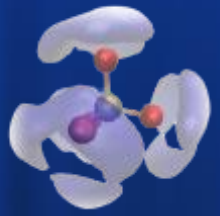
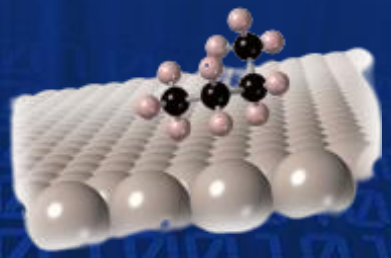
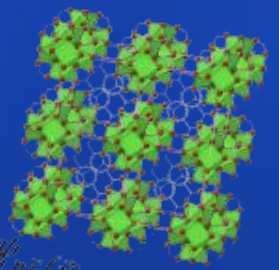


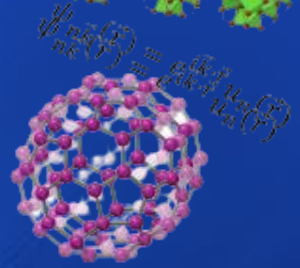


Machine Learning Models for Materials Discovery

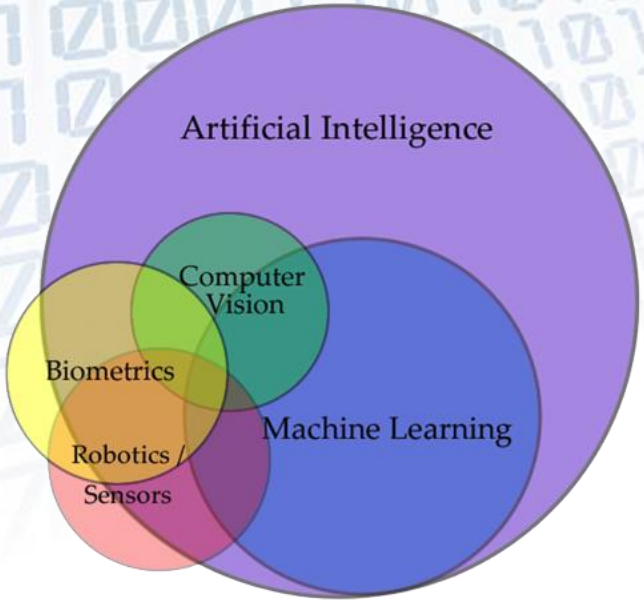
Marco Fronzi

University of Sydney



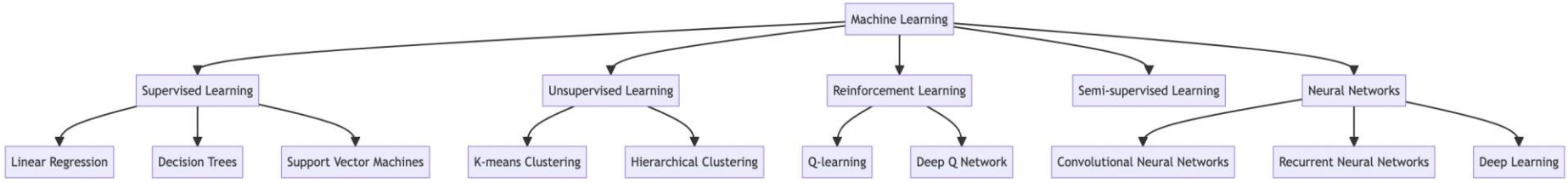


Artificial Intelligence and and Machine learning



Machine Learning:

subset of artificial intelligence that focuses on **algorithms and models that enable computers to learn and make predictions or decisions without explicit programming**. It involves the development of computational systems that can automatically learn and improve from experience, without being explicitly programmed for every specific task.

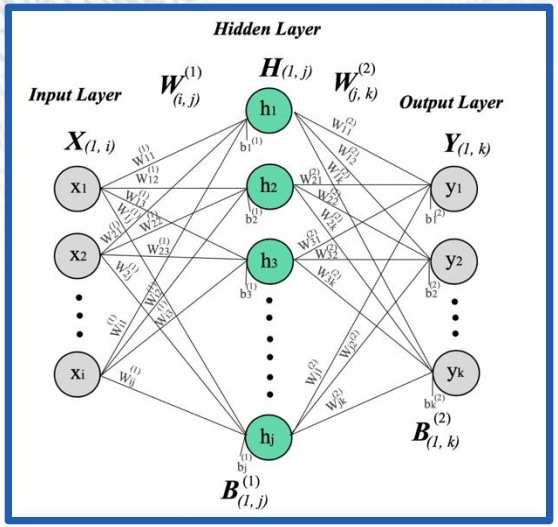


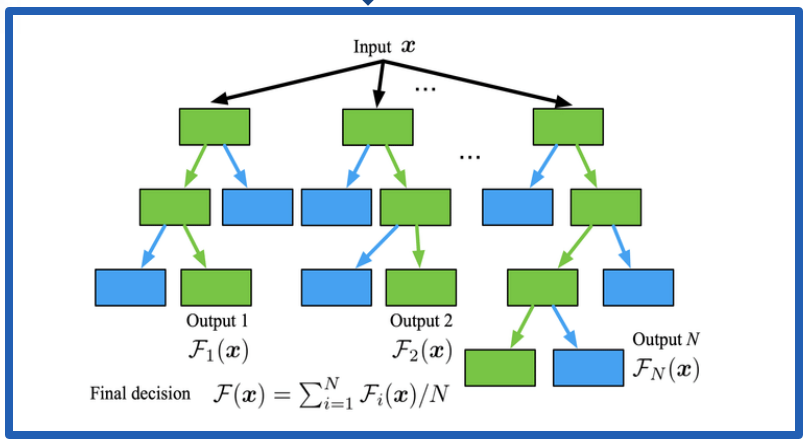
Second Slide Master

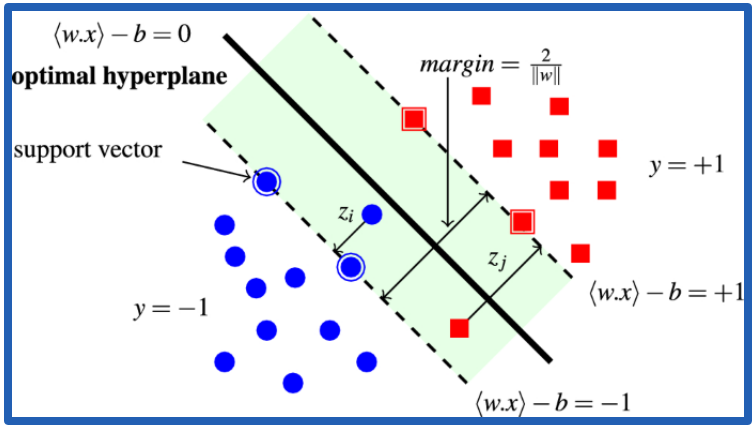
Neural Networks

Support Vector Machine

Random Forest







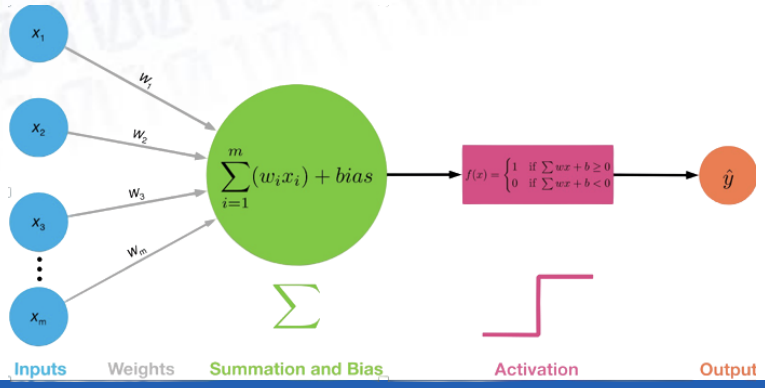
What is a Machine Learning Model?

Machine learning object is to find transfer function
To map structural parameters to target property

$\vec{x}$

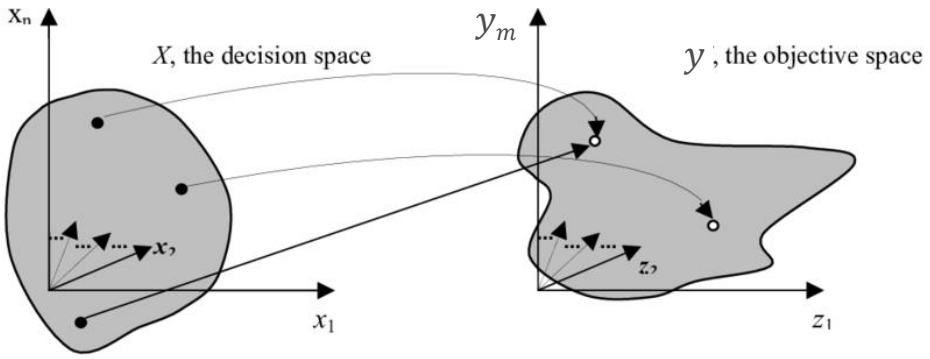
Parameter Space (Descriptors or Features)

$$f_{\lambda_1,\lambda_2,\ldots,\lambda_m}(x_1, x_2, \ldots, x_n) = y$$



y

Target Property



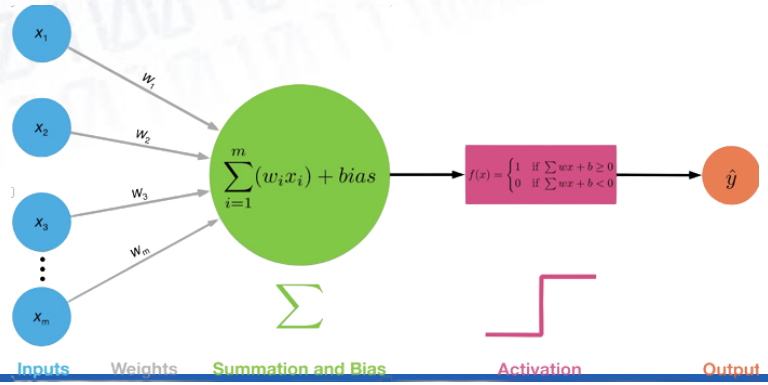
What is a Machine Learning Model?

Machine learning object is to find transfer function
To map structural parameters to target property

$\vec{x}$

Parameter Space (Descriptors or Features)

$$f_{\lambda_1,\lambda_2,\ldots,\lambda_m}(x_1, x_2, \ldots, x_n) = y$$



y

Target Property

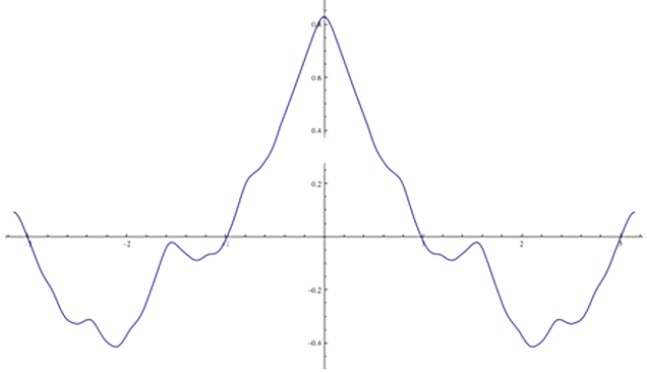
ML model is a function that maps n-dimensional space to a number
(regression = continue distribution)
(classification = discrete distribution)

If it maps to a discrete space (integer numbers) = classification
If it maps to a continuous space (real numbers) = regression

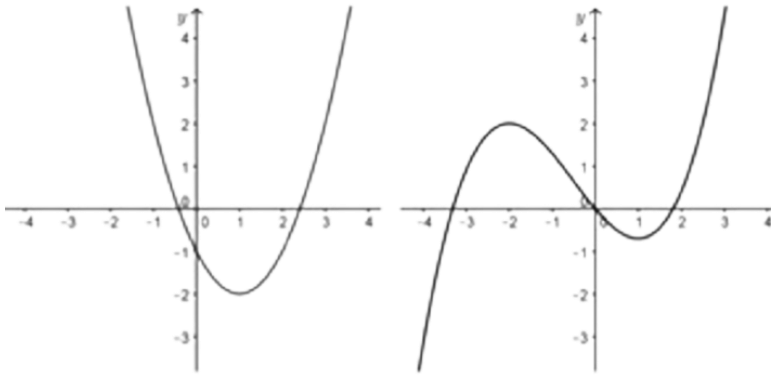
ML Models Properties

Machine learning object is to find transfer function
To map structural parameters to target property

Non-Analytical



Analytical



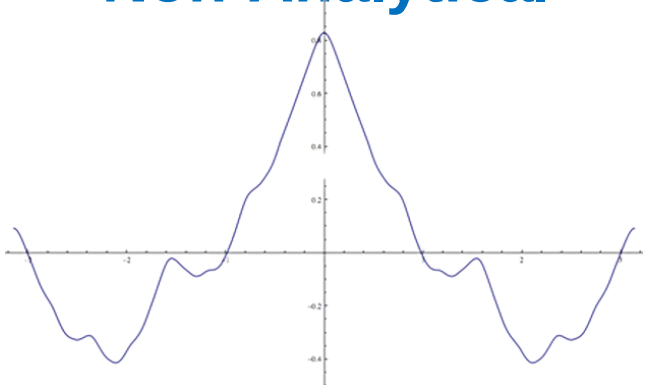
Properties= highly non linear; analytically unknown; numerically known;

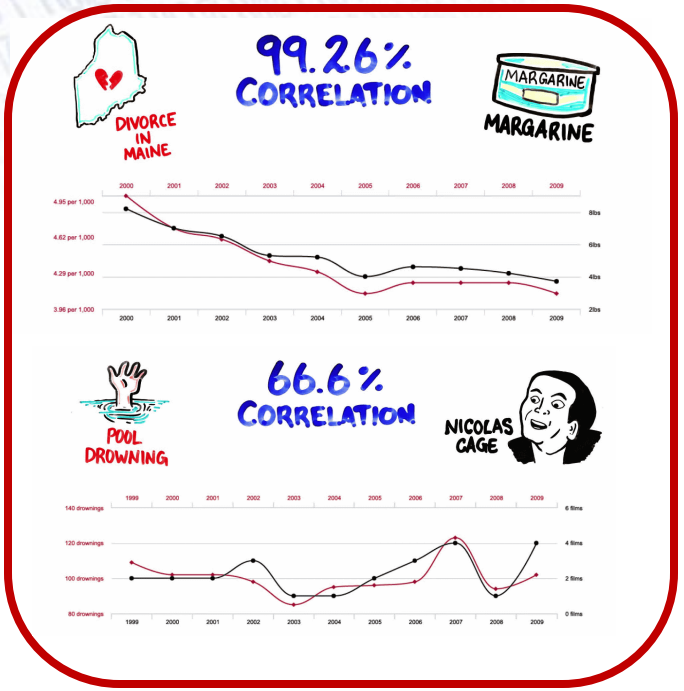
Depends on internal parameters (hyper-parameter)

ML Models Properties

Machine learning object is to find transfer function
To map structural parameters to target property

Non-Analytical





Machine learning models

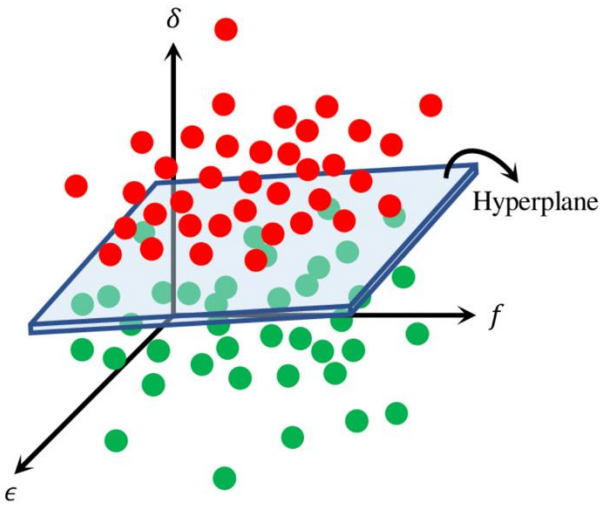
Pro: very good at finding correlations

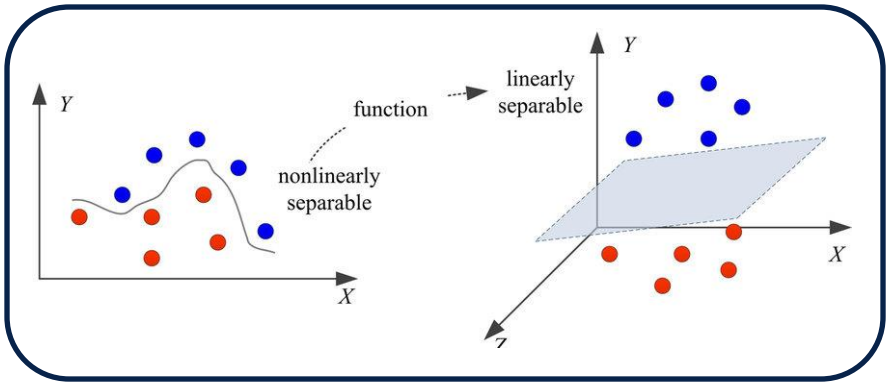
Cons: very bad at understanding causation

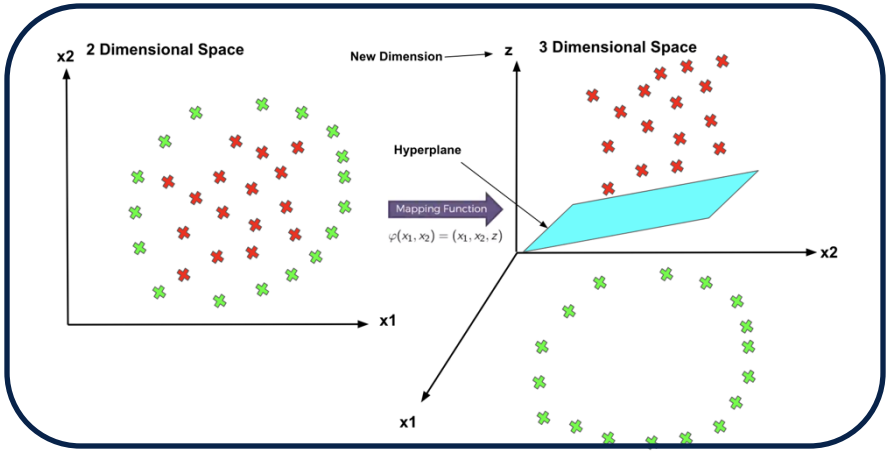
Conditions for Machine Learning Models

Separability is not necessary for models to work, but it **significantly influences the model's performance.** The degree of separability in the data determines how easily a model can learn to classify or make predictions.

Features Hyperspace







ML Models Properties

Pros:

1.Versatility: ML models can be applied to a vast array of problems, not just those in the physical sciences but across industries, such as finance, healthcare, marketing, etc.

2.Adaptability: ML models can learn from new data, improving their accuracy and effectiveness over time.

3.Efficiency: be faster and more computationally efficient than complex QM simulations.

Cons:

1.Data Dependence: The accuracy of ML models is heavily dependent on the quality and quantity of the training data.

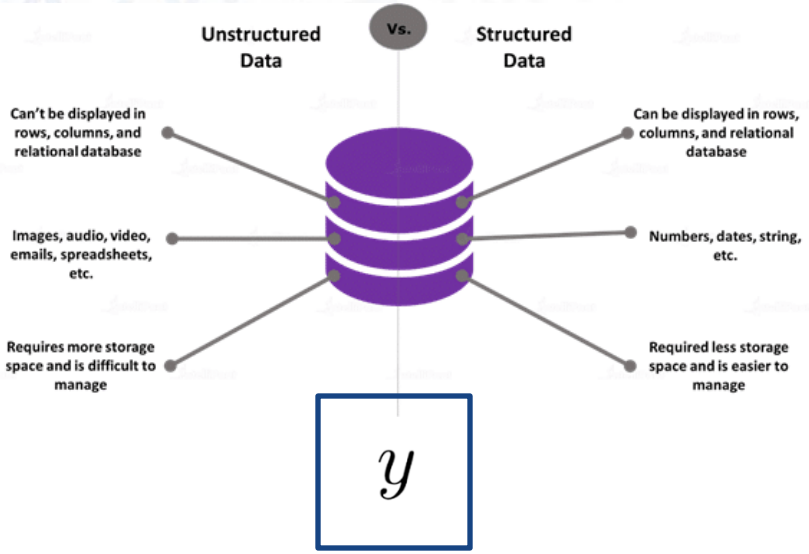
2.Black box: ML models can act as "black boxes," making it hard to understand why they make certain predictions Deep understanding of the science is needed

Building a ML Model

$$f_{\lambda_1,\lambda_2,\ldots,\lambda_m}(x_1, x_2, \ldots, x_n) = y$$

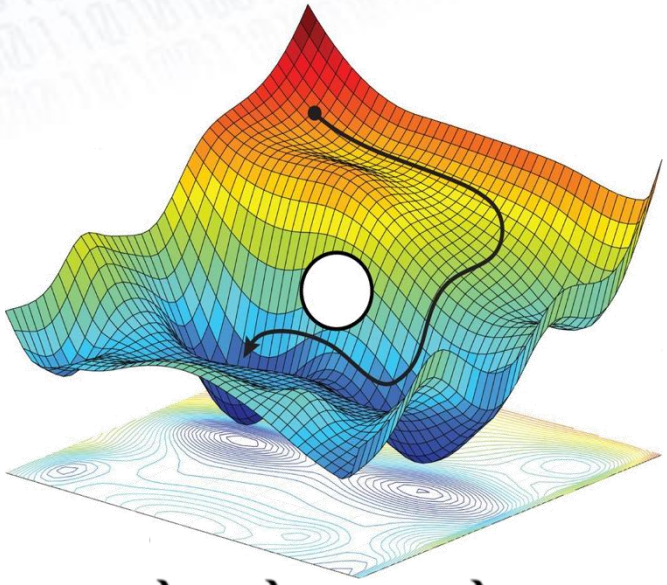
Data-set to provide as example (Comparable to Experience)

$$(x_1, x_2, \ldots, x_n)$$



$$y$$

Lot of work to do !!!



$$\lambda_1,\lambda_2,\ldots,\lambda_m$$

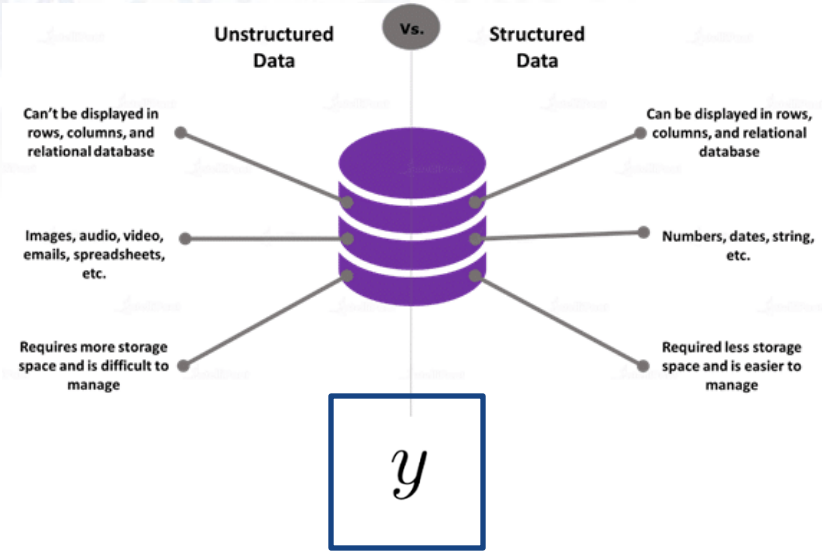
Easy task!
Many optimizers available

Building a Model Require a set of elements with known target property

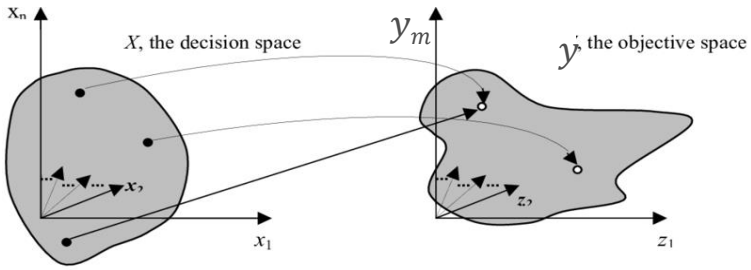
$$f_{\lambda_1,\lambda_2,\ldots,\lambda_m}(x_1, x_2, \ldots, x_n) = y$$

Data-set to provide as example (Comparable to Experience)

$$(x_1, x_2, \ldots, x_n)$$



- Must provide a set of elements that ML can use to learn how to map the two spaces

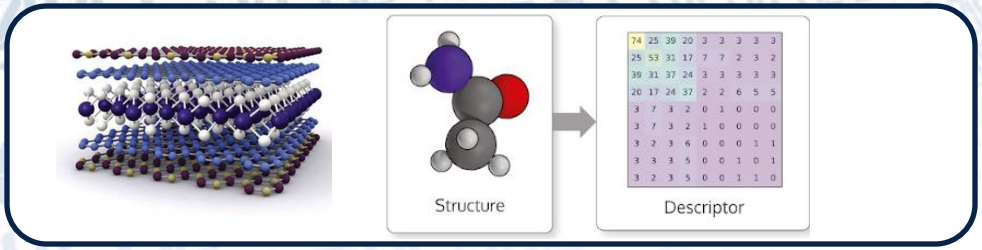


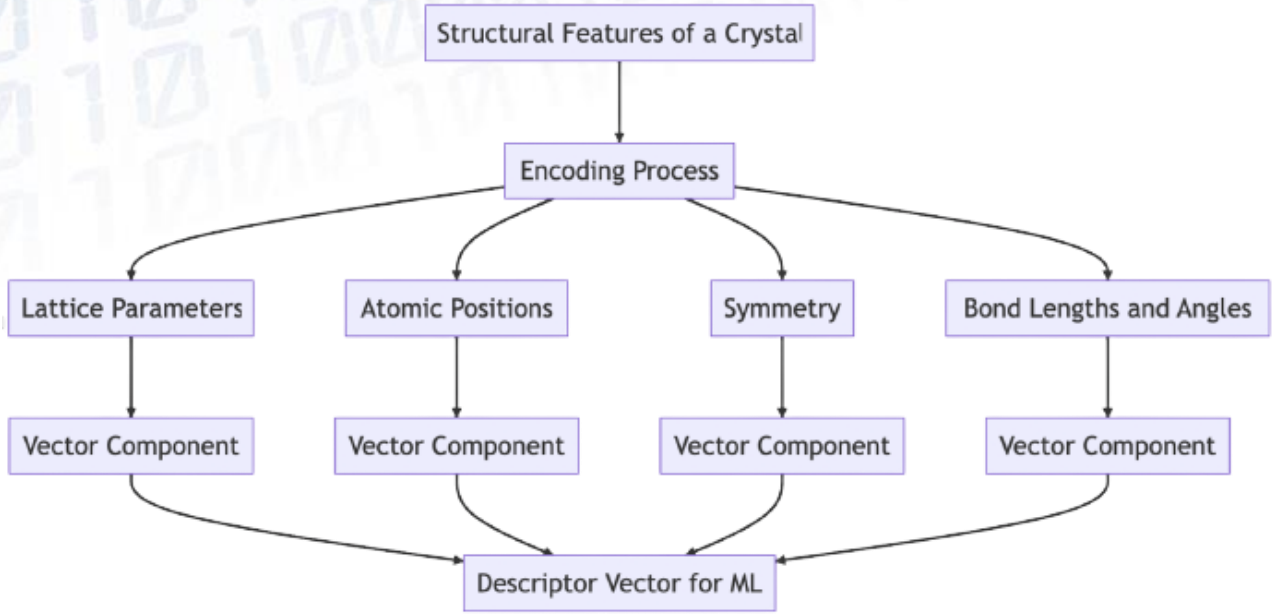
Target Property: Computed or measured

Parameter Space: Computational Featurization

Featurization

Convert available material information (e.g. structural, elemental) into a numerical vector that uniquely describe the crystal





Python lib

In this context, Feature and Descriptors are interchangeable words

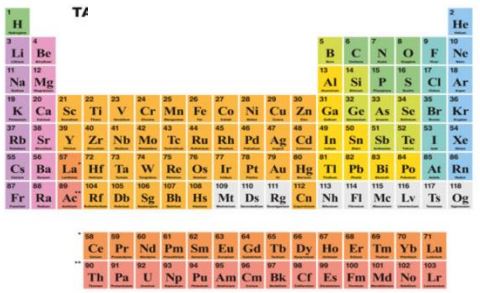
Featurization

Convert available material information (e.g. structural, elemental) into a numerical vector that uniquely describe the crystal

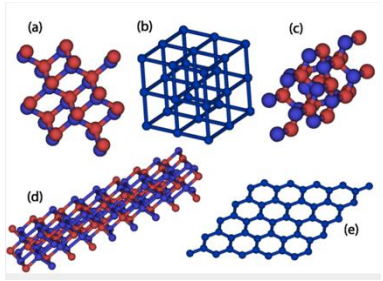
In materials informatics, featurization—the process of transforming raw material data into numerical descriptors for machine learning—can indeed be categorized into several main types:

1. **Composition-Based Featurizers**: These rely solely on the chemical formula of a material, extracting features from the types and ratios of elements present.
2. **Structure-Based Featurizers**: These utilize detailed crystallographic information, considering atomic positions and lattice parameters to derive descriptors related to geometry and bonding.
3. **Graph-Based Featurizers**: These represent materials as graphs, where atoms are nodes and bonds (or interatomic interactions) are edges, enabling the capture of complex structural relationships.
4. **Electronic Structure-Based Featurizers**: These use electronic properties, such as density of states or band structures, to derive features that reflect the electronic behavior of materials.

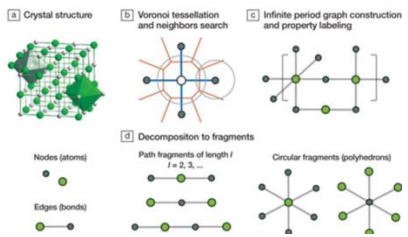
1



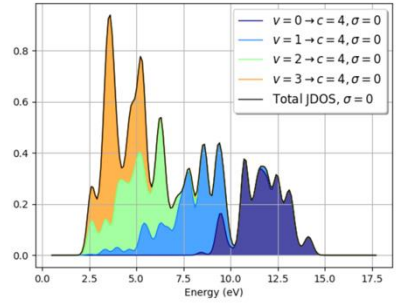
2



3



4



In this context, Feature and Descriptors are interchangeable words

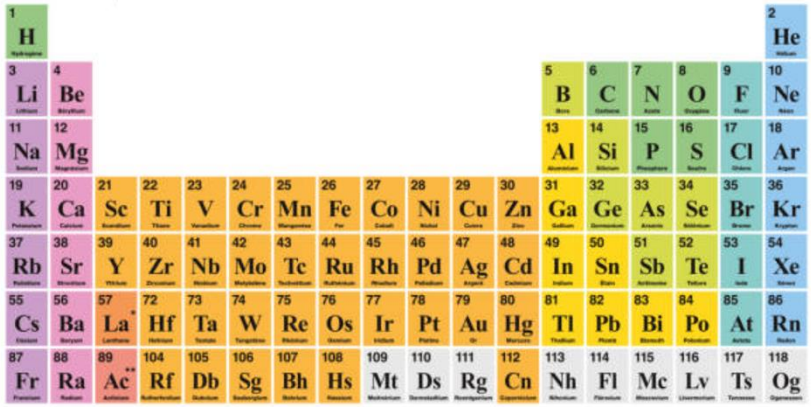
Featurization (Composition)

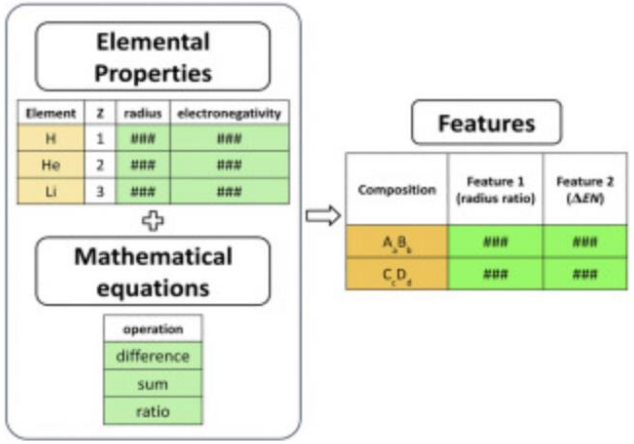
Composition-Based Featurizers

Objective: Derive descriptors solely from the chemical formula, capturing elemental properties and their statistical measures.

Common Features:

- Mean Atomic Number: $\bar{Z} = \sum_i x_i Z_i$
- Mean Atomic Mass: $\bar{M} = \sum_i x_i M_i$
- Mean Electronegativity: $\bar{\chi} = \sum_i x_i \chi_i$
- Variance of Property P: $\sigma_P^2 = \sum_i x_i (P_i - \bar{P})^2$





Featurization (Structure)

Structure-Based Featurizers

Objective: Utilize the crystal structure to extract geometrical and physical descriptors.

Common Features: • Density ρ: $\rho = \frac{\sum_i N_i M_i}{V}$

where N_i is the number of atoms of element i, M_i is the atomic mass, and V is the unit cell volume.

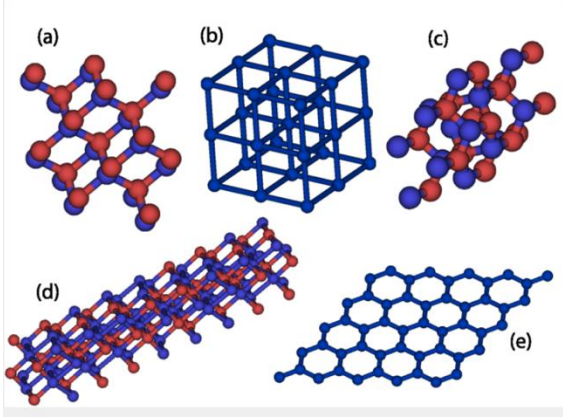
• Packing Fraction (ϕ):

$\phi = \frac{\sum_i N_i V_{\text{atom},i}}{V}$

where $V_{\text{atom},i}$ is the atomic volume of element i.

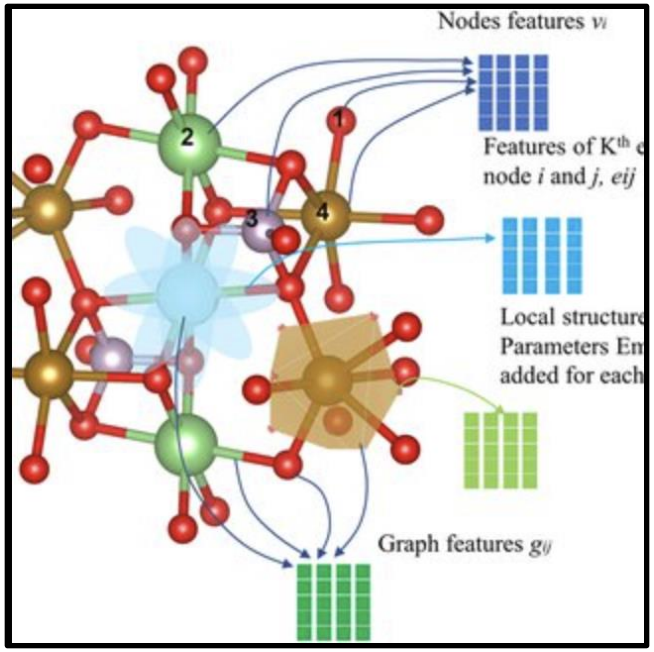
• Bond Length Distribution: Statistical measures (mean, variance) of the distances between neighboring atoms.

• Coordination Number: Average number of nearest neighbors per atom.



Featurization (Graph)

Graph-Based Featurizers



Objective: Represent the crystal structure as a graph to capture topological and connectivity features.

Common Features:

• Wiener Index:

$$W = \sum_{i<j} d_{ij}$$

where d_{ij} is the shortest path distance between nodes i and j in the graph. • Randić Index:

$$R = \sum_{(i,j)\in E} (d_i d_j)^{-0.5}$$

where d_i and d_j are the degrees of vertices i and j, respectively, and E is the set of edges.

• Clustering Coefficient:

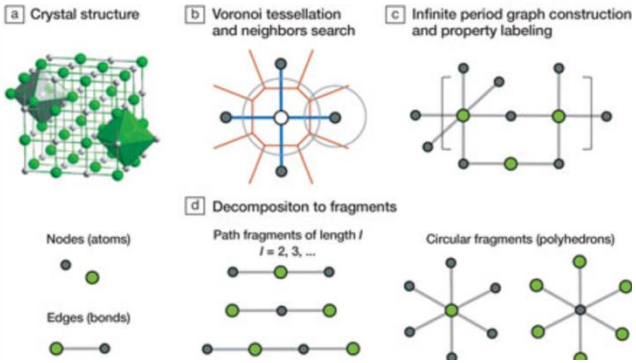
$$C_i = \frac{2e_i}{k_i(k_i-1)}$$

where e_i is the number of edges between the k_i neighbors of node i.

• Betweenness Centrality:

$$BC(i) = \sum_{s\neq i\neq t} \frac{\sigma_{st}(i)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(i)$ is the number of those pat



Featurization (Electronic Structure)

Electronic Structure-Based Featurizers

Objective: Extract features from the electronic properties of materials, such as band structures and density of states (DOS).

Common Features:

• Band Gap (E_g): Energy difference between the valence band maximum and conduction band minimum.

• Density of States at Fermi Level (D(E_F)):

$$D(E_F) = \frac{1}{N} \sum_i \delta(E_i - E_F)$$

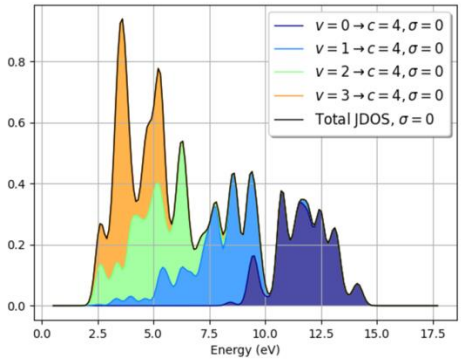
where E_i are the energy levels, and δ is the Dirac delta function.

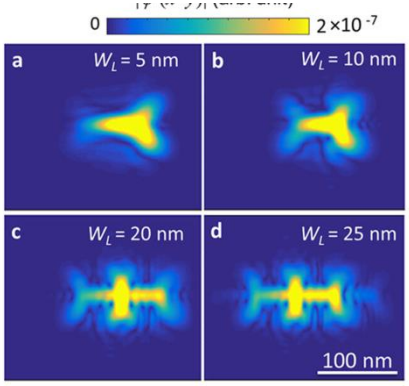
• Effective Mass $(m)*$: $\frac{1}{m^*} = \frac{1}{\hbar^2} \frac{\partial^2 E}{\partial k^2}$

where E is the energy and k is the wave vector.

• Work Function (Φ): $\Phi = -eV_{\text{vac}} + E_F$

where V_{vac} is the vacuum potential, and E_F is the Fermi energy.





Features Engineering

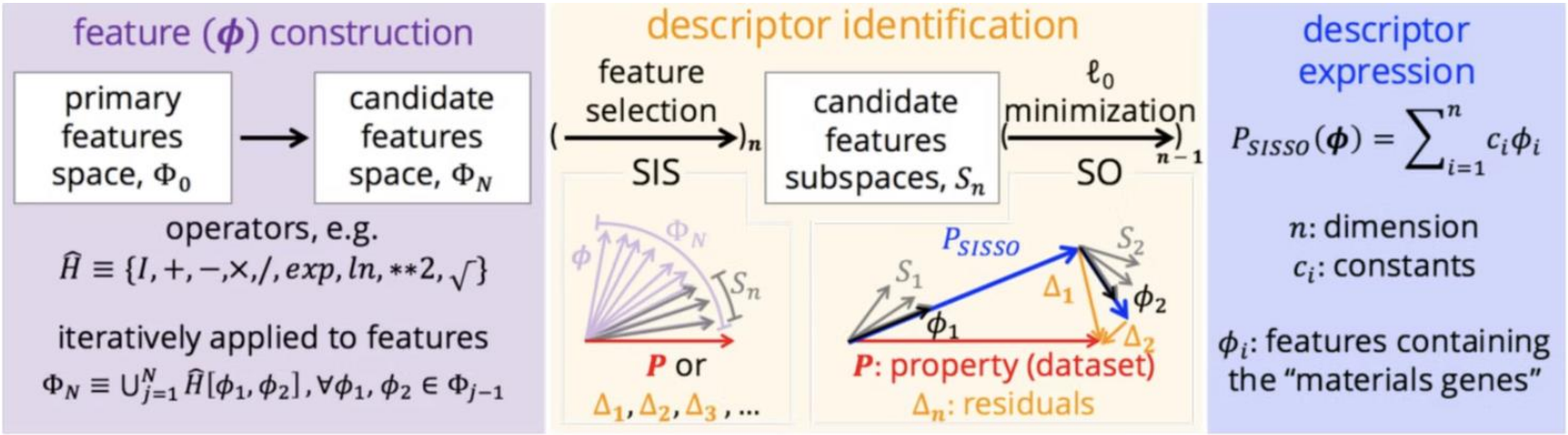
Key Objectives of Feature Engineering:

1. Improve Space Separability: models work better if the descriptors space is linearly separable

2. Better Capture of Non-Linear Relationships: Many real-world problems involve complex, non-linear interactions between variables. Feature engineering seeks to create features that can represent these intricate relationships, enabling models to learn from them effectively.

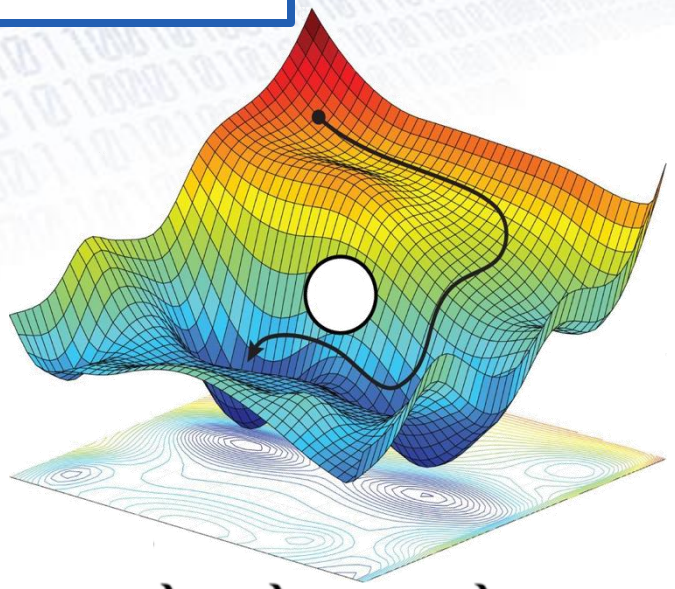
3. Improving Model Interpretability: By designing features that align with domain knowledge, feature engineering enhances the interpretability of models, allowing practitioners to understand and trust the model's decisions.

SISSO Engineering

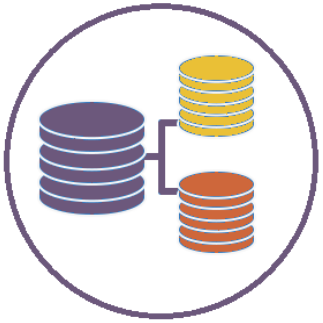


Building a ML Model

$$f_{\lambda_1,\lambda_2,\ldots,\lambda_m}(x_1, x_2, \ldots, x_n) = y$$



$\lambda_1,\lambda_2,\ldots,\lambda_m$

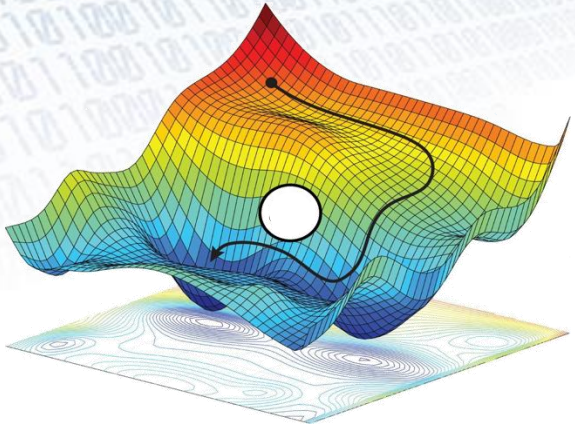


Futurized Dataset

Optimization

Model training is a search for (energy) global minimum

Algorithms = provide M examples and search for global minimum by changing hyper-parameters values

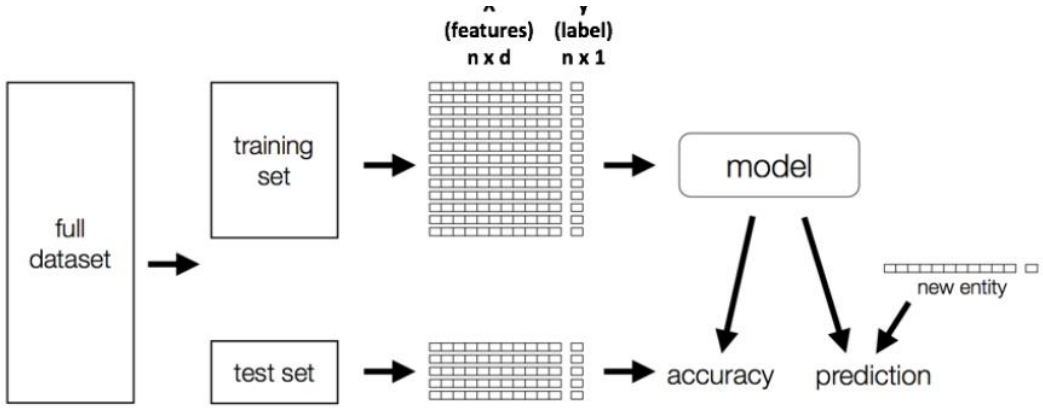


Metrics (Loss Functions)

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

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$



Gradient Descendent Optimization

At the core of many ML training processes lies Gradient Descent, an iterative optimization algorithm used to minimize a loss function L(θ), where θ represents the model parameters. Algorithm Steps:

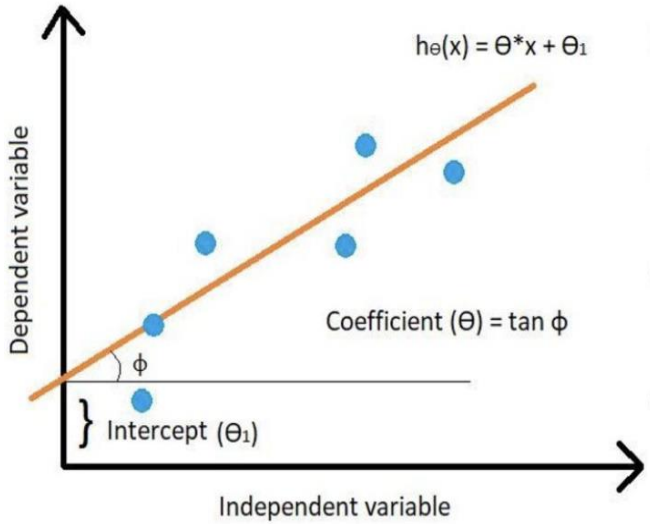
1. Initialize the parameters θ randomly.
2. Compute the gradient of the loss function with respect to the parameters:

$$\nabla_\theta L(\theta)$$

3. Update the parameters in the opposite direction of the gradient:

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

where η is the learning rate, a hyperparameter that controls the step size.
Convergence is achieved when the change in the loss function or parameters falls below a predefined threshold.



Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Stochastic Gradient Descendent

Stochastic Gradient Descent (SGD)

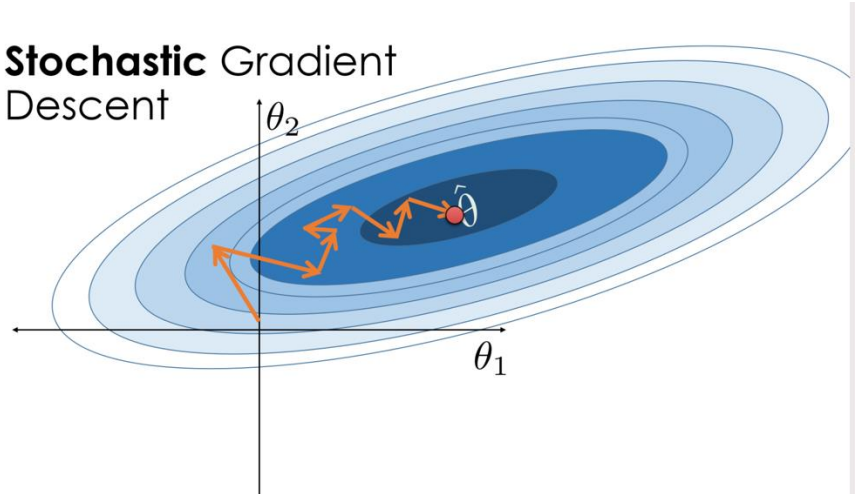
Stochastic Gradient Descent is a variant of gradient descent where the parameter update is performed using a single data point (or a mini-batch) instead of the entire dataset. This introduces noise into the optimization process but often leads to faster convergence.

Update Rule:

For a given data point (x_i, y_i):

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta; x_i, y_i)$$

This approach is particularly useful for large datasets where computing the gradient over the entire dataset is computationally expensive.



Adam Optimizer

The Adam (Adaptive Moment Estimation) optimizer is an extension of SGD that computes adaptive learning rates for each parameter by estimating the first and second moments of the gradients.

Algorithm: 1. Initialize: • Parameter vector θ • First moment vector m = 0 • Second moment vector v = 0 • Time step t = 0 • Hyperparameters: α (learning rate), β_1 (decay rate for the first moment), β_2 (decay rate for the second moment), ϵ (small constant to prevent division by zero)
2. Update: • Increment time step:

$$t \leftarrow t + 1$$

• Compute gradients:

$$g_t = \nabla_\theta L(\theta_{t-1})$$

• Update biased first moment estimate:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

• Update biased second moment estimate:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

• Compute bias-corrected first moment estimate:

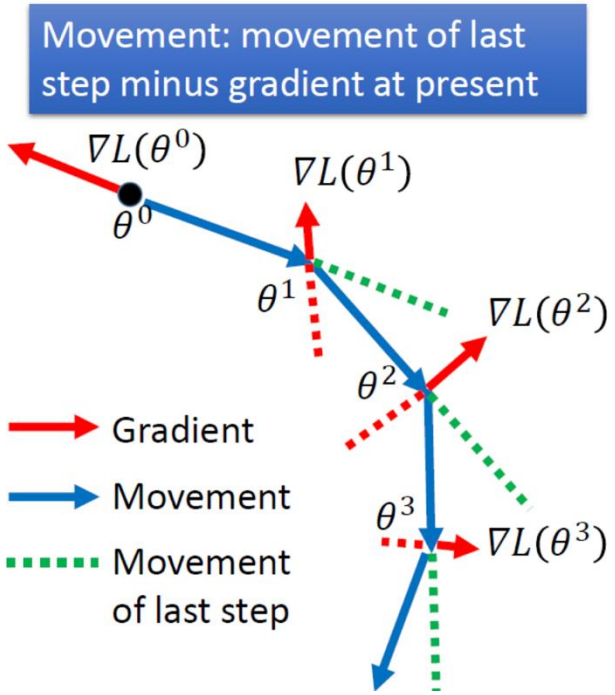
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

• Compute bias-corrected second moment estimate:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

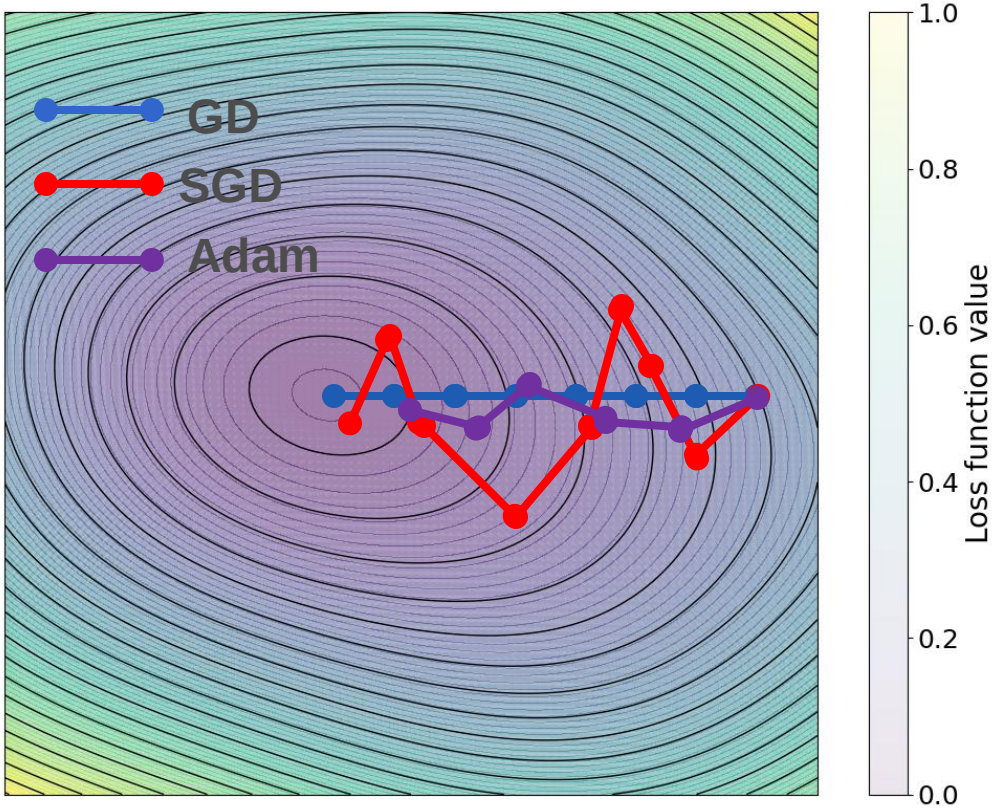
• Update parameters:

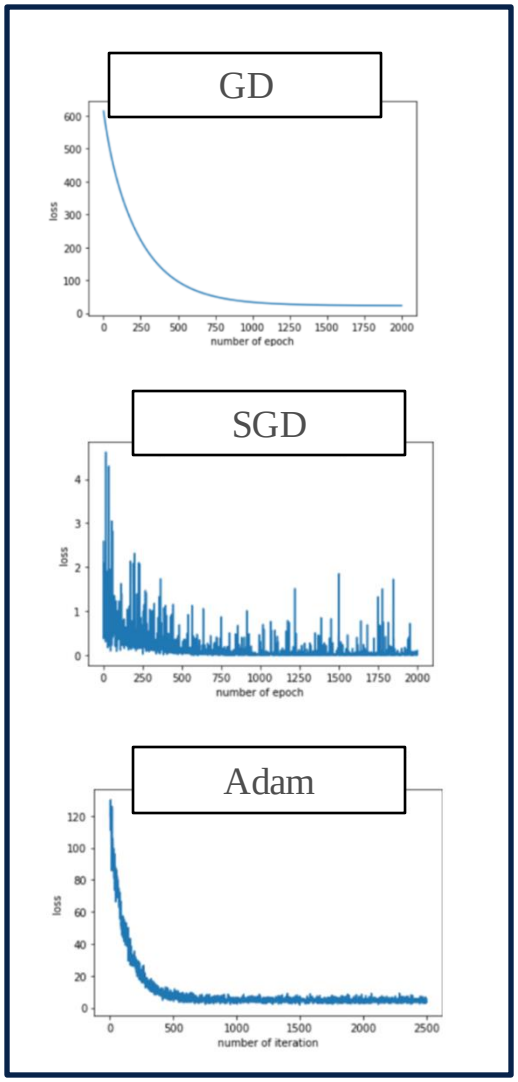
$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$



Adam provide an optimization algorithm that can handle sparse gradients on noisy problems.

Optimizer Efficiency vs Accuracy

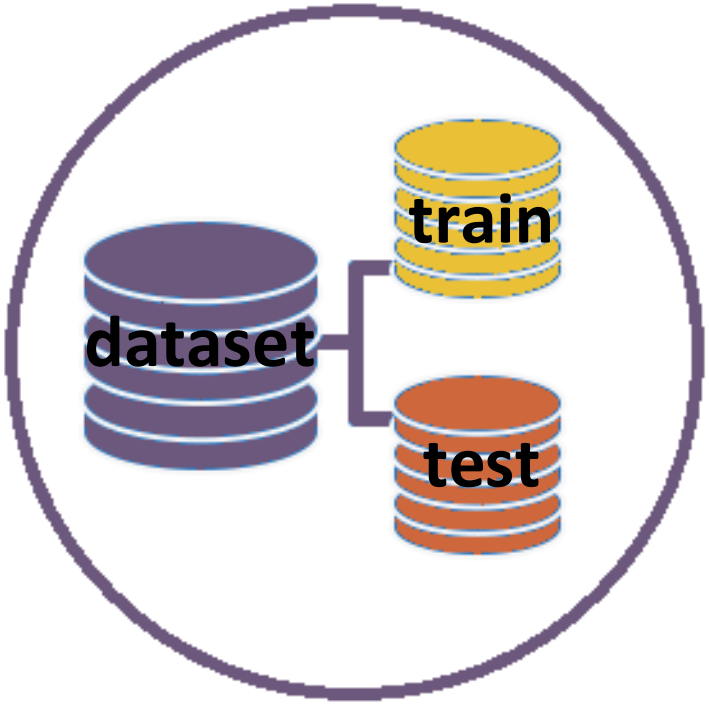




Select Samples

Two sets are necessary to build a model:
Training set used to optimize the model (tune the hyperparameters)

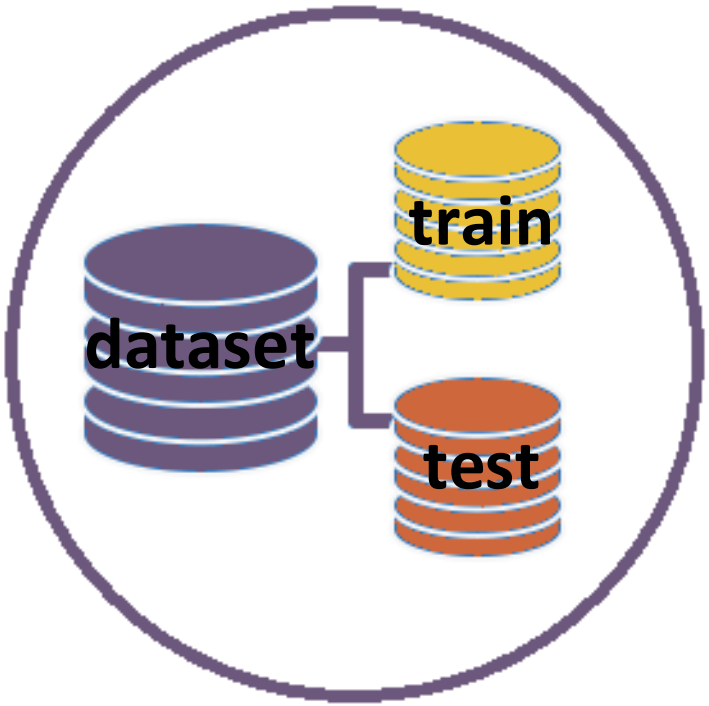
Test set: use to check quality of the model after optimization
Test if the model is able to generalize with reasonable error



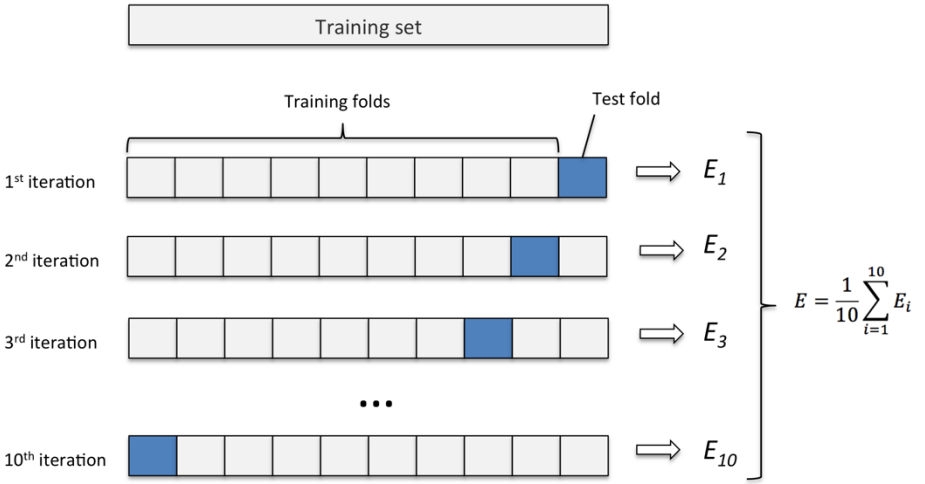
Train:test=80:20

Cross Validation

Perform multiple optimization choosing each time a different training test subset

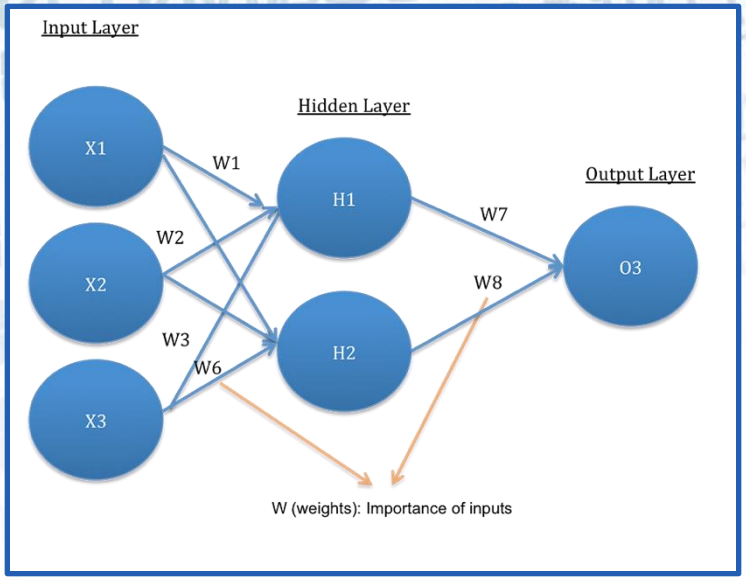


Train:test=80:20



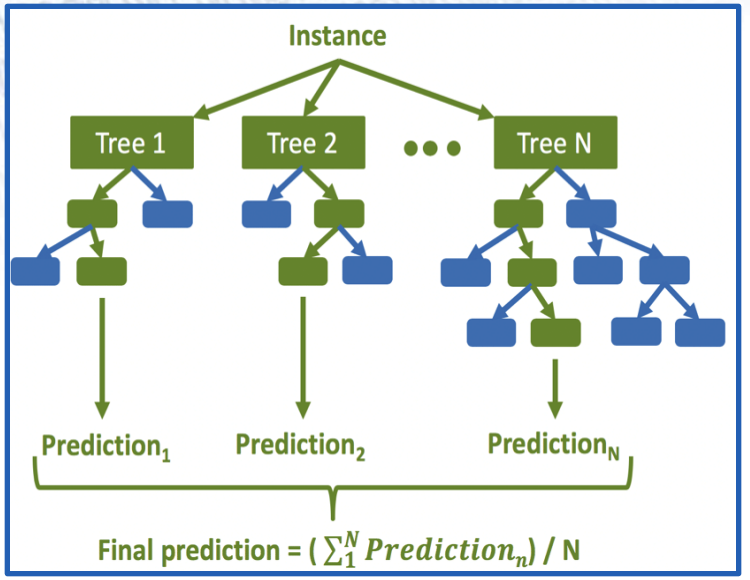
Neural Networks and Random Forest

Neural Networks



Neural networks are computational models inspired by the structure and functioning of the human brain. They consist of interconnected nodes, called neurons, organized in layers. Each neuron takes inputs, performs a computation, and produces an output. Neural networks can learn complex patterns and relationships in data through a process called training.

Random Forest



Random forests are ensemble learning models that combine multiple decision trees to make predictions. Each tree in the forest is built independently using a random subset of the training data and features. When making predictions, each tree or averages its individual output. A decision tree is a flowchart-like structure that uses rules based on features to make predictions. It starts with a root node, splits the data based on conditions, and reaches outcome predictions at the leaf nodes.

Random Forest

Decision Trees as Base Learners

A decision tree is a recursive partitioning of the input space. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^d$ are feature vectors and y_i are target labels, the tree partitions the space into regions R_m, where each region is associated with a prediction value:

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m). \tag{3}$$

Here, c_m is the prediction value (mean for regression, majority class for classification) in region R_m.

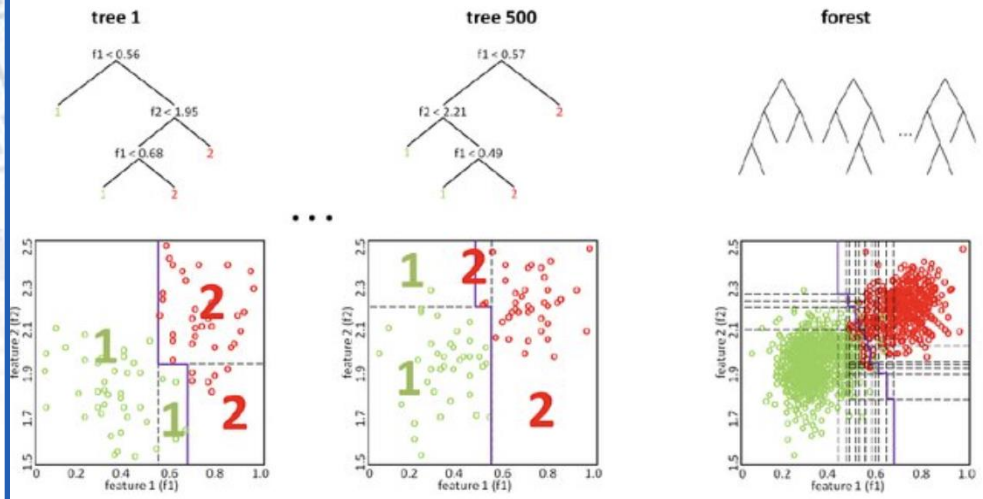
Random Forests: An Ensemble of Decision Trees

A random forest is a collection of T decision trees, each trained independently on a bootstrap sample of the data. The final prediction is obtained by aggregating the predictions of individual trees:

- **Classification:** Majority vote of tree predictions.
- **Regression:** Averaging tree outputs.

Formally, let $f_t(x)$ be the prediction from the t-th tree. The random forest prediction is:

$$\hat{f}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x). \tag{4}$$



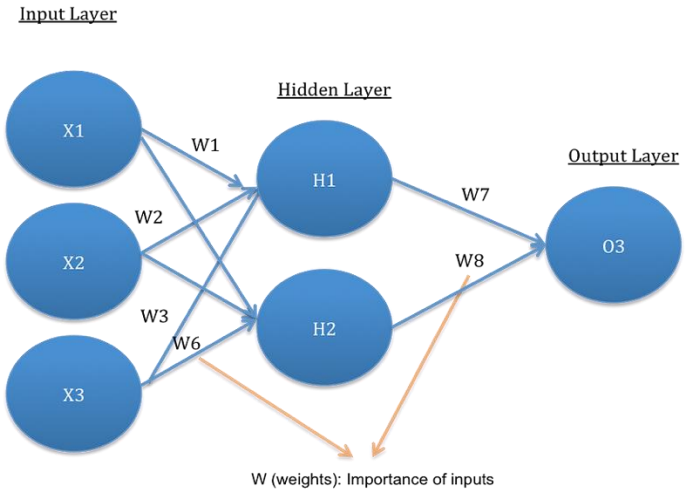
A decision tree is a recursive partitioning of the input space. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^d$ are feature vectors and y_i are target labels, the tree partitions the space into regions R_m, where each region is associated with a prediction value:

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m). \tag{3}$$

Generate M trees where:

- Each tree T_j builds a sequence of binary decisions, each **based on a single feature**, to split the data into regions.
- Each **leaf node** in the tree corresponds to a region in **feature space**, and that region is assigned an output value
- Each tree in the forest is trained on a **random subset** of the training data, drawn **with replacement** (a technique called *bootstrap aggregation* or *bagging*).

Neural Networks



$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) \tag{1}$$

Where:

- y is the output of the neuron,
- f is the activation function (e.g., sigmoid, ReLU, tanh),
- w_i are the weights,
- b is the bias,
- n is the number of inputs.

This equation represents the basic idea behind the weight update in neural networks. There are also advanced optimization algorithms like Adam, RMSprop, and others that have slightly different update rules, but they are all based on the concept of gradient descent.

During the optimization of Neural Networks, the weights w_i are adjusted using the gradient descent algorithm or its variants. The basic update rule for a weight w_i in gradient descent is:

Universal Approximation Theorem

The *Universal Approximation Theorem* states that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary precision, given an appropriate activation function.

Mathematical Formulation

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. The theorem states that for any $\varepsilon > 0$, there exists a neural network function $\hat{f}$ of the form:

$$\hat{f}(x) = \sum_{i=1}^{m} \alpha_i \sigma(w_i^T x + b_i) \tag{1}$$

where:

- σ is a non-linear activation function (such as the sigmoid function or ReLU);
- $w_i \in \mathbb{R}^n$ are weight vectors;
- $b_i \in \mathbb{R}$ are biases;
- $\alpha_i \in \mathbb{R}$ are output weights;
- m is the number of hidden neurons.

Such that:

$$\sup_{x \in K} |f(x) - \hat{f}(x)| < \varepsilon \tag{2}$$

where K is a compact subset of $\mathbb{R}^n$.

Kolmogorov-Arnold Networks

Kolmogorov-Arnold Representation Theorem

The theorem states that for any continuous function $f : [0, 1]^n \to \mathbb{R}$, there exist continuous functions ϕ_i and ψ such that:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{2n+1} \psi_i \left(\sum_{j=1}^{n} \phi_{ij}(x_j) \right). \tag{7}$$

This formulation shows that multivariate functions can be decomposed into sums and compositions of univariate functions, forming the mathematical basis for Kolmogorov-Arnold Networks.

Kolmogorov-Arnold Networks as Function Approximators

A Kolmogorov-Arnold Network (KAN) is constructed using layers of trainable univariate functions instead of traditional weighted sum operations. The function approximation follows:

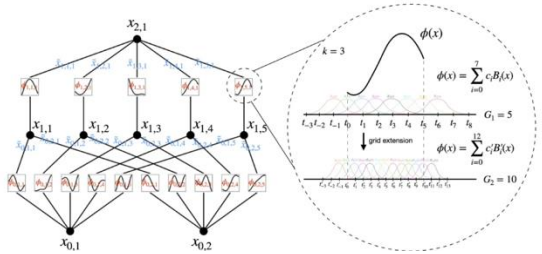
$$\hat{f}(x) = \sum_{i=1}^{m} \psi_i \left(\sum_{j=1}^{n} \phi_{ij}(x_j) \right), \tag{8}$$

where ϕ_{ij} and ψ_i are parameterized using neural network layers or splines.

Theoretical Properties of KANs

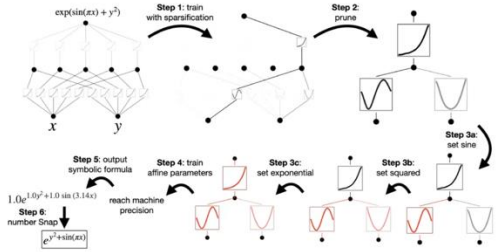
Universality

Kolmogorov-Arnold Networks are universal approximators since they can approximate any continuous function by choosing suitable ϕ_{ij} and ψ_i functions. This property follows directly from the Kolmogorov-Arnold theorem.



$$\text{spline}(x) = \sum_i c_i B_i(x)$$

Network Simplification



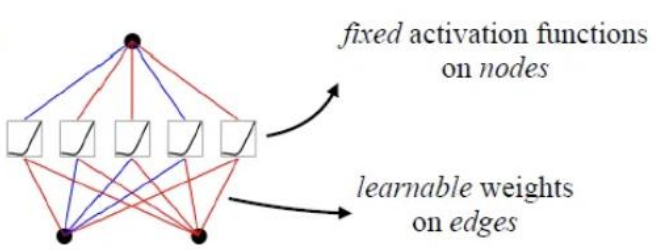
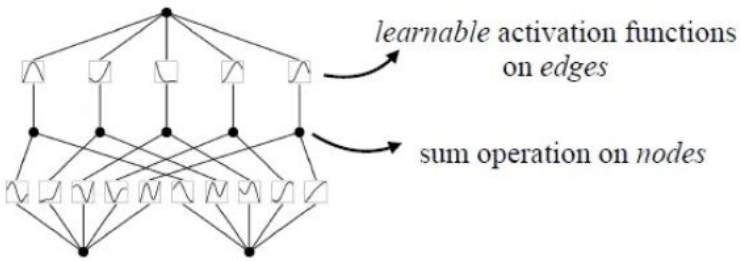
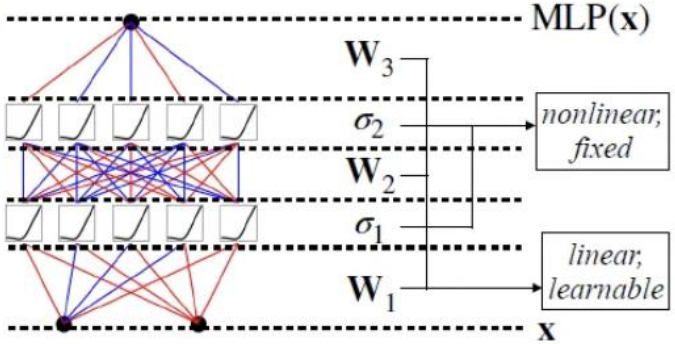
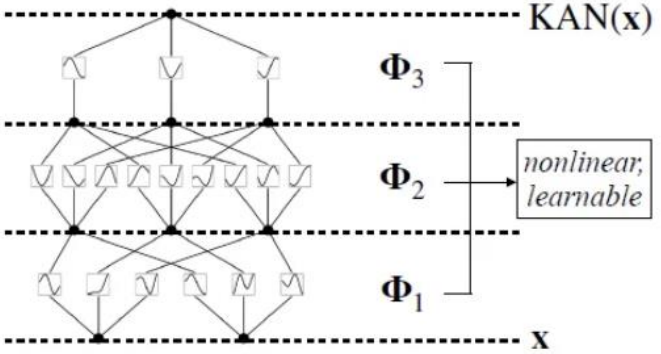
Interpretability

Unlike deep neural networks, which rely on black-box multi-layer compositions, KANs maintain an interpretable structure, making them attractive for scientific computing and symbolic regression.

Efficiency

Since KANs do not rely on traditional matrix multiplications, they can be more parameter-efficient than fully connected neural networks while maintaining high accuracy in function approximation.

KANs vs NNs

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) 	(b) 
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) 	(d) 

Machine Learning Workflow