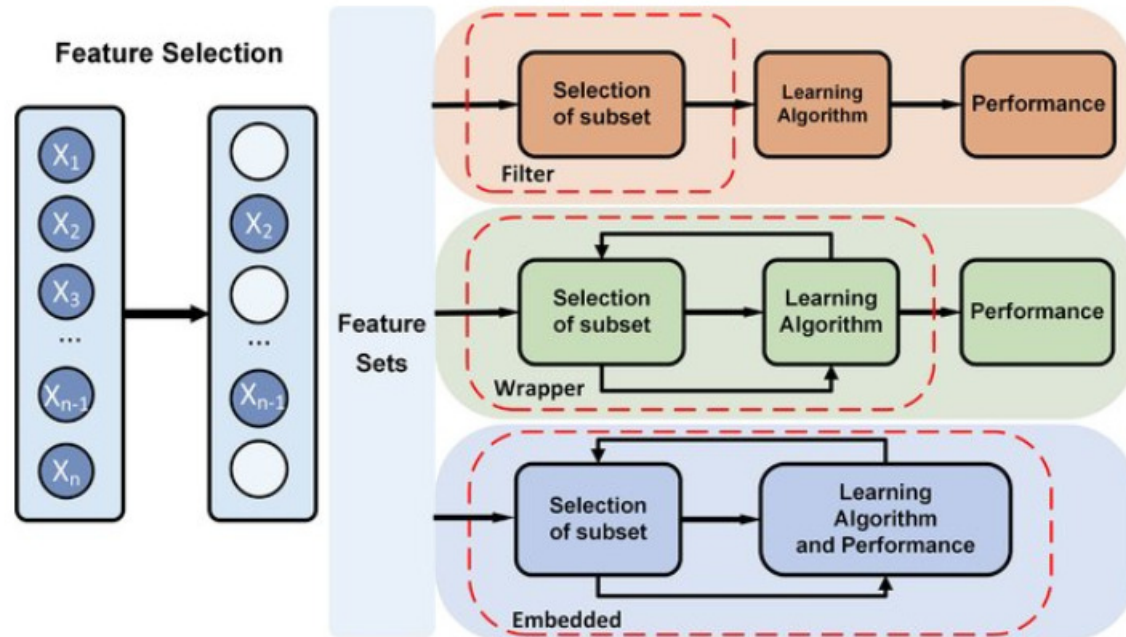
$$AUC = \int_0^1 \text{TPR}(FPR) d(FPR)$$

- Interpretation:
 - Probability that a randomly chosen positive is ranked higher than a randomly chosen negative.
 - $AUC = 1.0$ → perfect classifier.
 - $AUC = 0.5$ → random guessing.
 - $AUC < 0.5$ → worse than random (predictions reversed).

Data preparation

1. Feature Selection & Dimensionality Reduction

- **Why?**
 - Bioinformatics data often has $p \gg n$ (e.g., thousands of genes, few samples).
 - Redundant or irrelevant features → noise, overfitting.
- **Methods**
 - Filter: correlation, mutual information, statistical tests (t-test, ANOVA).
 - Wrapper: recursive feature elimination.
 - Embedded: LASSO, decision tree importance.
 - Dimensionality reduction: PCA, t-SNE, UMAP.



Data preparation

1. Filter Methods

- **Idea:** Select features *independently of the classifier*.
- **How?** Rank features by a statistical criterion → keep top ones.
- **Examples of criteria:**
 - Correlation with the class label.
 - Mutual information.
 - Statistical tests (t-test, ANOVA, chi-square).
- **Advantages**
 - Very fast, scalable to high-dimensional data (e.g. gene expression).
 - Classifier-agnostic.
- **Disadvantages**
 - Ignores interactions between features.
 - May discard useful combinations.

👉 **Bioinformatics example:** Selecting the top 100 genes most correlated with cancer vs. healthy samples before training a classifier.

Data preparation

2. Wrapper Methods

- **Idea:** Use the classifier itself to evaluate subsets of features.
- **How?** Try different feature subsets → train model → keep best-performing subset.
- **Search strategies:**
 - Forward selection (start empty, add features one by one).
 - Backward elimination (start full, remove features).
 - Recursive Feature Elimination (RFE).
- **Advantages**
 - Accounts for feature interactions.
 - Usually yields better performance than filters.
- **Disadvantages**
 - Computationally expensive (repeated model training).
 - Risk of overfitting if dataset is small.

👉 **Bioinformatics example:** Using recursive feature elimination with an SVM to identify the optimal set of genes for distinguishing tumor subtypes.

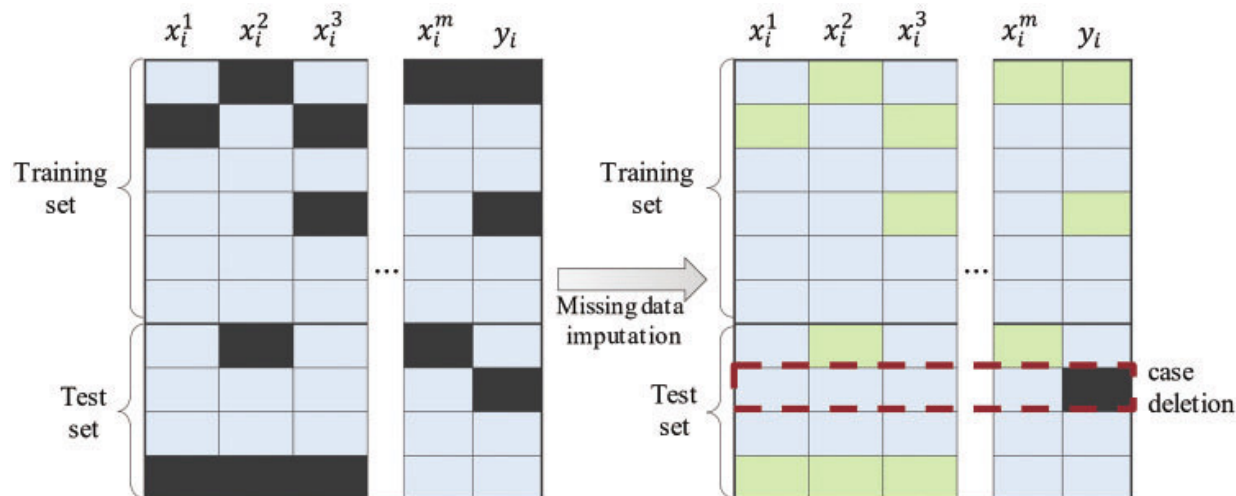
Data preparation

2. Handling Missing Values & Normalization

- **Missing values**
 - Deletion (if few).
 - Imputation (mean, kNN, model-based).
 - Important in genomic/proteomic data with experimental gaps.
- **Normalization**
 - Scale features to comparable ranges.
 - Log-transformation for skewed biological measures.
 - Z-score standardization, min-max scaling.

$$x' = \frac{x - \bar{x}}{s_x}$$

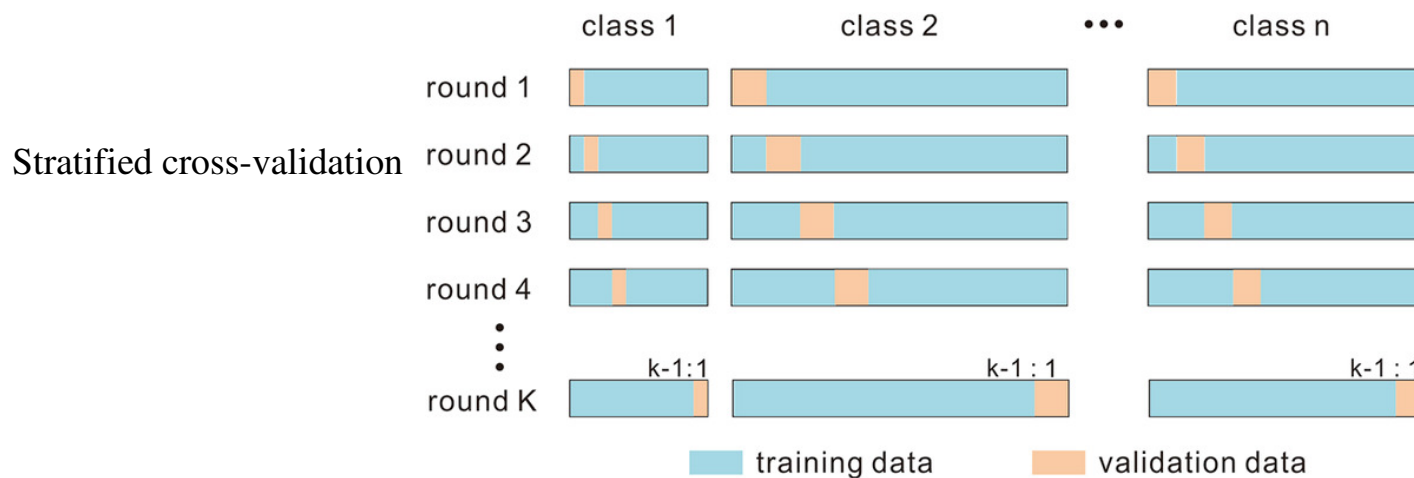
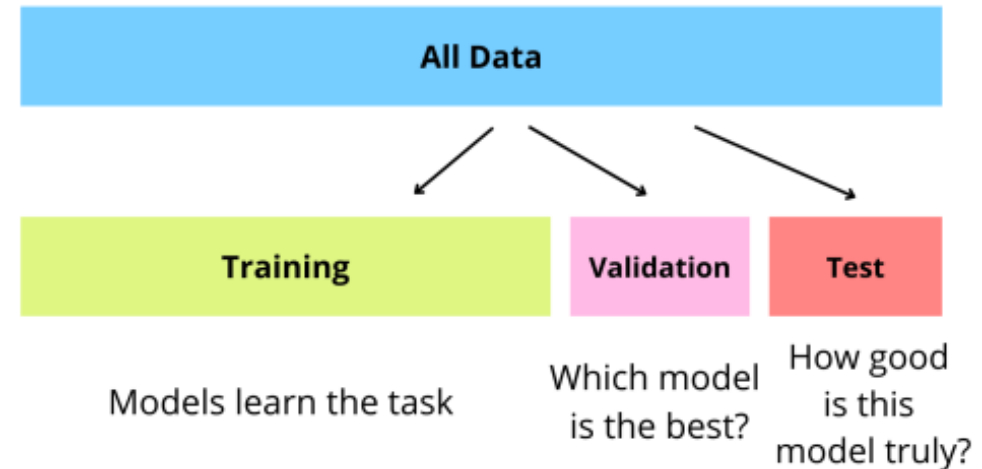
$$x' = \frac{x - m}{M - m}$$



Data preparation

3. Training, Validation & Test Sets

- **Why?** Prevent overfitting and ensure generalization.
- **Splits**
 - Training set: fit model parameters.
 - Validation set: hyperparameter tuning.
 - Test set: unbiased evaluation.
- **Cross-validation**
 - k-fold CV widely used with small bio datasets.
 - Stratified CV for class imbalance.



Data preparation

4. Class Imbalance in Bioinformatics

- **Problem:** Rare diseases or rare mutations → minority class.
- **Consequences:** Classifier biased toward majority class.
- **Solutions**
 - Resampling: oversample minority (SMOTE), undersample majority.
 - Cost-sensitive learning (higher penalty for misclassifying minority).
 - Evaluation metrics beyond accuracy (precision, recall, AUC).

Linear classifiers

Linear Discriminant Analysis (LDA)

1. Basic Idea

- LDA is a **linear classifier**.
- It assumes that data from each class follows a **multivariate normal distribution** with:
 - **Different means** (μ_k for each class k).
 - **Same covariance matrix** (Σ).
- It finds a **linear boundary** between classes that maximizes separation.

Strengths and Weaknesses



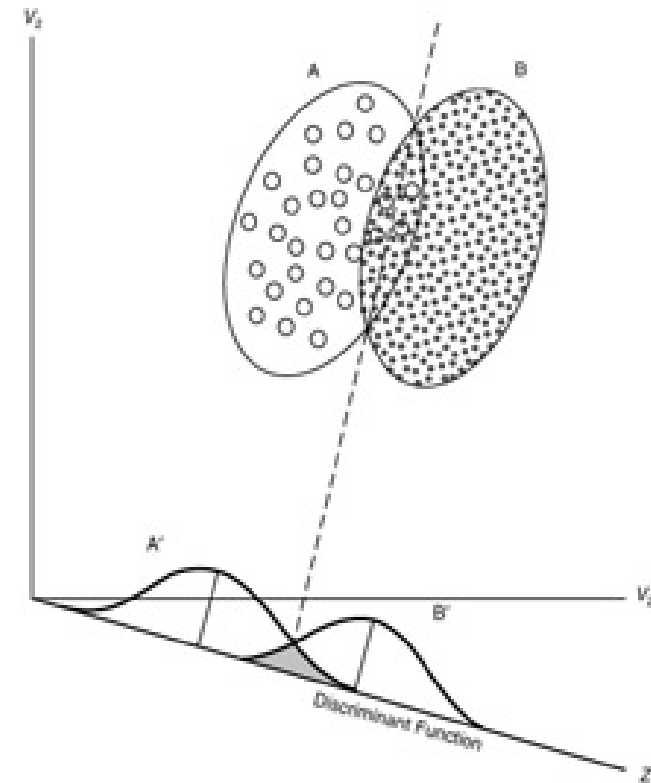
Strengths

- Simple, fast, interpretable.
- Works well with small datasets.



Weaknesses

- Assumes normality and equal covariance.
- Struggles with non-linear boundaries or highly correlated features.



Linear classifiers

Linear Discriminant Analysis (LDA)

2. Mathematical Formulation

- For class k :

$$p(x|y = k) \sim \mathcal{N}(\mu_k, \Sigma)$$

- By Bayes' theorem, the posterior is:

$$P(y = k|x) \propto \pi_k \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1}(x - \mu_k)\right)$$

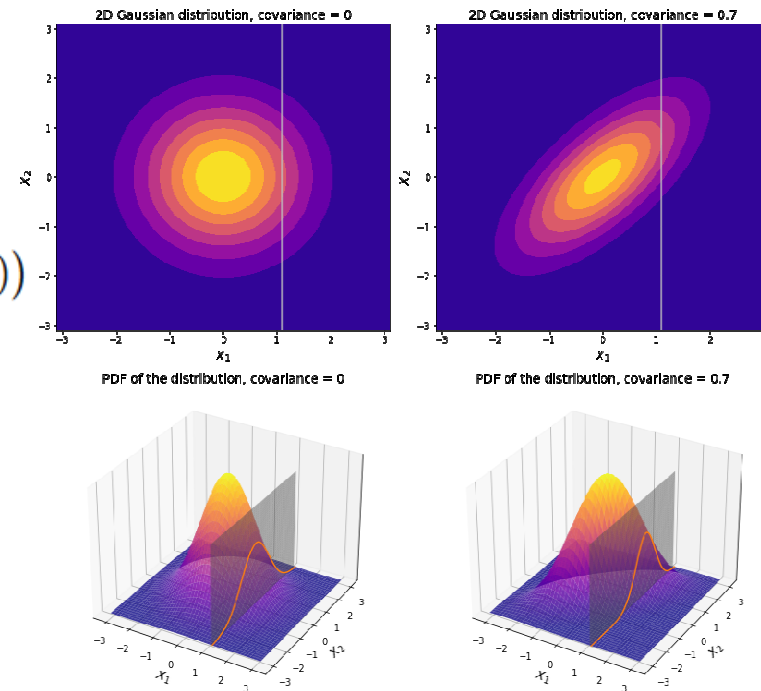
where π_k is the prior probability of class k .

- The **discriminant function** (log-posterior, ignoring constants):

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

- Classification rule:**

Assign x to the class k with the largest $\delta_k(x)$.



Linear classifiers

Linear Discriminant Analysis (LDA)

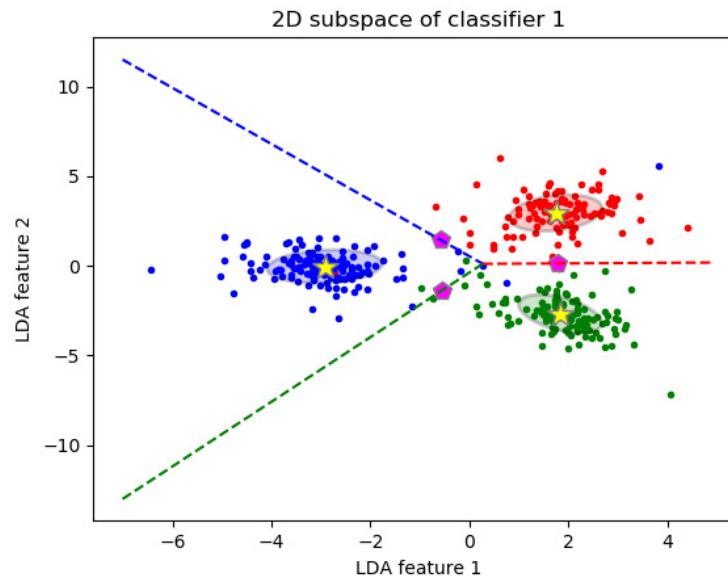
3. Decision Boundaries

- Boundaries between classes are **linear hyperplanes**.
- For two classes $k = 1, 2$, the decision boundary is:

$$(\mu_1 - \mu_2)^T \Sigma^{-1} x = c$$

→ a straight line in 2D, a hyperplane in higher dimensions.

- Intuition: LDA projects data onto a line that best separates classes, then sets thresholds.



Linear classifiers

Quadratic Discriminant Analysis (QDA)

1. Basic Idea

- QDA is an extension of LDA.
- Assumes that each class follows a **multivariate normal distribution**, but **allows different covariance matrices** for each class.
 - For class k :

$$p(x|y = k) \sim \mathcal{N}(\mu_k, \Sigma_k)$$

- This leads to **quadratic decision boundaries**, instead of linear ones.

2. Discriminant Function

- The log-posterior (up to a constant) becomes:

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k$$

- **Classification rule:** Assign x to the class with the largest $\delta_k(x)$.

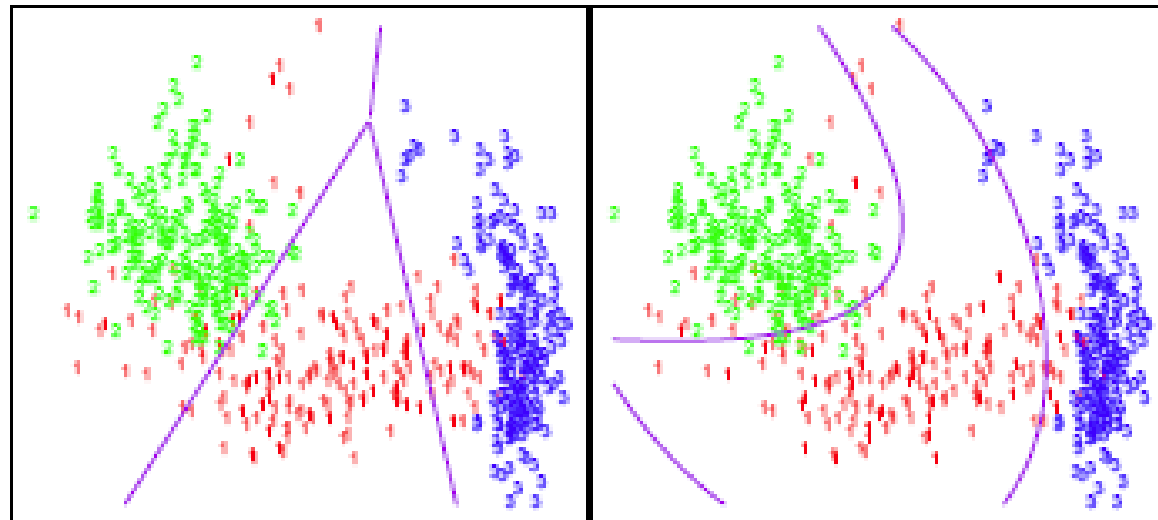
3. Decision Boundaries

- In QDA, the terms include **quadratic functions of x** .
- Boundaries between classes are **conic sections** (ellipses, parabolas, hyperbolas).
- Much more flexible than LDA → can adapt to more complex class distributions.

Linear classifiers

4. Comparison with LDA

| Aspect | LDA | QDA |
|-------------|---|------------------------------------|
| Covariance | Same across classes | Different for each class |
| Boundaries | Linear | Quadratic |
| Flexibility | Less flexible | More flexible |
| Data needs | Fewer parameters, works with small data | More parameters, needs larger data |
| Risk | May underfit | May overfit |



Linear classifiers

Logistic regression

1. Basic Idea

- Logistic regression is a **probabilistic linear classifier**.
- Instead of predicting a class directly, it models the **probability of class membership**.
- Useful for binary (yes/no) or multiclass problems.

2. Binary Logistic Regression

- Suppose classes are $y \in \{0, 1\}$.
- Model:

$$P(y = 1|x) = \sigma(\beta_0 + \beta^T x)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the **logistic (sigmoid) function**.

- Decision rule:
 - Predict class 1 if $P(y = 1|x) > 0.5$.
 - Otherwise predict class 0.
- Interpretation: Coefficients β_j correspond to the **log-odds ratio** of the outcome given feature x_j .

Linear classifiers

3. Multiclass Logistic Regression

- For K classes, use the **softmax function**:

$$P(y = k|x) = \frac{\exp(\beta_{0k} + \beta_k^T x)}{\sum_{j=1}^K \exp(\beta_{0j} + \beta_j^T x)}$$

- Strategies:
 - One-vs-Rest (OvR)**: Train one binary classifier per class.
 - Multinomial (Softmax)**: Single model, all classes handled together.

4. Decision Boundaries

- Logistic regression produces **linear decision boundaries**.
- In 2D, this is a straight line; in higher dimensions, a hyperplane.

Strengths and Weaknesses

✓ Strengths

- Probabilistic interpretation (confidence in predictions).
- Simple, interpretable coefficients.
- Works well for binary and multiclass classification.

✗ Weaknesses

- Only linear boundaries.
- Performance degrades with high-dimensional correlated features unless regularized.

Non-linear classifiers

Support Vector Machine (SVM)

1. Basic Idea

- SVMs are **powerful margin-based classifiers**.
- Goal: find the **hyperplane** that best separates classes.
- "Best" = the one that maximizes the **margin** (distance from hyperplane to nearest training points).
- The nearest training points are called **support vectors** — they define the boundary.

2. Linear SVM

For binary classification with labels $y_i \in \{-1, +1\}$:

- Decision function:
- Classification rule:
- Optimization problem (hard margin):

$$f(x) = w^T x + b$$

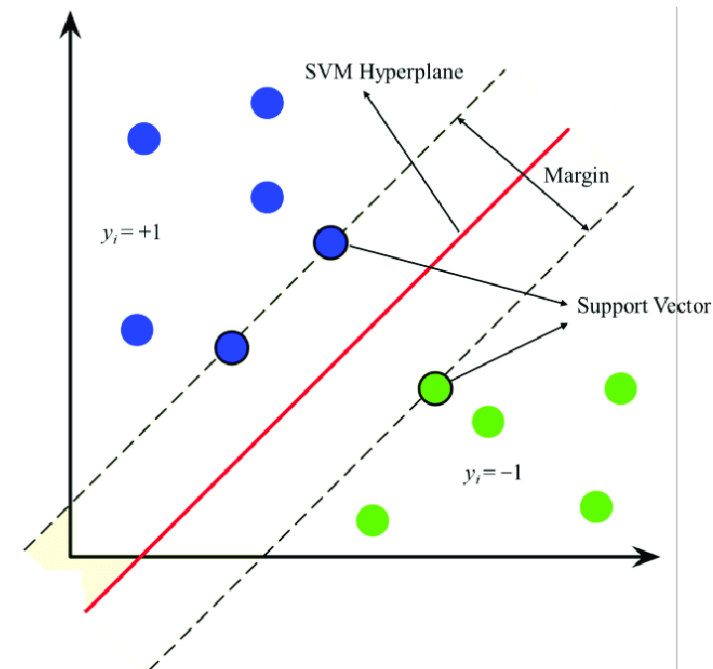
$$\hat{y} = \text{sign}(f(x))$$

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

subject to

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

- ☒ Maximizes margin $\frac{2}{\|w\|}$.



Non-linear classifiers

3. Soft Margin (C parameter)

- Real-world data are noisy and not perfectly separable.
- Allow **slack variables** ξ_i for misclassifications:

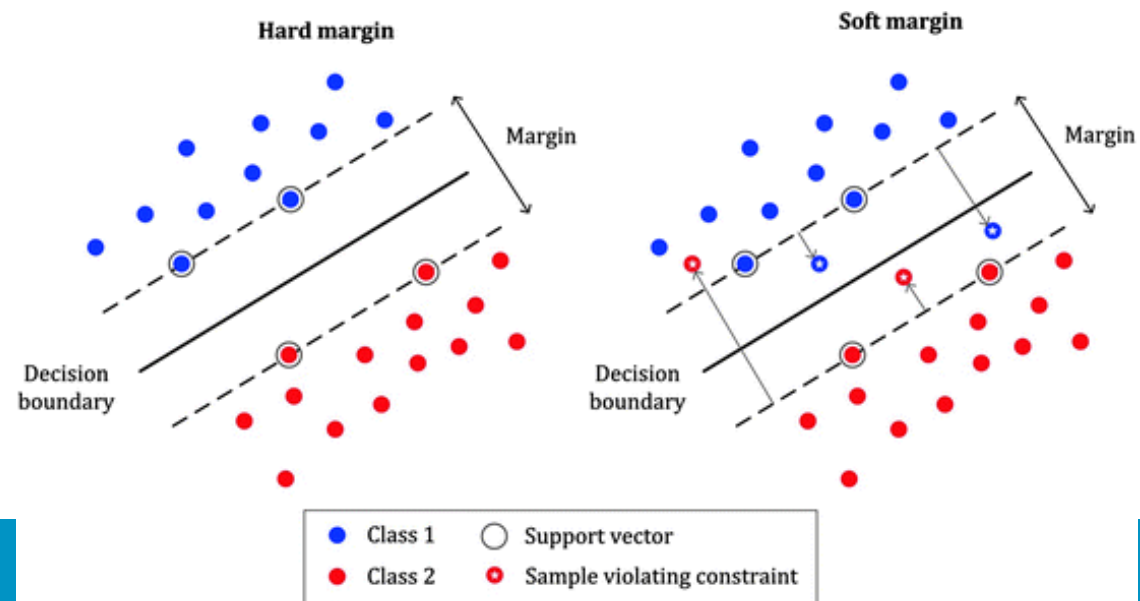
$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

- Objective:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i$$

- **Trade-off:**

- Large C : penalizes errors heavily \rightarrow smaller margin, fewer misclassifications.
- Small C : allows more violations \rightarrow larger margin, more generalization.



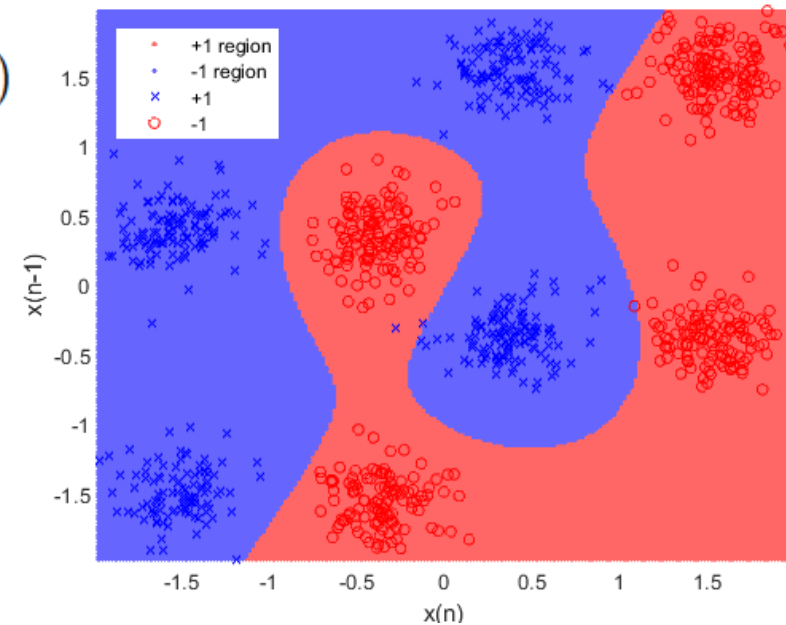
Non-linear classifiers

4. Kernel Trick

- Linear SVM only works if data are linearly separable.
- **Kernel functions** map data into higher-dimensional space without explicit computation.
- Decision boundary becomes non-linear in the original space.
- Common kernels:
 - **Polynomial:** $K(x, x') = (x^T x' + c)^d$.
 - **Radial Basis Function (RBF):**

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

- **Sigmoid:** $K(x, x') = \tanh(\alpha x^T x' + c)$.



Non-linear classifiers

1. Linear SVM Recap

We want to classify samples (x_i, y_i) , with $y_i \in \{-1, +1\}$.

- Decision function:

$$f(x) = w^T x + b$$

- Optimization problem (soft-margin):

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Non-linear classifiers

2. Dual Formulation

By introducing Lagrange multipliers $\alpha_i \geq 0$, we obtain:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

subject to

$$\sum_{i=1}^n \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C$$

- Only inner products $\langle x_i, x_j \rangle$ appear.
- Decision function:

$$f(x) = \sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b$$

where only support vectors ($\alpha_i > 0$) contribute.

Non-linear classifiers

3. Kernel Trick

- Replace inner product $\langle x_i, x_j \rangle$ with a **kernel function** $K(x_i, x_j)$.
- A kernel implicitly maps data into a higher-dimensional feature space $\phi(x)$:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

- Dual problem becomes:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

- Decision function:

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b$$

Non-linear classifiers

k-Nearest Neighbors (kNN) classifier

1. Basic Idea

- A **non-parametric** classifier.
- To classify a new sample x :
 1. Find the k training points closest to x .
 2. Assign the class that is most common among those neighbors.
- No explicit training — the “model” is just the training set.

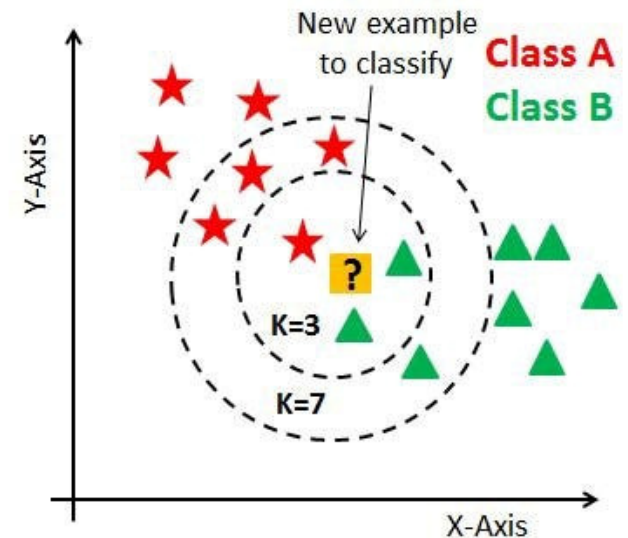
2. Distance Metrics

- The notion of **closeness** depends on the chosen metric.
- Common metrics:
 - **Euclidean distance:**


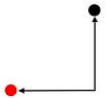
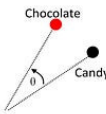
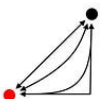
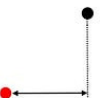

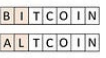
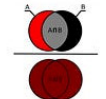
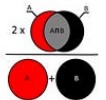
$$d(x, x') = \sqrt{\sum_j (x_j - x'_j)^2}$$

(default for continuous features).

- **Cosine similarity** (useful in high-dimensional data like gene expression).
- **Manhattan distance:** sum of absolute differences.



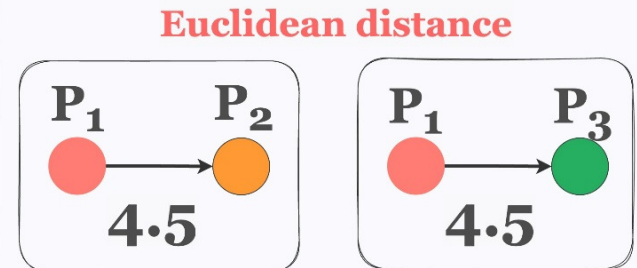
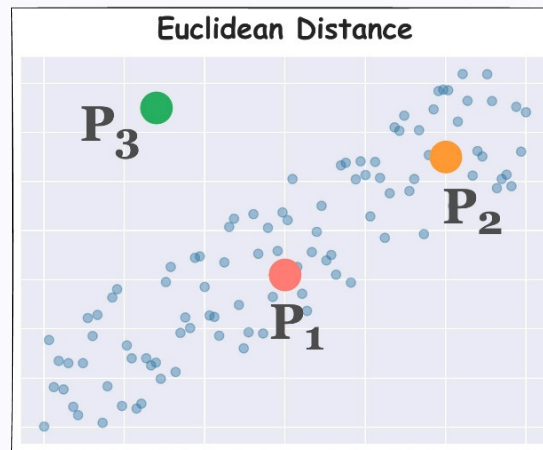
Non-linear classifiers

| Picture | Method | Application | Features | Disadvantages | Formula |
|---|-----------------------------|--|--|---|------------------|
|  | Euclidean Distance | General distance measurement, Clustering, Classification, Regression | Measures the straight line distance between two points in n-dimensional space. | Sensitive to outliers, Can be affected by scale differences | $O(n)$ Fast |
|  | Manhattan Distance | Distance on grid networks, Routing algorithms, Image processing | Measures the distance between two points on a grid network, where movement is limited. | Ignores diagonal movement, not useful for high-dimensional data, | $O(n)$ Fast |
|  | Cosine Similarity | Text document clustering, Text analysis, Recommendation systems | Measures the cosine of the angle between two vectors | Ignores magnitude of vectors, Not useful for negative values or high degree of correlation data | $O(n)$ Fast |
|  | Minkowski Distance | General distance measurement | Measures the distance between two points in n-dimensional space, where r determines the metric used. | Sensitive to outliers | $O(n)$ Fast |
|  | Chebyshev Distance | Measuring maximum difference, Clustering, Anomaly detection | Measures the maximum difference between corresponding components of two vectors | Only applicable for continuous data, Sensitive to outliers, may not be as useful for highly correlated data | $O(n)$ Fast |
|  | Hamming Distance | Measuring string similarity, Error-correcting codes, DNA sequencing | Measures the number of positions at which the corresponding symbols are different. | Only for same length strings, May not be as useful for continuous data | $O(n)$ Fast |
|  | Levenshtein Distance | Measuring string similarity | Measures the minimum number of single-character edits required to transform one string into another. | More expensive for long strings | $O(n^2)$ Slow |
|  | Jaccard Similarity | Set similarity measurement, Text analysis, recommendation systems | Measures the similarity between two sets by comparing their intersection and union. | Ignores magnitude of sets, May not be as useful for continuous data | $O(n)$ Fast |
|  | Sørensen-Dice Index | Measuring similarity of sets, Ecology, Biology, Genetics | Measures the similarity between two sets | May not be as useful for continuous data and Ignores magnitude of sets | $O(n)$ Fast |

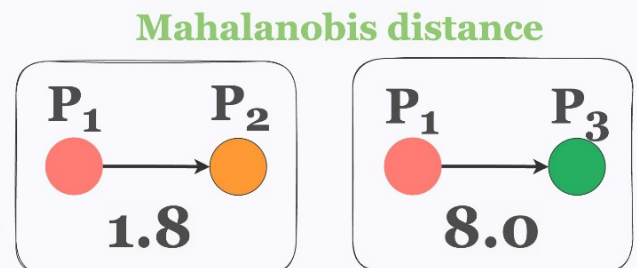
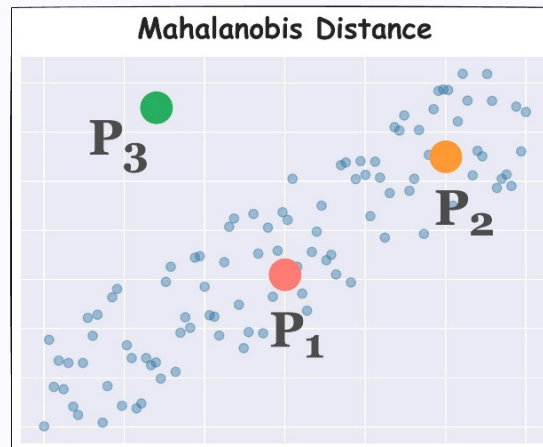
Non-linear classifiers

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}.$$

S=Covariance matrix



✗ P₂ and P₃ are equidistant to P₁



✓ P₂ is closer to P₁ than P₃

Non-linear classifiers

3. Choice of k

- **Small k** (e.g., $k = 1$):
 - Very flexible.
 - Low bias but high variance (sensitive to noise).
- **Large k :**
 - Smoother decision boundary.
 - Higher bias but lower variance.
- **Rule of thumb:** choose $k \approx \sqrt{n}$, then tune by cross-validation.

4. Curse of Dimensionality

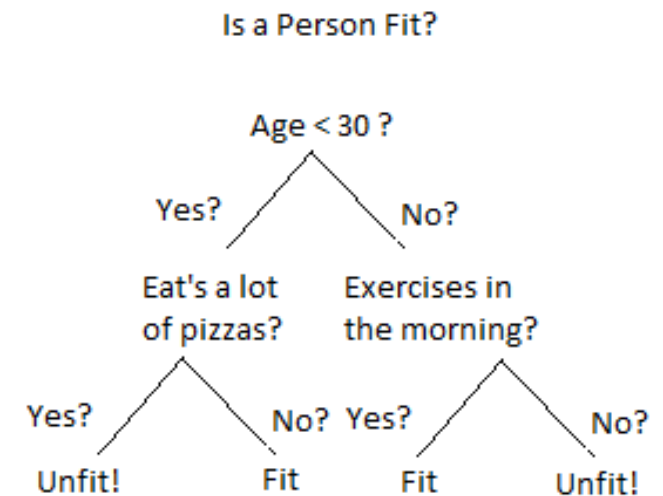
- In high-dimensional spaces (e.g., thousands of gene expression features):
 - Distances between points become less meaningful.
 - All samples tend to be “equally far apart”.
 - kNN performance deteriorates.
- **Typical fix in bioinformatics:** apply feature selection or dimensionality reduction (PCA, autoencoders) before using kNN.

Non-linear classifiers

Classification tree

1. Basic Idea

- A **tree-structured classifier**.
- Each **internal node**: a decision rule on a feature (e.g., "Gene X > 2.5?").
- Each **branch**: outcome of the rule.
- Each **leaf**: a class label (or probability distribution over classes).
- Classification = follow the path from root to leaf.



2. Splitting Criteria

At each node, we must decide **which feature and threshold** best split the data.

- **Gini Impurity** (used in CART):

$$G = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2$$

where p_k = proportion of samples of class k in the node.

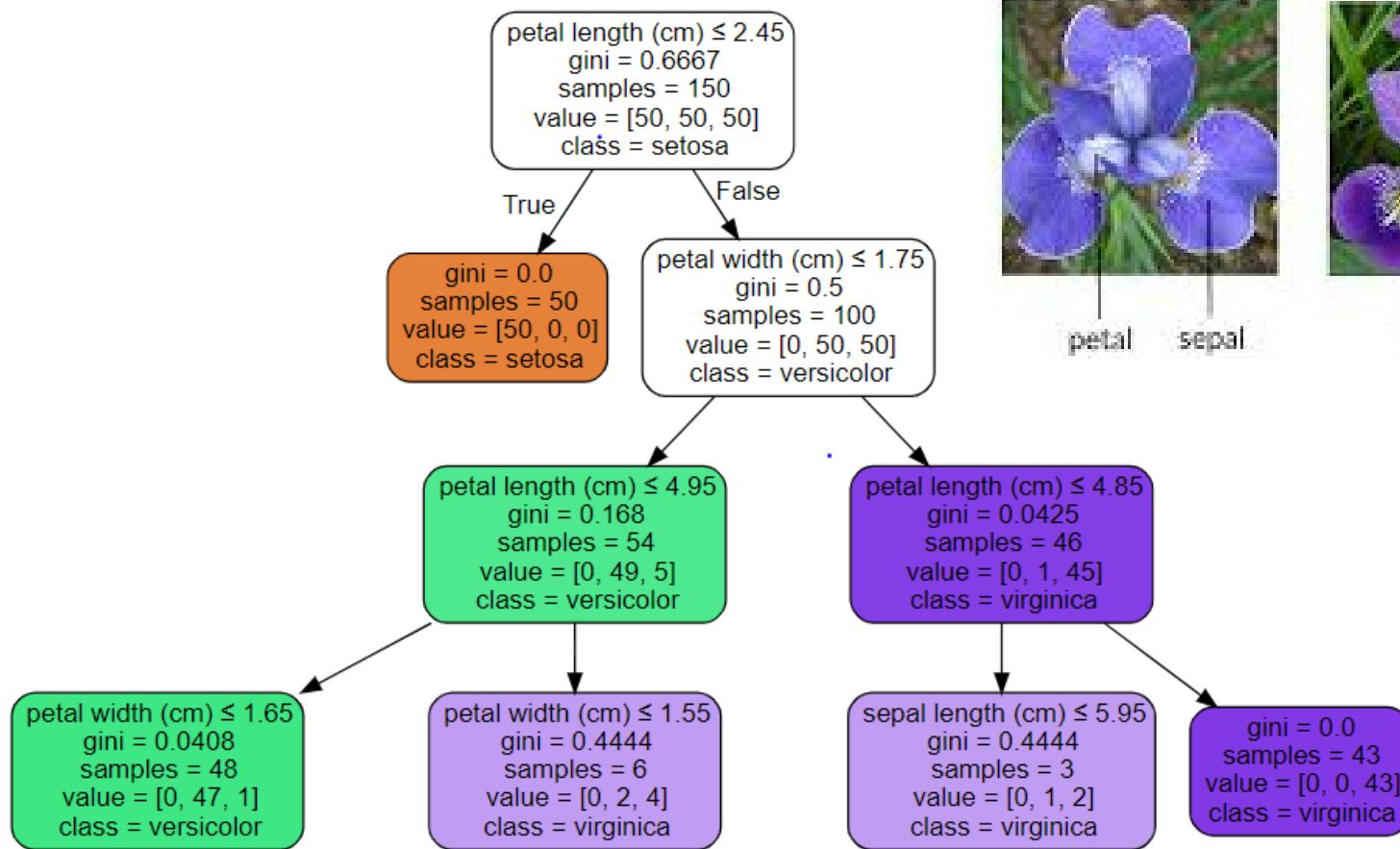
- Low G = node is "pure" (most samples in one class).
- **Entropy / Information Gain** (used in ID3, C4.5):

$$H = - \sum_{k=1}^K p_k \log_2(p_k)$$

- Splits chosen to **maximize information gain**: reduction in entropy.

👉 Both aim to create "pure" nodes where samples belong mostly to one class.

Non-linear classifiers



iris setosa



petal sepal

iris versicolor



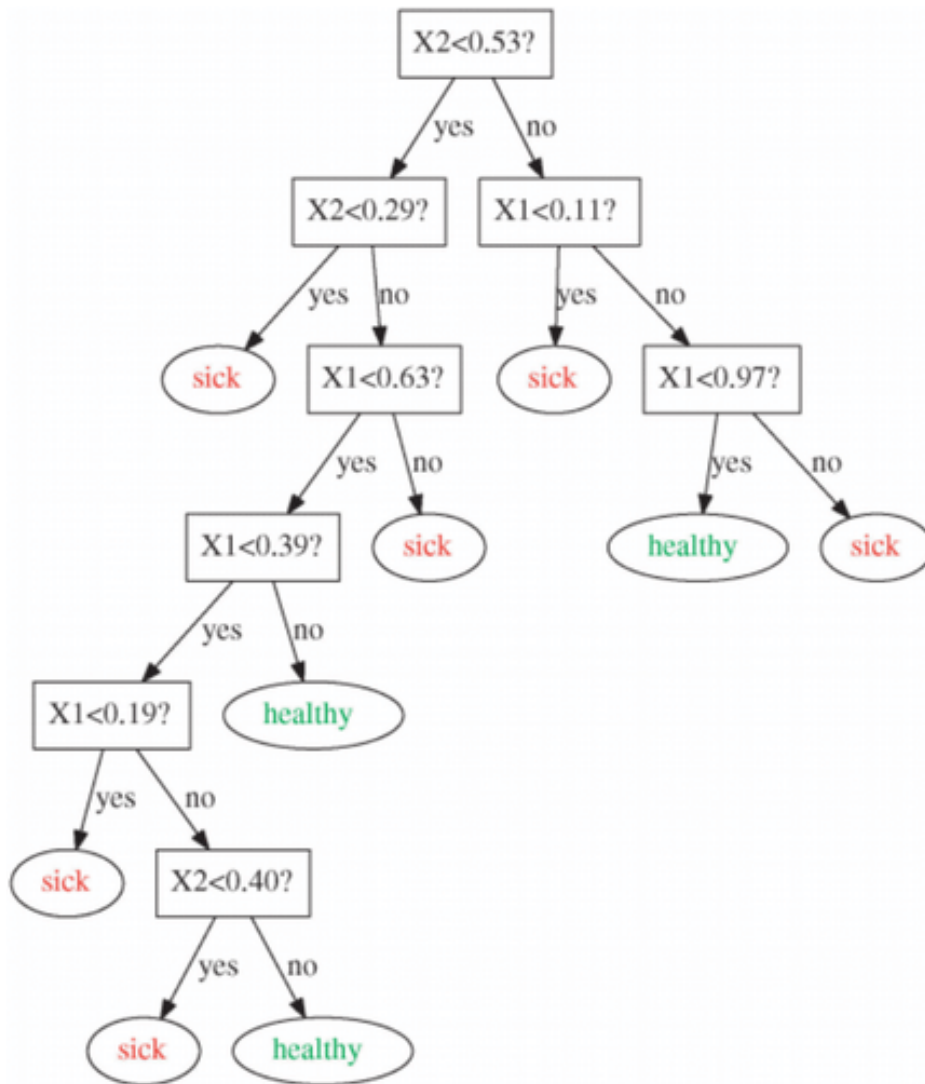
petal sepal

iris virginica

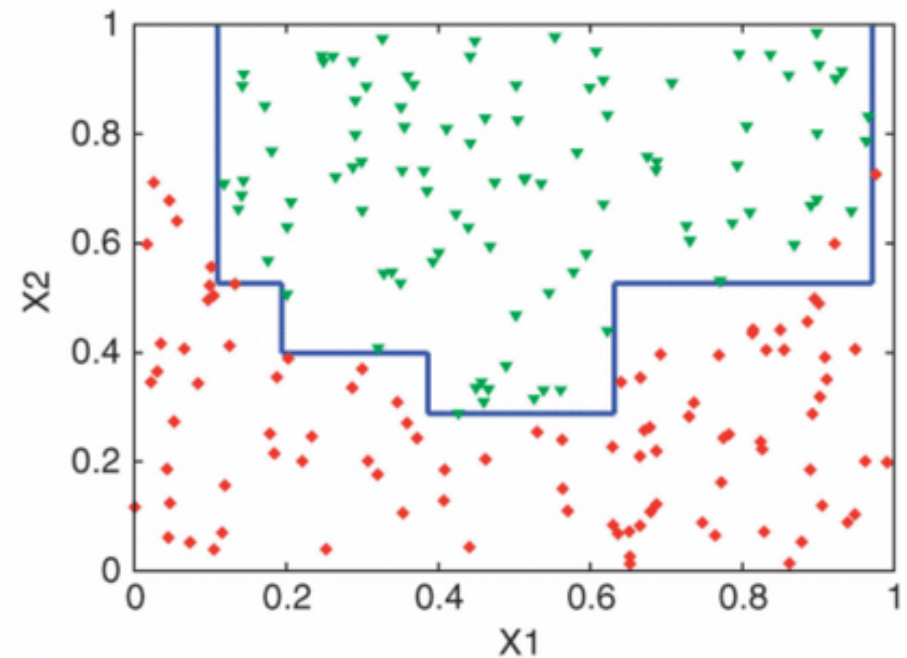


petal sepal

Non-linear classifiers



Boundaries



Non-linear classifiers

3. Interpretability

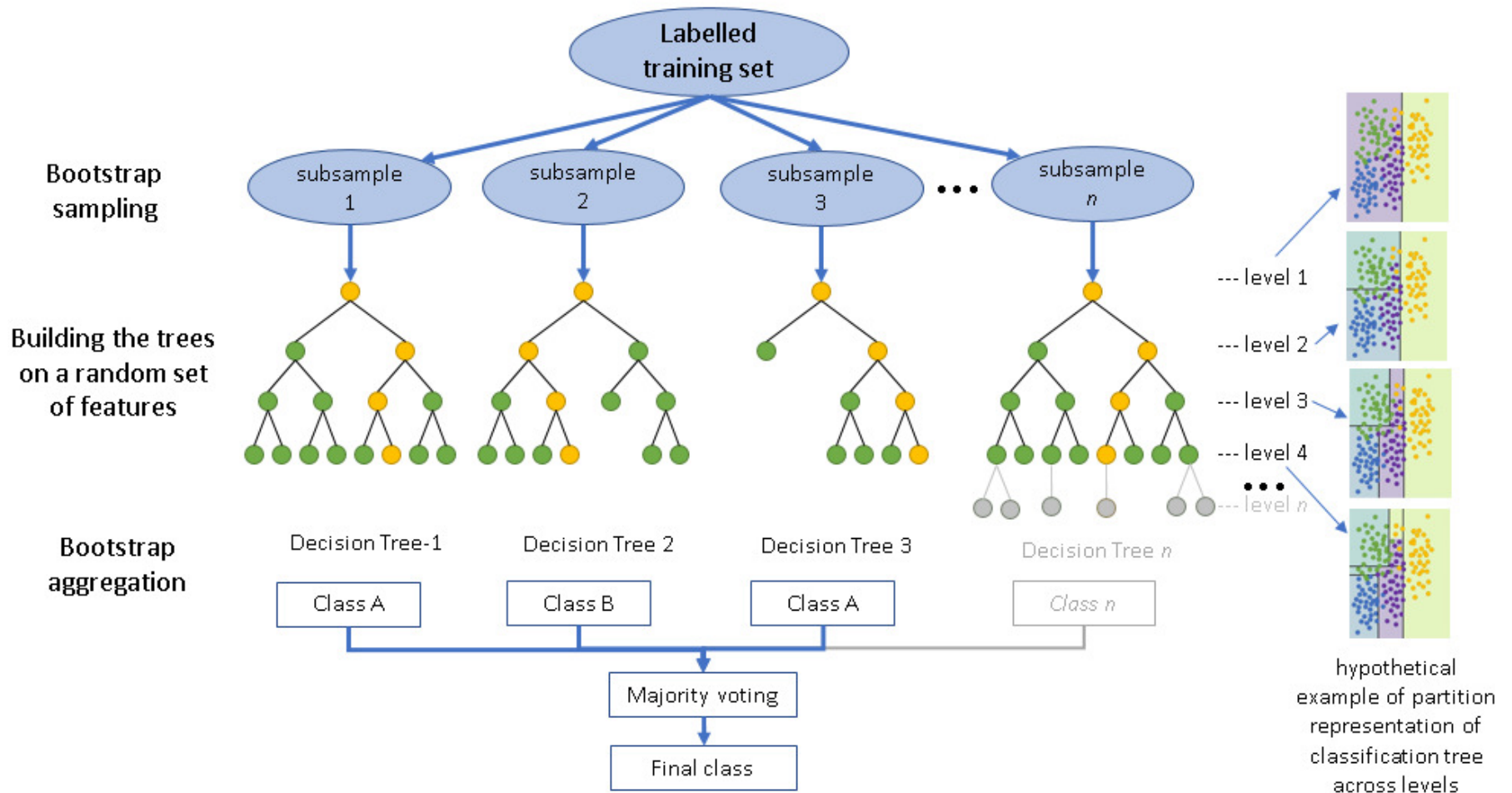
- Trees are **easy to interpret**:
 - Each path is a set of if-then rules.
 - Example in bioinformatics:
 - If “gene A expression > 5.2” and “protein B absent” → predict cancer subtype 1.
- Very useful when **explainability** is important (e.g., clinical decision support).

4. Overfitting and Pruning

- **Overfitting**:
 - If we grow the tree fully, it can memorize the training data.
 - Result: very low training error, poor generalization.
- **Pruning**:
 - Cut back branches that do not improve generalization.
 - Two approaches:
 - **Pre-pruning**: stop splitting when node has too few samples or impurity reduction is small.
 - **Post-pruning**: grow full tree, then remove weak branches using validation error.
- **Result**: simpler tree, better generalization.

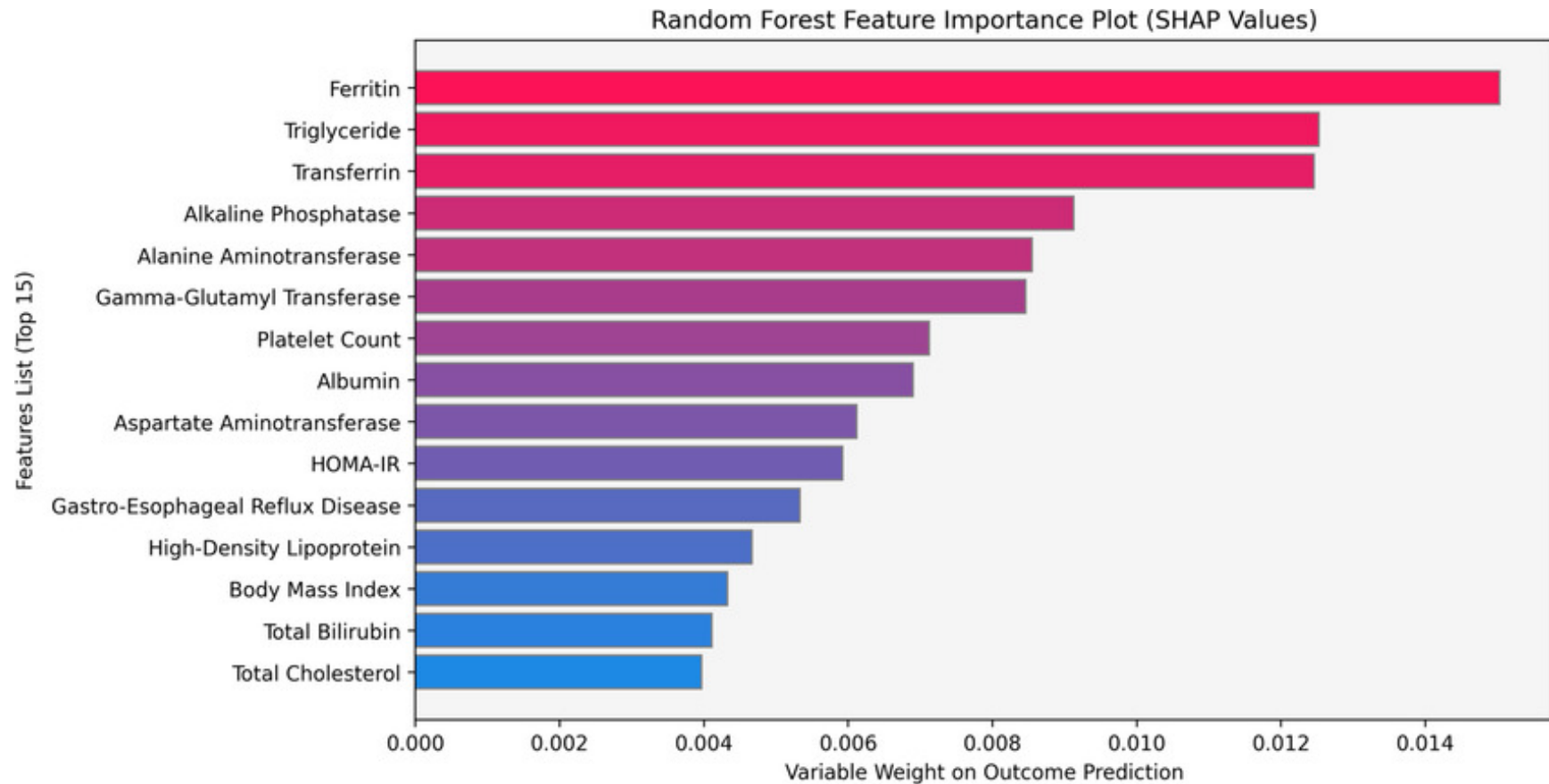
Non-linear classifiers

Random forests



Non-linear classifiers

Feature importance



Non-linear classifiers

1. Impurity-Based Importance (a.k.a. Gini Importance)

- At each split in a tree, we measure how much the chosen feature **reduces impurity** (e.g., Gini index or entropy).
- For a split s on feature j :

$$\Delta I(s, j) = I(\text{parent}) - \left(\frac{n_L}{n} I(\text{left}) + \frac{n_R}{n} I(\text{right}) \right)$$

- $I(\cdot)$: impurity (Gini or entropy).
- n_L, n_R : samples in left/right child nodes.
- n : samples in parent node.
- Importance of feature j :

$$FI(j) = \sum_{\text{splits on } j} \Delta I(s, j)$$

- Averaged across all trees in the forest.

✓ **Intuition:** a feature is important if it is often chosen to split, and those splits greatly reduce impurity.

Non-linear classifiers

2. Permutation Importance

- Alternative, model-agnostic method.
- Steps:
 1. Compute accuracy of the trained forest on the test set.
 2. Randomly **permute feature j** across samples (break its relationship with the target).
 3. Recompute accuracy.
 4. Importance = drop in accuracy.
- If accuracy drops a lot → feature was important.
- If accuracy barely changes → feature was not important.

✓ **Intuition:** permuting an important feature destroys predictive structure.

Non-linear classifiers

3. Shapley values

For a model f , feature set N , and feature i :

$$\phi_i(f) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! (|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)]$$

- S : subset of features not including i .
- $f(S)$: model prediction using only features in S .
- Weight: fraction of permutations where S precedes i .

✓ Intuition: contribution of i = improvement in prediction when adding i , averaged over all possible feature orders.

Key Properties (Why It's "Fair")

Efficiency: total contribution = difference between prediction and baseline.

Symmetry: if two features contribute equally, they get equal Shapley values.

Dummy: a feature that contributes nothing always has Shapley = 0.

Additivity: values can be added across models.

Non-linear classifiers

3. Shapley values

Shapley values explain **individual predictions**, not just global feature importance.

For a sample x :

$$f(x) = f(\text{baseline}) + \sum_i \phi_i(x)$$

- Baseline = average prediction.
- Each $\phi_i(x)$ = how much feature i pushed the prediction up or down.

Advantages

Theoretically grounded (game theory).

Works for any model (tree, linear, deep learning).

Provides **local** explanations (per sample) and **global** importance (average across samples).

Disadvantages

Computationally expensive: need to consider many feature subsets (2^p).

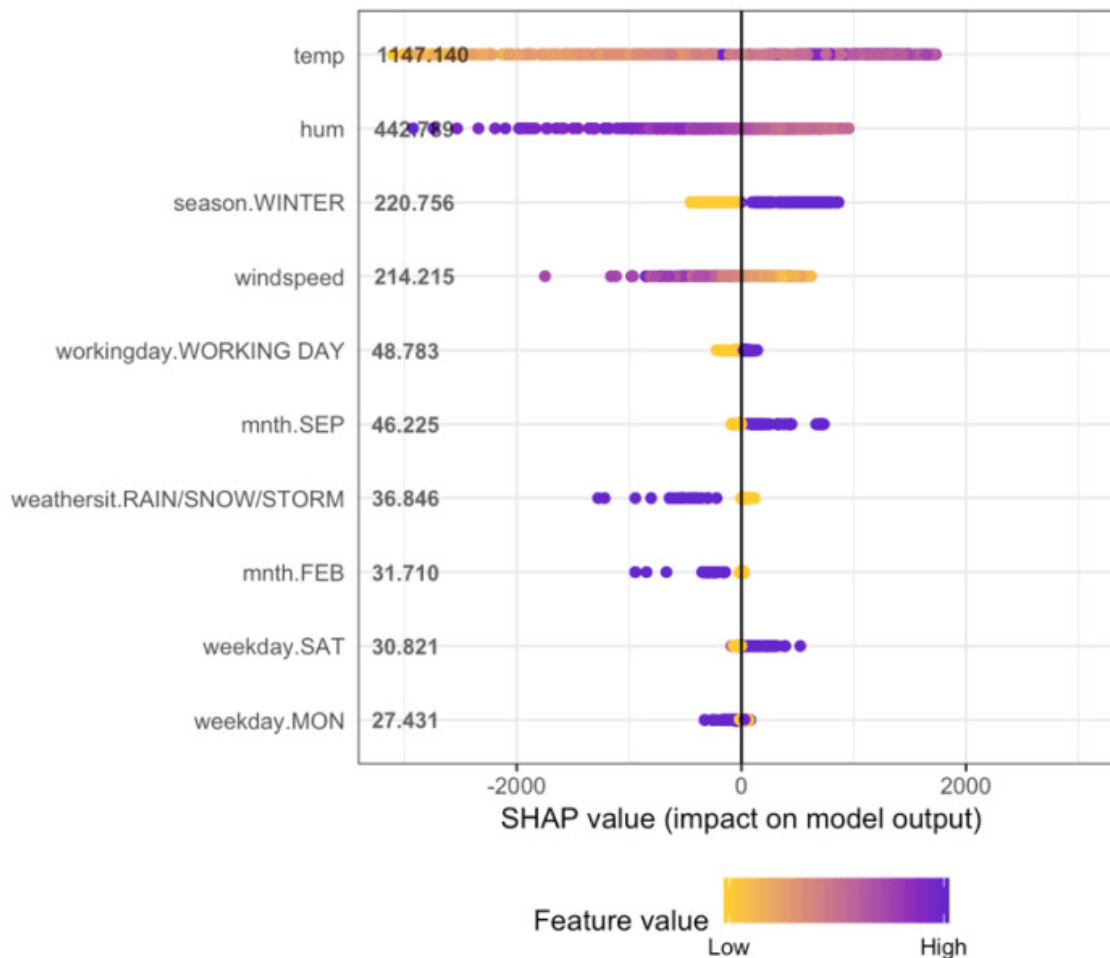
Approximations (e.g., **SHAP** for tree ensembles) are used in practice.

Applications in Bioinformatics

- Identifying **which genes** drive a disease prediction in a given patient.
- Explaining **why a sample** was classified as resistant vs. sensitive to a drug.
- Providing interpretable explanations for complex ML models in biomedical contexts.

Non-linear classifiers

3. Shapley values



How to interpret the shap summary plot?

- The y-axis indicates the variable name, in order of importance from top to bottom. The value next to them is the mean SHAP value.
- On the x-axis is the SHAP value. Indicates how much is the change in log-odds. From this number we can extract the probability of success.
- Gradient color indicates the original value for that variable. In booleans, it will take two colors, but in number it can contain the whole spectrum.
- Each point represents a row from the original dataset.

Non-linear classifiers

Naïve Bayes

1. Basic Idea

- A **probabilistic classifier** based on **Bayes' theorem**:

$$P(y|x) = \frac{P(x|y) P(y)}{P(x)}$$

- Goal: choose the class y that maximizes the posterior probability $P(y|x)$.
- **"Naïve" assumption**: features are conditionally independent given the class.

2. Mathematical Formulation

For input vector $x = (x_1, x_2, \dots, x_p)$:

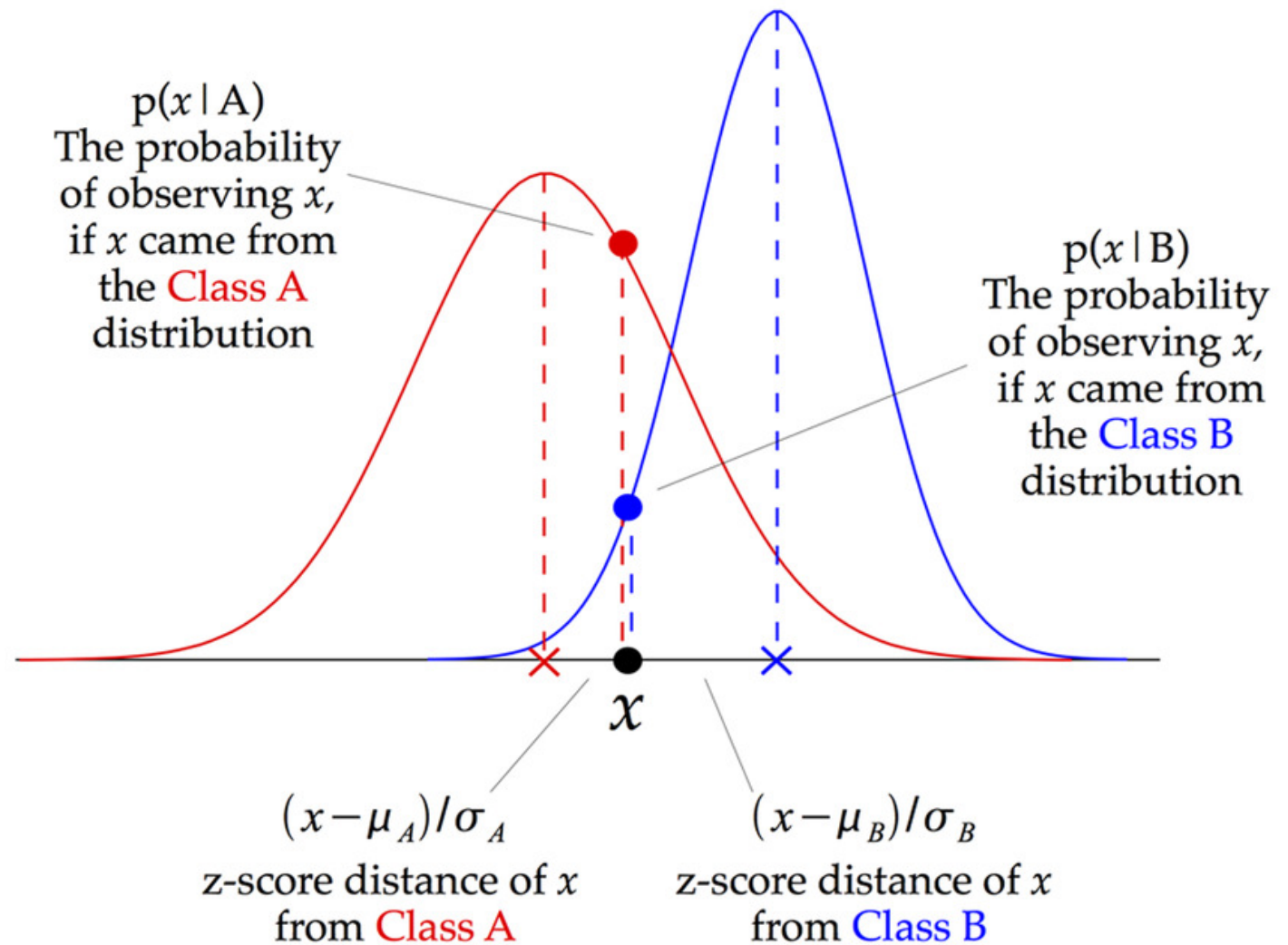
$$P(y|x) \propto P(y) \prod_{j=1}^p P(x_j|y)$$

- Prior: $P(y)$ = class probability.
- Likelihood: $P(x_j|y)$ estimated from training data.
- Posterior: proportional to prior \times likelihoods.

👉 Classification rule:

$$\hat{y} = \arg \max_y P(y) \prod_{j=1}^p P(x_j|y)$$

Non-linear classifiers



Gaussian Naïve Bayes

Non-linear classifiers

3. Types of Naïve Bayes

- **Gaussian NB:** assumes continuous features x_j are Gaussian.
- **Multinomial NB:** common for text (word counts, k-mers in genomics).
- **Bernoulli NB:** for binary features (presence/absence of mutation, motif).

4. Why It Works Despite “Naïve” Assumption

- Independence rarely holds in real data (genes are highly correlated!).
- But the model often still performs well in practice, especially when the number of features is very large.
- The independence assumption makes estimation feasible even with small sample sizes.

Strengths and Weaknesses

✓ Strengths

- Very fast to train and predict.
- Works well with **high-dimensional data** (common in genomics).
- Requires few training samples.

✗ Weaknesses

- Independence assumption often violated.
- Probabilities may be poorly calibrated (outputs can be overconfident).
- Less flexible than more complex classifiers.

Ensemble methods

1. Basic Idea

- Instead of relying on a single classifier, **combine many “weak learners”** to form a **stronger classifier**.
- Principle: *“Wisdom of the crowd”* → multiple models, when combined, usually generalize better than one model.

Advantages of Ensembles

Higher accuracy than single models.

Reduce overfitting (especially bagging/random forests).

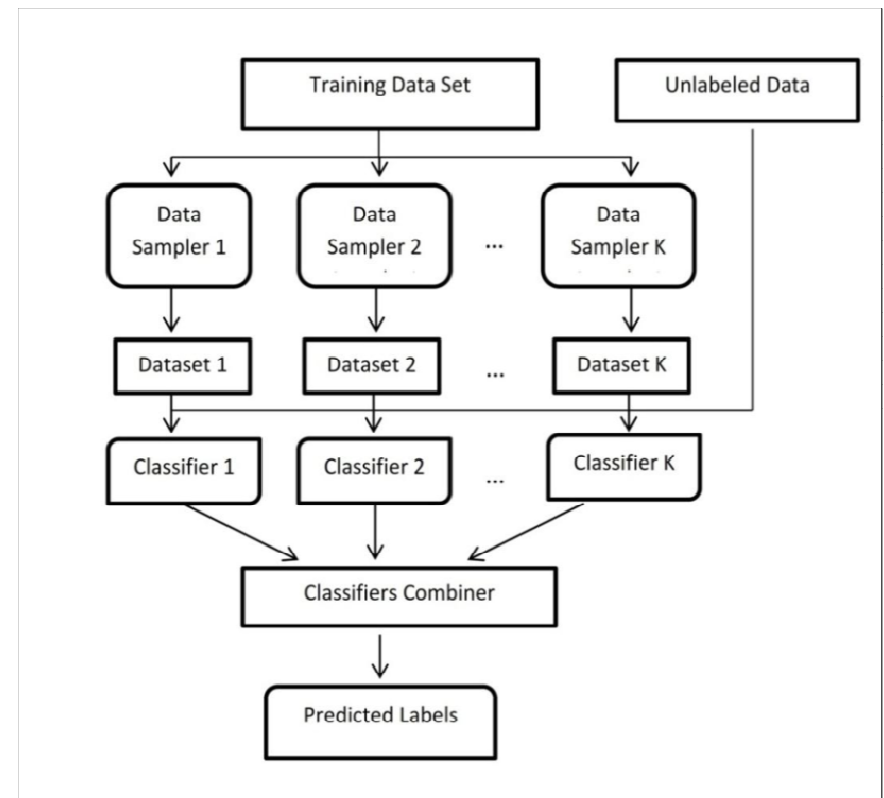
Handle complex data distributions.

Disadvantages

Less interpretable (especially boosting and stacking).

Computationally more expensive.

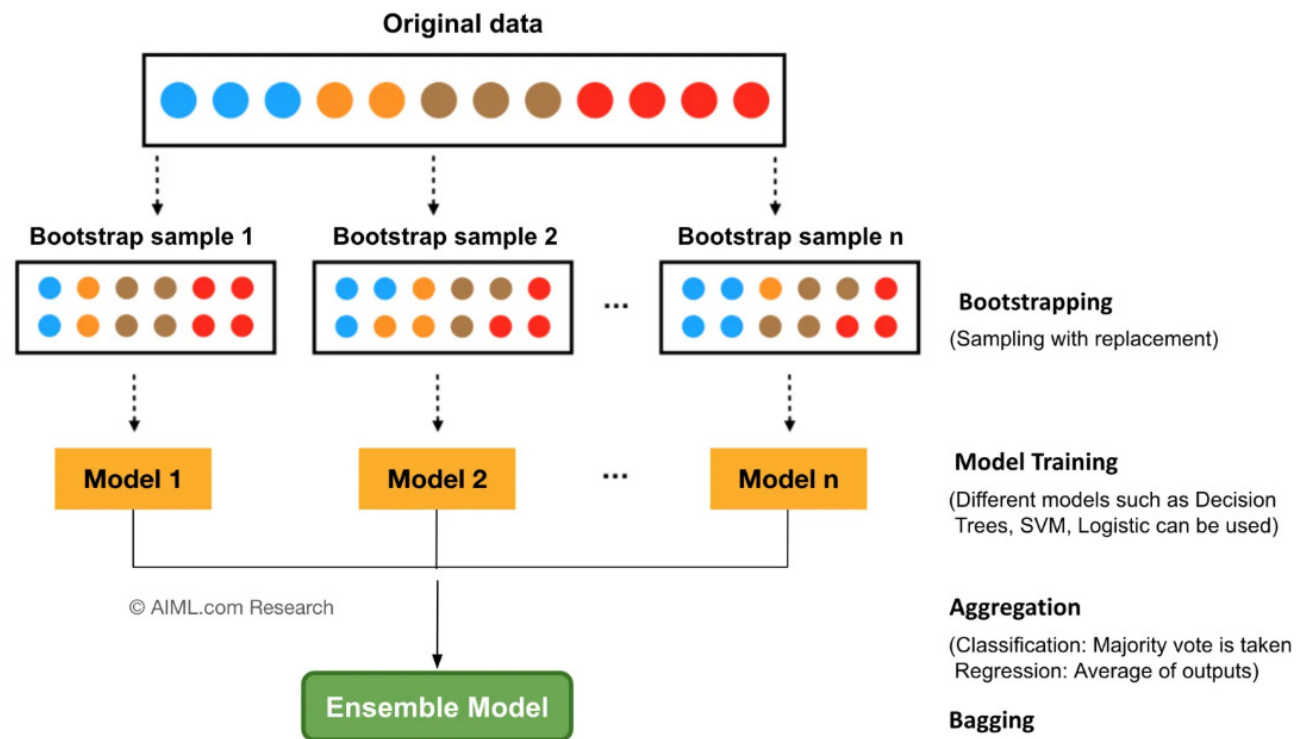
Risk of overfitting if not tuned (especially boosting).



Ensemble methods

(a) Bagging (Bootstrap Aggregating)

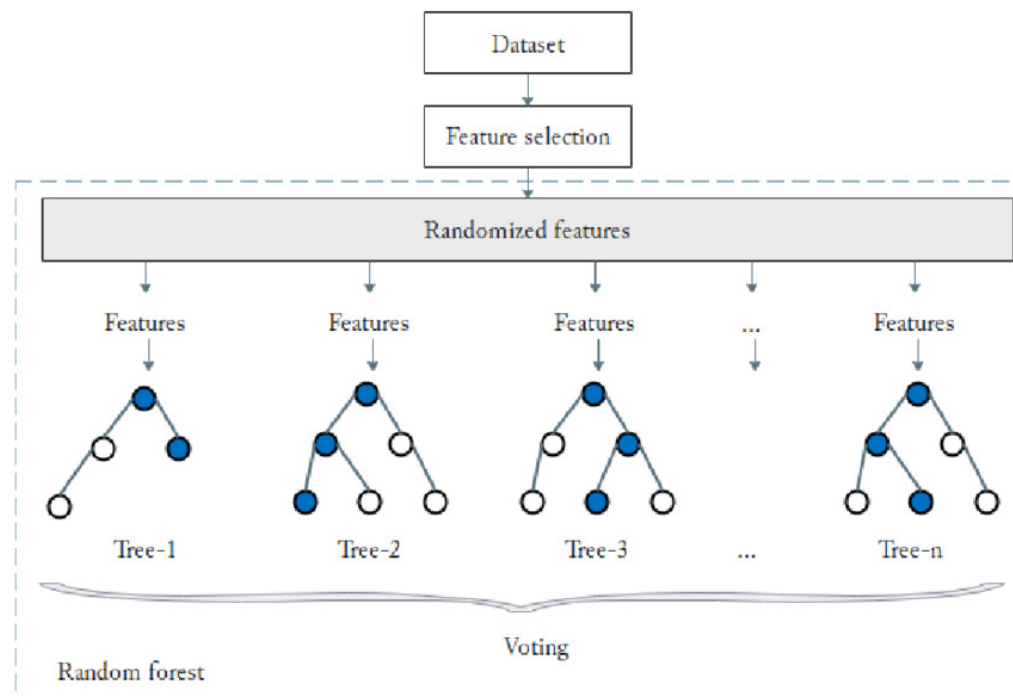
- Train multiple models on **bootstrapped samples** of the training set.
- Aggregate predictions by majority vote (classification) or averaging (regression).
- Reduces variance, stabilizes unstable learners (like decision trees).
- ☒ Example: **Random Forests**.



Ensemble methods

(b) Random Forests

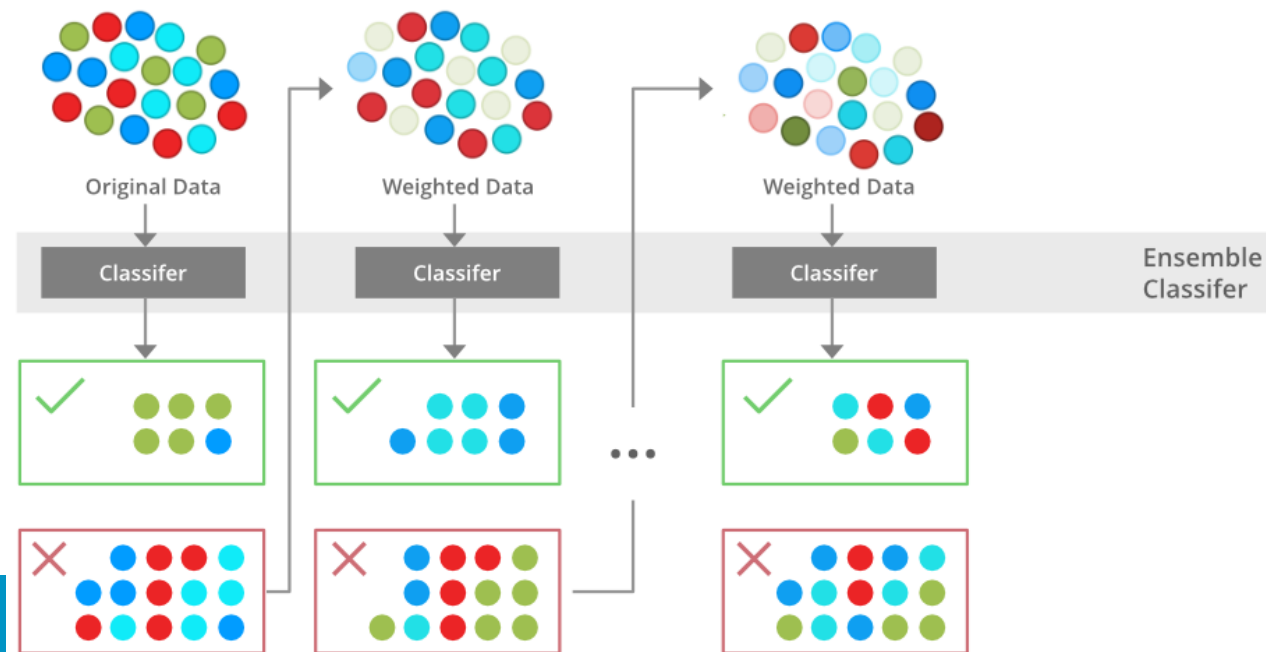
- Extension of bagging applied to decision trees.
- At each split:
 - Select a **random subset of features** instead of all features.
- Decorrelates trees → more diversity → stronger ensemble.
- Provides **feature importance** measures.
- Widely used in bioinformatics (gene expression classification, biomarker discovery).



Ensemble methods

(c) Boosting

- Sequentially train weak learners (often shallow trees).
- Each new learner focuses on **samples misclassified** by previous ones.
- Final prediction: weighted vote of all learners.
- Reduces bias and variance, often very accurate.
- Key algorithms:
 - **AdaBoost** (Adaptive Boosting).
 - **Gradient Boosting Machines (GBM)**.
 - **XGBoost, LightGBM, CatBoost** (modern scalable implementations).



Ensemble methods

AdaBoost

1. AdaBoost (Adaptive Boosting)

- **Core idea:** build an ensemble of *weak learners* (often shallow decision stumps).
- **Training process:**
 - Start with equal weights for all training samples.
 - Train a weak learner → compute its error.
 - Increase weights of misclassified samples (make them “harder to ignore”).
 - Next learner focuses more on previously misclassified data.
- **Final model:** weighted vote of all weak learners.
- **Loss function:** based on **exponential loss**.

Exponential Loss (AdaBoost)

$$L(y, f(x)) = \exp(-yf(x)), \quad y \in \{-1, +1\}$$

- Penalizes misclassified points **exponentially**.
- Correct predictions with large margin → very small loss.
- Wrong predictions → rapidly growing loss.
- This is why AdaBoost **focuses on misclassified samples** in later iterations.

Ensemble methods

XGBoost

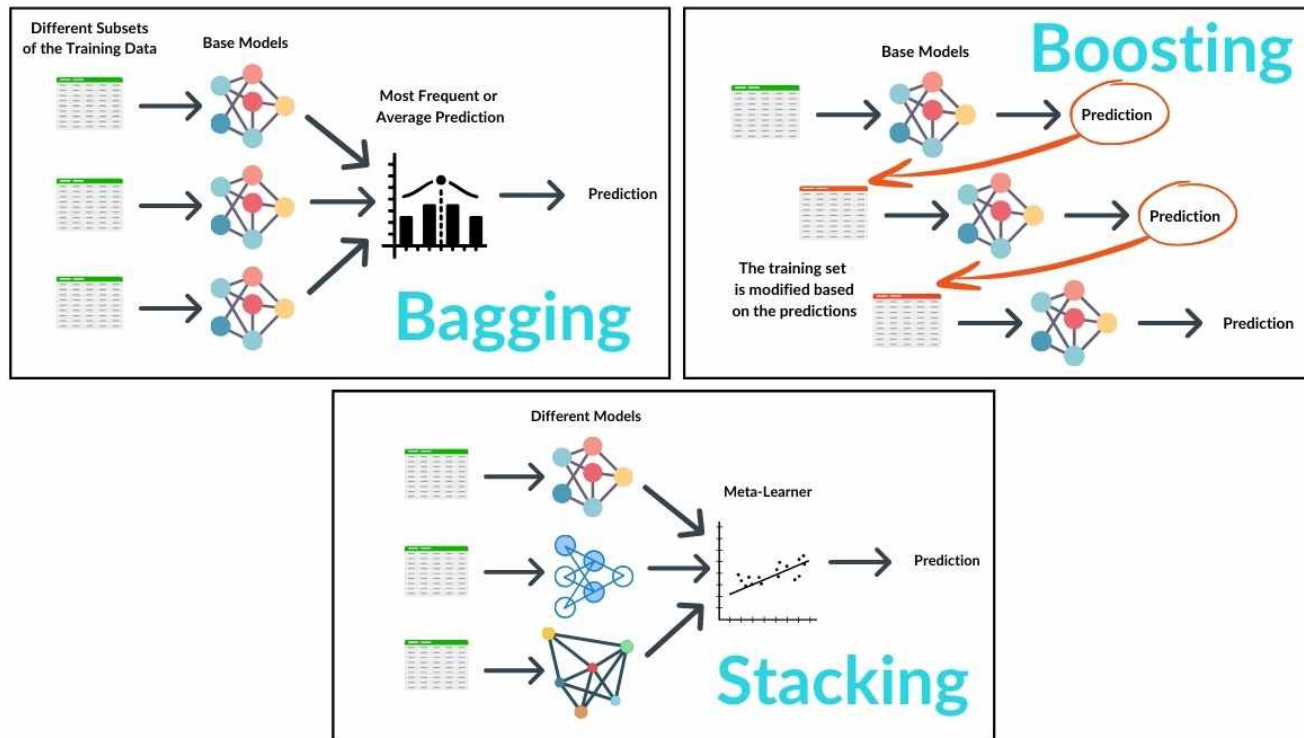
- A modern, highly optimized implementation of **gradient boosting**.
- **Key differences from AdaBoost:**

| Aspect | AdaBoost | XGBoost |
|-------------------------|---|--|
| Optimization principle | Reweights samples → minimizes exponential loss implicitly. | Directly optimizes a differentiable loss (e.g., logistic loss, squared error) via gradient descent. |
| Weak learner | Usually decision stumps (depth = 1). | Uses deeper CART trees (depth 4–10 common). |
| Regularization | No explicit regularization → risk of overfitting. | Built-in L1/L2 regularization on weights to control complexity. |
| Handling missing values | Not explicitly handled. | Handles missing data automatically by learning default split directions. |
| Computation | Sequential, slower for large data. | Highly optimized (parallelization, sparsity-aware, cache-efficient, GPU support). |
| Interpretability | Simple, since stumps are used. | More complex, but still provides feature importance and SHAP values . |

Ensemble methods

(d) Stacking (Stacked Generalization)

- Train **different types of classifiers** (e.g., SVM, logistic regression, trees).
- Combine their outputs using a **meta-learner** (often logistic regression).
- Flexible — can exploit strengths of different models.
- More complex, but often achieves state-of-the-art results.



Challenges in Bioinformatics

1. High-dimensionality vs. Small Sample Size ($p \gg n$)

- In many omics datasets:
 - p = number of features (e.g., genes, SNPs, proteins) can be **thousands or millions**.
 - n = number of samples (patients, experiments) is often **dozens or hundreds**.
- Consequences:
 - Models can easily **overfit** — memorize the training data but fail to generalize.
 - Distances (in kNN) and covariance estimates (in LDA/QDA) become unreliable.
- Solutions:
 - **Feature selection** (filters, wrappers, embedded methods).
 - **Dimensionality reduction** (PCA, autoencoders).
 - **Regularization** (LASSO, ridge regression).

👉 Example: Gene expression data with 20,000 genes but only 100 patients.

Challenges in Bioinformatics

2. Noisy Data

- Bioinformatics data are prone to **measurement errors**:
 - Sequencing artifacts, dropouts in single-cell RNA-seq.
 - Variability between labs, instruments, or protocols.
- Consequences:
 - Noise can mask true biological signals.
 - Classifiers may pick up **spurious correlations**.
- Solutions:
 - Careful **quality control** and **data cleaning**.
 - **Robust classifiers** (Random Forests, SVMs with soft margins).
 - **Cross-validation** to ensure stability of results.

👉 Example: Technical noise in RNA-seq can look like differential expression.

Challenges in Bioinformatics

3. Class Imbalance

- Rare diseases, rare mutations → **one class is much smaller than others.**
- Consequences:
 - Classifier biased toward the majority class.
 - Accuracy misleading (e.g., 95% accuracy by always predicting “healthy”).
- Solutions:
 - **Resampling:** oversample minority class (SMOTE), undersample majority.
 - **Cost-sensitive learning:** higher penalty for misclassifying minority.
 - Use **metrics beyond accuracy:** precision, recall, F1, ROC-AUC.

👉 Example: In a cancer study, only 5% of patients respond to a given drug.

Challenges in Bioinformatics

4. Interpretability vs. Accuracy Trade-off

- **Simple models** (LDA, logistic regression, decision trees):
 - Easy to interpret, but may miss complex non-linear patterns.
- **Complex models** (Random Forests, Boosting, SVM with RBF):
 - Higher accuracy, but often “black-box” models.
- Challenge in bioinformatics:
 - Clinicians and biologists often **require interpretability** to trust predictions.
 - Sometimes a slightly less accurate but interpretable model is more useful.
- Solutions:
 - Use **explainable AI tools**: feature importance, SHAP values, LIME.
 - Balance accuracy with interpretability depending on application.

👉 Example: Logistic regression provides clear odds ratios for biomarkers, while XGBoost may classify better but is harder to interpret directly.

Contents

- Overview
- Data preparation
- Linear classifiers
 - LDA, QDA, Logistic regression
- Non-linear
 - SVM, kNN, Tree, Random Forest, Naïve Bayes
- Ensemble methods
- Challenges in Bioinformatics



CEU

*Universidad
San Pablo*

Lesson 7. Dimensionality reduction

Medicin School

Contents

- Overview
- Linear dimensionality reduction
 - PCA
- Non-linear dimensionality reduction
 - Manifolds, t-SNE, UMAP

Overview

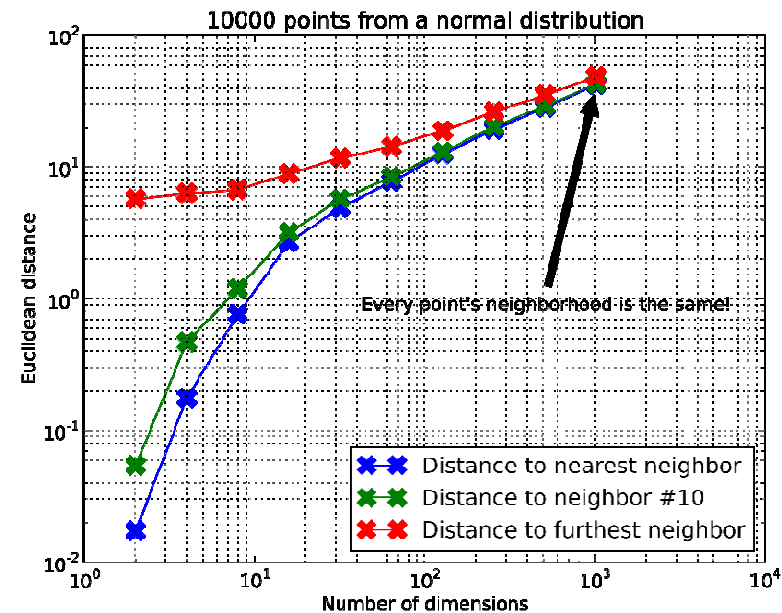
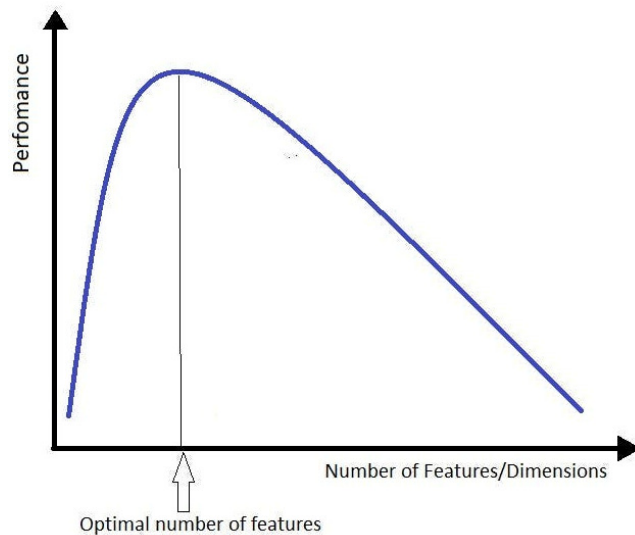
The Curse of Dimensionality

In bioinformatics we routinely deal with **very high-dimensional data**:

- Gene expression microarrays or RNA-seq: tens of thousands of genes per sample.
- SNP arrays: millions of genetic variants per individual.
- Proteomics and metabolomics: thousands of features, often with missing values.

As the number of features (dimensions) grows:

- Data points become **sparse** in high-dimensional space → similarity measures (e.g., Euclidean distance) lose discriminative power.
- Volume grows exponentially → we need exponentially more samples to cover the space.
- Many dimensions may contain **noise** rather than signal.



Overview

Problems with High-dimensional Data

Computational cost

- Algorithms scale poorly with the number of features (e.g., covariance matrix computation in PCA, training deep models).
- Storage and memory requirements explode for omics datasets.

Overfitting in predictive models

- With many more features than samples ($p \gg n$ scenario, typical in biology), models can fit noise rather than signal.
- Leads to poor generalization to new data (e.g., biomarkers that “work” only in the discovery cohort).

Difficulty in visualization and interpretation

- Humans can visualize only 2D or 3D.
- Without dimensionality reduction, it's impossible to see structure such as clusters of cell types or disease subgroups.

Overview

Benefits of Dimensionality Reduction

Data compression

- Represent thousands of genes with a handful of principal components.
- Useful for downstream storage, faster computation, and efficient search.

Noise reduction

- By projecting onto the dominant axes of variation, we filter out measurement noise and batch effects.
- Especially valuable in high-throughput experiments where technical noise is common.

Feature extraction and biological insight

- Latent dimensions can correspond to **biological processes** (e.g., immune activation, cell cycle, metabolic states).
- Helps to identify biomarkers, stratify patients, or reveal hidden structure in the data.

Overview

| Aspect | Feature Selection | Feature Extraction |
|---------------------------|--|---|
| Definition | Keep a subset of original features | Create new transformed features |
| Interpretability | High (original biological features preserved) | Often low (latent variables, PCs hard to interpret) |
| Computation | Simpler, cheaper | Sometimes expensive (matrix decomposition, neural nets) |
| Risk | May ignore feature interactions | May obscure biological meaning |
| Example in Bioinformatics | Selecting 50 most variable genes in cancer study ↑ We saw this in the previous lecture | Using PCA to reduce 20,000 genes to 20 components |

Overview

Feature extraction

- **Idea:** Instead of picking features from the original set, we construct *new* features that capture the essence of the data.
- **Goal:** Represent the data in a lower-dimensional space while retaining most of its variance or structure.

Linear methods

- **Principal Component Analysis (PCA):** new axes that maximize variance.
- **Linear Discriminant Analysis (LDA):** new axes that maximize class separability.

Nonlinear methods (Manifold learning)

- **t-SNE / UMAP:** nonlinear embeddings that preserve local/global structure.
- **Autoencoders:** neural networks that learn compact latent representations.

Advantages

- Can uncover hidden patterns not obvious in the original features.
- Often very effective for visualization and compression.

Limitations

- New features (principal components, latent variables) are often not directly interpretable biologically.
- Risk of losing important signal if reduction is too aggressive.

Linear dimensionality reduction

Principal Components Analysis (PCA)

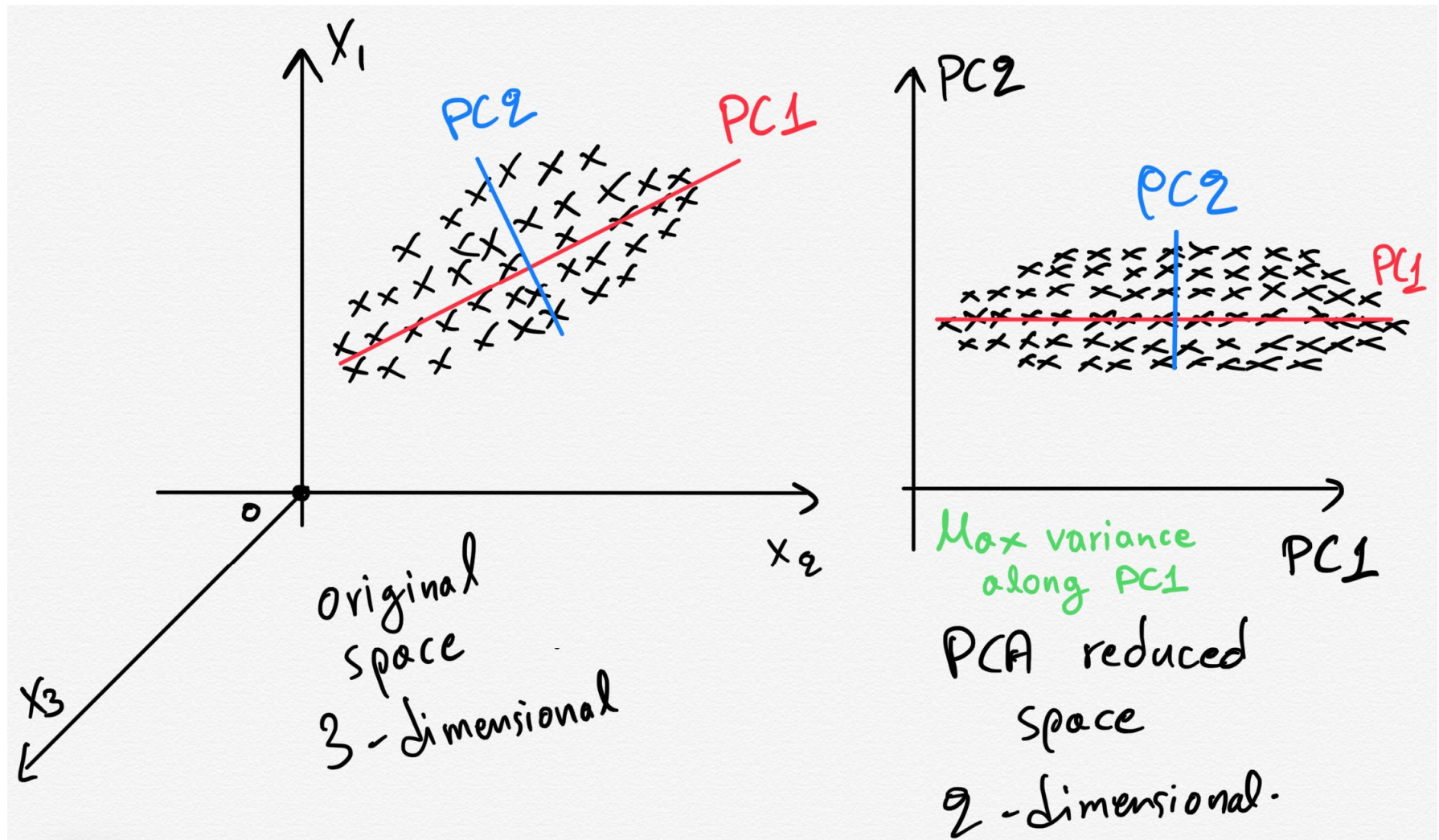
Intuition

- **Goal:** Find a new coordinate system (set of axes) in which:
 - The first axis captures the maximum variance in the data.
 - The second axis captures the maximum variance orthogonal to the first.
 - And so on.
- The new coordinates are called **principal components (PCs)**.
- By keeping only the first few PCs, we approximate the dataset in fewer dimensions while retaining most of the variability.

Geometric Interpretation


- Imagine gene expression profiles of thousands of genes plotted in a 20,000-dimensional space.
- PCA finds the directions in that space where samples differ the most.
- Each sample is then projected onto those new axes (principal components).
- Instead of describing a patient by 20,000 genes, we can describe them by a handful of PCs (e.g., PC1, PC2).

Linear dimensionality reduction



Linear dimensionality reduction

Given a dataset with n samples and p features, represented as a matrix

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}, \quad X \in \mathbb{R}^{n \times p}.$$


Each row corresponds to a sample (e.g., a patient or a cell), and each column to a feature (e.g., gene expression level).

Step 1: Compute the mean of each feature

For feature j (column j of X):

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}.$$

This is the **average expression** of gene j across all samples.

Linear dimensionality reduction

Step 2: Center the data

We subtract the mean from each entry in that column:

$$\tilde{x}_{ij} = x_{ij} - \mu_j.$$

The **centered data matrix** is

$$\tilde{X} = X - \mathbf{1}_n \mu^\top,$$

where $\mathbf{1}_n$ is a column vector of ones of length n , and μ is the vector of feature means.

Result: Each feature now has **zero mean**, which ensures that PCA captures **variance** and not absolute offsets.

Step 3: Compute the covariance matrix

Using the centered matrix \tilde{X} :

$$\Sigma = \frac{1}{n-1} \tilde{X}^\top \tilde{X}.$$

- $\Sigma \in \mathbb{R}^{p \times p}$.
- Entry Σ_{jk} is the covariance between feature j and feature k .
- The diagonal entries Σ_{jj} are the variances of each feature.

Linear dimensionality reduction

Step 4: Eigenvalue Decomposition

We now solve the **eigenvalue problem** for the covariance matrix:

$$\Sigma v_i = \lambda_i v_i, \quad i = 1, \dots, p.$$

- $v_i \in \mathbb{R}^p$ are the **eigenvectors** (principal directions, or loadings).
- $\lambda_i \geq 0$ are the **eigenvalues**, corresponding to the variance explained by each principal component.

Since Σ is symmetric and positive semi-definite:

- All eigenvalues are real and non-negative.
- Eigenvectors form an orthogonal basis.

Step 5: Ordering Components

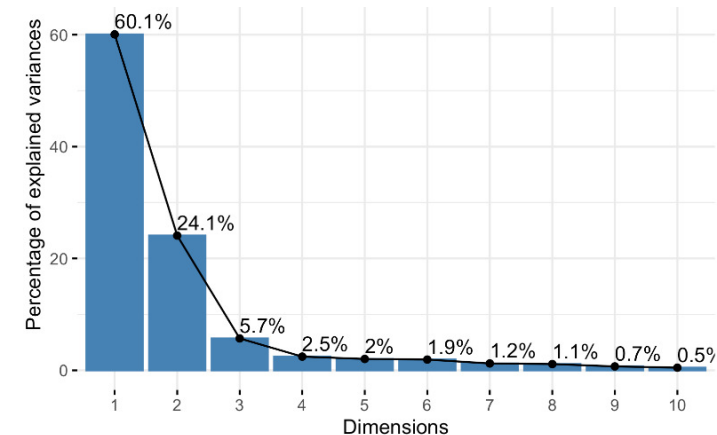
- Sort eigenvalues in descending order:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p.$$

- The first eigenvector v_1 corresponds to the direction of **maximum variance**.
- The fraction of variance explained by component i is:

$$\text{VarExpl}(i) = \frac{\lambda_i}{\sum_{j=1}^p \lambda_j}.$$

We often visualize this with a **scree plot** to decide how many components k to retain.



Linear dimensionality reduction

Step 6: Projection onto Principal Components

To transform the original data into the lower-dimensional space:

$$Z = \tilde{X}V_k,$$

where:

- $\tilde{X} \in \mathbb{R}^{n \times p}$ is the centered data.
- $V_k \in \mathbb{R}^{p \times k}$ contains the first k eigenvectors as columns.
- $Z \in \mathbb{R}^{n \times k}$ are the new coordinates (principal component scores).

Each row of Z represents one sample in the reduced k -dimensional space.

Step 7: Reconstruction (Optional)

If we keep only $k < p$ components, we can reconstruct an **approximation** of the original data:

$$\hat{X} = ZV_k^\top + \mathbf{1}_n\mu^\top,$$

where we add back the feature means.

- This reconstruction minimizes the squared error between the original data X and \hat{X} .
- In practice: dimension reduction is a trade-off between compression and accuracy.

Linear dimensionality reduction

Step 8: Practical Notes

- In practice, we often use **Singular Value Decomposition (SVD)** of \tilde{X} rather than explicitly computing Σ .

$$\tilde{X} = U\Sigma V^T$$

Here, columns of V are the eigenvectors, and the squared singular values correspond to eigenvalues.

- SVD is numerically stable and efficient when $n \ll p$ or $p \ll n$, which is common in bioinformatics (e.g., 100 samples \times 20,000 genes).

Non-linear dimensionality reduction

Manifolds

Motivation

Linear methods like PCA assume that the variability in the data can be well represented as a linear combination of features.

But in many biological datasets, the structure is **nonlinear**:

- Single-cell RNA-seq data often follows **trajectories** (e.g., cell differentiation paths).
- Protein conformations can lie on **curved manifolds** in high-dimensional structural space.
- Population genetics data may form clusters connected by nonlinear relationships (migration, admixture).

The idea of **manifold learning**:

- Data lies (approximately) on a low-dimensional nonlinear manifold embedded in a high-dimensional space.
- Nonlinear methods try to recover that manifold for visualization and analysis.

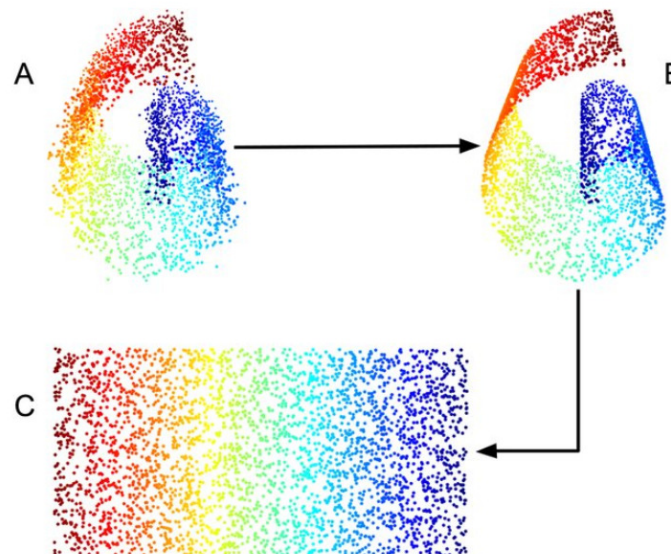
Non-linear dimensionality reduction

What is a Manifold? (Informal Introduction)

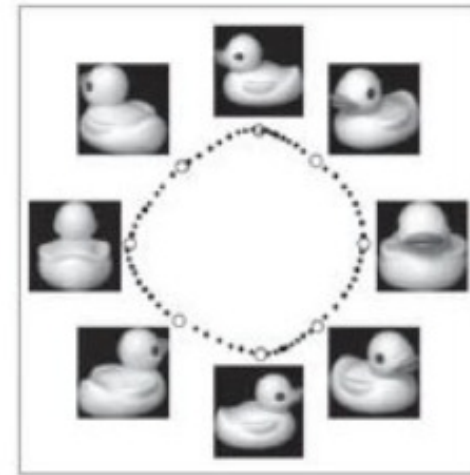
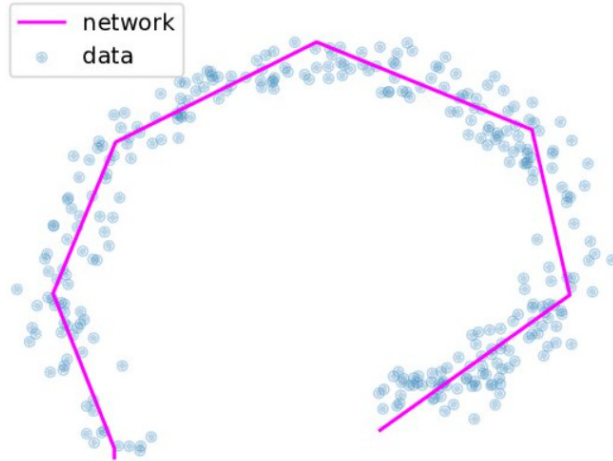
- A **manifold** is a space that may be **curved** and **complex** in high dimensions, but if you zoom in close enough, it looks **flat** like **ordinary Euclidean space**.

Everyday Examples

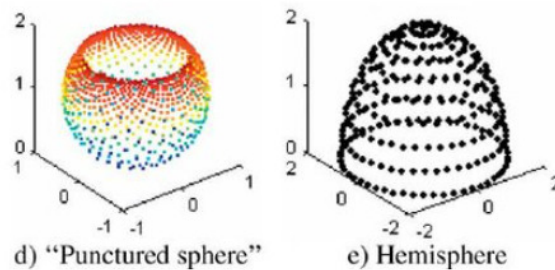
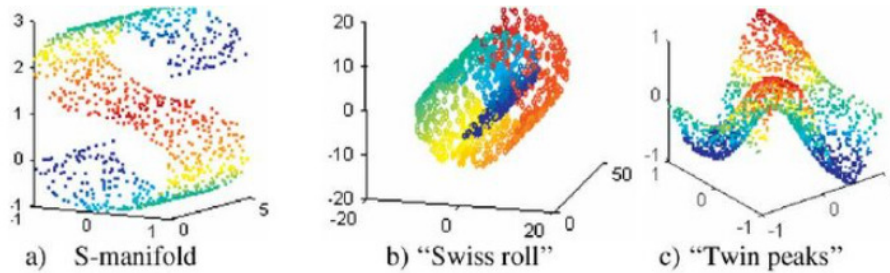
- The **surface of the Earth**:
 - Globally it's curved (a sphere).
 - Locally (in a small city or neighborhood), it looks flat and can be described with 2D coordinates.
- A **rolled sheet of paper** (Swiss roll):
 - Globally it's curled up in 3D.
 - Locally it's still a flat 2D sheet.



Non-linear dimensionality reduction



a) $k = 4$



Non-linear dimensionality reduction

t-SNE (t-distributed Stochastic Neighbor Embedding)

Intuition

- **Goal:** Map high-dimensional data (e.g., thousands of genes) into 2D or 3D while preserving **local neighborhoods**.
- Think: if two cells look similar in gene expression space, they should appear close together in the 2D plot.
- Unlike PCA (which preserves global variance), t-SNE focuses on **who is close to whom**.
- Widely used for **visualization of clusters** in single-cell RNA-seq, flow cytometry, and other high-dimensional bioinformatics data.

Formal Construction

1. High-dimensional similarities

- For two points $x_i, x_j \in \mathbb{R}^p$, define the probability that x_j is a neighbor of x_i :

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}.$$

- Here, σ_i is chosen so that the “effective number of neighbors” matches a user-defined **perplexity** parameter (\approx neighborhood size).
- The joint probability is symmetrized:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}.$$

Non-linear dimensionality reduction

2. Low-dimensional similarities

- Each point is mapped to $y_i \in \mathbb{R}^2$ (or \mathbb{R}^3).
- Define similarities with a **Student-t distribution** (heavy tails prevent crowding):

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

3. Optimization

- Find embeddings $\{y_i\}$ that minimize the **Kullback–Leibler divergence** between the two distributions:

$$C = \text{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

- Optimization is done with gradient descent.

Non-linear dimensionality reduction

Key Features

- **Preserves local neighborhoods:** clusters remain intact.
- **Student-t distribution** in low-dim space avoids the “crowding problem.”
- **Perplexity parameter** controls the balance between local vs. more global structure.

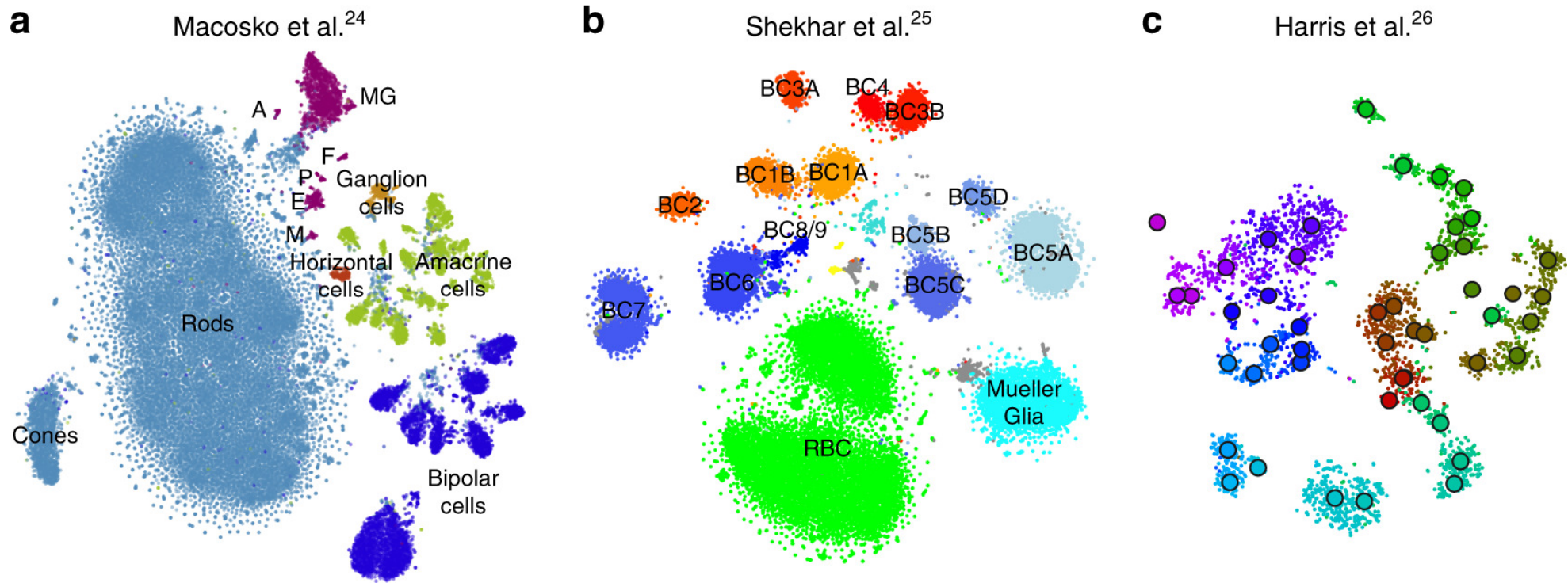
Limitations

- **Global distances not preserved:** two clusters far apart in 2D may not be as distant in reality.
- **Non-deterministic:** different runs may yield slightly different embeddings.
- **Hyperparameter sensitive** (especially perplexity).
- **Computationally expensive** for very large datasets (though approximations exist).

Applications in Bioinformatics

- **Single-cell RNA-seq:** visualize cell populations and subtypes.
- **Flow/mass cytometry:** cluster immune cell phenotypes.
- **Metagenomics:** detect microbial community structure.

Non-linear dimensionality reduction



<https://www.nature.com/articles/s41467-019-13056-x>

Non-linear dimensionality reduction

UMAP (Uniform Manifold Approximation and Projection)

Intuition

- **Goal:** Like t-SNE, embed high-dimensional data into 2D/3D while preserving neighborhood structure.
- **Difference:** UMAP is based on **manifold theory** and **algebraic topology**, aiming to preserve both **local** and **global structure**.
- Think of it as: *"find a fuzzy graph of neighborhoods in high-dim space, then optimize a low-dim embedding that preserves this graph."*
- Widely adopted in **single-cell transcriptomics** because it often gives clearer trajectories and faster computation than t-SNE.

Formal Construction (sketch)

1. High-dimensional fuzzy simplicial complex

- For each data point $x_i \in \mathbb{R}^p$, find its k -nearest neighbors.
- Define edge weights using a smooth exponential kernel:

$$w_{ij} = \exp\left(-\frac{\max(0, d(x_i, x_j) - \rho_i)}{\sigma_i}\right),$$

where:

- $d(x_i, x_j)$ = distance between points (e.g., Euclidean).
- ρ_i = distance to the closest neighbor (local connectivity).
- σ_i = scaling parameter ensuring smooth decay.
- This defines a **fuzzy topological representation** (a weighted graph approximating the manifold).

Non-linear dimensionality reduction

2. Low-dimensional embedding graph

- Place points $y_i \in \mathbb{R}^2$ or \mathbb{R}^3 .
- Define a similar fuzzy graph in the low-dimensional space using:

$$\tilde{w}_{ij} = \left(1 + a\|y_i - y_j\|^{2b}\right)^{-1},$$

where a, b are parameters controlling the shape of the curve (learned from data).

3. Optimization

- Minimize the **cross-entropy** between the high-dimensional and low-dimensional graphs:

$$C = \sum_{i,j} \left[w_{ij} \log \frac{w_{ij}}{\tilde{w}_{ij}} + (1 - w_{ij}) \log \frac{1 - w_{ij}}{1 - \tilde{w}_{ij}} \right].$$

- This ensures that strong connections in the high-dim graph are preserved in the low-dim embedding.

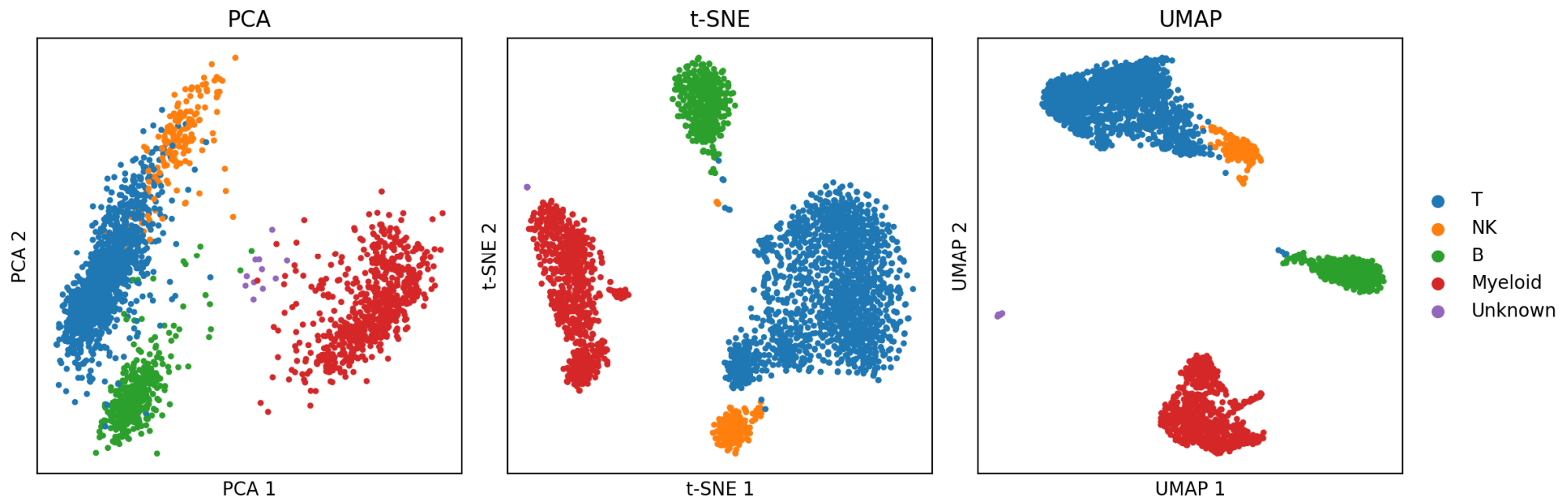
Key Features

- **Local + global preservation:** captures both clusters and trajectories.
- **Speed and scalability:** typically faster than t-SNE, works well for millions of points.
- **Deterministic** (given fixed random seed).
- **Parameters:**
 - `n_neighbors` : controls local vs. global balance (like t-SNE's perplexity).
 - `min_dist` : controls how tightly points are packed in the embedding.

Non-linear dimensionality reduction

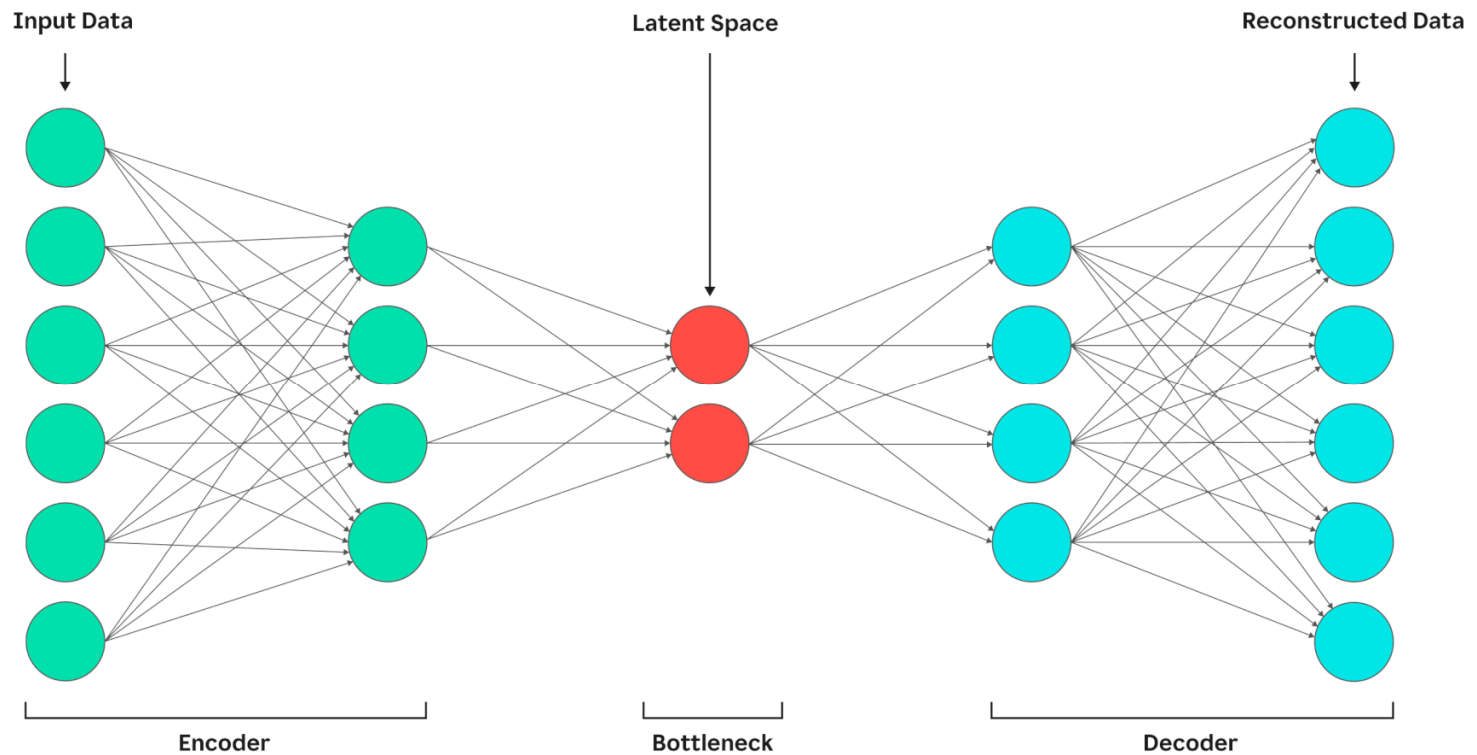
Limitations

- **Less interpretable** mathematically than PCA (nonlinear).
- **Hyperparameters** can affect visualization strongly.
- Like t-SNE, **distances in 2D/3D are not absolute**: structure is qualitative, not quantitative.



Non-linear dimensionality reduction

Autoencoders



Non-linear dimensionality reduction

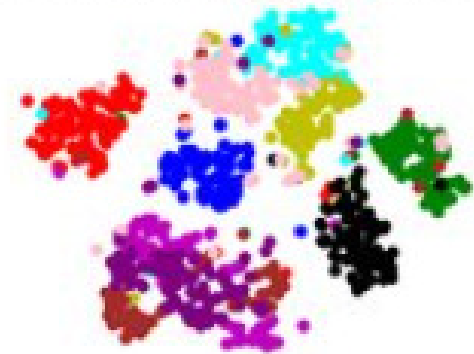
Epoch 0, accuracy: 0.171



Epoch 20, accuracy: 0.752



Epoch 40, accuracy: 0.817



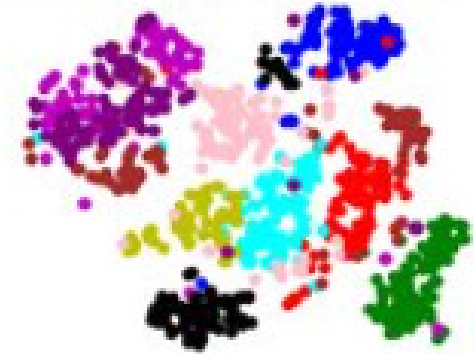
Epoch 60, accuracy: 0.833



Epoch 80, accuracy: 0.851



Epoch 100, accuracy: 0.856



Non-linear dimensionality reduction

| Method | Reconstruction possible? | How? | Bioinformatics interpretation |
|--------------|--|---|---|
| PCA | ✓ Yes | Linear projection + inverse | Compress omics data, denoise, approximate reconstruction of expression profiles |
| t-SNE | ✗ No | Not a projection, embedding only | Only for visualization (clusters, cell types) |
| UMAP | ✗/≐ Not standard (only with parametric UMAP) | Embedding preserves neighborhoods, but no inverse | Visualization and manifold discovery |
| Autoencoders | ✓ Yes | Neural network encoder–decoder | Nonlinear compression and reconstruction of omics data |

Contents

- Overview
- Linear dimensionality reduction
 - PCA
- Non-linear dimensionality reduction
 - Manifolds, t-SNE, UMAP

Practice

<https://github.com/sdawley1/ML-Cancer-Classification/blob/main/Final%20Project.ipynb>

This dataset comes from a proof-of-concept study published in 1999 by Golub et al. It showed how new cases of cancer could be classified by gene expression monitoring (via DNA microarray) and thereby provided a general approach for identifying new cancer classes and assigning tumors to known classes. These data were used to classify patients with acute myeloid leukemia (AML) and acute lymphoblastic leukemia (ALL).

There are two datasets containing the training/initial (`training` , 38 samples) and test/independent (`independent` , 34 samples) datasets used in the paper. These datasets contain measurements corresponding to ALL and AML samples from Bone Marrow and Peripheral Blood. Intensity values have been re-scaled such that overall intensities for each chip are equivalent.

References

Golub, et al.

Information on data set

<https://www.kaggle.com/crawford/gene-expression>



CEU

*Universidad
San Pablo*

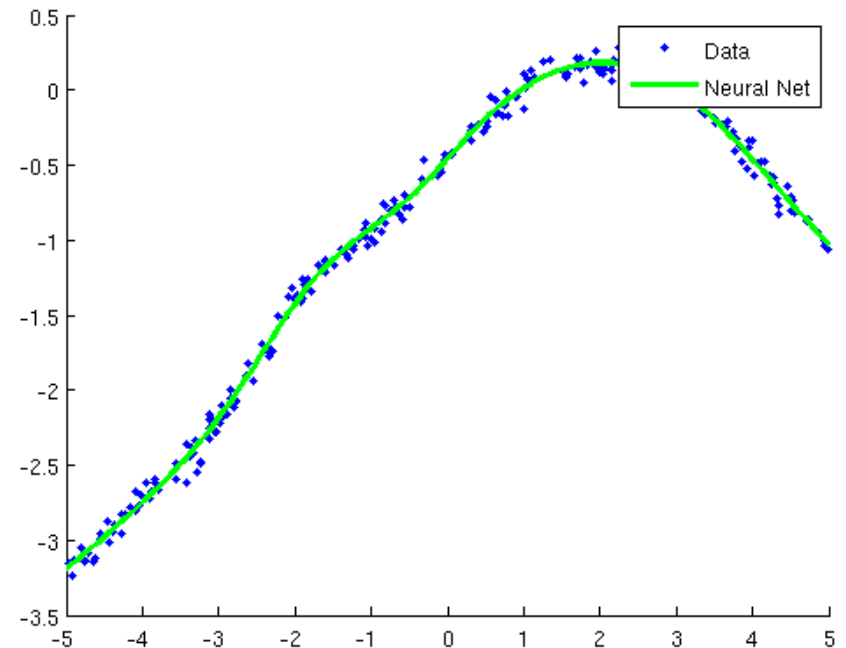
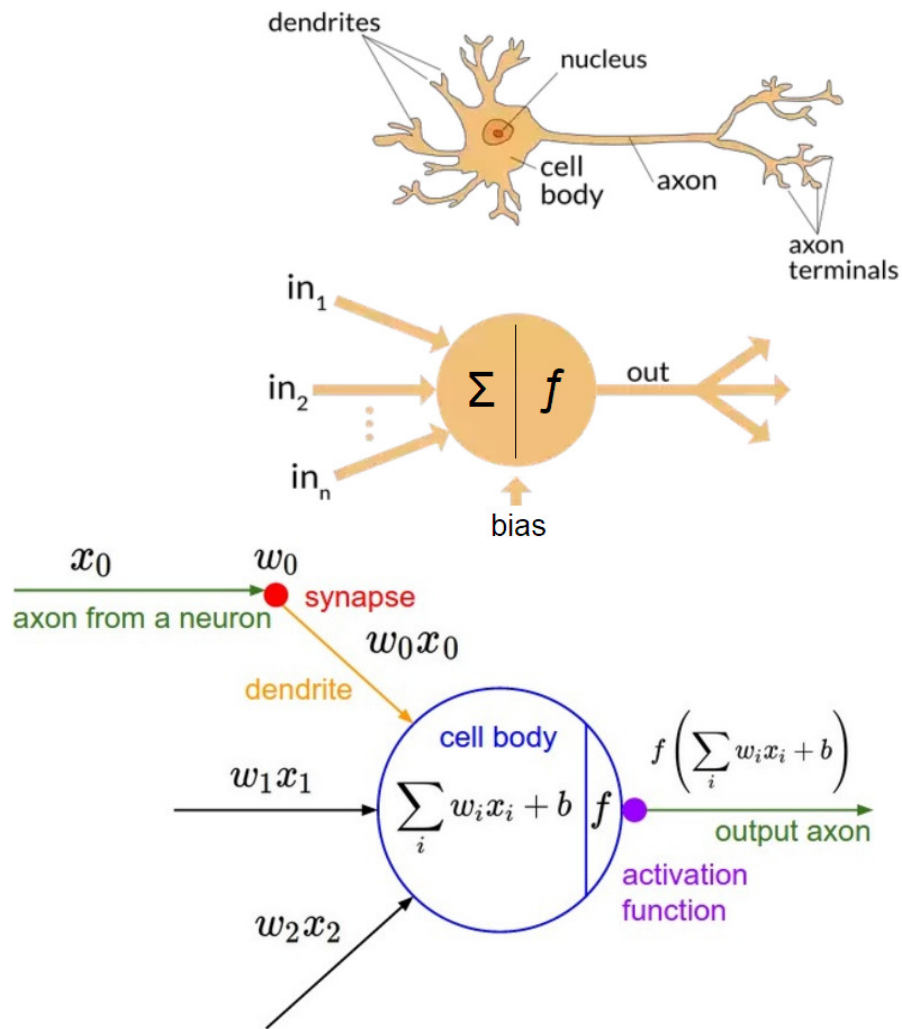
Lesson 8. Neural networks

Medicin School

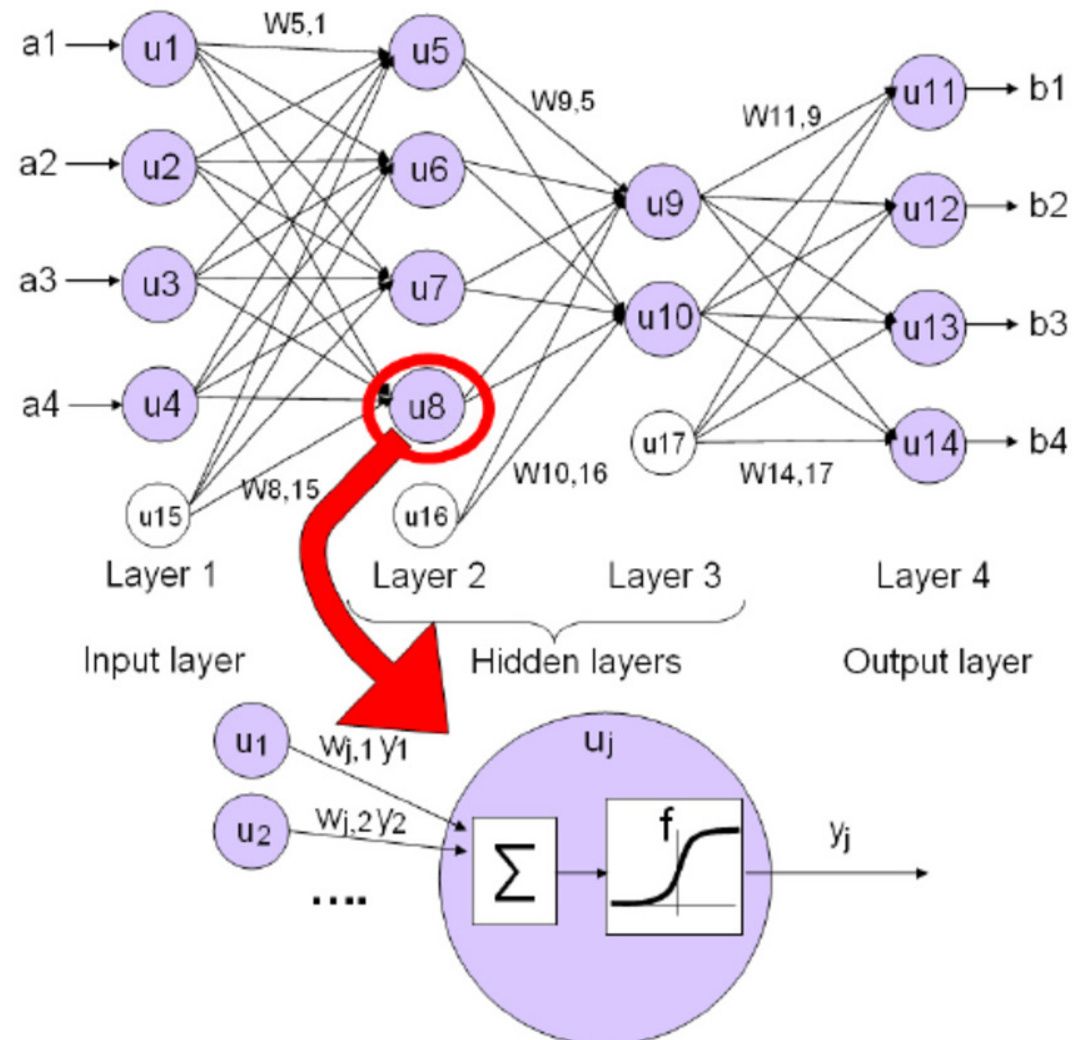
Contents

- Overview
- Basic networks
- Regularization
- Architectures
 - Convolutional, RNN, LSTM, Transformer, GPT, BERT, Autoencoders, Variational autoencoders, Generative Adversarial networks, Diffusion networks
- Applications

Overview

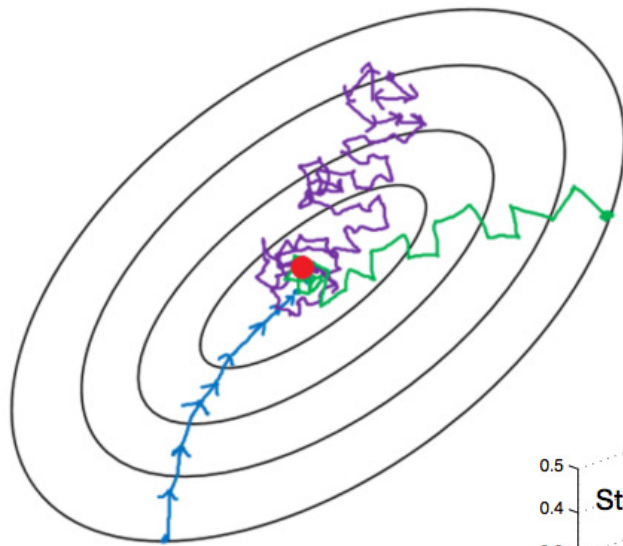


Overview

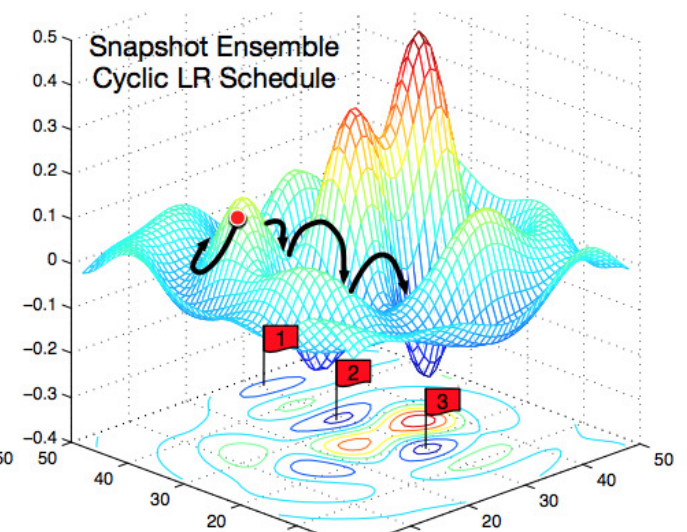
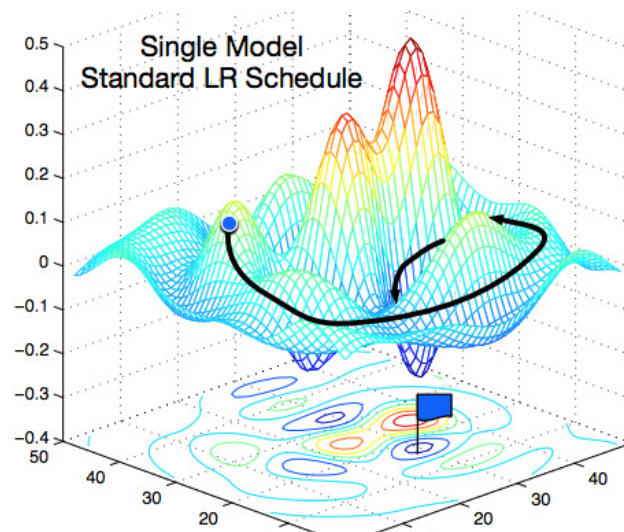


Overview

Stochastic optimization



- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent



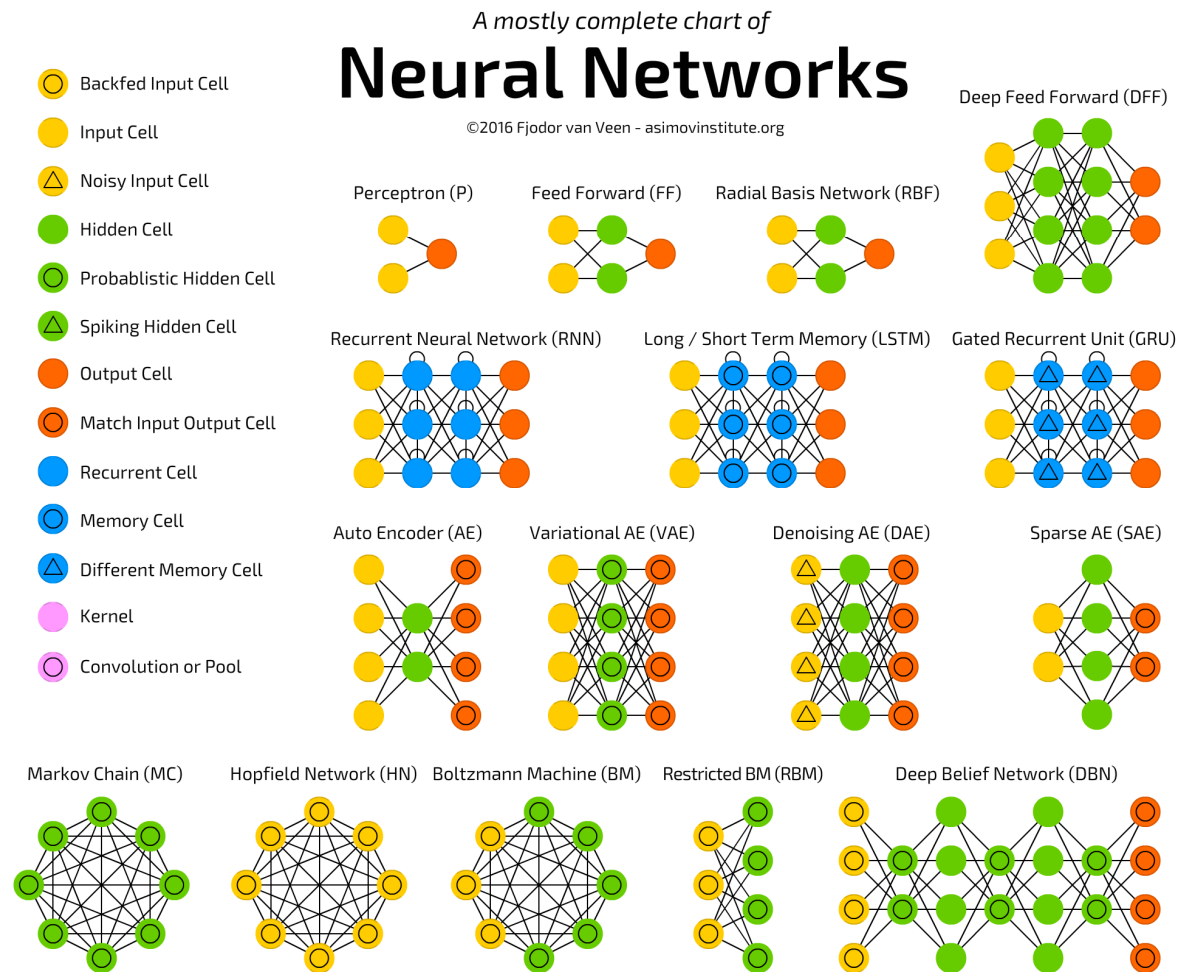
Overview

Why deep learning now

- Many parameters require many training examples (big data).
- GPUs have largely accelerated the calculations:
 - Tensorflow/Keras (Google)
 - Pytorch (Facebook)
 - Jax (Google)
- Advances in stochastic optimization.
- Advances in network architectures.

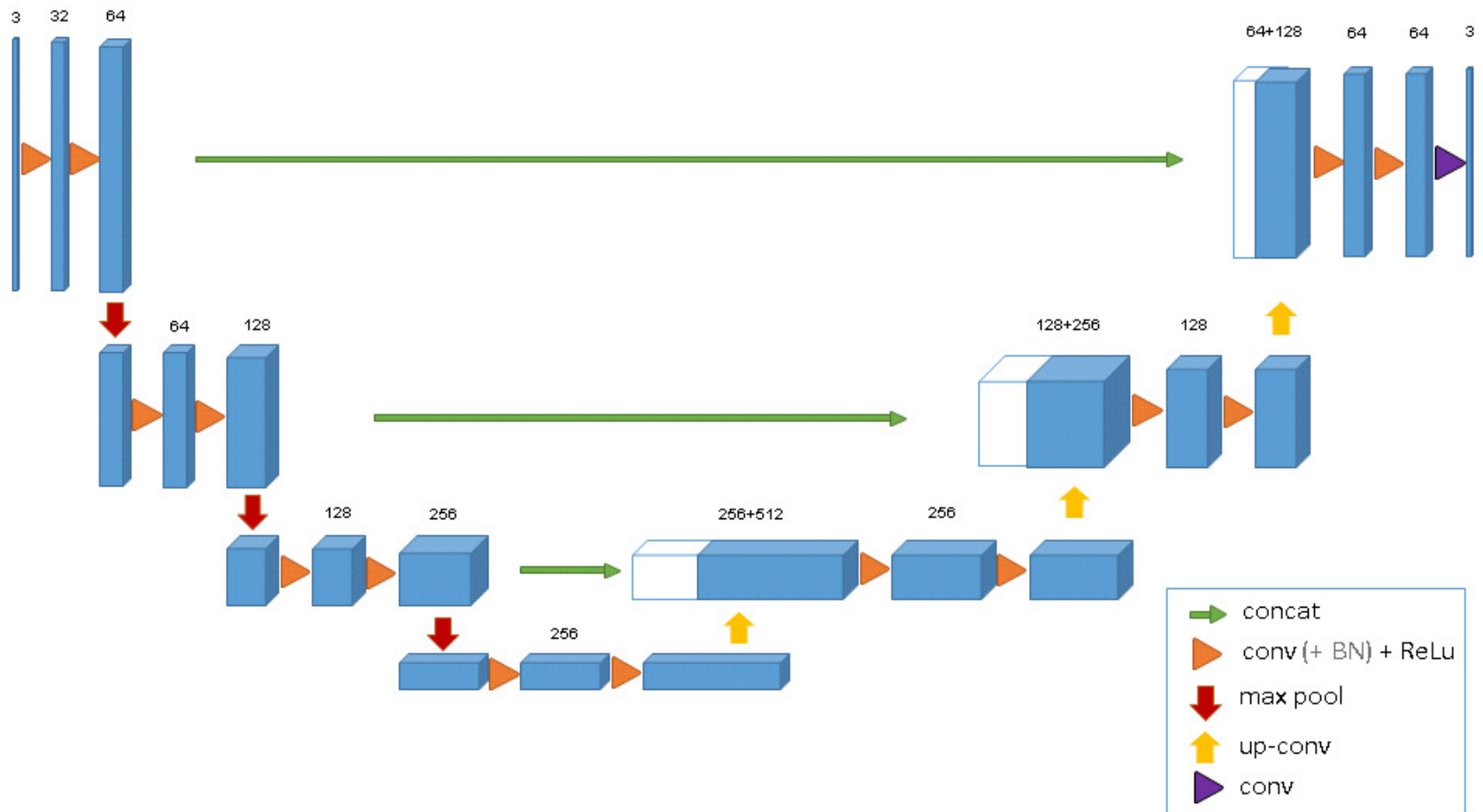
Overview

Architecture

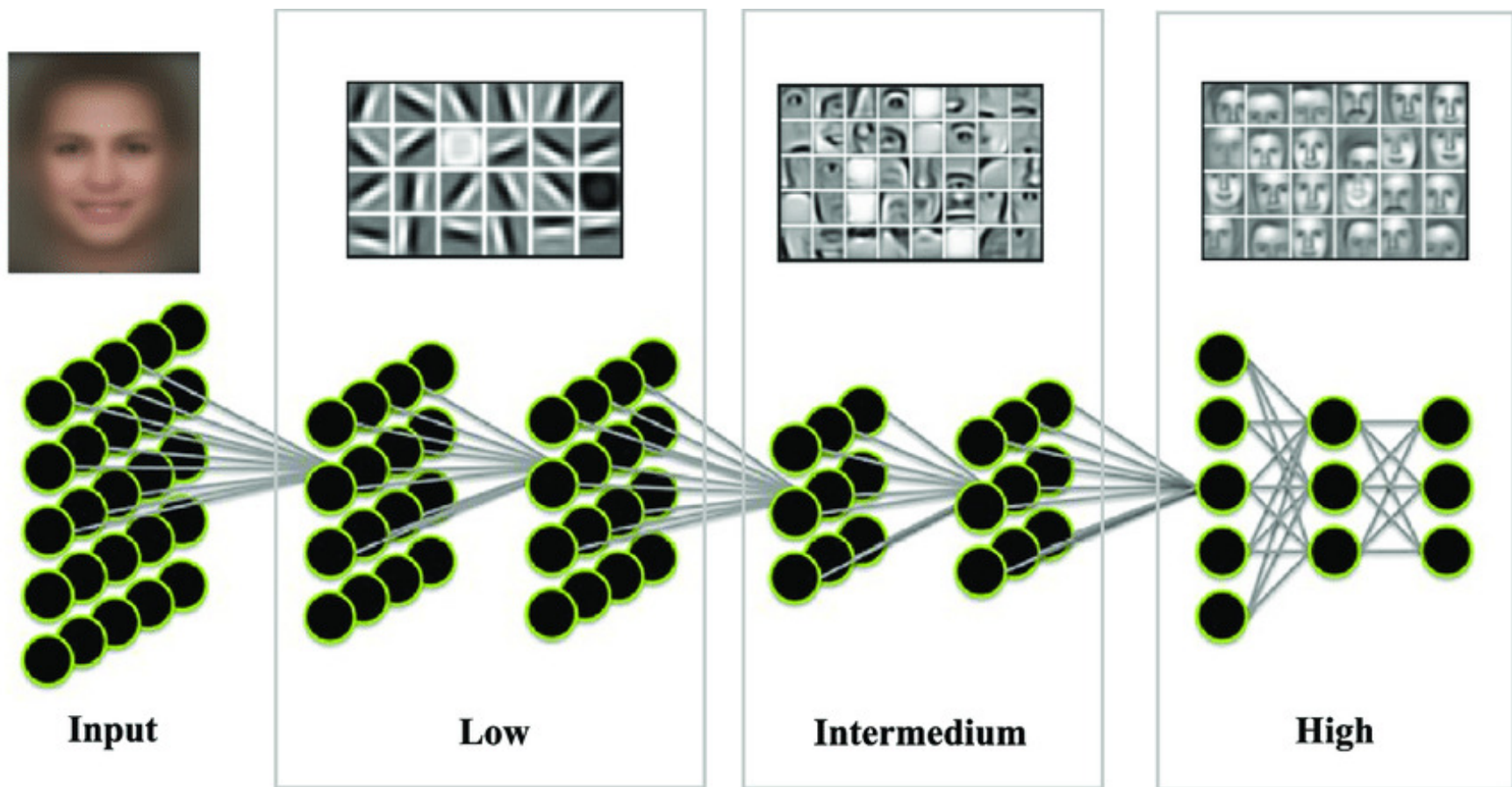


Deep neural network

Architecture

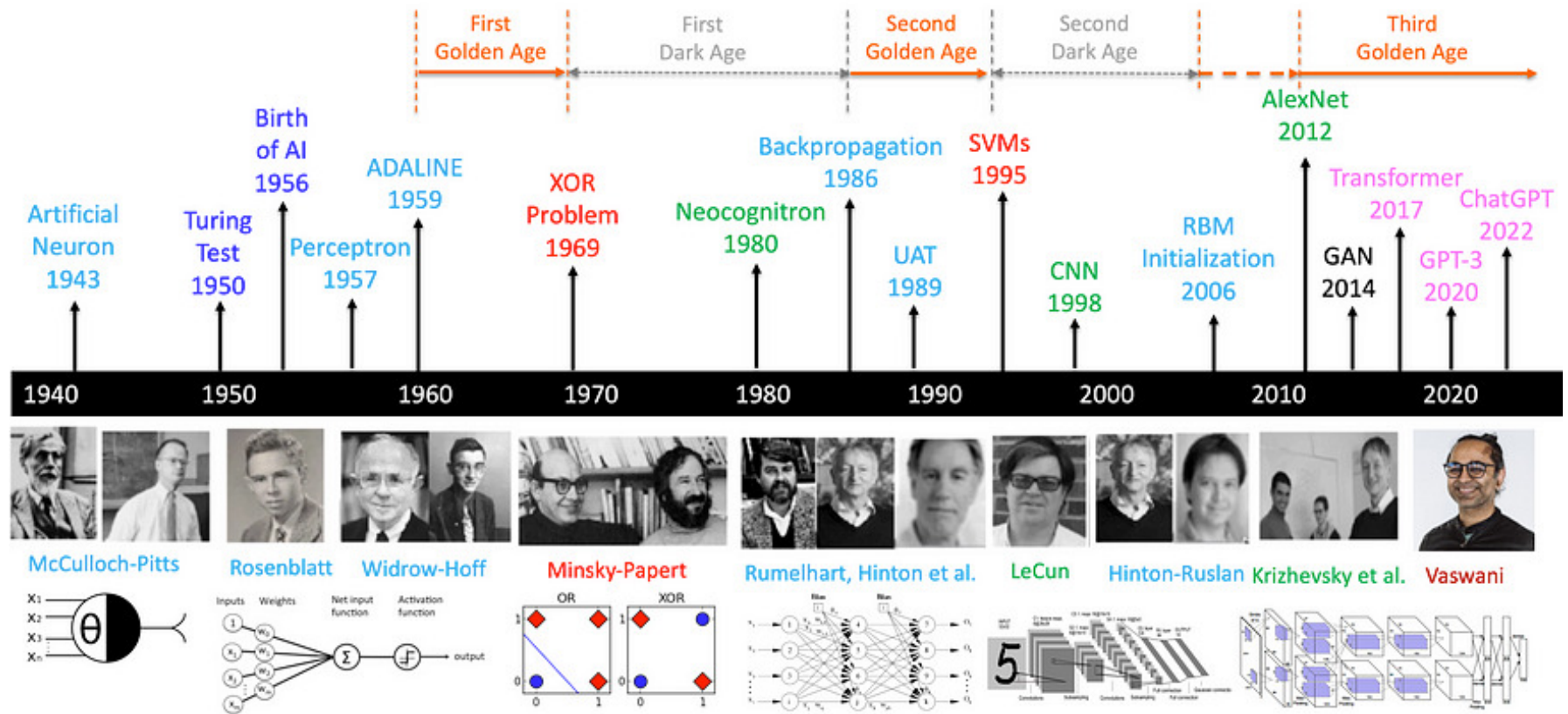


Overview



Overview

A Brief History of AI with Deep Learning



<https://medium.com/@lmpo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b>

Overview

DL \subset **NN** \subset **ML** \subset **AI**

Artificial Intelligence (AI)

- AI refers to a set of technique that enable computers **to mimic human behavior**.

Machine Learning (ML)

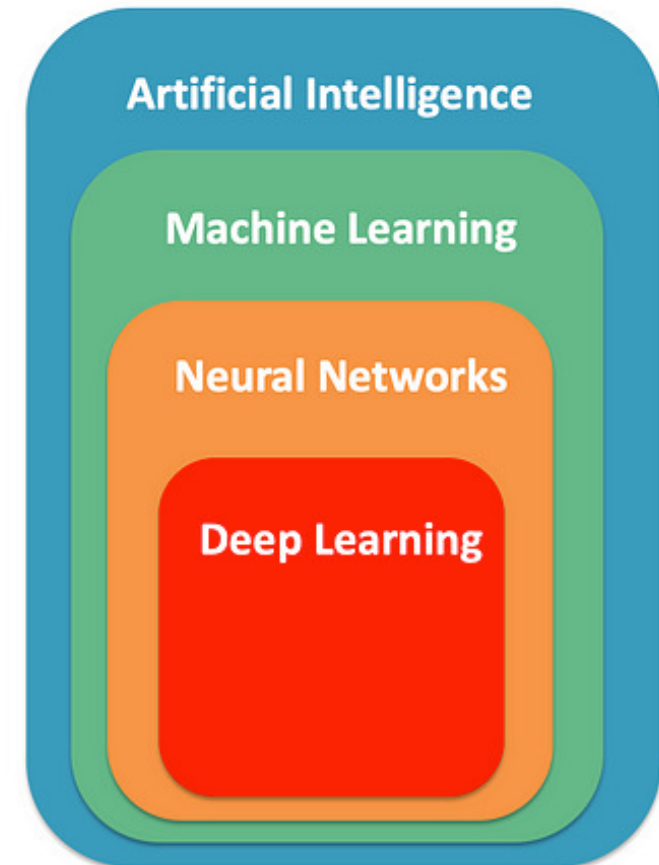
- ML is a subfield of AI , which enables machines **to learn and make prediction based on data**.

Neural Networks (NN)

- NN are a subfield of ML that **use artificial neural networks** to extract patterns from data, inspired by the human brain.

Deep Learning (DL)

- DL is a subfield of NN that **utilizes multi-layered neural networks** to achieve high performance on complex tasks.



Basic networks

Loss Functions in Deep Learning

Role of the loss function

- The loss function measures **how far the network's predictions are from the true labels**.
- Training a neural network means **minimizing this loss** by adjusting the weights through optimization (e.g., gradient descent).
- The choice of loss function depends on the **task type** (regression vs. classification).

1. Regression Losses

When the output is a continuous value (e.g., predicting gene expression levels, protein stability scores):

- **Mean Squared Error (MSE):**

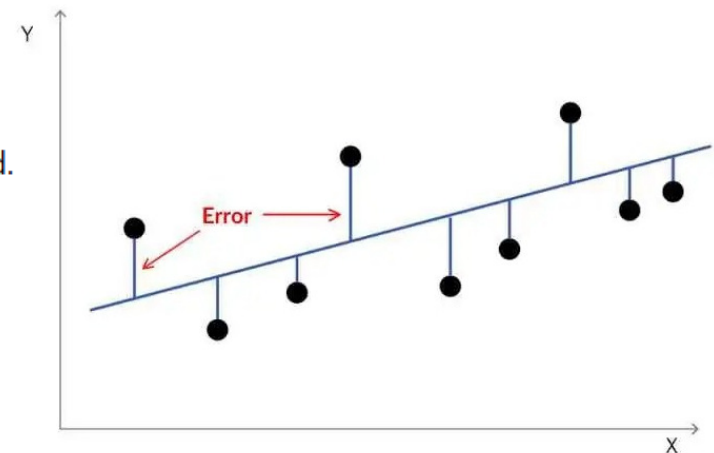
$$L = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Penalizes large errors heavily (quadratic growth). Smooth, differentiable, widely used.

- **Mean Absolute Error (MAE):**

$$L = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

More robust to outliers, but less smooth for optimization.



Basic networks

2. Classification Losses

When the output is categorical (e.g., classifying tumor vs. normal samples, predicting protein family):

- **Binary Cross-Entropy (logistic loss, for 2 classes):**

$$L = -\frac{1}{N} \sum_{i=1}^N \left[y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i) \right]$$

Encourages predicted probability \hat{p}_i to be close to the true label $y_i \in \{0, 1\}$.

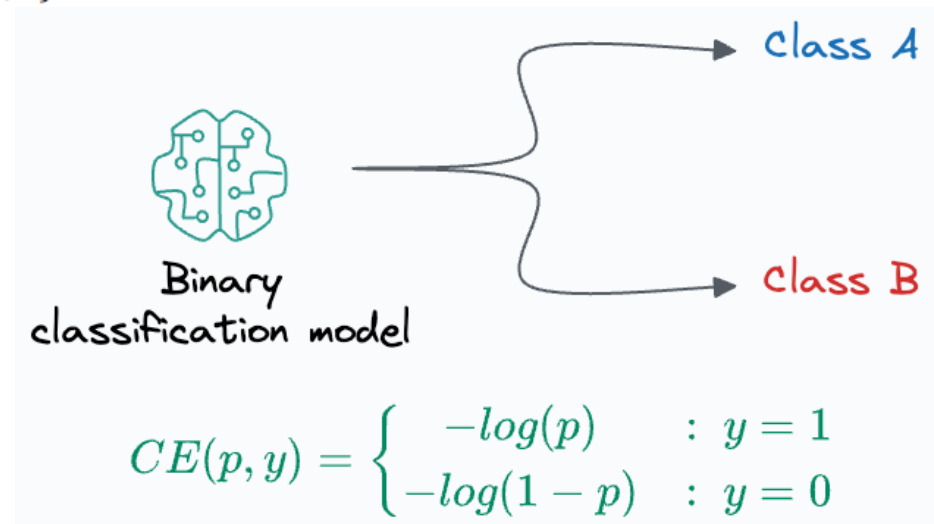
- **Categorical Cross-Entropy (multiclass):**

$$L = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log(\hat{p}_{ik})$$

Here y_{ik} is 1 if sample i belongs to class k , else 0.

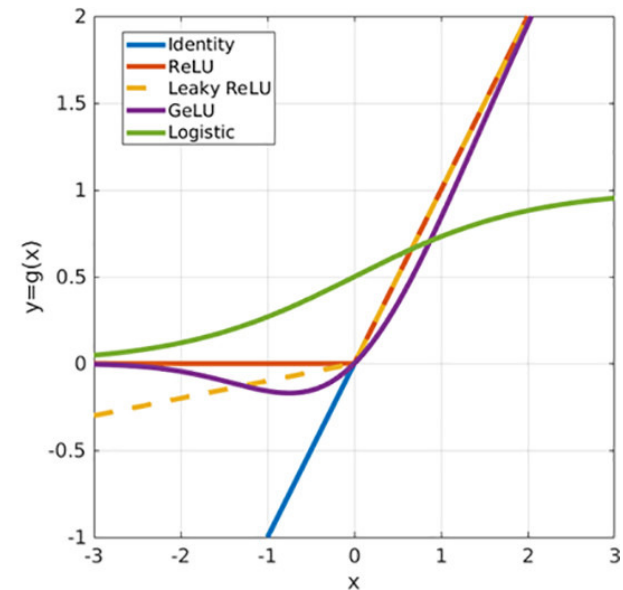
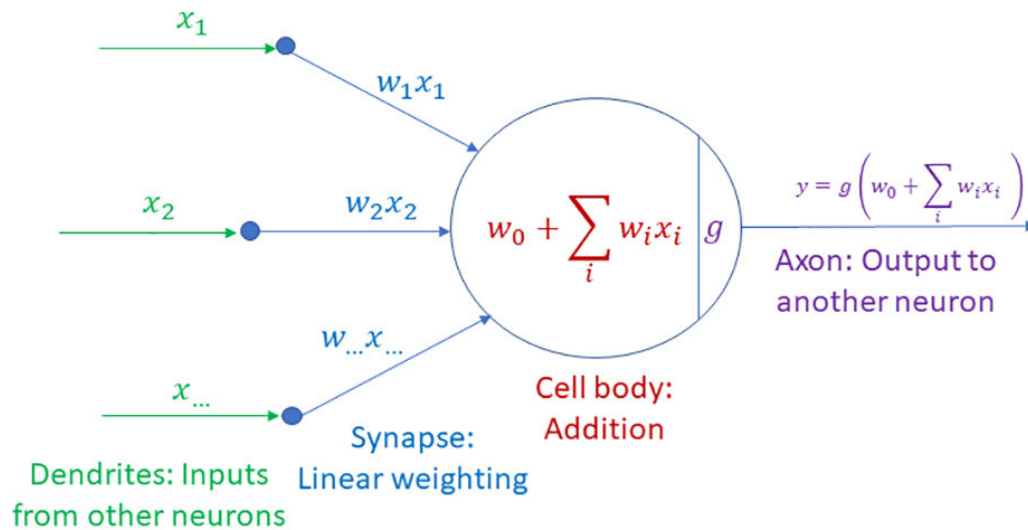
Used with **softmax** outputs, where \hat{p}_{ik} are predicted class probabilities.

| | | |
|--------------------|---------------------------------------|---------------------------------------|
| True label | $y = 1$ <small>minority class</small> | $y = 0$ <small>majority class</small> |
| Probability output | $\hat{y} = 0.3$ | $\hat{y} = 0.7$ |
| log loss | $-\log(0.3)$ | $-\log(0.3)$ |



Basic networks

A single neuron



$$u(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$RELU(x) = xu(x)$$

$$Leaky\ RELU(x) = xu(x) + axu(-x)$$

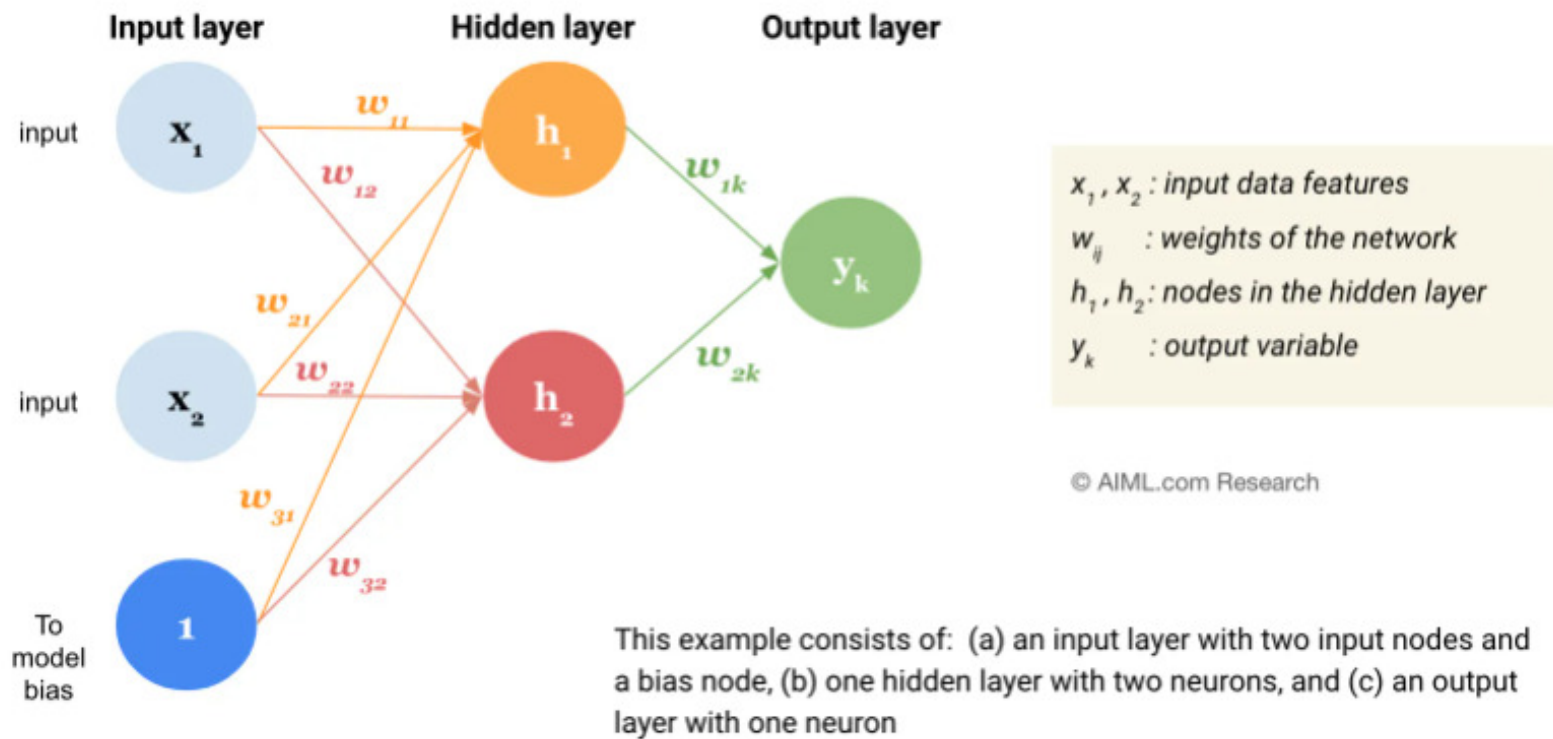
$$Logistic(x) = 1/(1 + \exp(-x))$$

$$GELU(x) = xCDF_{Gaussian}(x)$$

Basic networks

Multilayer Perceptron

Illustrative example of Multilayer perceptron, a Feedforward neural network



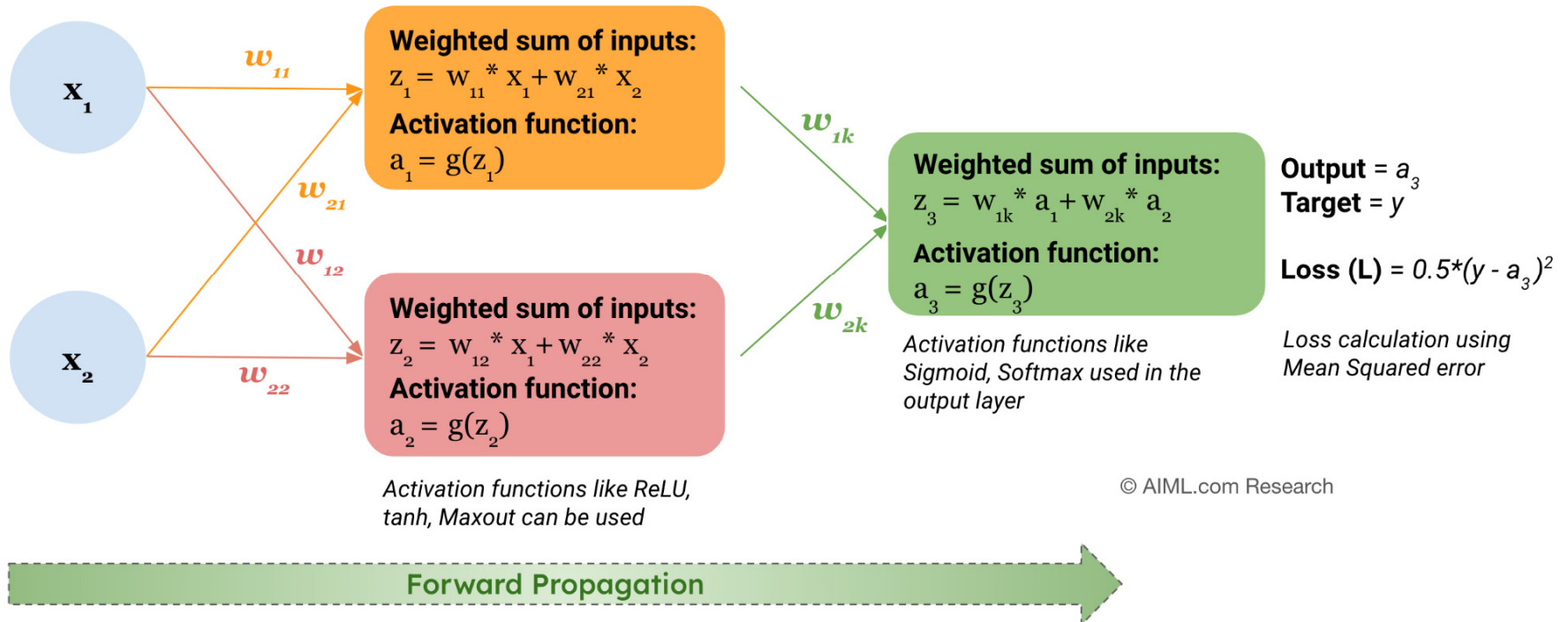
Basic networks

Forward pass

Input layer

Hidden layer

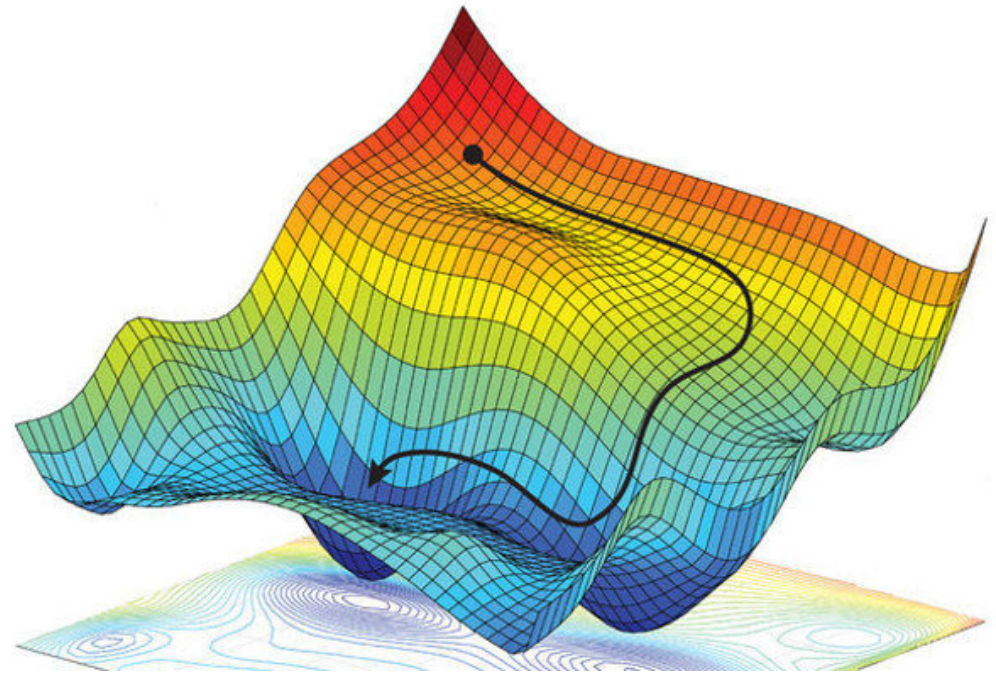
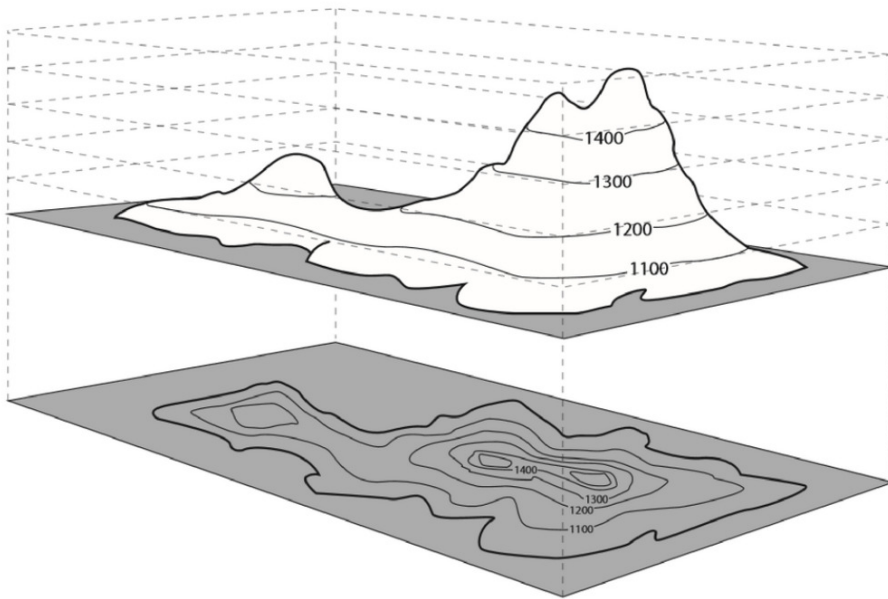
Output layer



© AIML.com Research

Basic networks

Loss function



Basic networks

Backward pass

Backpropagation

Goal

- Compute gradients of the loss with respect to each parameter (weights, biases).
- Needed to update parameters via gradient descent.

Key Idea

- Neural networks are **compositions of functions**:

$$\hat{y} = f(x; \theta)$$

- Loss function:

$$L(\theta) = \mathcal{L}(\hat{y}, y)$$

- Backpropagation uses the **chain rule** to compute $\frac{\partial L}{\partial \theta}$.

Basic networks

Steps

1. Forward pass

- Input flows through the network.
- Compute activations layer by layer until output \hat{y} .
- Evaluate the loss L .

2. Backward pass

- Start at the loss and propagate gradients backwards.
- For each layer:

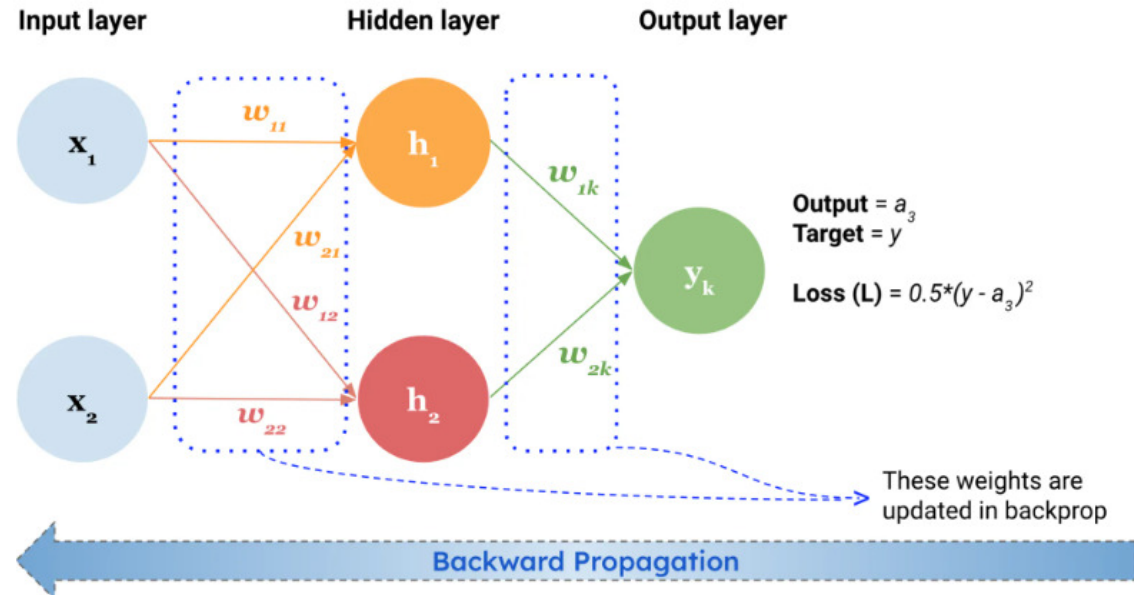
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial W}$$

where $z = Wx + b$, $a = \sigma(z)$.

3. Update parameters

- Gradient descent step:

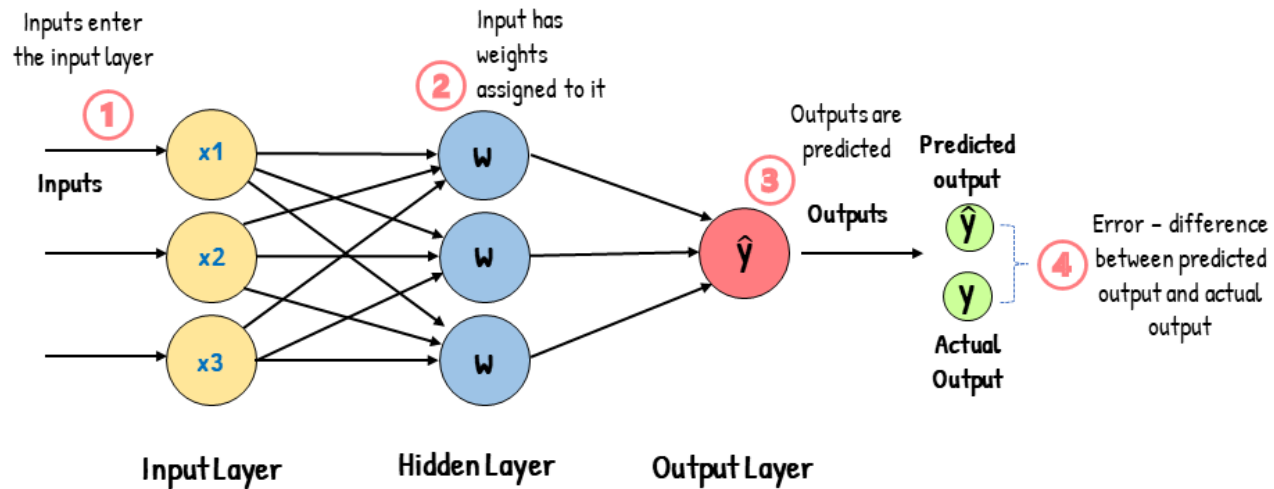
$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$



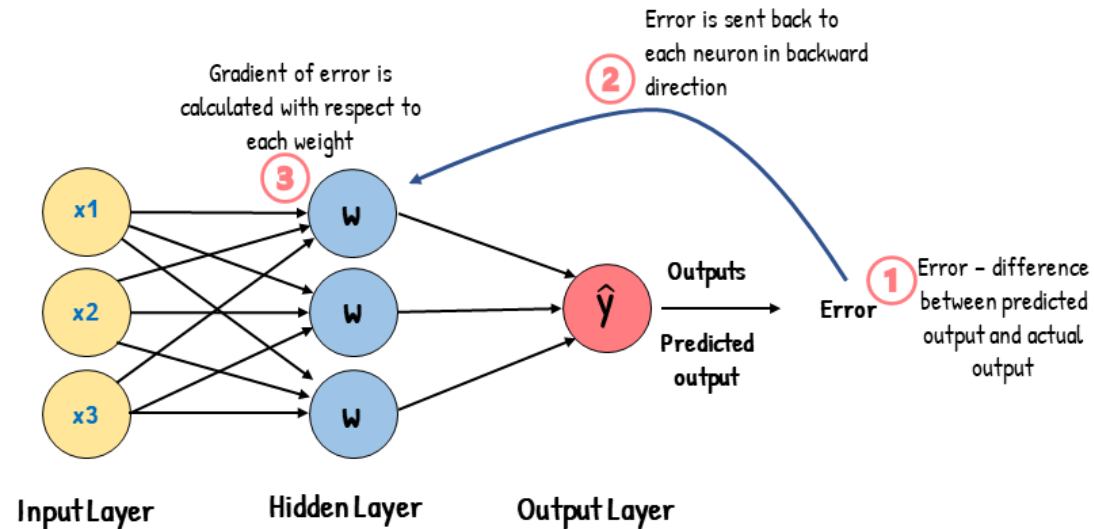
- In stochastic gradient descent, weights (w_{ij}) are updated as below:
 $w_{ij} := w_{ij} + \Delta w_{ij}$
 $\Delta w_{ij} = -\eta * \partial L(w_{ij}) / \partial w_{ij}$,
 where η is the learning rate and $\partial L(w_{ij}) / \partial w_{ij}$ is the gradient of loss w.r.t the model weights
- Intermediate variables calculated during forward prop ($z_1, z_2, z_3, a_1, a_2, a_3$) are used for gradient calculation $\partial L(w_{ij}) / \partial w_{ij}$

Basic networks

Forward pass



Backward pass



Basic networks

Gradient Descent

- Update rule:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta)$$

- η = learning rate (step size).
- Uses the full dataset \rightarrow computationally expensive.

Stochastic Gradient Descent

- Instead of all data, update with **one sample** (or a small batch) at a time.
- Update rule for sample i :

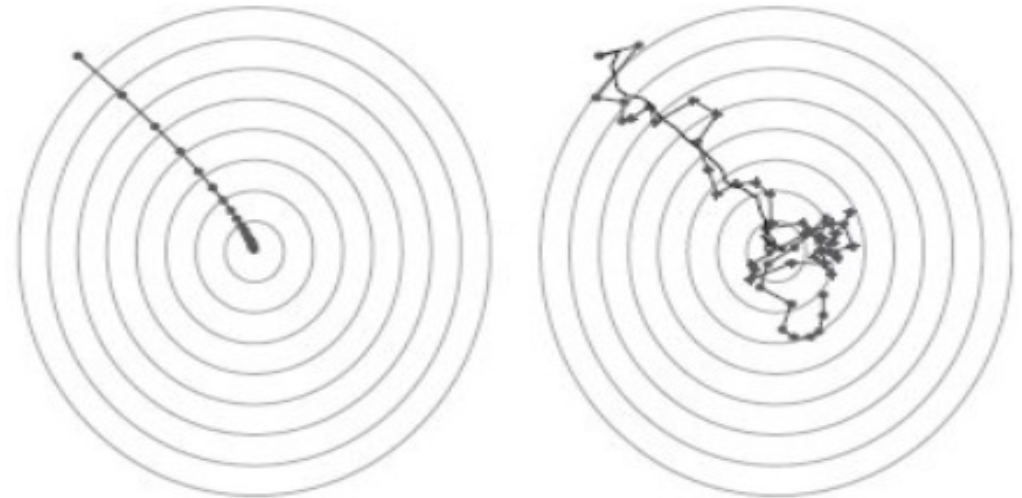
$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(f(x_i; \theta), y_i)$$

- Faster, introduces **noise** in updates (helps escape local minima).

Mini-batch SGD

- Balance between efficiency and stability.
- Use batches of size B :

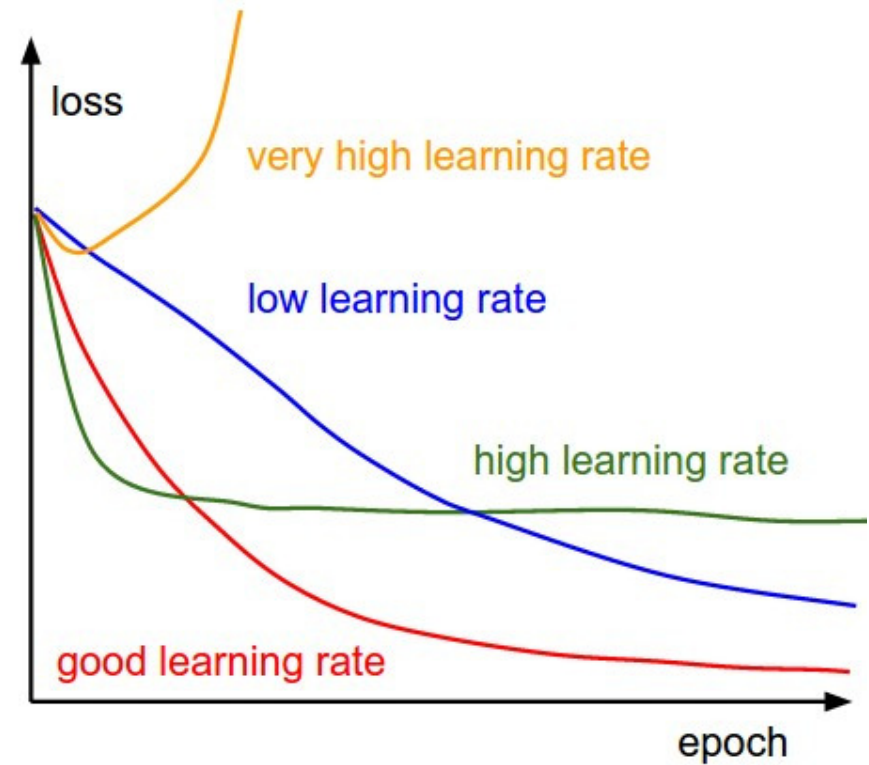
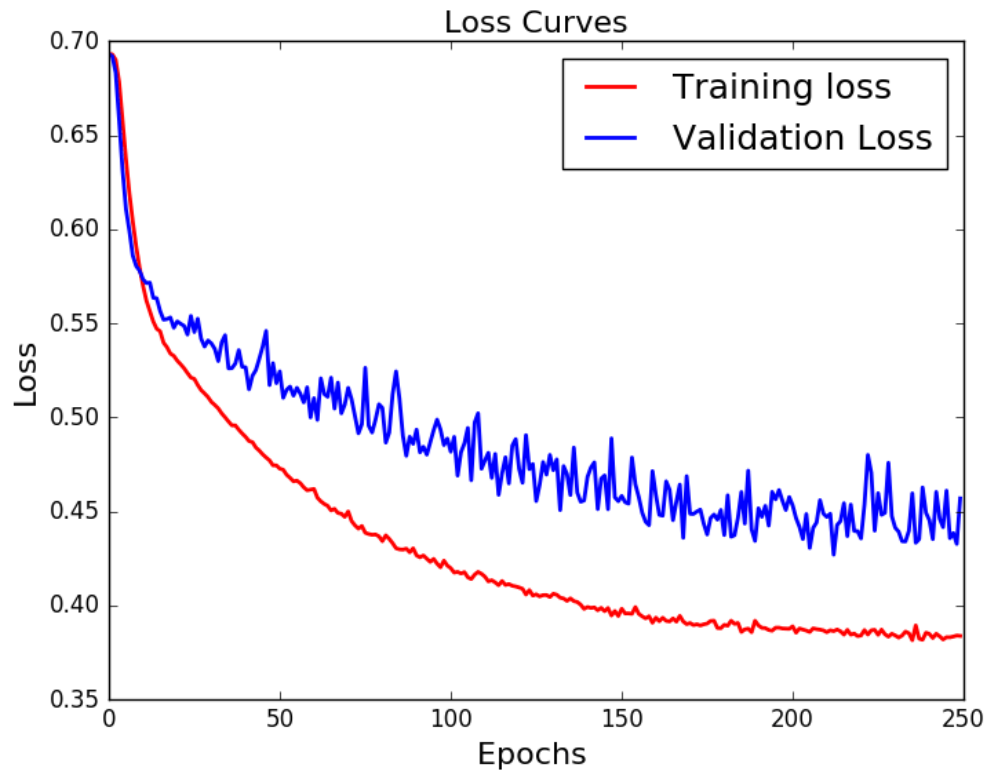
$$\theta \leftarrow \theta - \eta \frac{1}{B} \sum_{i=1}^B \nabla_{\theta} \mathcal{L}(f(x_i; \theta), y_i)$$



Hyperparameters

- Learning rate η** : too high \rightarrow divergence, too low \rightarrow slow learning.
- Batch size**: small \rightarrow noisy but fast, large \rightarrow stable but expensive.

Basic networks



Regularization

Why?

- Deep networks have millions of parameters → risk of **overfitting**.
- Regularization = techniques to improve **generalization**.

1. Weight Decay (L2 Regularization)

- Add penalty on large weights:

$$L_{\text{reg}} = L + \lambda \sum_j w_j^2$$

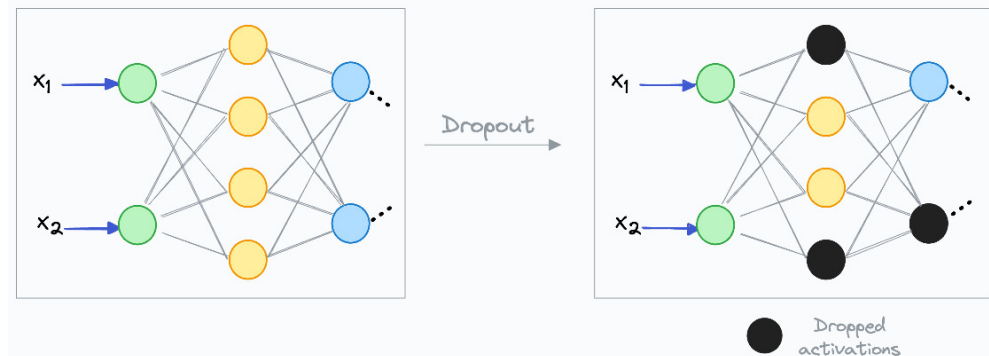
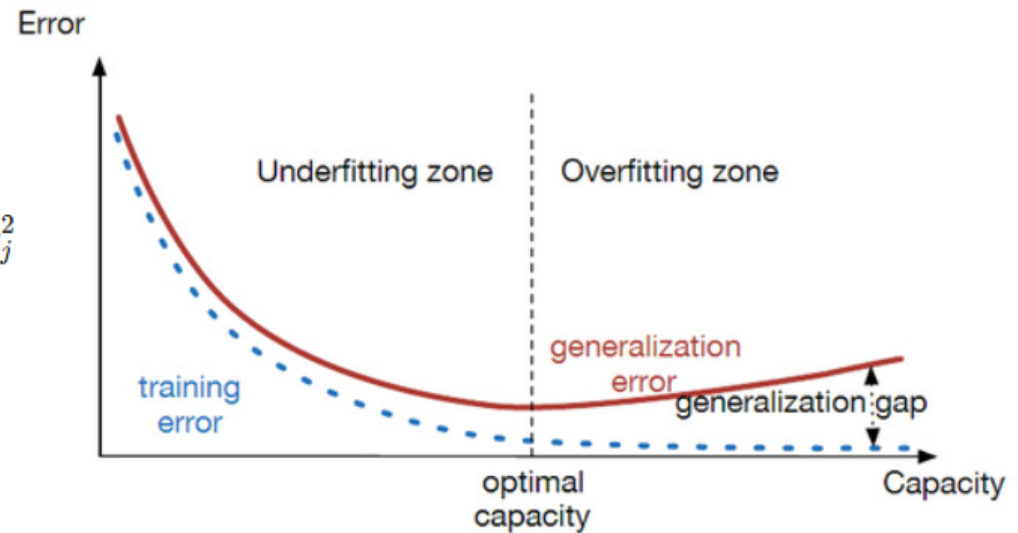
- Encourages smaller, smoother weights.
- Widely used (default in most frameworks).

2. Dropout

- Randomly “turn off” neurons during training (with probability p).
- Prevents co-adaptation of features.
- At inference: scale activations to use the full network.

3. Early Stopping

- Monitor validation loss.
- Stop training when validation loss stops decreasing.
- Simple and very effective.



Regularization

4. Batch Normalization

- Normalize activations within a mini-batch.
- Reduces **internal covariate shift**.
- Acts as both **accelerator** (faster convergence) and **regularizer**.

1. Per-feature mean and variance (across the batch)

For feature j :

$$\mu_j = \frac{1}{M} \sum_{i=1}^M x_{ij}, \quad \sigma_j^2 = \frac{1}{M} \sum_{i=1}^M (x_{ij} - \mu_j)^2$$

2. Normalization

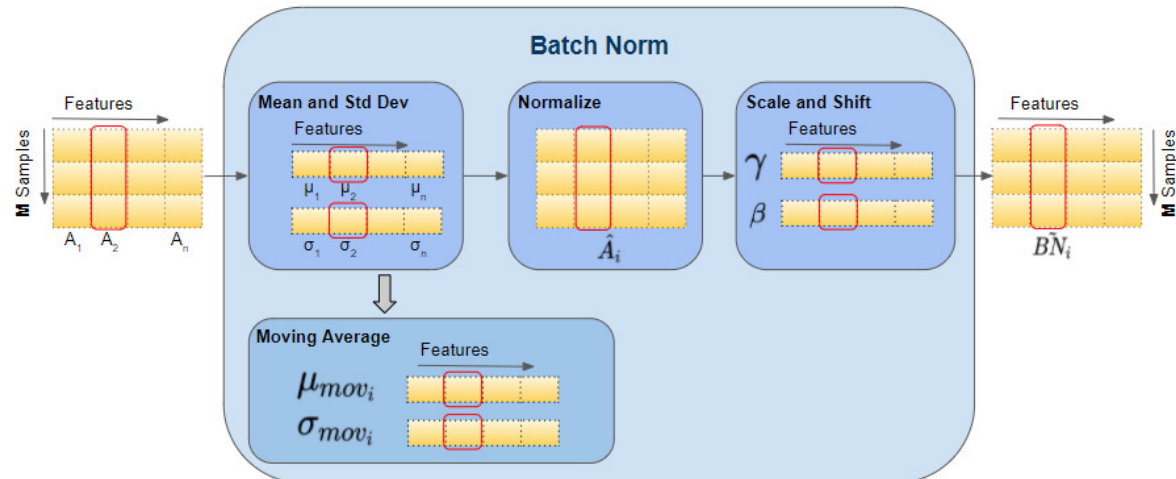
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

3. Scale and shift (learnable parameters)

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$$

4. BatchNorm output

$$BN(x_{ij}) = y_{ij}$$



Architectures

Convolution Operation

- Instead of connecting every input to every neuron, a **filter (kernel)** slides over the input.
- Each filter detects a specific **local pattern** (edges, motifs, secondary structures).
- Formula for 2D convolution (image-like input):

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

where I = input, K = kernel (shared weights).

Source layer

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 2 | 6 | 8 | 2 | 0 | 1 | 2 |
| 4 | 3 | 4 | 5 | 1 | 9 | 6 | 3 |
| 3 | 9 | 2 | 4 | 7 | 7 | 6 | 9 |
| 1 | 3 | 4 | 6 | 8 | 2 | 2 | 1 |
| 8 | 4 | 6 | 2 | 3 | 1 | 8 | 8 |
| 5 | 8 | 9 | 0 | 1 | 0 | 2 | 3 |
| 9 | 2 | 6 | 6 | 3 | 6 | 2 | 1 |
| 9 | 8 | 8 | 2 | 6 | 3 | 4 | 5 |

Convolutional kernel

| | | |
|----|----|---|
| -1 | 0 | 1 |
| 2 | 1 | 2 |
| 1 | -2 | 0 |

Destination layer

| | | | | | | | |
|--|---|--|--|--|--|--|--|
| | | | | | | | |
| | 5 | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

$$\begin{aligned} &(-1 \times 5) + (0 \times 2) + (1 \times 6) + \\ &(2 \times 4) + (1 \times 3) + (2 \times 4) + \\ &(1 \times 3) + (-2 \times 9) + (0 \times 2) = 5 \end{aligned}$$

Architectures

Sobel Edge Detection

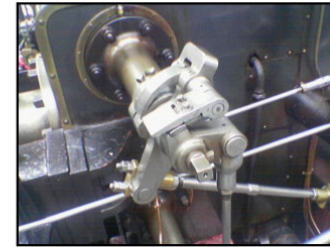
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

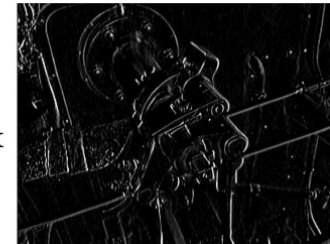
- Find pixels with large gradients

$$G = \sqrt{G_x^2 + G_y^2}$$

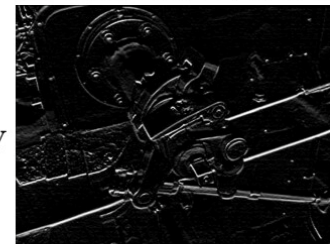
Pixel-wise operation on images



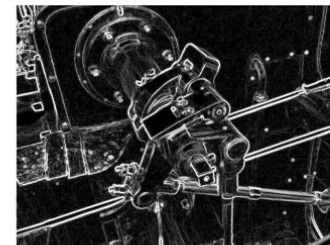
G_x



G_y



G



Architectures

Pooling operation

Motivation

- After convolutions, feature maps are often still large.
- Pooling **reduces dimensionality** while keeping the most important information.
- Makes the representation more **robust to small shifts/noise** in the input.

1. Max Pooling

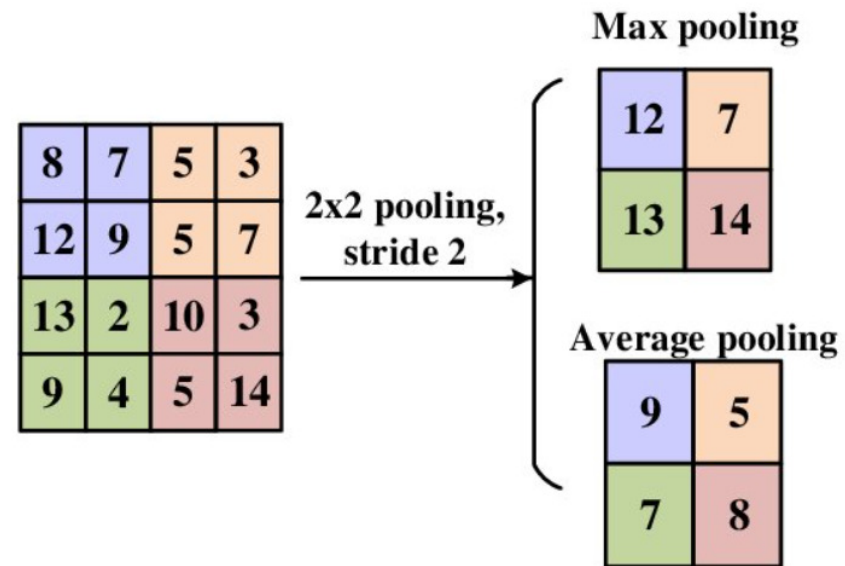
- Takes the maximum value in a local patch.
- Keeps the strongest activation (most confident feature detection).
- Formula (patch size $p \times p$):

$$y_{ij} = \max_{(m,n) \in p \times p} x_{i+m,j+n}$$

2. Average Pooling

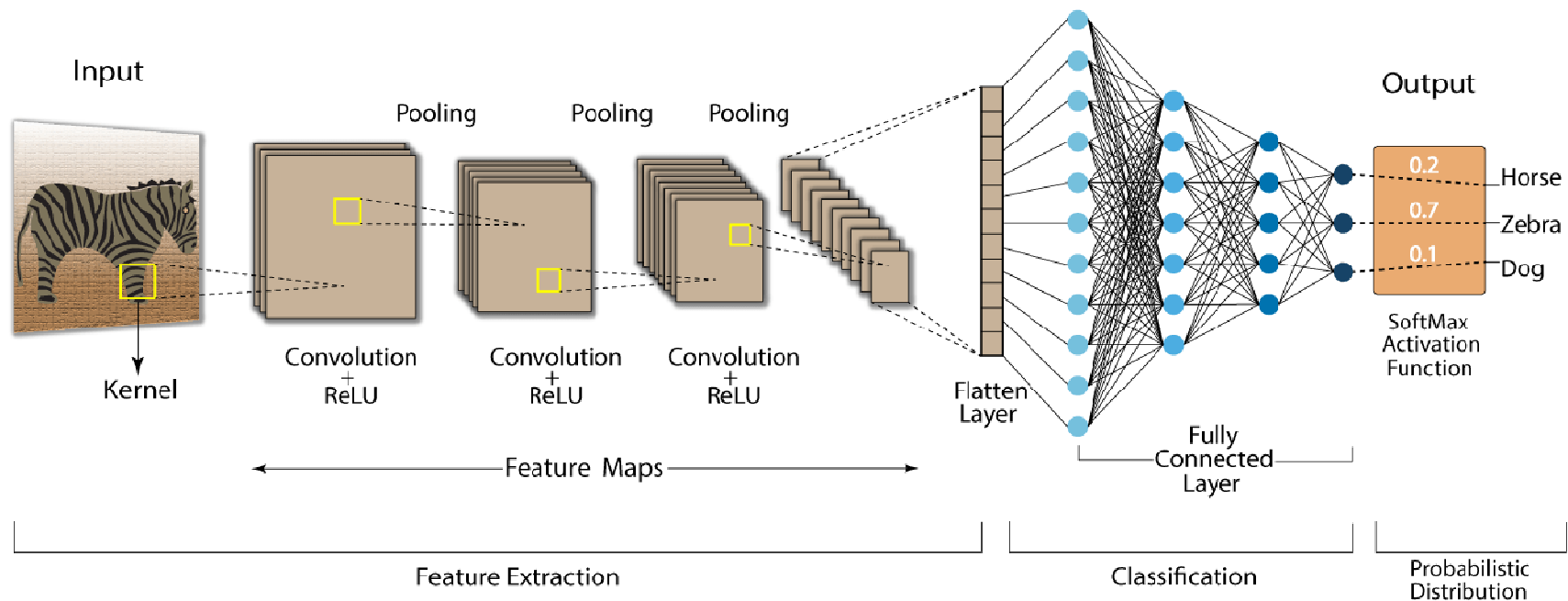
- Takes the average of the patch.
- Smooths the representation, keeps overall intensity.
- Formula:

$$y_{ij} = \frac{1}{p^2} \sum_{(m,n) \in p \times p} x_{i+m,j+n}$$



Architectures

Convolution Neural Network (CNN)



Weight Sharing

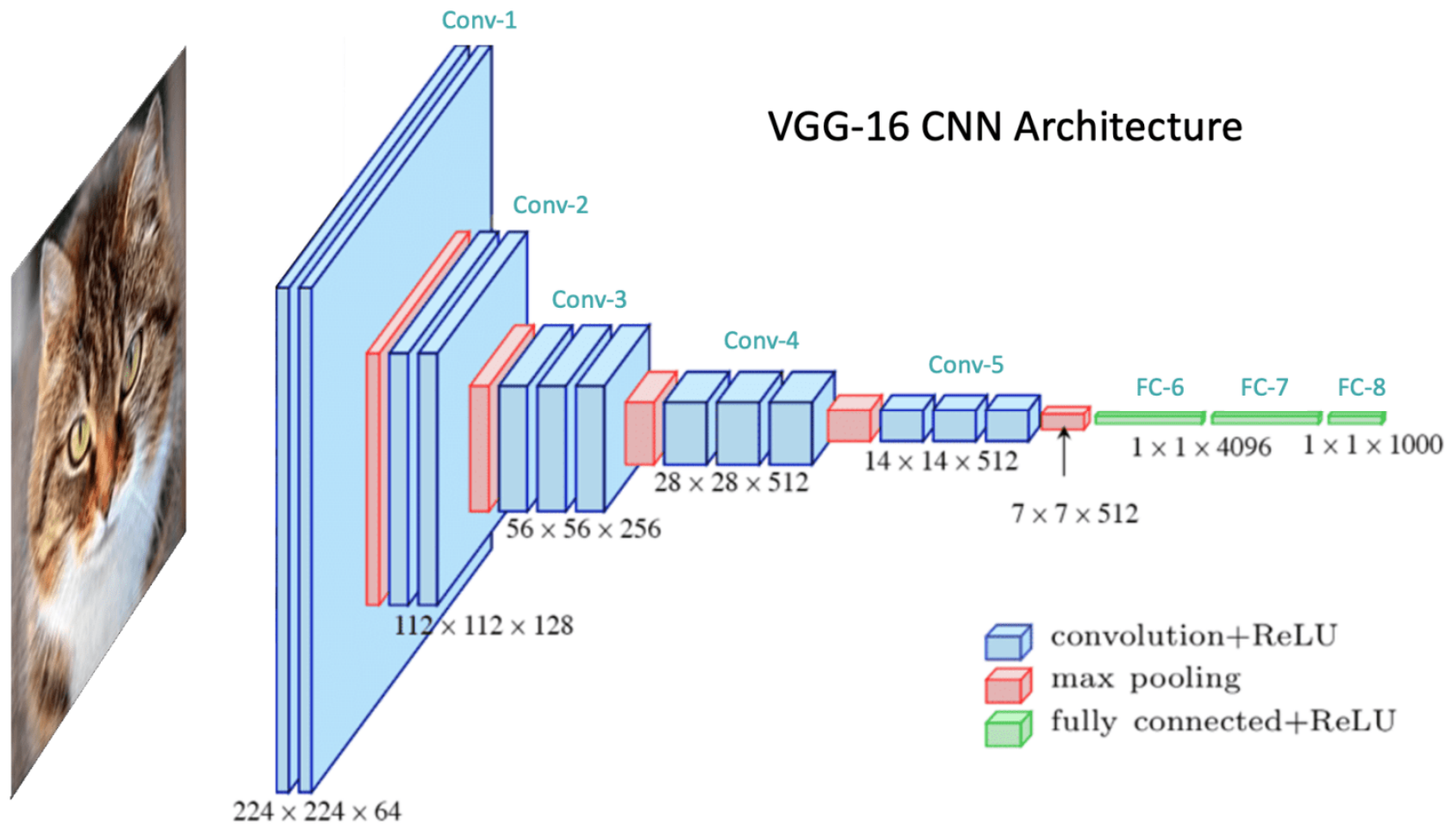
In fully connected networks, each neuron has its **own set of weights**.

In CNNs, the **same kernel (set of weights)** is applied across the whole input.

This means:

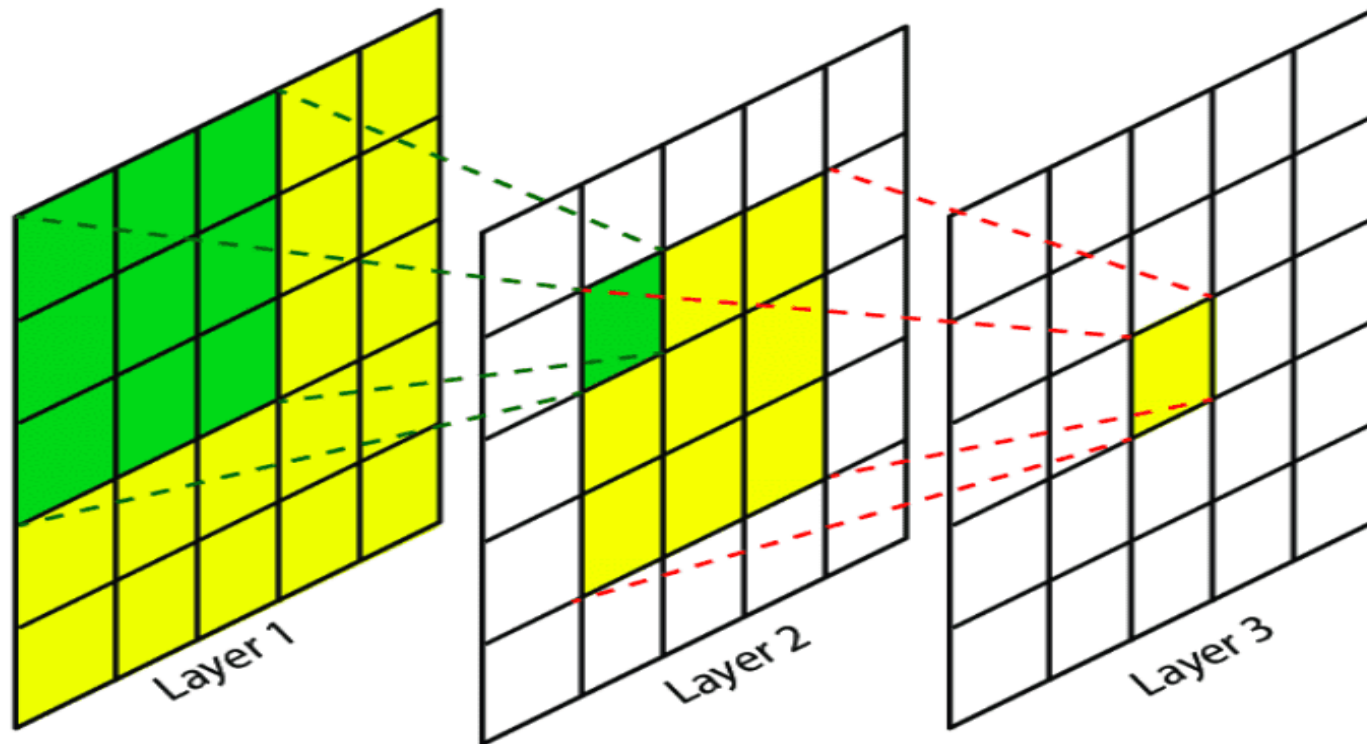
- One filter detects the **same feature** (e.g., an edge, a DNA motif) at **any position** in the input.
- Greatly reduces the number of parameters.
- Gives the network **translation invariance**: it doesn't matter where the feature occurs, only that it occurs.

Architectures



Architecture

Local receptive field

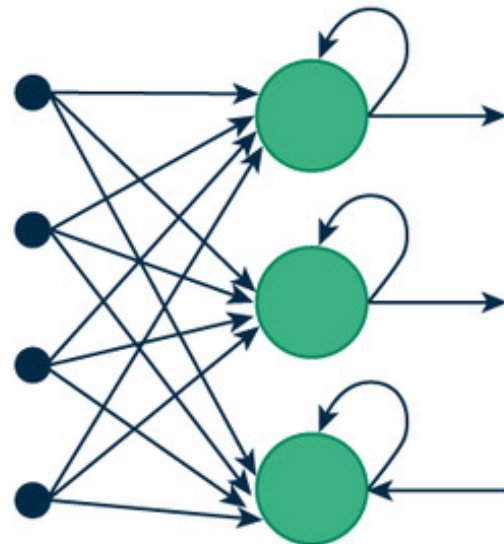


Architectures

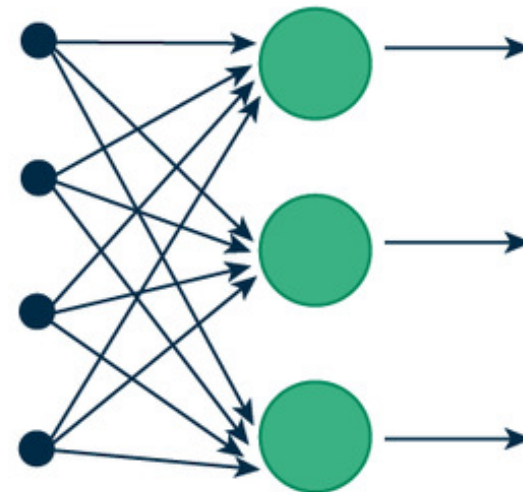
Recurrent Neural Networks (RNNs)

Motivation

- Standard neural networks assume **independent inputs**.
- Sequences (text, genomes, signals) have **dependencies over time/position**.
- RNNs introduce **recurrence**: the hidden state carries information from the past.

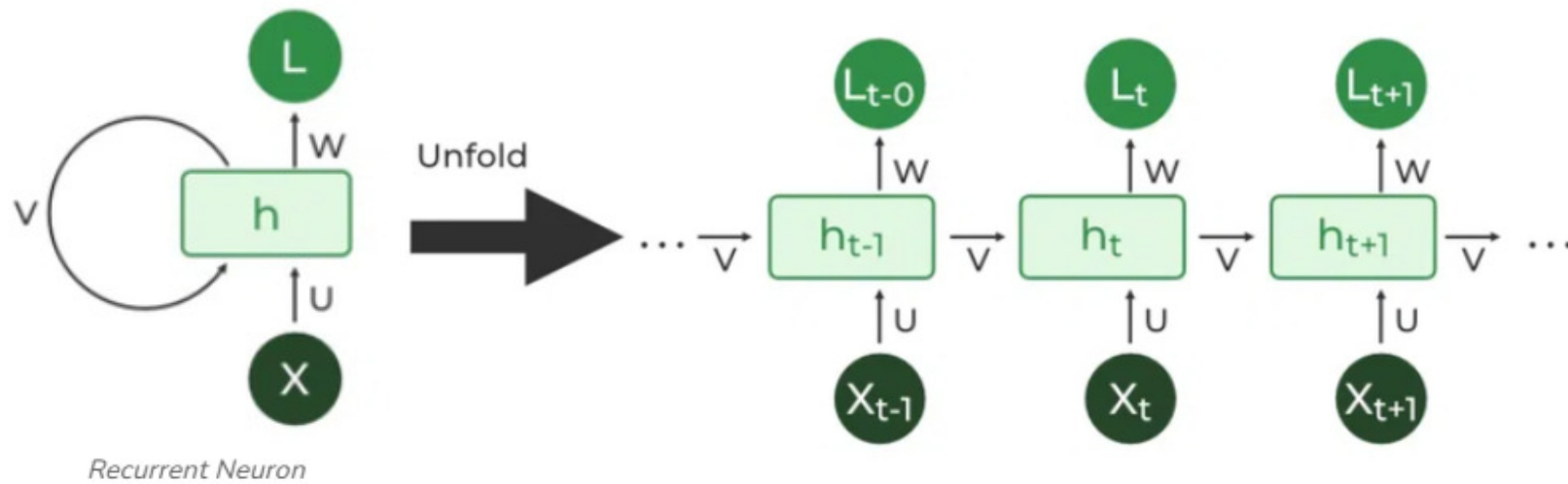


(a) Recurrent Neural Network



(b) Feed-Forward Neural Network

Architecture



1. Hidden state update

$$h_t = \tanh(UX_t + Vh_{t-1})$$

2. Output (before loss)

$$\hat{y}_t = Wh_t$$

3. Loss at each step (depends on task, e.g., classification with cross-entropy)

$$L_t = \mathcal{L}(\hat{y}_t, y_t)$$

Architecture

The problem with basic RNNs

- RNNs are trained with **Backpropagation Through Time (BPTT)**.
- At each step, gradients are multiplied by the recurrent weight matrix V .
- If eigenvalues of V are < 1 , gradients shrink exponentially → **vanishing gradients**.
- If eigenvalues of V are > 1 , gradients grow uncontrollably → **exploding gradients**.
- As a result:
 - RNNs can learn **short-term dependencies** (a few steps).
 - They fail to capture **long-term dependencies** (e.g., motifs far apart in DNA, long protein domains, long sentences).

The LSTM solution (Hochreiter & Schmidhuber, 1997)

- Introduces a **cell state** c_t that acts like a “memory conveyor belt”.
- Adds **gates** (input, forget, output) that regulate information flow.
- Crucial: the cell state allows **gradients to flow unchanged across many steps** → solves the vanishing gradient problem.
- Result:
 - Can learn **long-range dependencies**.
 - Training is more stable.
 - Became the backbone of sequence modeling for two decades (until Transformers).

Architecture

Long Short-Term Memory (LSTM)

Inputs

- Current input: x_t
- Previous hidden state: h_{t-1}
- Previous cell state: C_{t-1}

1. Forget gate

Decides what fraction of the previous cell state to keep:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

2. Input gate

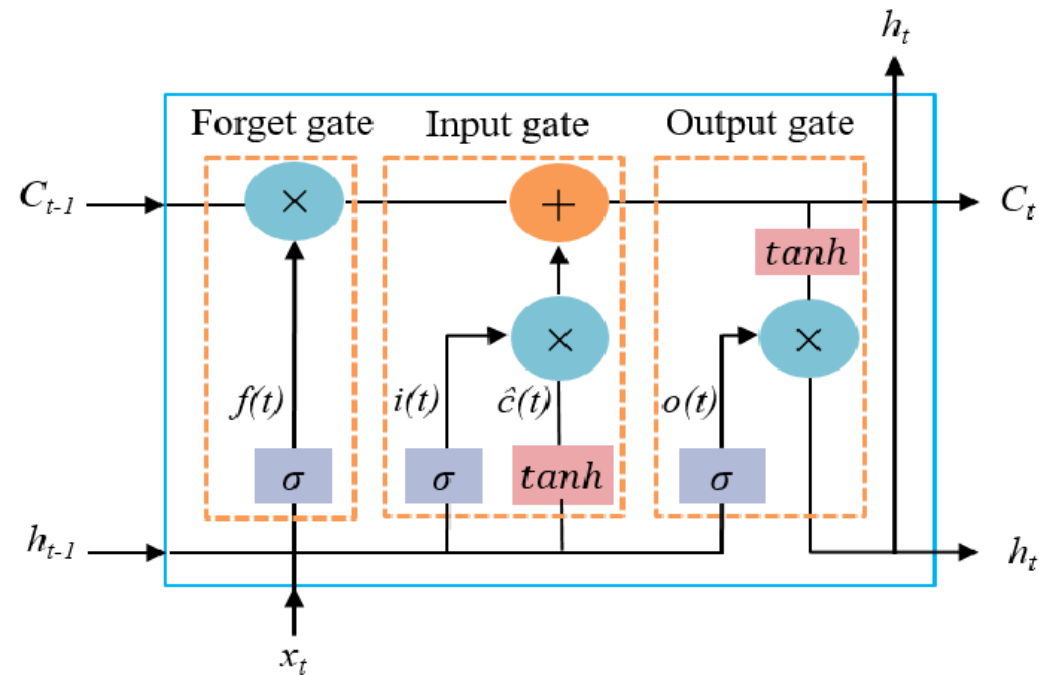
Decides how much new information to write:

- Candidate cell content:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

- Input gate activation:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$



Architecture

3. Cell state update

Combine the "old memory" and the "new candidate":

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

4. Output gate

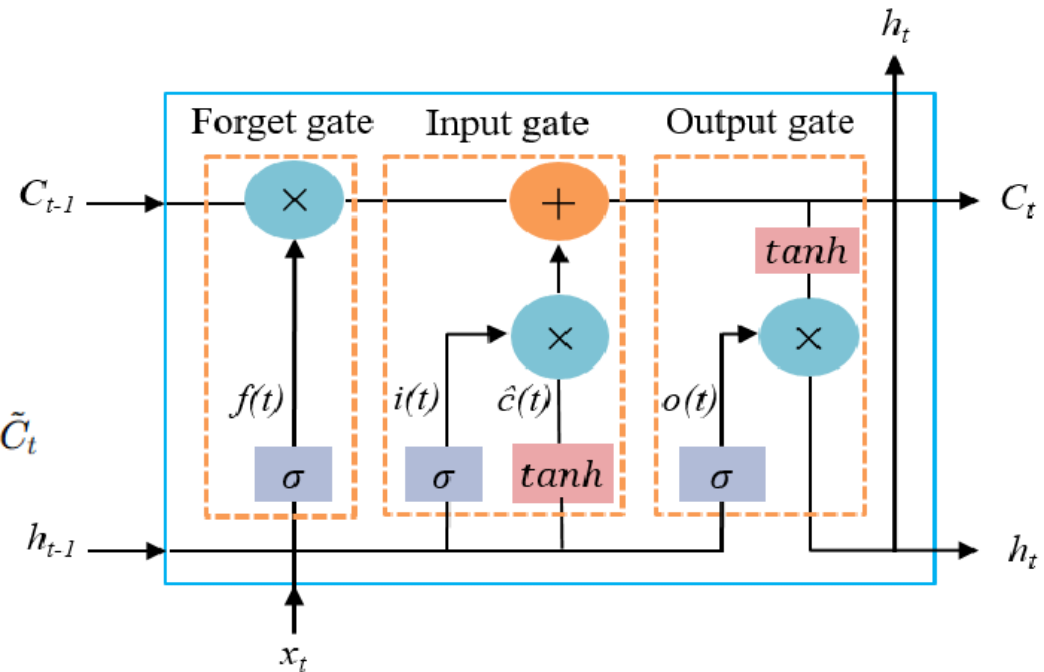
Controls how much of the cell state is exposed as hidden state:

- Output gate activation:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

- Hidden state update:

$$h_t = o_t \odot \tanh(C_t)$$



Limitations of RNNs/LSTMs

- Sequential processing: must handle tokens one after another → **slow, hard to parallelize.**
- Long-range dependencies: information must pass through many steps → still prone to forgetting.

Architecture

Transformers

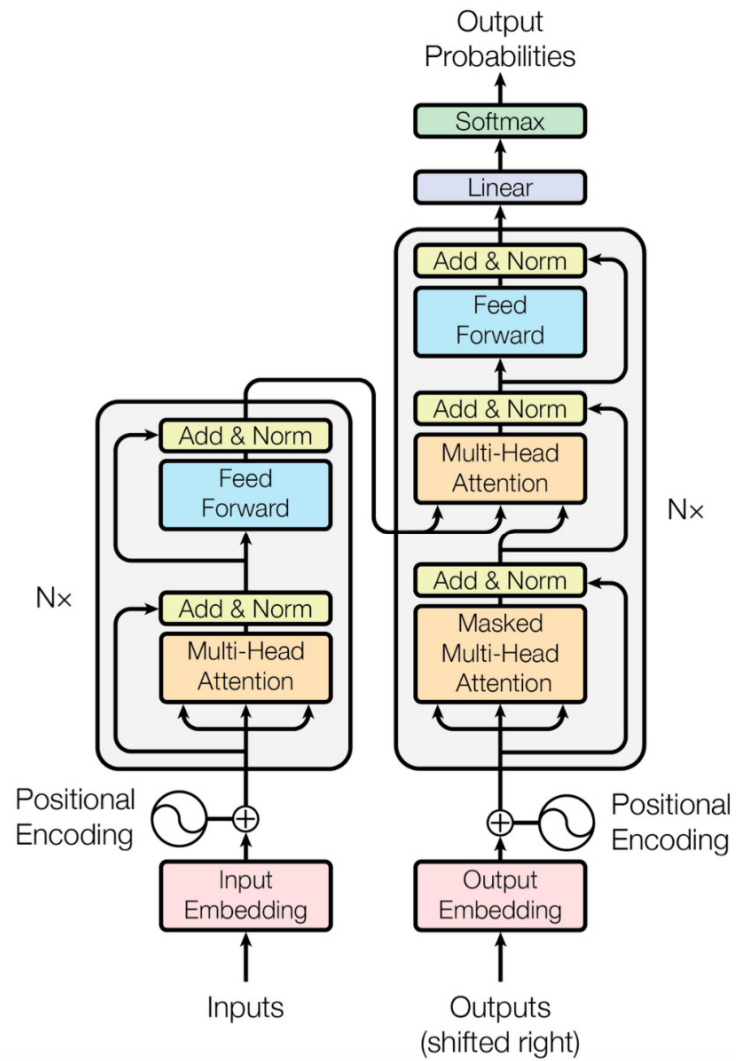
Core Idea of Transformers (Vaswani et al., 2017)

- **Attention mechanism:** each element in the sequence can **directly attend to all others**, no matter their distance.
- Replace recurrence with **parallel attention layers**.
- This enables:
 - **Parallel training** (all tokens processed at once).
 - **Direct modeling of long-range dependencies**.
 - **Scalability** to very large models (e.g., protein language models, AlphaFold, GPT).

Key Concept: “Attention is All You Need”

- Every token computes a **weighted combination of all other tokens**.
- The weights represent **relevance** (learned by the model).
- Result: flexible, global context capture.

Architecture



Attention

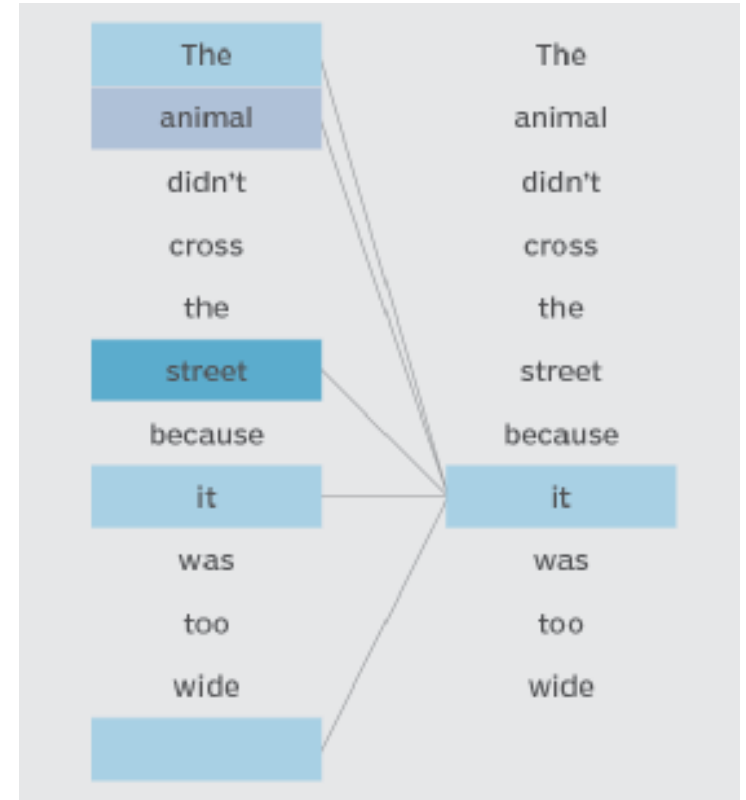
Can you me help this sentence to translate

Kannst du mir helfen diesen Satz zu uebersetzen ?

Can you help me to translate this sentence

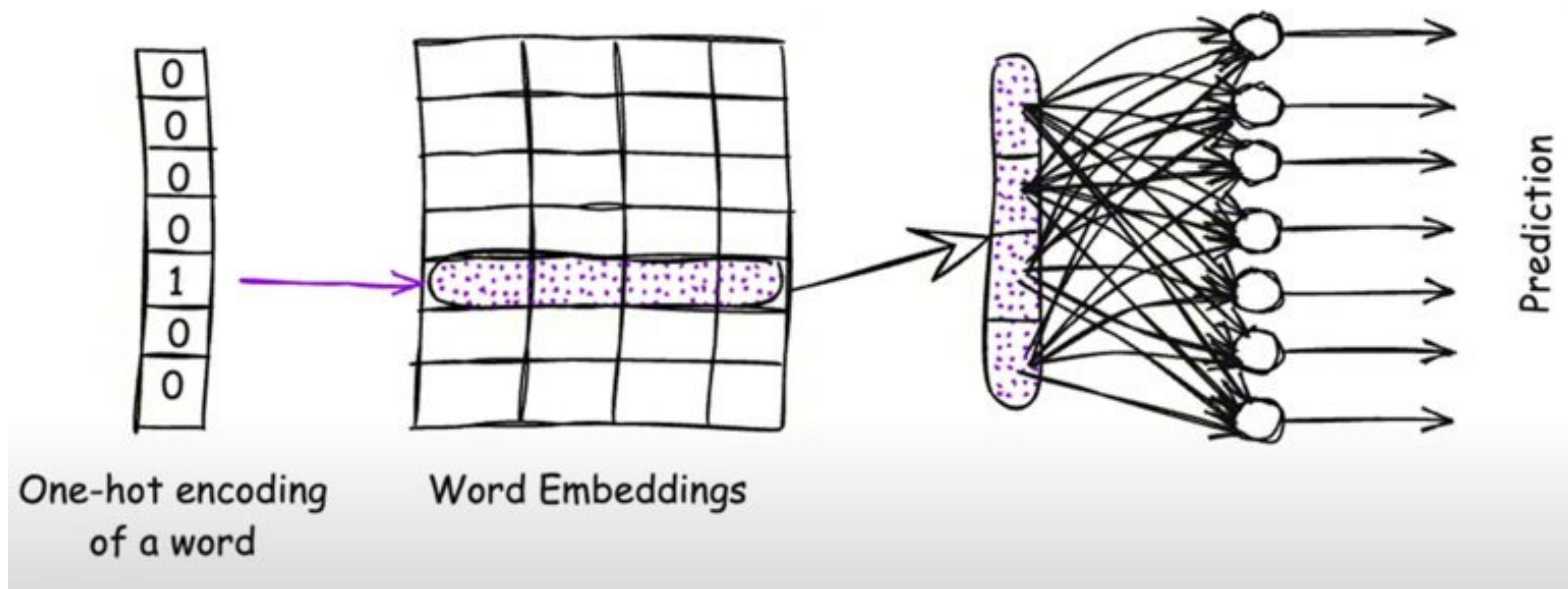
Kannst du mir helfen diesen Satz zu uebersetzen ?

We can think of self-attention as a mechanism that **enhances the information content of an input embedding by including information about the input's context**. In other words, the self-attention mechanism enables the model to weigh the importance of different elements in an input sequence and dynamically adjust their influence on the output.



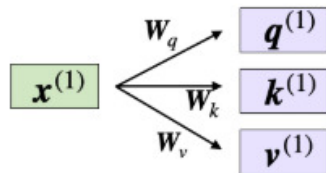
<https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html>

Self Attention, Step 1: Embedding

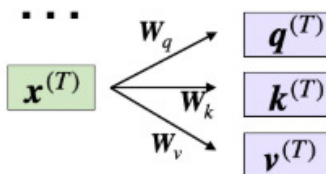
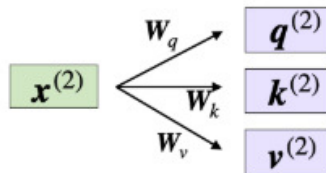


Self Attention, Step 2: Query, key, value construction

- Query sequence: $\mathbf{q}^{(i)} = \mathbf{W}_q \mathbf{x}^{(i)}$ for $i \in [1, T]$
- Key sequence: $\mathbf{k}^{(i)} = \mathbf{W}_k \mathbf{x}^{(i)}$ for $i \in [1, T]$
- Value sequence: $\mathbf{v}^{(i)} = \mathbf{W}_v \mathbf{x}^{(i)}$ for $i \in [1, T]$



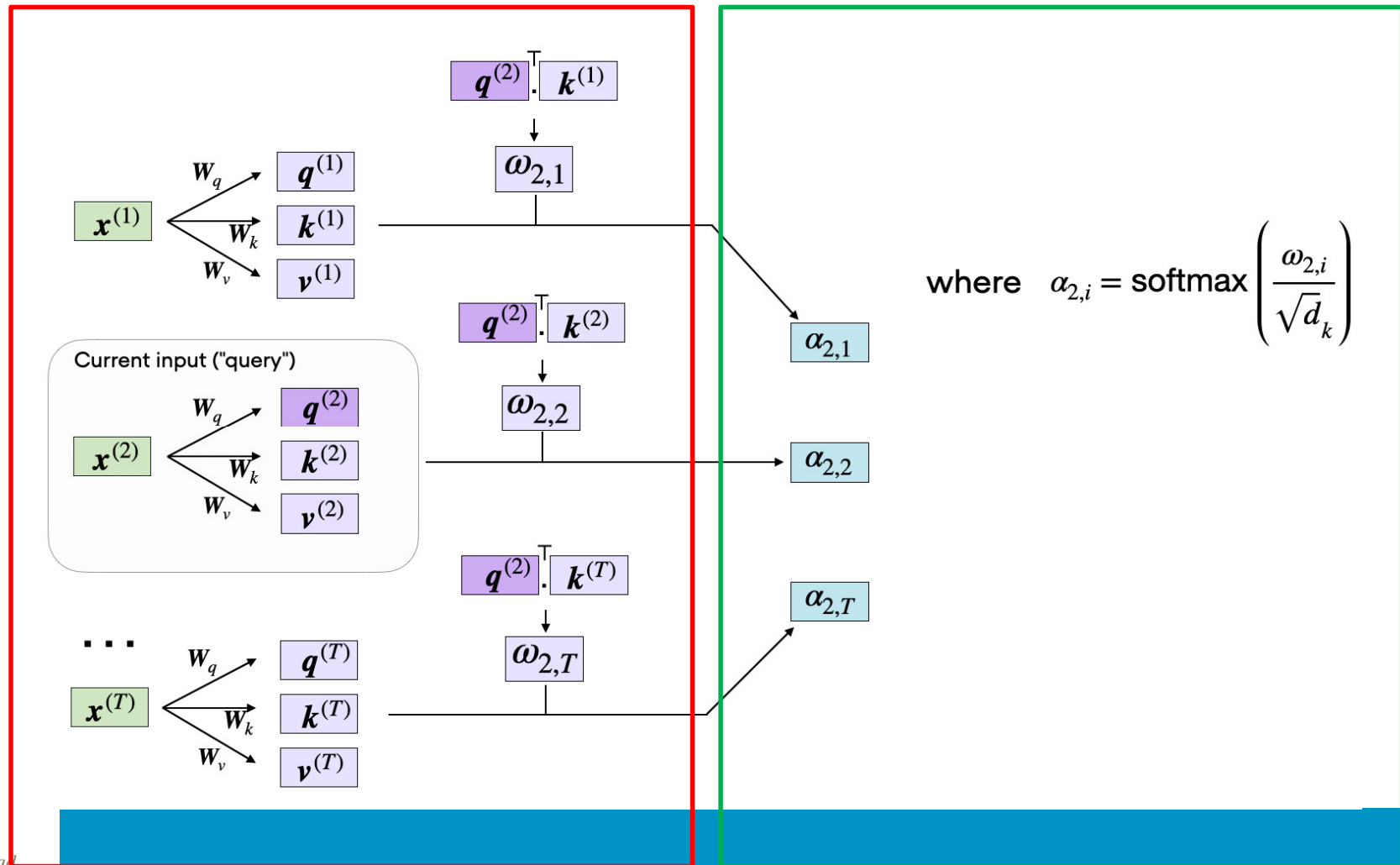
\mathbf{W}_q and \mathbf{W}_k are $d_k \times d$ matrices
 \mathbf{W}_v is a $d_v \times d$ matrix



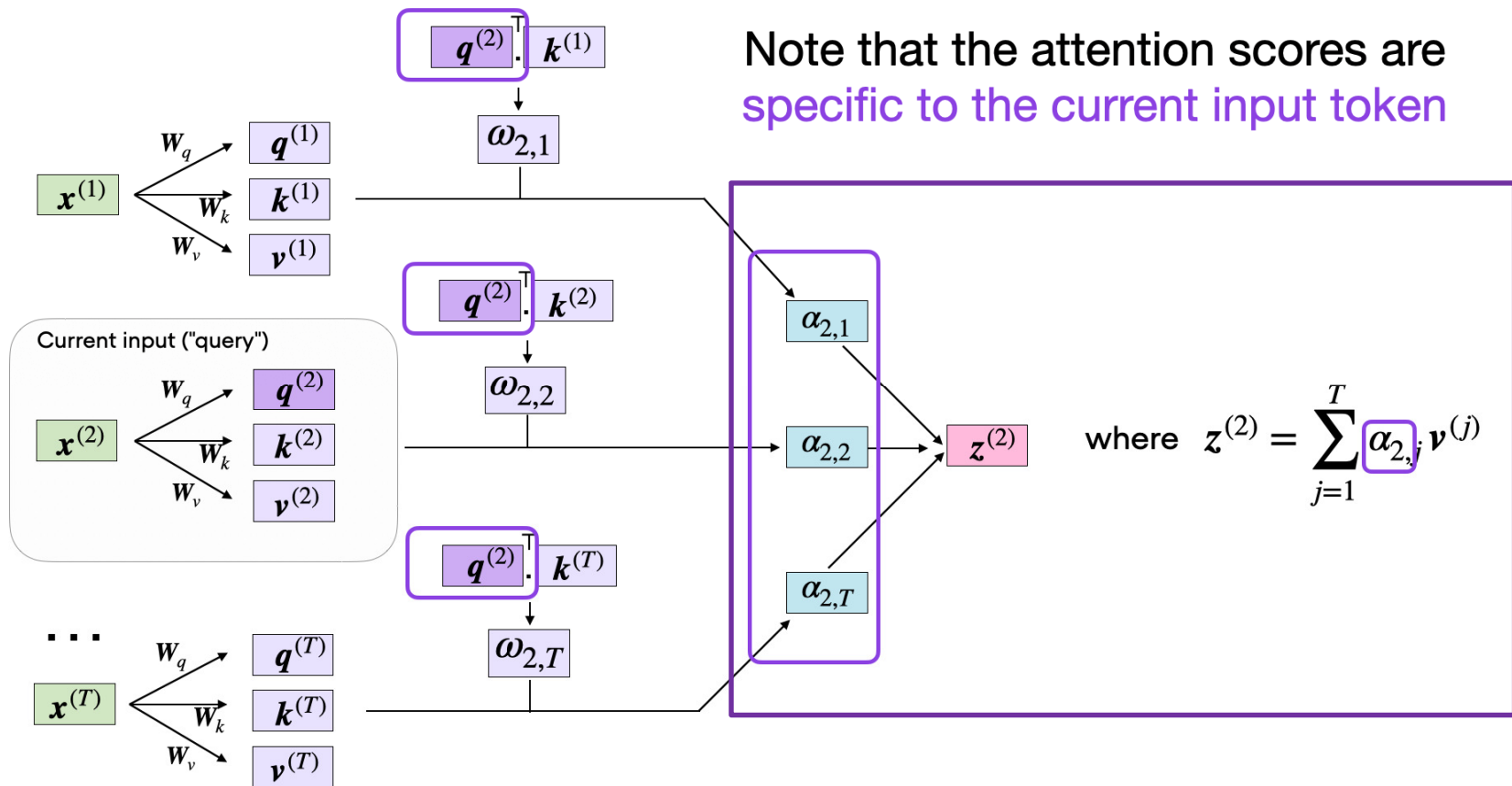
Self Attention, Step 3: Compute attention scores

Unnormalized query

Normalized attention scores



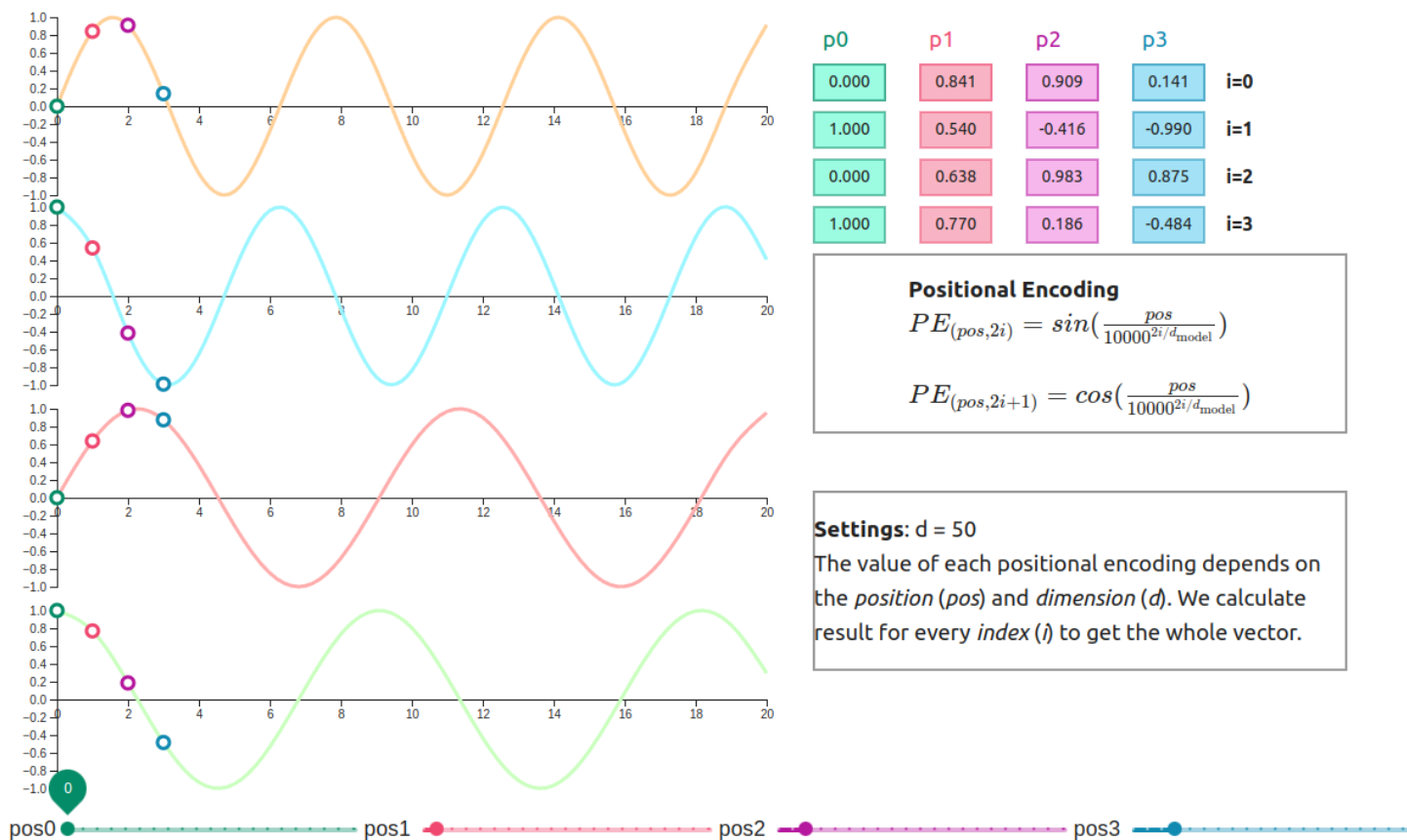
Self Attention, Step 4: Compute weighted value



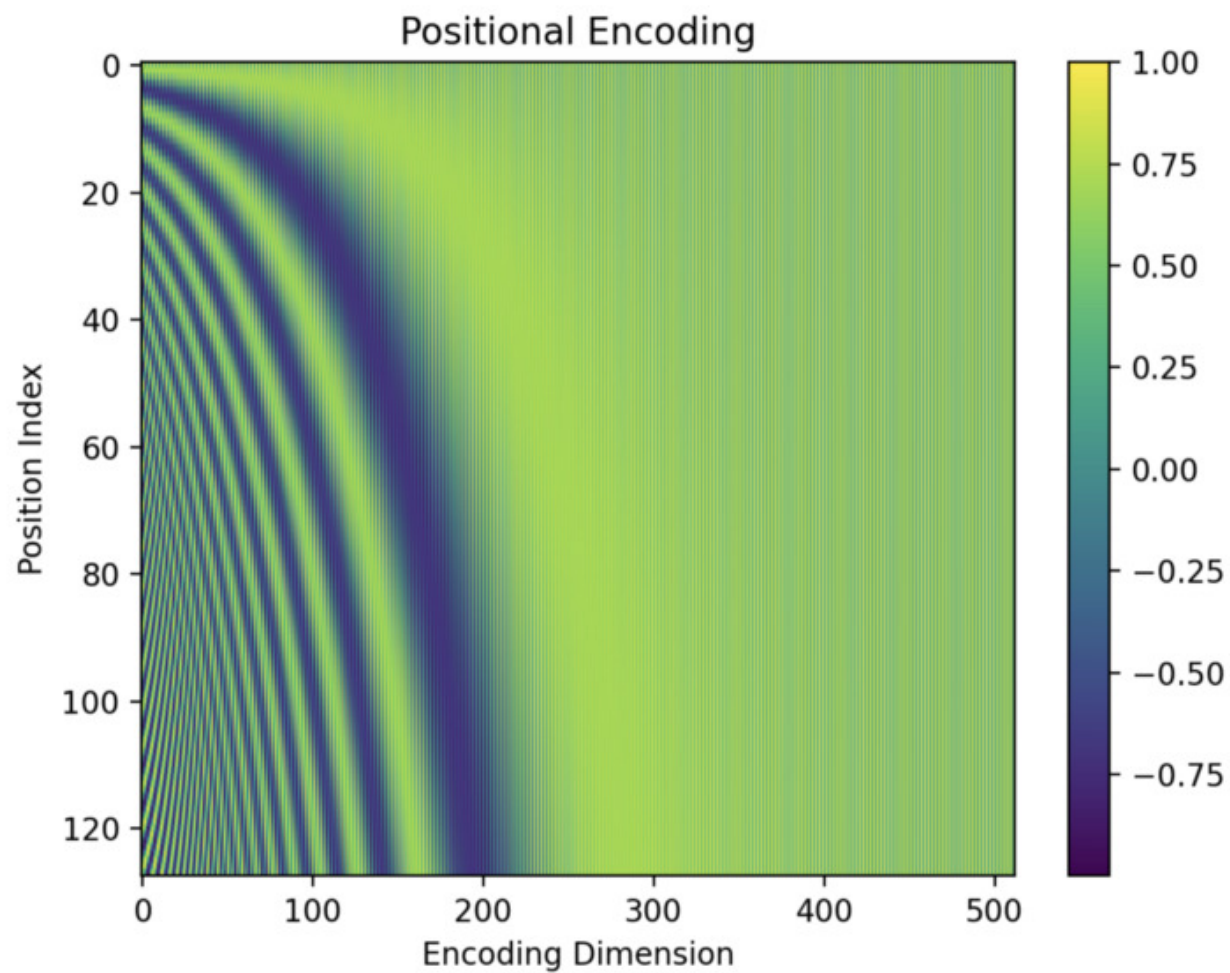
Positional encoding

Without positional encoding, the input symbols would be treated as independent tokens. Positional encoding allows the network to consider the location of the symbol within the sequence.

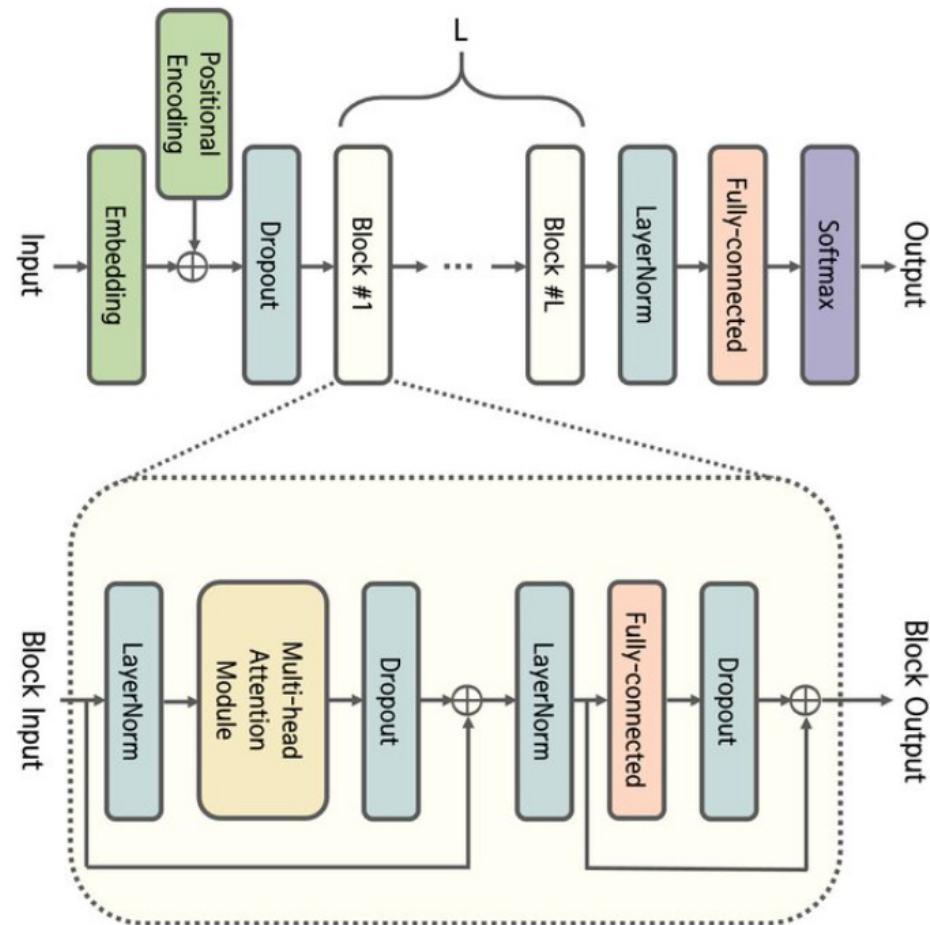
Positional encoding visualization



Positional encoding

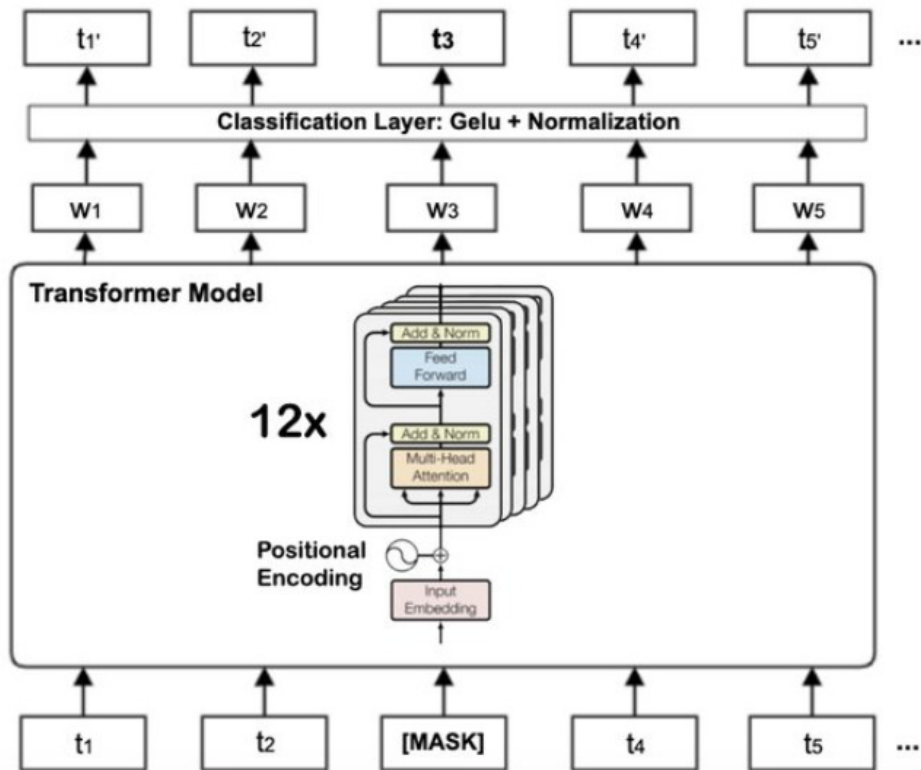


Generative Pre-trained Transformer (GPT)

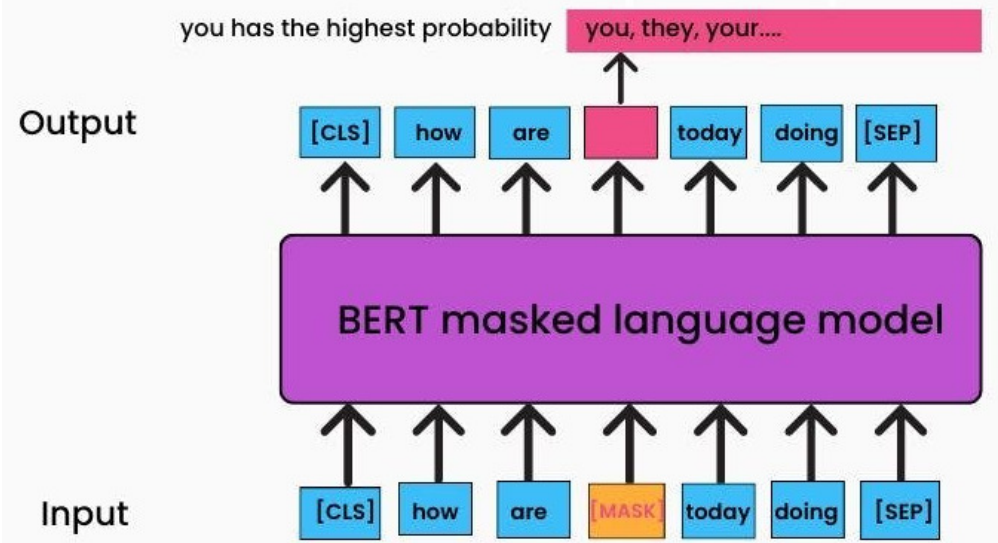


<https://www.mdpi.com/2227-7390/11/11/2451>

BERT



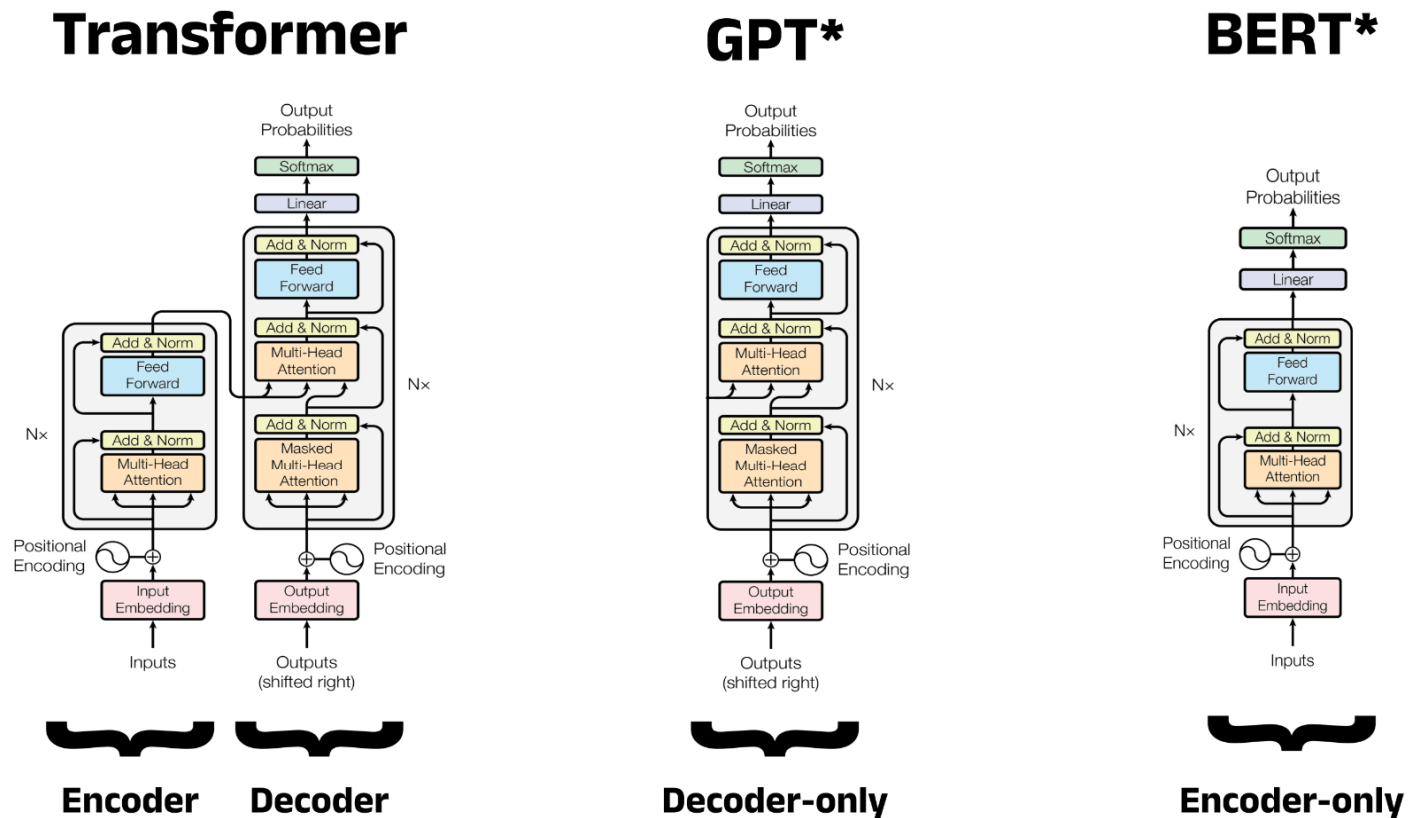
Masked language training



Next sentence training

| Sentence 1 | Sentence 2 | Next Sentence? |
|---------------------|-----------------------------|----------------|
| I am going outside. | I will be back after 6. | YES |
| I am going outside. | You know nothing John snow. | NO |

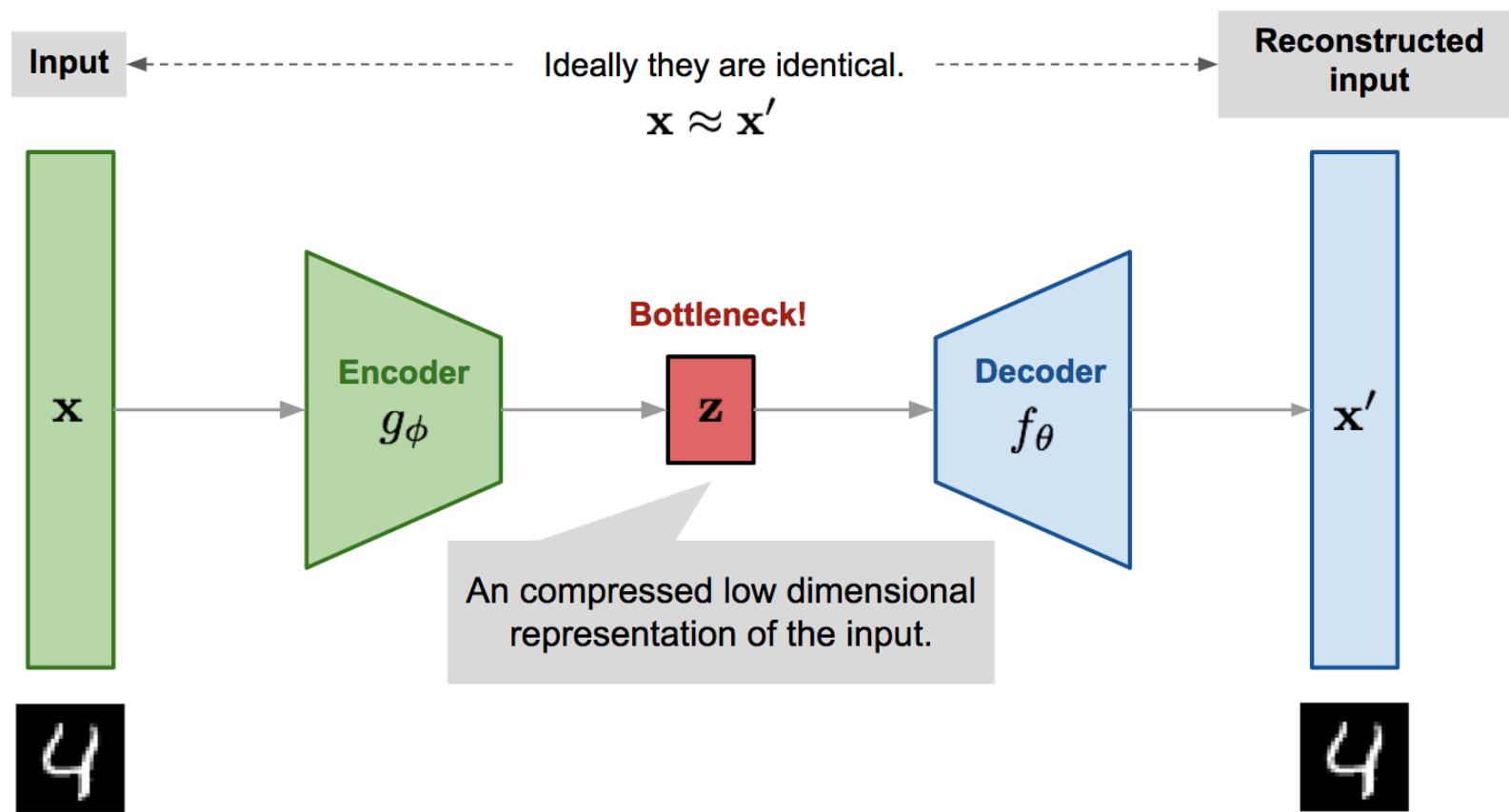
BERT



*Illustrative example, exact model architecture may vary slightly

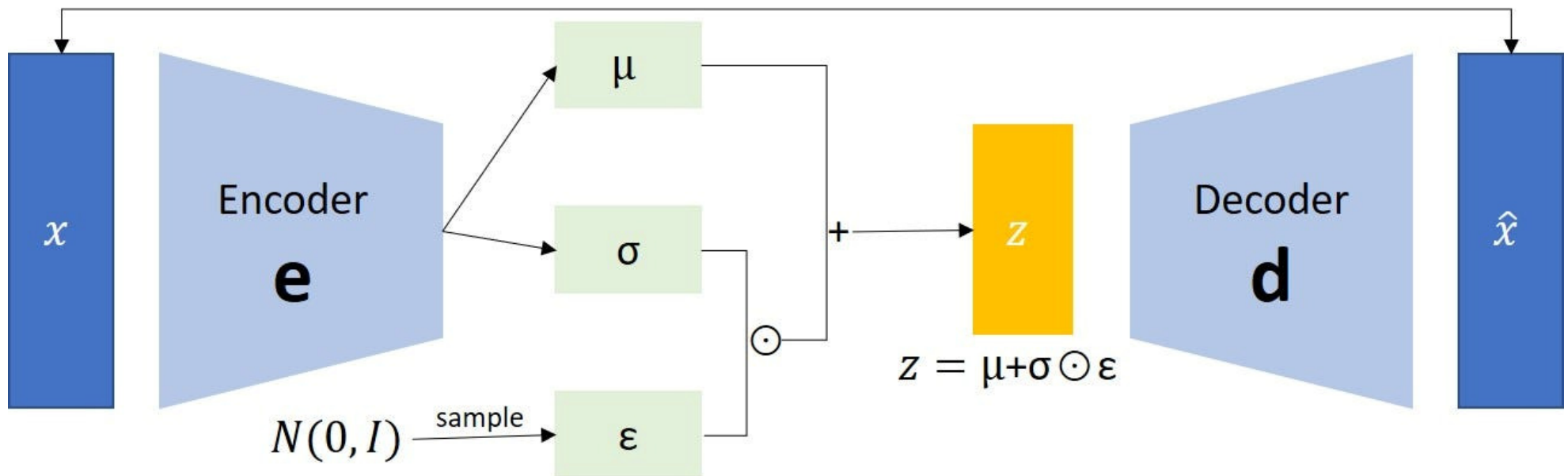
<https://towardsdatascience.com/a-complete-guide-to-bert-with-code-9f87602e4a11/>

Autoencoders



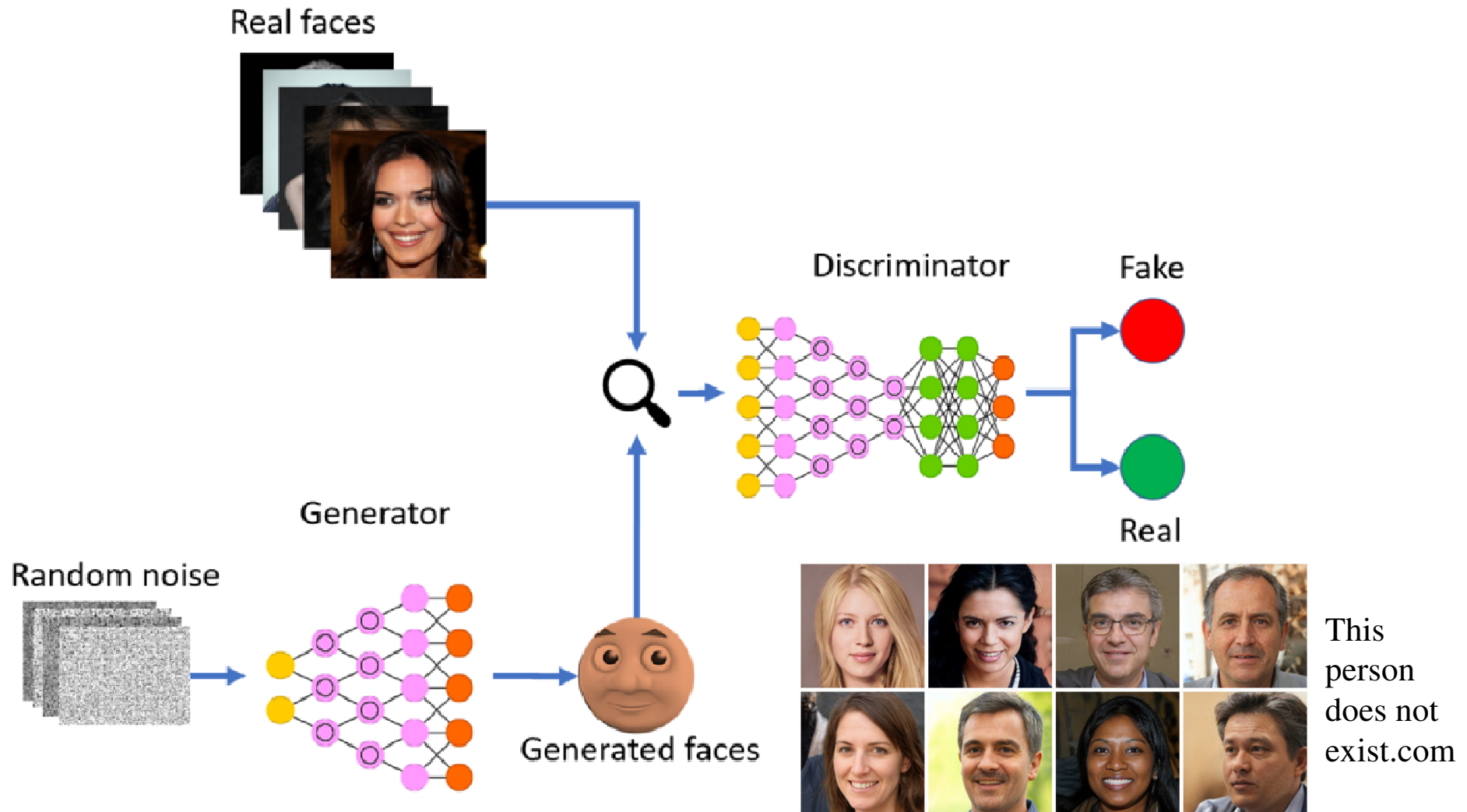
Variational autoencoders

Minimize 1: $(x - \hat{x})^2$

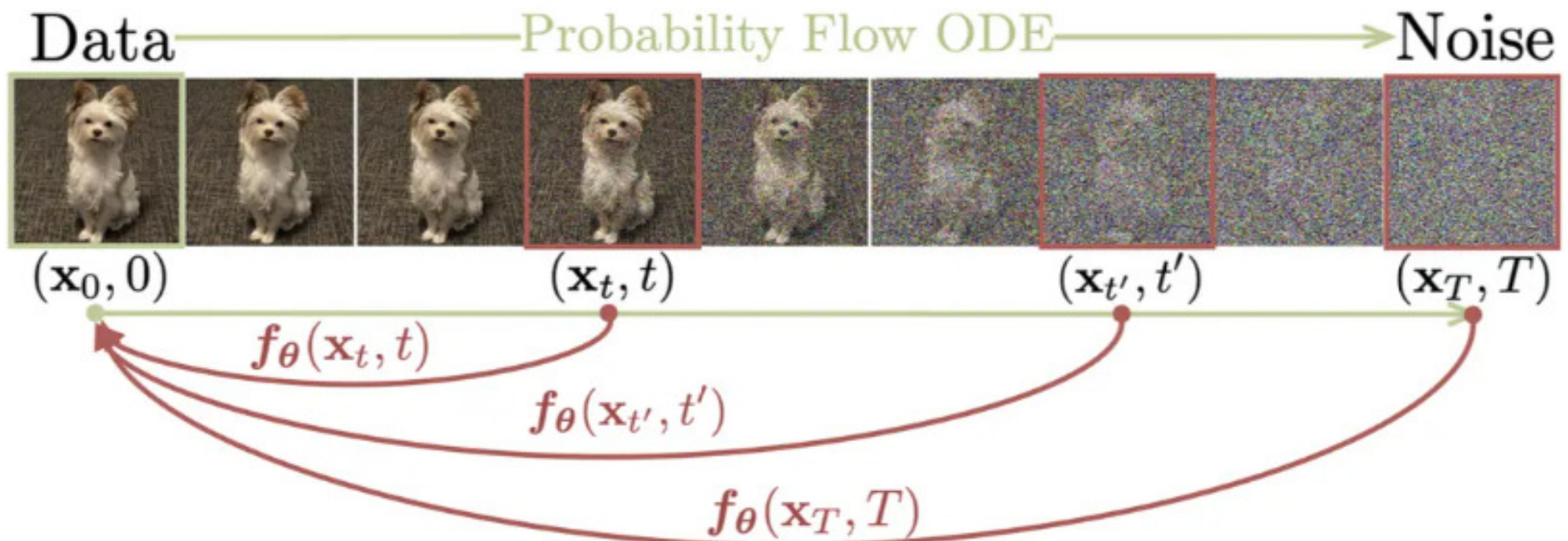


Minimize 2: $\frac{1}{2} \sum_{i=1}^N (\exp(\sigma_i) - (1 + \sigma_i) + \mu_i^2)$

Generative Adversarial Networks (GANs)



Diffusion models



Applications

1. Protein Structure Prediction & Modeling

- **AlphaFold2 / AlphaFold3 (DeepMind)**
Transformer-based architecture predicting 3D protein structures from sequences; revolutionized structural biology.
- **RoseTTAFold (UW / Baker Lab)**
Three-track network integrating sequence, distance, and coordinate information; inspired by AlphaFold but open-source.
- **OmegaFold**
Lightweight model for protein structure prediction using large-scale pretraining.
- **Graph-based protein models (e.g., GearNet, GVP-GNN)**
Neural networks operating on protein contact graphs, capturing spatial interactions.

2. Protein Language Models

- **ESM (Evolutionary Scale Modeling, Meta)**
Transformer protein language models (ESM-1b, ESM-2) trained on massive sequence datasets; used for embeddings, structure prediction, and function annotation.
- **ProtTrans (ProtBERT, ProtT5, ProtXLNet)**
Transfer of NLP architectures to proteins; useful for embedding, secondary structure prediction, mutational effect prediction.
- **ProGen**
Generative Transformer for protein design and novel enzyme discovery.

Applications

3. Genomics & Epigenomics

- **Basenji / Basenji2 (Kundaje Lab)**
Deep CNNs predicting gene expression and regulatory activity from DNA sequences.
- **Enformer (DeepMind)**
Transformer-based model extending Basenji to predict gene expression across long genomic contexts.
- **DeepSEA**
CNN predicting chromatin effects and transcription factor binding from raw sequence.
- **BPNet**
Base-pair resolution CNN for transcription factor binding profiles.

4. Drug Discovery & Chemoinformatics

- **Graph Neural Networks for molecules (MPNN, Chemprop, GIN)**
Learning molecular properties from molecular graphs.
- **Diffusion-based generative models (e.g., DiffDock)**
Predict protein–ligand docking poses.
- **MolFormer**
Transformer for SMILES-based or graph-based molecular representation learning.

Applications

5. Single-cell Omics & Systems Biology

- **scVI / totalVI (Bayesian VAE frameworks)**

Variational autoencoders modeling single-cell transcriptomics and multi-omics data.

- **CellBERT / scBERT**

Transformers embedding single-cell data and gene–cell relationships.

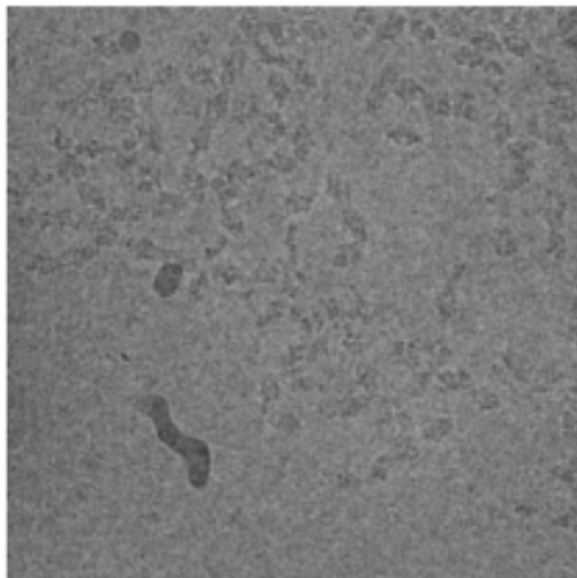
Contents

- Overview
- Basic networks
- Regularization
- Architectures
 - Convolutional, RNN, LSTM, Transformer, GPT, BERT, Autoencoders, Variational autoencoders, Generative Adversarial networks, Diffusion networks
- Applications

Practice

https://github.com/cossorzano/COSS_DataAnalysis_notebooks/blob/main/ImageProcessing/micCleaner_Unet.ipynb

Image



GT mask



Pred mask





CEU

*Universidad
San Pablo*

Lesson 9. Reinforcement learning

Medicin School

Contents

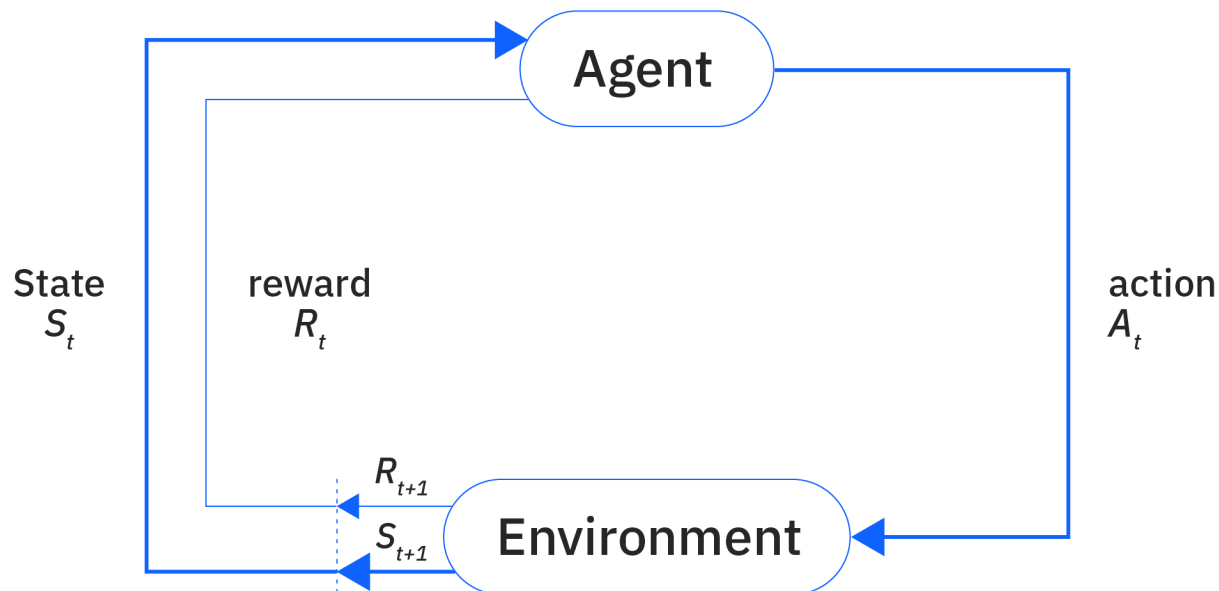
- Overview
- Multiarmed bandits
- Algorithms
 - UCB, Dynamic programming, Q-learning, SARSA, Policy gradient, Deep Q-Network
- Applications

Overview

<https://www.youtube.com/watch?v=C5LOgWMtFrY>

<https://www.youtube.com/watch?v=fBiataDpGIo>

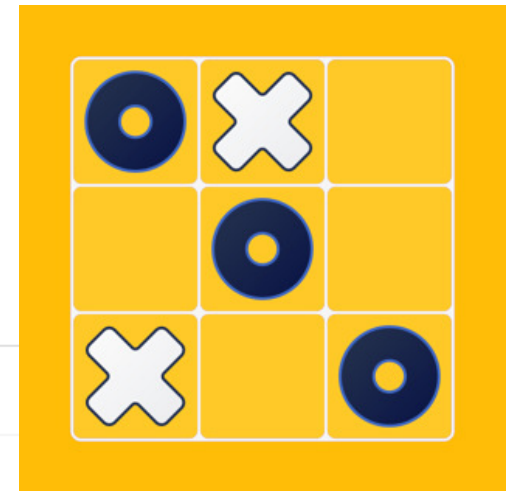
This diagram illustrates the interaction loop between an agent and its environment in a reinforcement learning setting. At each time step t , the **agent** observes the current **state** S_t of the environment and selects an **action** A_t . The **environment** responds by transitioning to a new state S_{t+1} and providing a **reward** R_{t+1} . The agent uses this feedback to improve its policy in order to maximize cumulative future rewards.



Overview

- **Learn Optimal Policies:** The primary goal is to learn a policy (a mapping from states to actions) that maximizes cumulative reward over time.
- **Maximize Long-Term Reward:** RL seeks immediate gains and long-term benefits by optimizing for the expected return, often discounted over time.
- **Balance Exploration and Exploitation:** An RL agent must explore new actions to discover their potential while exploiting known actions that yield high rewards.
- **Adapt to Dynamic Environments:** RL systems aim to learn and adapt their behavior through interaction in possibly changing or unknown environments.
- **Learn from Interaction Without Supervision:** Unlike supervised learning, RL does not rely on labeled input/output pairs; instead, it learns through trial and error by interacting with an environment.

Overview



| RL Concept | Tic-Tac-Toe Example |
|------------------|--|
| Agent | The player learning to play (could be X or O). |
| Environment | The game board and the rules of tic-tac-toe. |
| State S_t | The current configuration of the board (e.g., 3×3 grid with Xs, Os, and empty cells). $3^9 = 19683$ states |
| Action A_t | The move chosen by the agent (i.e., placing an X or O in an empty cell). |
| Reward R_{t+1} | Numerical feedback: +1 for a win, 0 for a draw, -1 for a loss, or sometimes 0 for non-terminal moves. |
| Policy π | The agent's strategy for selecting actions based on the current state. |
| Episode | A complete game from an empty board to a win, loss, or draw. |
| Goal | Learn a policy that maximizes the expected reward (i.e., learns to win or draw rather than lose). |

Overview

How It Works in Practice

1. **Initial State:** The board is empty.
2. **Agent Makes a Move:** Based on its policy, the agent picks an action (e.g., put X in the center).
3. **Environment Responds:** The opponent (could be another agent, a fixed strategy, or human) plays, updating the board.
4. **State Transitions:** The board changes to a new configuration.
5. **Reward Given:** After a terminal state (win/loss/draw), a reward is given.
6. **Learning:** The agent updates its policy using learning algorithms (like Monte Carlo methods or Q-learning) to improve performance over time.

Multiarmed bandits



Imagine you have several slot machines, each giving random rewards from unknown distributions. Some pay off more often than others, but you don't know which ones. The challenge is to pull the levers in a way that maximizes your total reward over time, balancing exploration of new machines with exploitation of those that seem best.

<https://gibberblot.github.io/rl-notes/single-agent/multi-armed-bandits.html>

Multiarmed bandits

In the **k-armed bandit problem**, you are faced with k different actions (or "arms"), each associated with an unknown **expected reward**. The true value of an action a is denoted by:

$$q^*(a) = \mathbb{E}[R_t \mid A_t = a]$$

This is the expected value (mean) of the reward R_t received at time t , given that you selected action $A_t = a$. Since $q^*(a)$ is unknown, the agent maintains an **estimate** of it, denoted by $Q_t(a)$, which is updated over time based on observed rewards.

At each time step t , the agent can either:

- **Exploit:** Choose the action a with the highest estimated value $Q_t(a)$ (a *greedy action*), aiming to maximize immediate reward.
- **Explore:** Choose a non-greedy action, which might have a lower $Q_t(a)$, in order to gather more information and potentially improve future estimates.

This creates a **trade-off** between:

- **Short-term gain** (exploitation): Selecting the action believed to be best now.
- **Long-term gain** (exploration): Trying other actions to reduce uncertainty and possibly discover better options.

The optimal strategy for this trade-off depends on factors such as the accuracy of the current estimates $Q_t(a)$, the variance in rewards, and the number of remaining steps. This is a fundamental challenge in reinforcement learning and is addressed by various algorithms (e.g., ϵ -greedy, UCB, Thompson sampling).

Multiarmed bandits

Action-value methods

Action-value methods are strategies for estimating the value of each action a , denoted by $q^*(a) = \mathbb{E}[R_t \mid A_t = a]$, and using these estimates to guide action selection.

A basic way to estimate $q^*(a)$ is through the **sample average** of past rewards received when action a was taken:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{I}\{A_i = a\}}{\sum_{i=1}^{t-1} \mathbb{I}\{A_i = a\}},$$

where $\mathbb{I}\{A_i = a\}$ is an indicator function that equals 1 if action a was taken at time i , and 0 otherwise. If action a has never been selected before time t , $Q_t(a)$ is initialized to a default value (e.g., 0). By the law of large numbers, $Q_t(a) \rightarrow q^*(a)$ as the number of times a is selected approaches infinity.

Multiarmed bandits

Action-value methods: ϵ -greedy strategy

To **select actions**, the **greedy strategy** chooses the action with the highest current estimate:

$$A_t = \arg \max_a Q_t(a).$$

This method **exploits** current knowledge to maximize immediate reward but never explores other actions.

To balance **exploration and exploitation**, the **ϵ -greedy method** selects the greedy action with probability $1 - \epsilon$, and with small probability ϵ , it chooses an action uniformly at random:

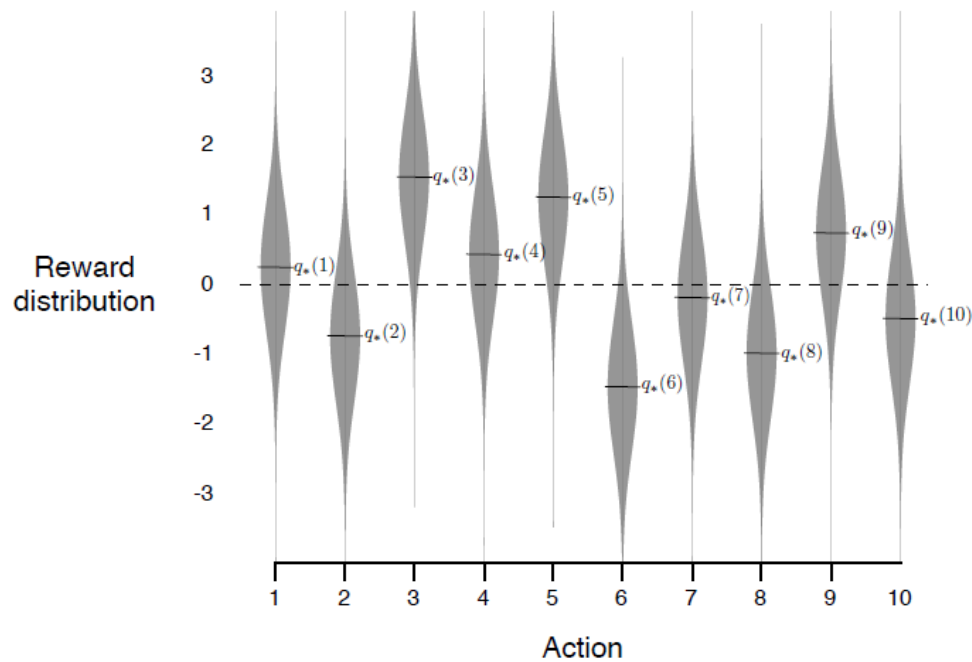
- Ensures all actions are tried infinitely often (in expectation).
- Guarantees that $Q_t(a) \rightarrow q^*(a)$ for all a , and that the probability of selecting the optimal action converges to at least $1 - \epsilon$.

Though these guarantees are **asymptotic**, ϵ -greedy remains simple and effective in practice for many problems.

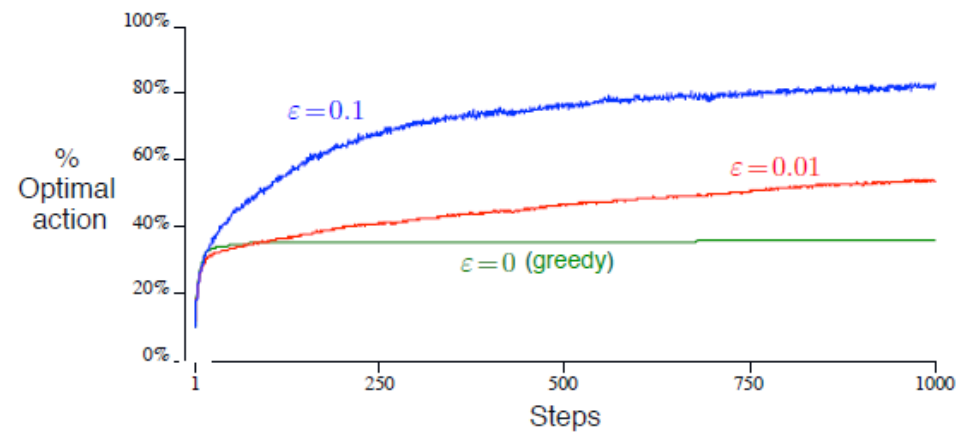
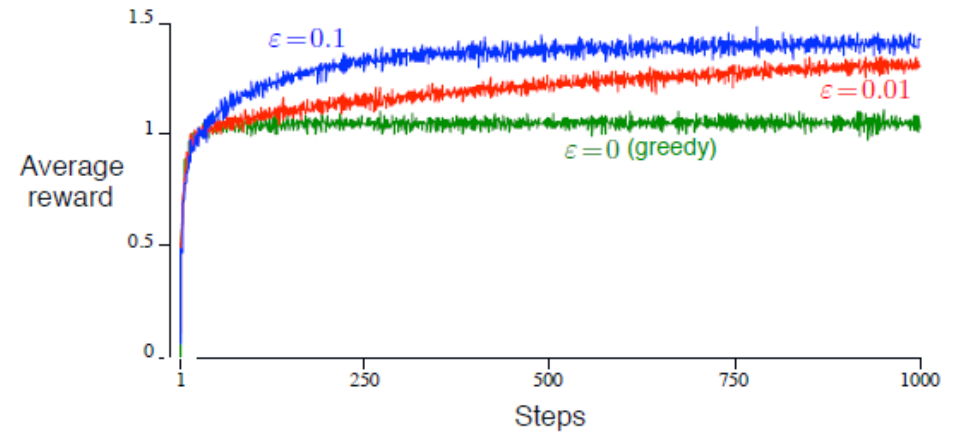
Multiarmed bandits

Action-value methods: ϵ -greedy strategy

True distribution



The first 10 actions are exploratory.



Multiarmed bandits

Action-value methods: bandit algorithm

Stationary bandits

A simple bandit algorithm

Initialize, for $a = 1$ to k :

$$Q(a) \leftarrow 0$$

$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \quad (\text{breaking ties randomly}) \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$$R \leftarrow \text{bandit}(A)$$

$$N(A) \leftarrow N(A) + 1$$

$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$$

Non-stationary bandits

Initialize, for $a = 1$ to k :

$$Q(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$$R \leftarrow \text{bandit}(A)$$

$$Q(A) \leftarrow Q(A) + \alpha [R - Q(A)]$$

Algorithms

Action-value methods: UCB algorithm

Upper-Confidence-Bound (UCB) Action Selection

UCB is a method that balances **exploration** and **exploitation** by selecting actions based not only on their estimated value $Q_t(a)$, but also on the **uncertainty** or **potential** for that estimate to improve.

At each time step t , the action selected is:

$$A_t = \arg \max_a \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

where:

- $Q_t(a)$ is the estimated value of action a at time t ,
- $N_t(a)$ is the number of times action a has been selected up to time t ,
- $\ln t$ grows slowly over time and promotes exploration,
- $c > 0$ is a tunable parameter that controls the degree of exploration.

Algorithms

Dynamic programming

Dynamic Programming (DP) refers to a class of algorithms for solving **Markov Decision Processes (MDPs)** when the full model of the environment is known. That is, the **transition probabilities** $p(s', r \mid s, a)$ are available for all states $s \in \mathcal{S}$, actions $a \in \mathcal{A}(s)$, next states $s' \in \mathcal{S}_+$, and rewards $r \in \mathcal{R}$.

The key idea in DP is to use **value functions** to iteratively compute optimal or policy-specific behavior.

1. Bellman Equations for Optimality

The optimal state-value function $v^*(s)$ satisfies:

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a] = \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v^*(s')]$$

The optimal action-value function $q^*(s, a)$ satisfies:

$$q^*(s, a) = \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q^*(s', a') \right]$$

$$v^*(s) = \max_a q^*(s, a)$$

These form the **Bellman optimality equations**, which define fixed points that DP algorithms aim to approximate.

Algorithms

Dynamic programming

2. Policy Evaluation (Prediction)

Given a policy $\pi(a \mid s)$, the **state-value function** $v^\pi(s)$ is:

$$v^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v^\pi(s')]$$

We can **approximate** v^π iteratively by computing a sequence v_0, v_1, \dots , using the update:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')]$$

This is called **iterative policy evaluation**.

Algorithms

Dynamic programming

Policy Iteration: Policy Evaluation + Policy Improvement

Input:

- Initial policy π
- Transition model $p(s', r \mid s, a)$
- Small threshold $\theta > 0$ for evaluation accuracy
- Discount factor $\gamma \in [0, 1]$

Initialize:

- Arbitrary $V(s)$ for all $s \in \mathcal{S}^+$, with $V(\text{terminal}) = 0$

Loop (until policy is stable):

1. Policy Evaluation:

- Repeat until $\Delta < \theta$:
 - $\Delta \leftarrow 0$
 - For each $s \in \mathcal{S}$:
 - $v \leftarrow V(s)$
 - $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma V(s')]$
 - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

2. Policy Improvement:

- $\text{policy_stable} \leftarrow \text{True}$
- For each $s \in \mathcal{S}$:
 - $\text{old_action} \leftarrow \pi(s)$
 - $\pi(s) \leftarrow \arg \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma V(s')]$
 - If $\pi(s) \neq \text{old_action}$, then set $\text{policy_stable} \leftarrow \text{False}$

Until policy_stable is True

Algorithms

What is Q-Learning?

Q-Learning is one of the simplest and most widely used **reinforcement learning** algorithms. It teaches an agent **how good each action is in each state** so it can act optimally, even without knowing the environment's dynamics.

- **Goal:** Learn an action-value function $Q(s, a)$ — the expected cumulative reward starting in state s , taking action a , and then following the best policy.
- **Key feature: Off-policy** — the agent can explore using one policy (e.g., ϵ -greedy) but updates as if it followed the optimal policy afterwards.

Core Idea

For each state–action pair (s, a) , Q-learning updates its estimate of the long-term return by combining:

- the **immediate reward** received after taking a in s ,
- the **best future Q-value** from the next state.

This is based on the **Bellman optimality equation**.

Key Advantages

- Simple, intuitive.
- Works without a model of the environment (model-free).
- The basis for many modern RL algorithms, including **DQN**.

Exploration vs Exploitation

- **Exploration:** Try new actions to discover rewards.
- **Exploitation:** Choose the action with highest Q-value.
- Balance via ϵ -greedy or other strategies.

Algorithms

Update Rule (Tabular Case)

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Where:

- s : current state
- a : action taken
- r : reward received
- s' : next state
- α : learning rate
- γ : discount factor (0–1)
- $\max_{a'} Q(s', a')$: best future action value

The Learning Loop

1. Initialize $Q(s, a)$ arbitrarily.
2. Observe the current state s .
3. Choose an action a (ϵ -greedy).
4. Take the action, observe reward r and new state s' .
5. Update $Q(s, a)$ using the rule above.
6. Repeat until convergence.

Why It Works

- Q-learning converges to the **optimal Q-function** Q^* under mild conditions (sufficient exploration, decreasing learning rate).
- Once Q^* is known, the **optimal policy** is simply:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

Algorithms

What is SARSA?

SARSA (State–Action–Reward–State–Action) is a **value-based, on-policy** reinforcement learning algorithm. Like Q-learning, it estimates $Q(s, a)$, but instead of updating towards the *best possible* next action, it updates towards the **actual action the policy chose** in the next state.

Core Idea

- Learn an action-value function $Q(s, a)$.
- Update using the **sequence actually experienced**:
 (s, a, r, s', a') .
- Thus, SARSA learns the value of the **current policy** (on-policy), not an implicitly optimal one.

Why Use SARSA?

- **On-policy**: Learns about the behavior you're actually executing (important if exploration has significant cost).
- **Safer learning**: Better in environments where "risky" exploratory actions might lead to large penalties.
- **Smooth transition**: Conceptually close to Q-learning but highlights the on/off policy distinction.

Algorithms

Update Rule (Tabular Case)

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$$

Where:

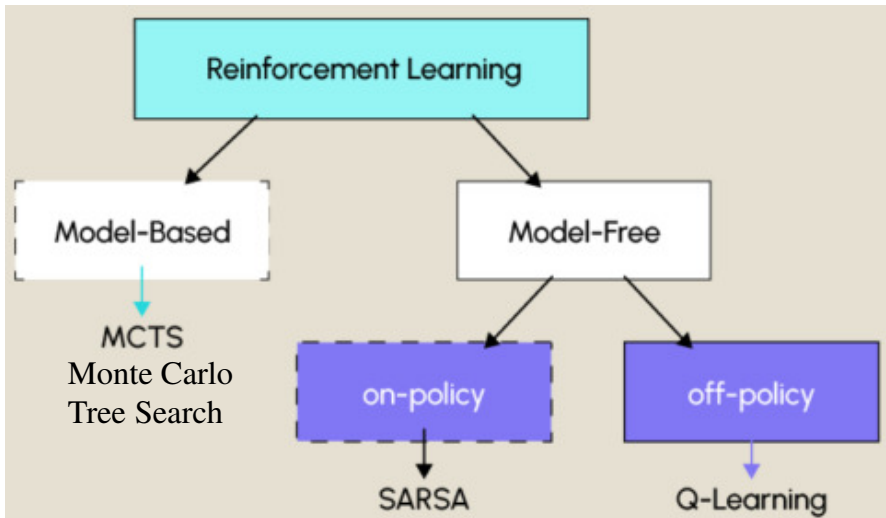
- s : current state
- a : action taken
- r : reward received
- s' : next state
- a' : next action chosen by the same policy
- α : learning rate
- γ : discount factor

The Learning Loop

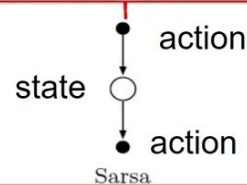
1. Initialize $Q(s, a)$ arbitrarily.
2. Observe initial state s .
3. Choose an action a from s (ϵ -greedy).
4. Take action, observe reward r and next state s' .
5. Choose next action a' from s' using the *same policy*.
6. Update $Q(s, a)$ using the SARSA rule.
7. Set $s \leftarrow s', a \leftarrow a'$ and repeat.

Algorithms

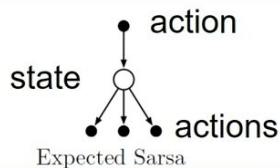
Comparison SARSA vs Q-learning



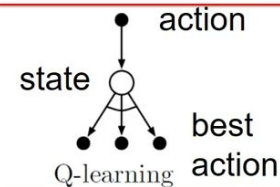
Summary: SARSA and related algorithms



SARSA: you actual perform next action, according to the policy, and then you update $Q(s,a)$



Exp. SARSA: you look ahead and average over **potential next** actions and then you update $Q(s,a)$



Q-learning: you look ahead and **imagine greedy next** action to update $Q(s,a)$ (but you then perform the actual next action based on your current policy)

Algorithms

✴ Introduction to Policy Gradient Methods

From Value Functions to Policies

- **Value-based methods** (Q-learning, SARSA) learn a **value function** (e.g., $Q(s, a)$) and derive a policy indirectly by choosing actions with the highest estimated value.
- **Policy Gradient methods** skip this step and **directly learn the policy** $\pi_\theta(a|s)$, parameterized by θ (often a neural network).

Why Use Policy Gradients?

- Handle **continuous action spaces** naturally (no need to pick max over discrete actions).
- Learn **stochastic policies** directly (important for exploration and uncertainty).
- Flexible: can incorporate constraints, prior knowledge, or complex architectures.

When to Use Policy Gradients

- **Continuous control** tasks (adjusting lab conditions, dosing, tuning hyperparameters).
- **Stochastic decision making** (sampling molecules, simulating biological systems).
- When value-based methods become impractical due to large/discrete action spaces.

Algorithms

Objective

Maximize the expected return under the policy:

$$J(\theta) = \mathbb{E}_{\pi_{\theta}}[R]$$

We perform **gradient ascent** on $J(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

Policy Gradient Theorem

It provides a way to compute the gradient of $J(\theta)$ without knowing the environment's dynamics:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi_{\theta}}(s, a) \right]$$

This is the foundation of all policy-gradient algorithms.

Basic Algorithm: REINFORCE

1. **Collect trajectories** (s_0, a_0, r_1, \dots) by following π_{θ} .
2. For each step, compute the return G_t .
3. **Update parameters:**

$$\theta \leftarrow \theta + \alpha G_t \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)$$

4. Repeat until convergence.

Algorithms

Variance Reduction

- Use **baseline functions** (e.g. value estimates $V(s)$) to reduce gradient variance:

$$\nabla_{\theta} J(\theta) \approx \mathbb{E}[\nabla_{\theta} \log \pi_{\theta}(a|s) (Q(s, a) - V(s))]$$

This leads to **Actor-Critic methods**, where:

- **Actor** = the policy $\pi_{\theta}(a|s)$,
- **Critic** = estimates $V(s)$ or $Q(s, a)$.

Policy Gradient Family Tree

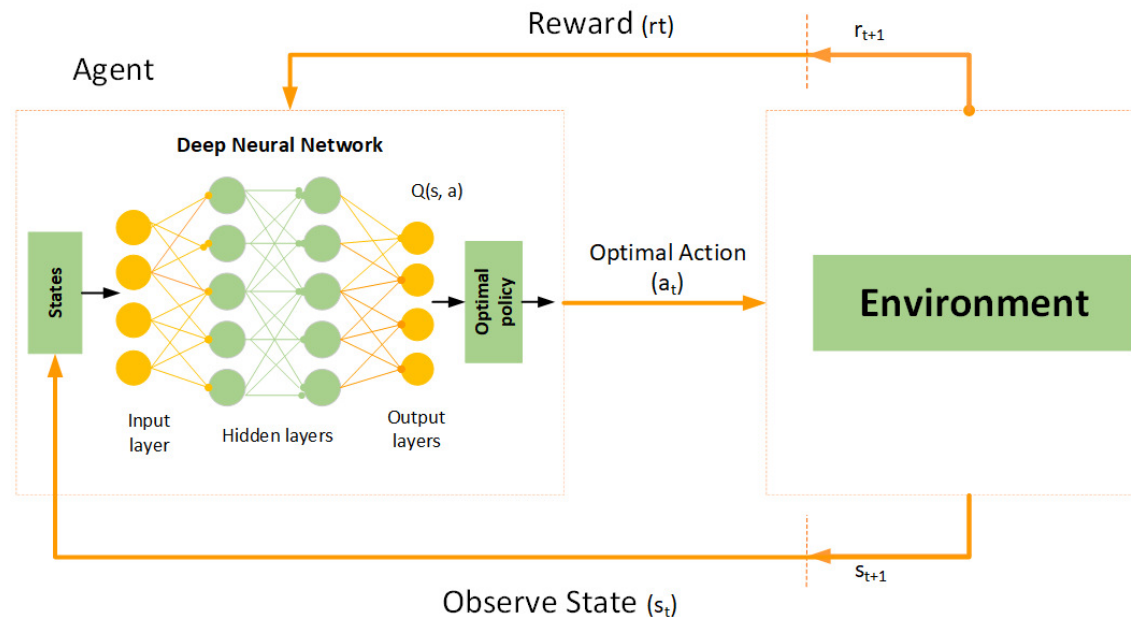
| Method | Learns | Main Trait |
|------------------|----------------|--------------------------------------|
| REINFORCE | Policy only | Simple Monte-Carlo gradient |
| Actor-Critic | Policy + Value | Lower variance, faster learning |
| PPO / TRPO / A3C | Advanced forms | Stability and efficiency for deep RL |

Algorithms

✴ Introduction to Deep Q-Networks (DQN)

From Q-Learning to Deep Q-Learning

- **Problem with Q-learning:** The Q-table explodes for large or continuous state spaces (like images, sequences, or molecular graphs).
- **Solution:** Use a **neural network** to approximate $Q(s, a; w)$ instead of a table.
- This approach — first popularized by DeepMind — is called a **Deep Q-Network (DQN)**.



Algorithms

Core Idea

- Input: state (often high-dimensional).
- Output: Q-values for each possible action.
- Update network weights w to minimize the **temporal-difference (TD) error**:

$$\mathcal{L}(w) = \left[r + \gamma \max_{a'} Q(s', a'; w^-) - Q(s, a; w) \right]^2$$

where w^- are the parameters of a **target network** (see below).

Key Innovations Over Plain Q-Learning

1. Experience Replay Buffer

- Store past experiences (s, a, r, s') in memory.
- Sample mini-batches randomly to break correlations and improve data efficiency.

2. Target Network

- Maintain a separate, slowly updated copy of the Q-network.
- Stabilizes learning by fixing the TD target for several updates.

3. Neural Network Approximation

- Deep network encodes states into useful representations.
- Allows working with images or high-dimensional feature spaces.

Algorithms

The Learning Loop (Simplified)

1. Initialize Q-network $Q(s, a; w)$ and target network $Q(s, a; w^-)$.
2. For each step:
 - Observe state s , choose action a (ϵ -greedy from current Q-network)
 - Execute a , observe reward r and next state s' .
 - Store (s, a, r, s') in replay buffer.
 - Sample a mini-batch from the buffer.
 - Compute TD targets $y = r + \gamma \max_{a'} Q(s', a'; w^-)$.
 - Update Q-network weights w to minimize $(y - Q(s, a; w))^2$.
 - Periodically update the target network $w^- \leftarrow w$.
3. Repeat.

Advantages

- Scales to very large and complex state spaces.
- Still value-based, off-policy like Q-learning.
- Basis for many extensions (Double DQN, Dueling DQN, Rainbow DQN).

Algorithms

When to Use DQN

- Large discrete action spaces (like selecting a mutation, a molecular configuration, or a lab action from many possibilities).
- When you have high-dimensional inputs (images, protein graphs, gene expression vectors).

Applications



1. Drug & Molecule Design

- **De novo molecule generation:** RL agents propose molecular structures (SMILES strings or graphs) and get rewards based on predicted binding affinity, toxicity, or pharmacokinetics.
Example: REINVENT, MolDQN — RL to generate molecules satisfying multiple biological constraints.
- **Lead optimization:** Agent modifies existing compounds step by step to improve binding scores or ADMET properties.



2. Protein & Nucleic Acid Design

- **Protein sequence optimization:** RL to find amino acid substitutions that improve stability, binding, or enzymatic activity.
- **RNA secondary structure design:** Agents generate sequences folding into desired target structures (Inverse RNA Folding).

Applications

| Application Area | Example Task | Reward Signal |
|-----------------------|---------------------------------|--|
| Molecule design | Generate high-binding molecules | Docking score, toxicity penalties |
| Protein design | Optimize stability of sequence | Energy/stability metrics |
| Adaptive experiments | Choose next CRISPR target | Information gain, prediction error reduction |
| Workflow optimization | Schedule HPC jobs | Throughput, waiting time |
| Clinical decisions | Recommend drug dosing | Patient outcomes, toxicity penalties |

Practice

Dynamic programming

https://github.com/cossorzano/COSS_DataAnalysis_notebooks/blob/main/MachineLearning/reinforcementLearning_Grid.ipynb

✿ Problem Being Solved: Gridworld Navigation (Toy MDP)

The code implements **Policy Iteration** to solve a simple finite **Markov Decision Process (MDP)** defined as follows:

🚩 Environment: 4×4 Gridworld

- There are **16 states**, arranged in a 4x4 grid.
- Each state represents a position in the grid, numbered from 0 (top-left) to 15 (bottom-right).

```
0  1  2  3
4  5  6  7
8  9 10 11
12 13 14 15
```

📄 Copiar ✎ Editar

🚩 Goal: Reach a Terminal State Efficiently

- **States 0 and 15** are terminal.
- All other states are **non-terminal**: the agent can move **up, down, left, or right**, unless it would fall off the grid (in which case it stays in place).
- The agent receives a **reward of -1 per move**, encouraging it to reach a terminal state in as few steps as possible.

