# Introduction to Machine learning theory

# Why learning

- Machine learning is programming computers to optimize a performance criterion using example data or past experience
  - Learning general models from a data of particular examples
  - Build a model that is *a good and useful approximation* to the data.
- Data is cheap and abundant (data warehouses, data marts); knowledge is expensive and scarce.

# When learning

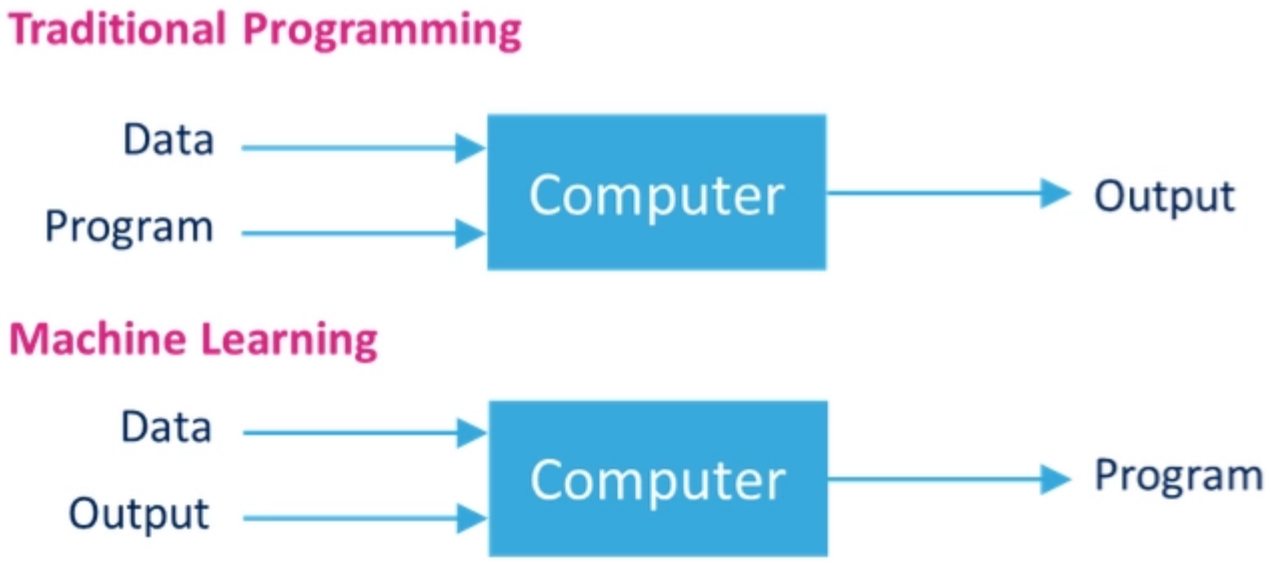
- Machine learning is suitable for problems where:
  - A pattern exists
  - We can not pin it down mathematically
  - We have data on it

- Learning is used when:
  - Human expertise does not exist (navigating on Mars)
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)

# Why learning

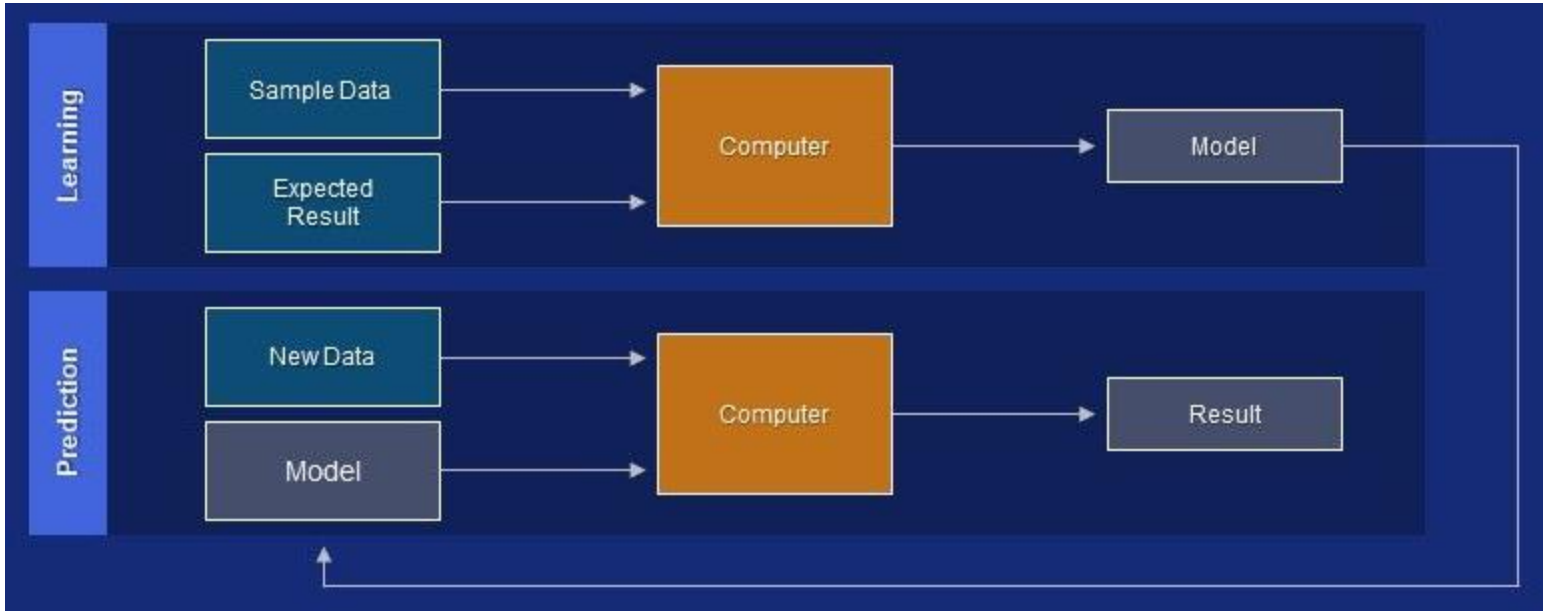
If you don't know how to solve a problem, write it out as an optimization problem

# Learning vs Programming




- Learning algorithm is implemented as a program

# Learning vs Programming




- The Learning algorithm and data are implemented as a program

# ML : The Classification problem

- The problem of classification is defined as:

- **Given**: A set of training data

  $(x_1, y_1), \ldots (x_n, y_n)$ where $x_i \; in \; \mathbb{R}^n$ and $y_i$ is the class label

- **Find :** A classification function

  $f$: $\mathbb{R}^n \rightarrow \{c_1, \ldots, c_k\}$ which classifies well additional samples $\{x_k\}$

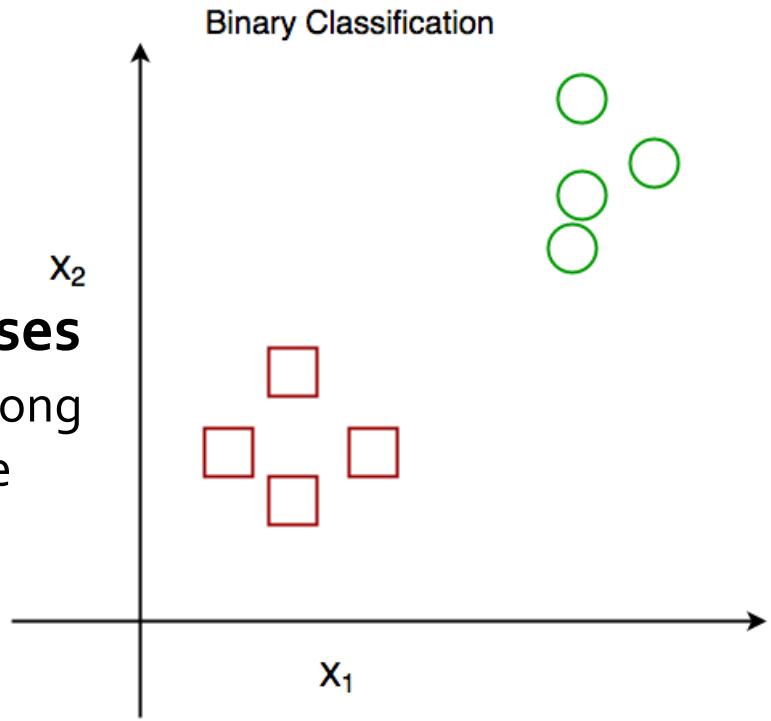
  k is the number of classes

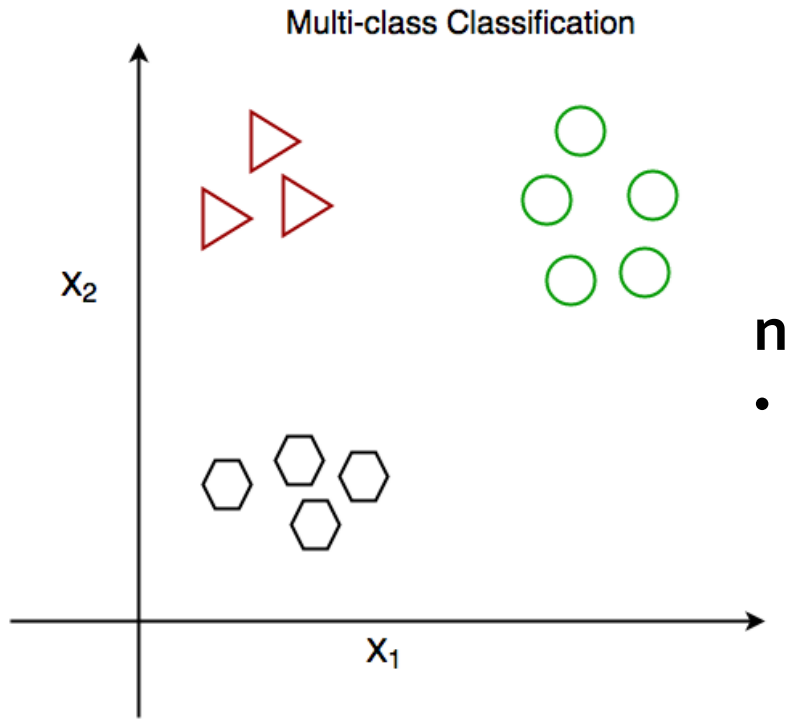
# Types of classification

Binary vs multiclass classification

**2 target classes**

- Samples belong to one of the two classes




**n target classes**

- Samples belong to one of the n classes

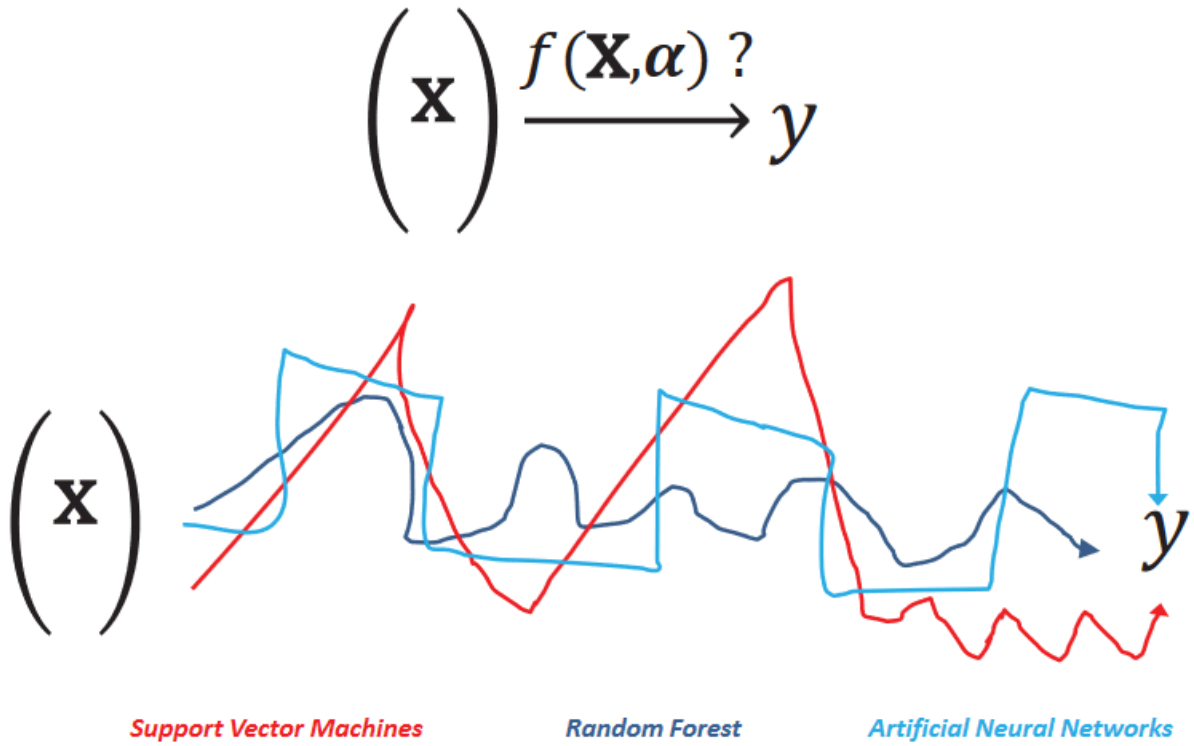
# ML: Classification methods

Approaches to learn classifiers (learning algorithms):

- Linear classifiers: Logistic Regression, Bayesian classifier
- Support Vector Machines (SVM)
- Decision trees
- Random Forest
- K-Nearest Neighbor
- Neural networks
- …

# ML: Classification methods

Approaches to learn classifiers (learning algorithms):



# ML: Main components

- Model representation:
  - Structure of the functional form of the knowledge to be extracted (Trees, partition, graph,…)
- Objectif function:
  - Measure the quality of the model (Gini, Entropy, RMSE, logloss, …)
- Search method (learning algorithm):
  - Strategy used to explore the search space and find the optimal or "good" model (backpropagation, local search, divide-and-conquer, greedy search, …)

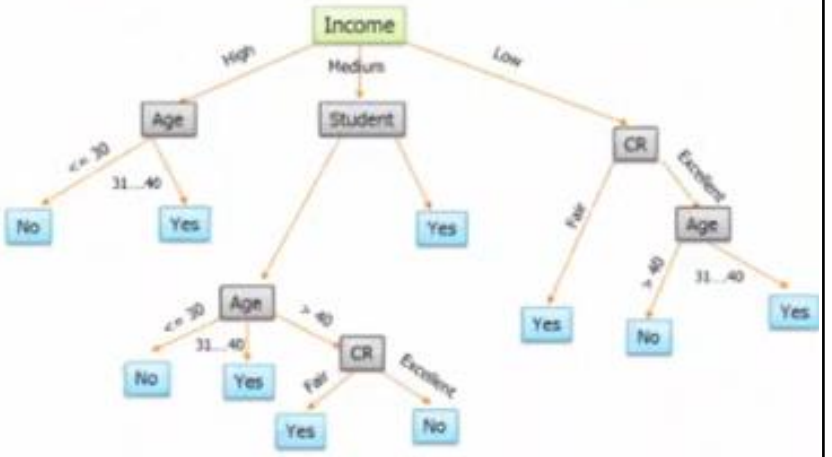
# ML: Main components

- Hypothesis space: Tree-based models

Representation = Tree

Construction of the tree

- **Greedy algorithms**: Split the data based on an attribute that optimizes certain criterion
  - CHAID, CART
  - ID3, C4.5, SLIQ, SPRINT



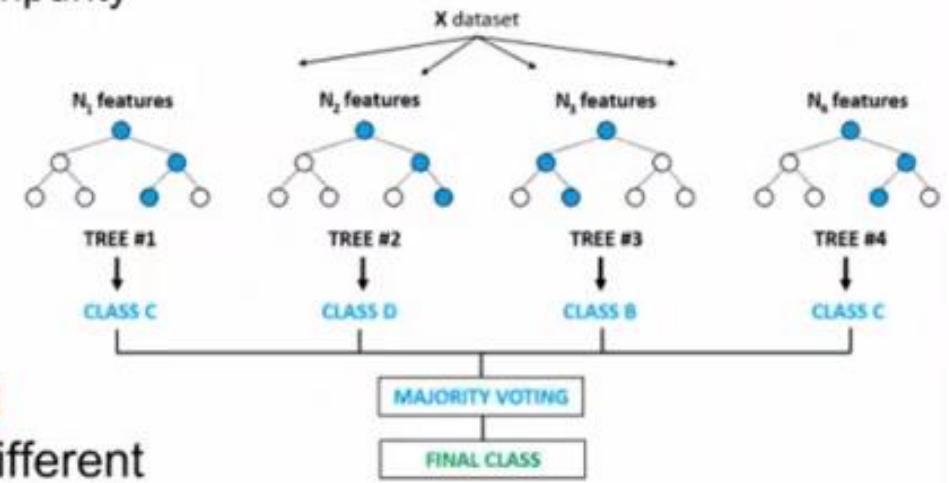

Attribute selection - Greedy heuristics

Nodes with homogeneous class distribution are preferred - node impurity

- Gini Index
- Entropy, Gain, Khi2, …

**BIAS-VARIANCE TRADEOFF**

- Multiple objectives
  - Misclassification error,
  - Tree size (PRUNING), …




Random forest : **Randomized Greedy Heuristic** → Use of different features

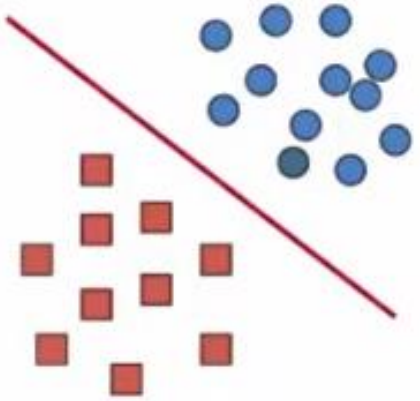
# ML: Main components

- Hypothesis space: SVM models

SVMs maximize the *margin* around the separating hyperplane.

- A.k.a. large margin classifiers

The decision function is fully specified by a subset of training samples, *the support vectors*.



Linear SVM

- **Simplest** model
- Solving SVMs is a *quadratic optimization* problem with linear constraints

Non-linear SVM – Kernel functions

- Polynomial, Radial Basis Functions, Sigmoid
- More **difficult** optimization problems → Metaheuristics

# ML: Main components

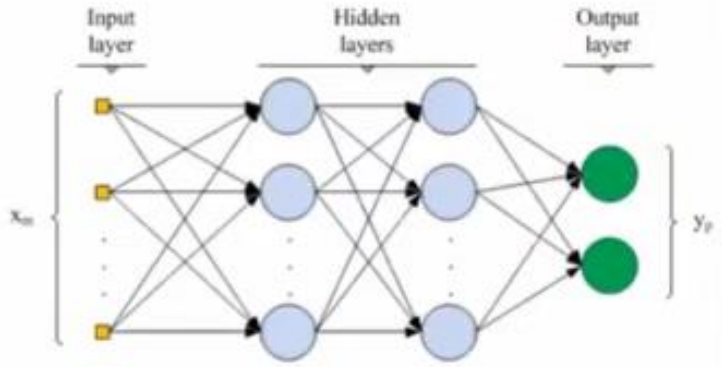
- Hypothesis space: NN models

**Optimization of NN architecture**

- NP-hard and complex problem
- Metaheuristics have been widely used

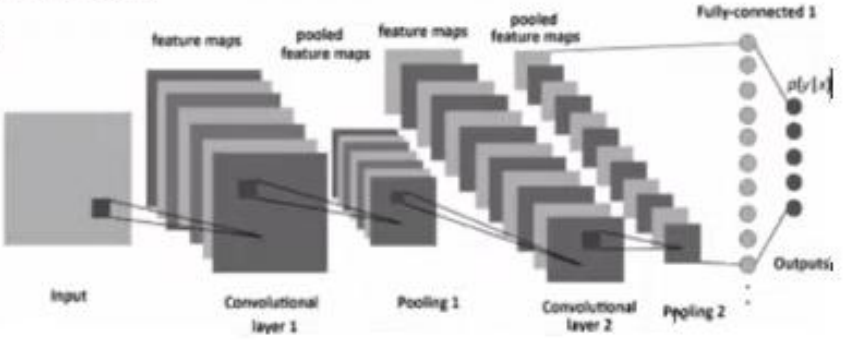
**Learning phase**

- Initial solution
  - Random weights
- Objective function
  - Classification error
- Learning algorithms
  - Continuous optimization problem
  - Most popular: Gradient-based algorithms




**Deep learning issues**

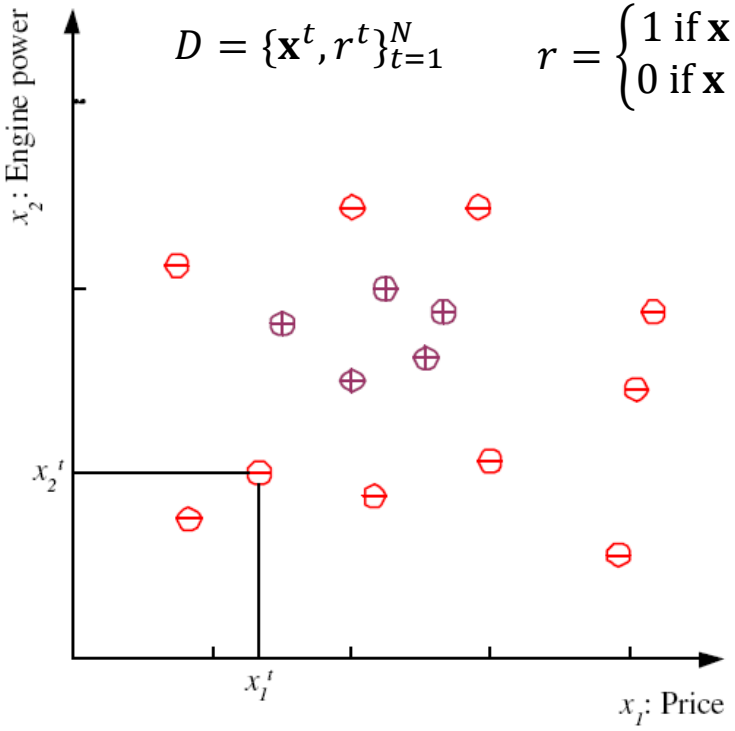
- Architecture of the heterogeneous network
  - Type of layers, Hyperparameters
- Learning algorithm
  - Big optimization (4M variables)
  - Greedy + iterative gradient
    - Greedy layer-wise training



# Hypothesis space (class) $\mathcal{H}$

- Let start with a sample problem of binary classification

$$D = \{\mathbf{x}^t, r^t\}_{t=1}^{N} \qquad r = \begin{cases} 1 \text{ if } \mathbf{x} \text{ is positive} \\ 0 \text{ if } \mathbf{x} \text{ is negative} \end{cases}$$

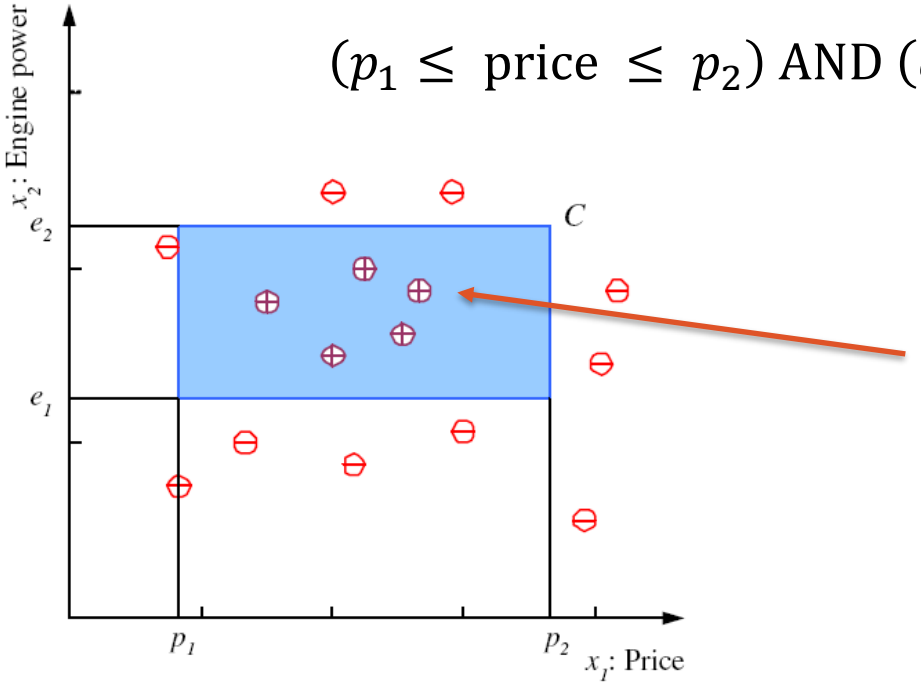



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Prediction of "family car"
  - Prediction: Is car x a family car?
- Knowledge extraction:
  - What do people expect from a family car?
- Output:
  - Positive (+) and negative (–) examples
- Input representation:
  - x1: price, x2 : engine power

# Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - Hypothesis is any axis aligned Rectangle. Inside rectangle is positive

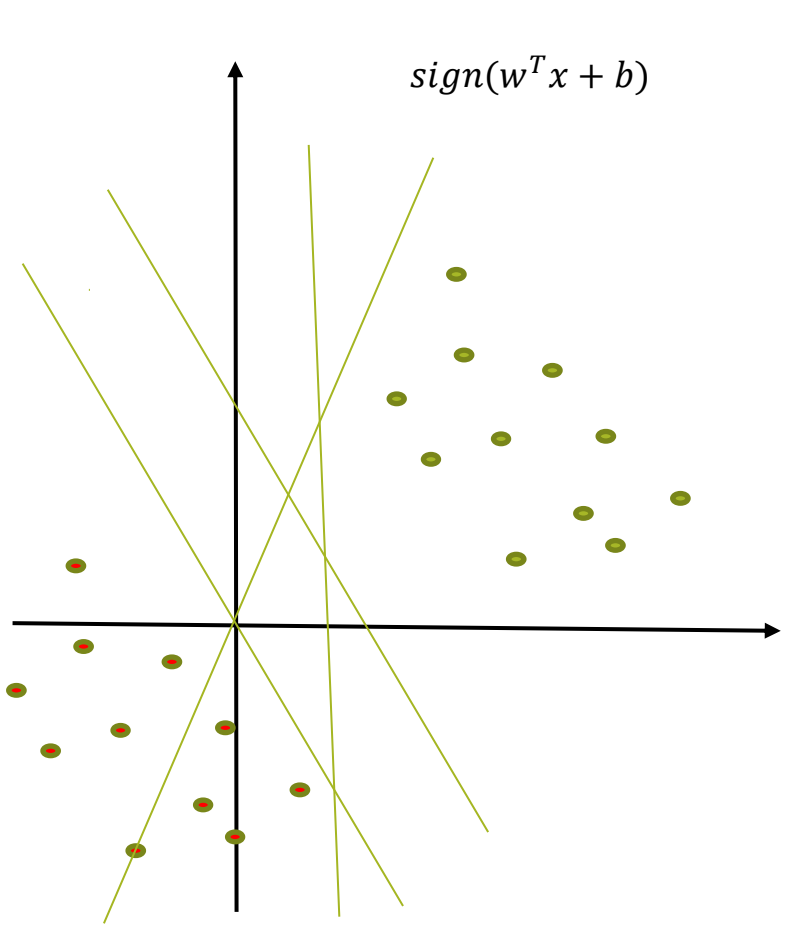



$$(p_1 \leq \text{price} \leq p_2) \text{ AND } (e_1 \leq \text{engine power} \leq e_2)$$

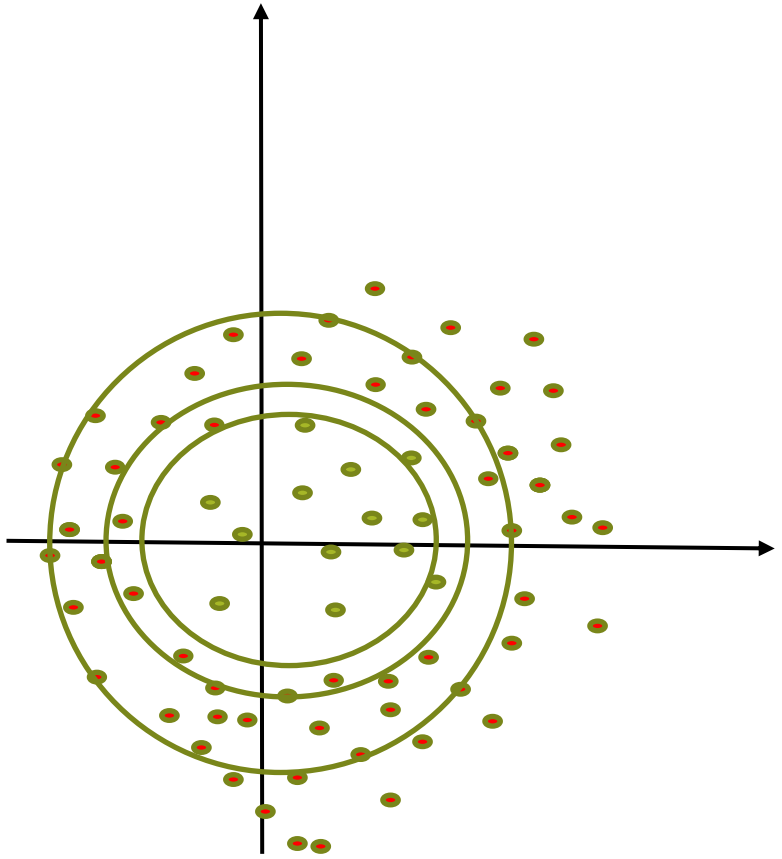
$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } h \text{ says } \mathbf{x} \text{ is positive} \\ 0 \text{ if } h \text{ says } \mathbf{x} \text{ is negative} \end{cases}$$

  - $\mathcal{H}$ is the set of all possible rectangle called the hypothesis space

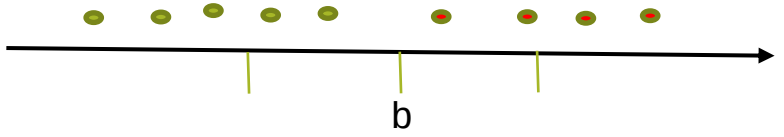
# Hypothesis space (class) $\mathcal{H}$



$sign(w^T x + b)$



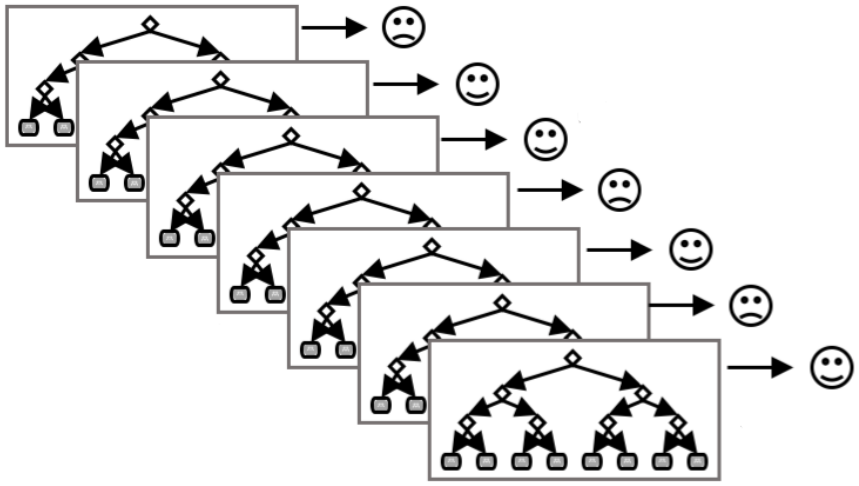
$sign(x^2 - b)$



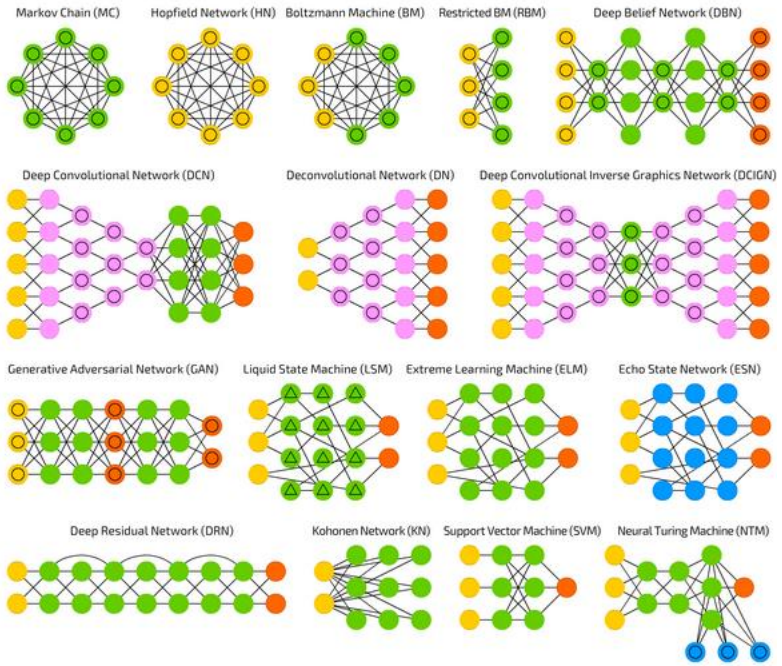

$sign(x - b)$

# Hypothesis space (class) $\mathcal{H}$

- Tree hypothesis space

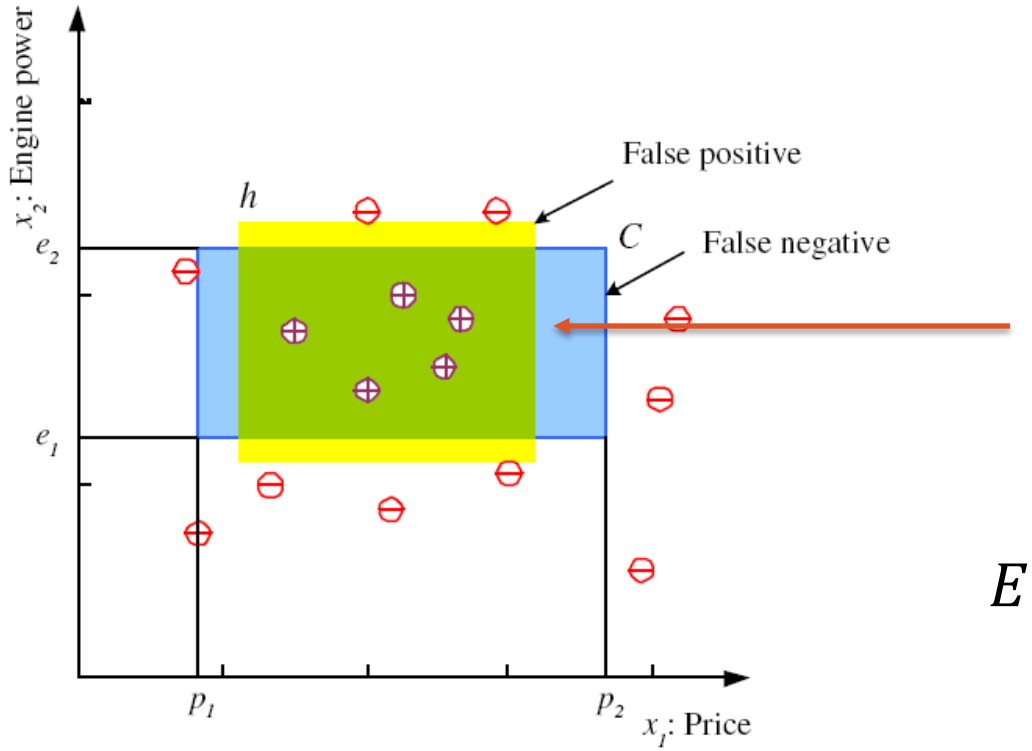


- NN hypothesis space



# Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - Assume that *h* is given: what is the Error of *h* on $\mathcal{H}$?



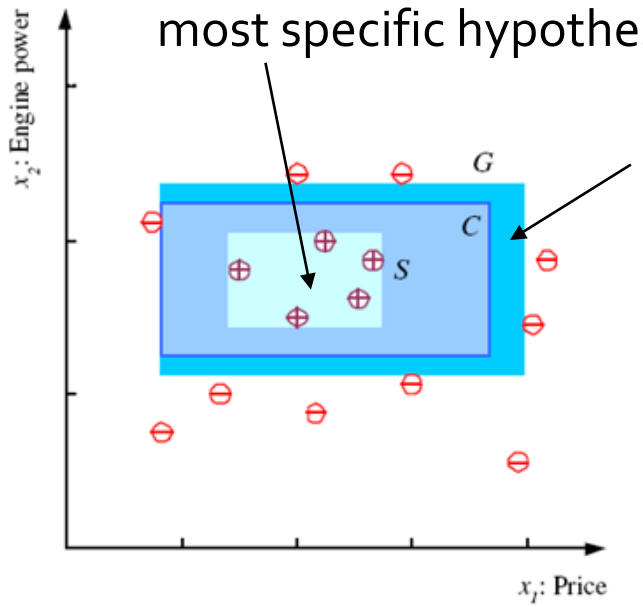

$$D = \{\mathbf{x}^t, r^t\}_{t=1}^N \subset X$$

$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } h \text{ says } \mathbf{x} \text{ is positive} \\ 0 \text{ if } h \text{ says } \mathbf{x} \text{ is negative} \end{cases}$$

$$E(h|D) = \sum_{t=1}^{N} 1(h(\mathbf{x}^t) \neq r^t)$$

# Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - True hypothesis (concept to be learned): c of $\mathcal{H}$ such that $c(\mathbf{x}^t) \neq r^t$ for $\mathcal{D} \subset X$
  - Specific S, general G



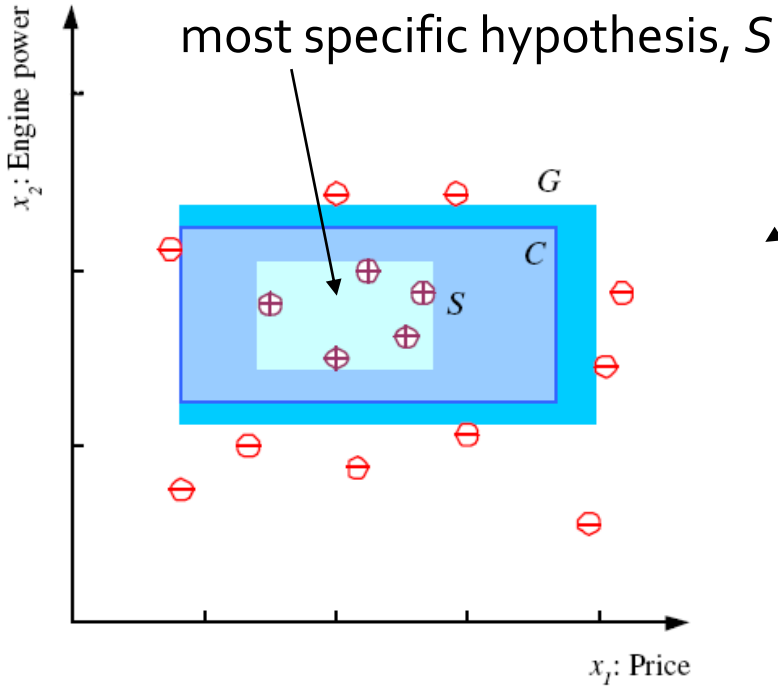

$h \in \mathcal{H}$, between S and G is

**Consistent ($E(h \mid D) = 0$)**

($h(\mathbf{x}^t) = c(\mathbf{x}^t)$ for $\mathbf{x}^t$ of $\mathcal{D}$)

# Hypothesis space $\mathcal{H}$

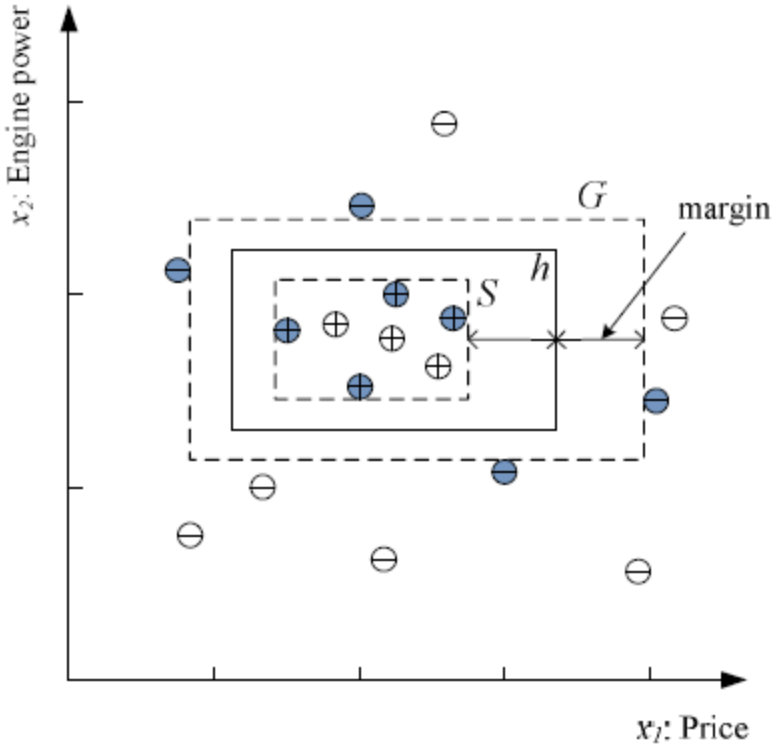
- Rectangle learning:
  - The version space (VS) with the respect to a hypothesis space $\mathcal{H}$ and the training set D is the subset of hypothesis from $\mathcal{H}$ consistant with the training set D




$h \in \mathcal{H}$, between S and G is **Consistent ($E(h \mid D) = 0$) and** make up the version space (Mitchell, 1997)

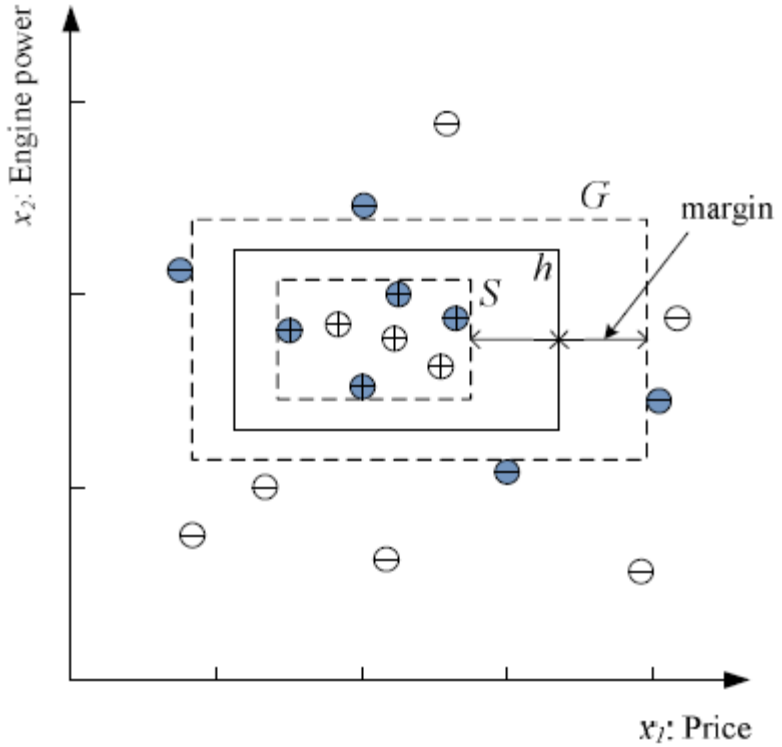
# Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - To generalize better, choose *h* with largest margin



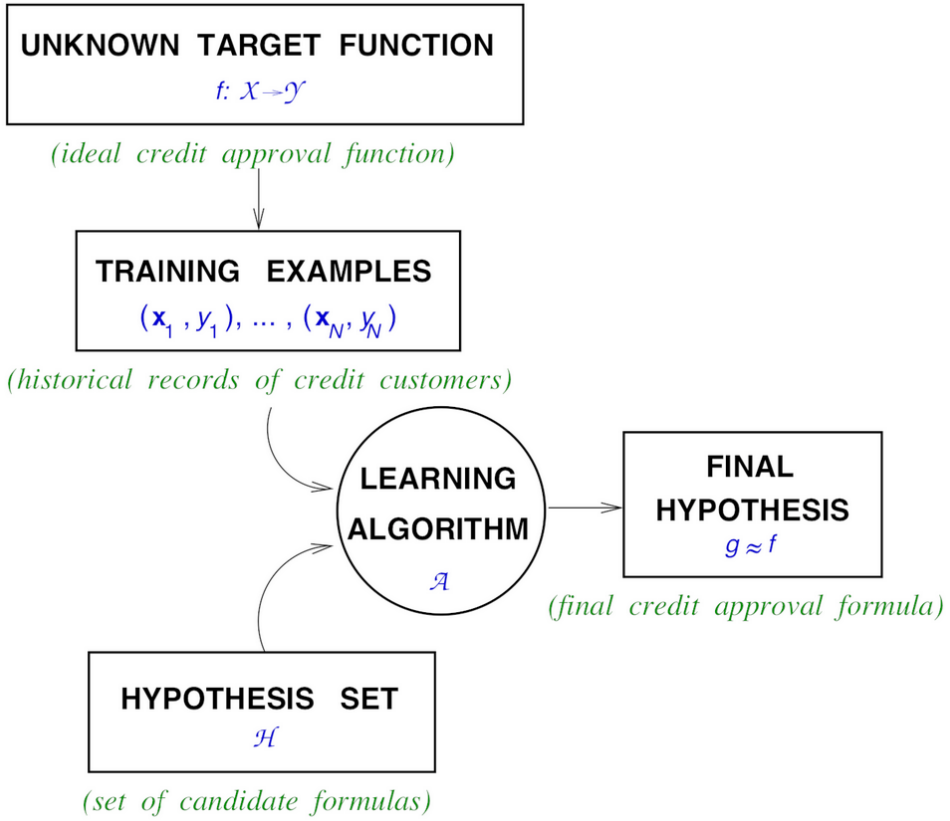
# Hypothesis space $\mathcal{H}$

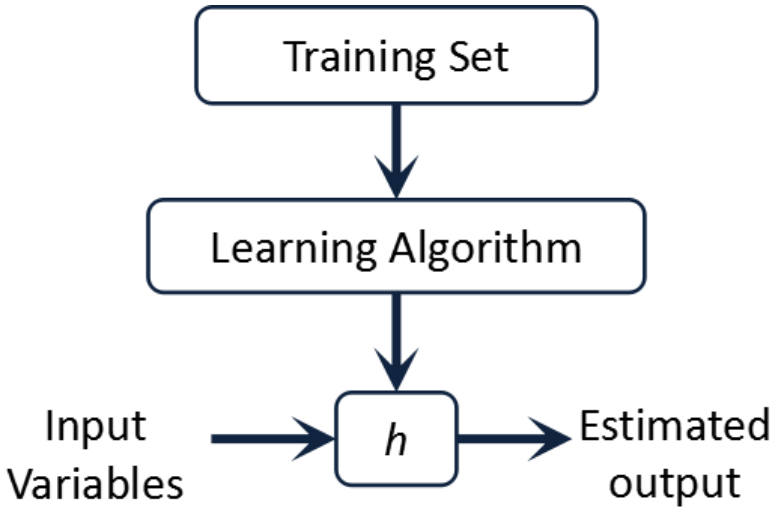
- Rectangle learning:
  - But for a given dataset X, how to compute h? **Learning algorithm (or learner)**



# Hypothesis space $\mathcal{H}$

- Each hypothesis space has it learning algorithm
  - The hypothesis set and learning algorithm together, is called a **Learning Model**



# Hypothesis space $\mathcal{H}$

- The hypothesis set and learning algorithm together, is called a **Learning Model**
  - **Question 1: what is the best learning model?**
  - **Question2: how good is a learning model, given dataset ?**
  - **Question 3: what is the capacity (complexity, expressive power, richness, or flexibility) of a learning model ?**

# Q1: No Free Launch (NFL) theorem

- In his 1996 paper, "**The Lack of A Priori Distinctions Between Learning Algorithms"**, Wolpert introduced the NFL theorem for supervised machine learning.

" The theorem states that given a noise-free dataset, for any two machine learning algorithms A and B (learning model), the average performance of A and B will be the same across all possible problem instances drawn from a uniform probability distribution"

# Q1: No Free Launch (NFL) theorem

- Does it mean that all algorithms are equal? No, of course not.
  - **Some algorithms may generally perform better than others on certain types of problems, but every algorithm has disadvantages and advantages due to the prior assumptions that come with that algorithm.**
  - XGBoost may win hundreds of Kaggle competitions yet fail miserably at forecasting tasks because of the limiting assumptions involved in tree-based models.
  - Neural networks may perform really well when it comes to complex tasks like image classification and speech detection, yet suffer from overfitting due to their complexity if not trained properly.

# Q1: No Free Launch (NFL) theorem

- In practice, this is what "no free lunch" means for you:
  - No single algorithm will solve all your machine learning problems better than every other algorithm.
  - Make sure you completely understand a machine learning problem and the data involved before selecting an algorithm to use.
  - All models are only as good as the assumptions that they were created with and the data that was used to train them.
  - Simpler models like logistic regression have more bias and tend to underfit, while more complex models like neural networks have more variance and tend to overfit.
  - The best models for a given problem are somewhere in the middle of the two bias-variance extremes.
  - To find a good model for a problem, you may have to try different models and compare them using a robust cross-validation strategy.

# Q1: Universal Approximation Theorem.

- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.

**Universal Approximation Theorem:** Fix a continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ (activation function) and positive integers $d, D$. The function $\sigma$ is not a polynomial if and only if, for every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$ (target function), every compact subset $K$ of $\mathbb{R}^d$, and every $\epsilon > 0$ there exists a continuous function $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ (the layer output) with representation

$$f_\epsilon = W_2 \circ \sigma \circ W_1,$$

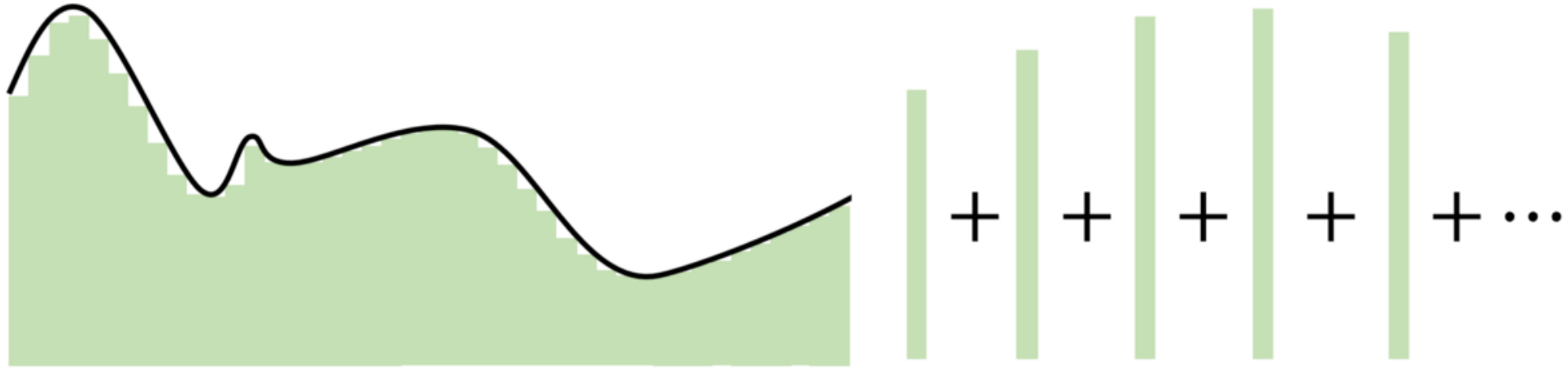
where $W_2, W_1$ are composable affine maps and $\circ$ denotes component-wise composition, such that the approximation bound

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

holds for any $\epsilon$ arbitrarily small (distance from $f$ to $f_\epsilon$ can be infinitely small).

# Q1: Universal Approximation Theorem.

- The key point in the Universal Approximation Theorem is that instead of creating complex mathematical relationships between the input and output, it uses simple linear manipulations to divvy up the complicated function into many small, less complicated pieces, each of which are taken by one neuron.



# Q1: Universal Approximation Theorem.

- The number of hidden neurons should be between the size of the input layer and the output layer.
  - The most appropriate number of hidden neurons is sqrt(input layer nodes * output layer nodes)

# Q1: Universal Approximation Theorem.

- A model with too little capacity cannot learn the problem, whereas a model with too much capacity can learn it too well and overfit the training dataset. Both cases result in a model that does not generalize well.
  - The capacity of a neural network model is defined by both it's structure in terms of nodes and layers and the parameters in terms of its weights. Therefore, we can reduce the complexity of a neural network to reduce overfitting in one of two ways:
    - Change network complexity by changing the network structure (number of weights): grid search until a suitable number of nodes and/or layers
    - Change network complexity by changing the network parameters (values of weights): keeping network weights small