1 Introduction

Machine learning is programming computers to improve a performance criterion using example data or past experience

Machine learning is the task of producing knowledge from data.

it is useful when it is not possible to model a problem:

- Human expertise does not exist
- Solution changes in time
- solution needs to be adapted to particular cases

ML allows the user to not make effort to create a model

ML is provided by:

→ Huge availability of data (Big Data)
→ Increasing computational power (GPU)
→ Recent progress in algorithms and theory

What is a learning problem?

$\boxed{\text{LEARNING = Improving with experience at some task}}$

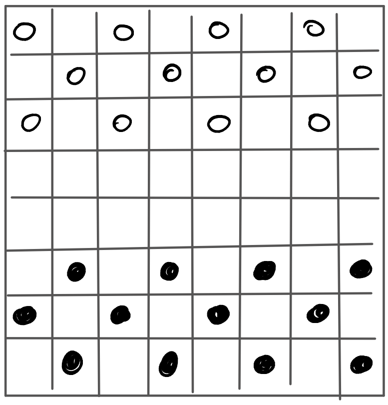
1. Improve over task T
2. with respect to performance measure P
3. based on experience E information given to the system

If you don't specify these three elements you don't have a well defined problem

If the problem is not well defined you cannot find the solution.

An example is to play checkers:

- Test : Play checkers (T)
- Performance : % games won in a world tournament (P)
- Experience : opportunity to play against self (E)



To solve this problem we have to answer some questions, and choosing the algorithm is the very last question.

- What experience? (experience is the input)
- What exactly should be learned ?
- How shall it be represented ?
  (How to represent the information ?)
- What specific algorithm to learn ?

Type of training experience
→ Human expert suggests optimal move for each configuration of the board
→ Human expert evaluates each configuration of the board
→ Computer plays against itself

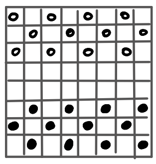
Depending on this choice, the other choices will be taken accordingly. The choice of how collecting experience is very important.

Choose a way of getting experience that is representative for the problem

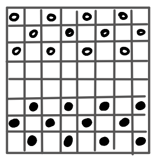
big issue in ML!

Now that we have the experience we have to decide what to learn; TARGET FUNCTION:

- ChooseMove : Board $\mapsto$ Move
- V: Board $\mapsto \mathbb{R}$



$\xrightarrow{\text{Move}}$ New configuration



possible moves

- 1.2
- 3.6
- 1.7
- 2.9

new configurations associated to values

Let's imagine a possible definition based on b that is is a particular configuration of the board, a snapshot taken at one moment during the game:

$V(b) = +100$ if b is FINAL & game is WIN

$V(b) = -100$ if b is FINAL & game is LOSS

$V(b) = 0$ if b is FINAL & game is DRAWN

If b is not a final state, then $V(b) = V(b')$ where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

This function seems perfect, but the problem is that it cannot be computed since we don't know how to play optimally.

We need an approximation. We need to represent this function (there are many ways), for the moment we consider a linear combination of some features, that we can compute given the configuration of the board.

Representation of Target functions
- → collection of rules
- → neural network
- → polynomial function of board features
- → ...

A representation for learned function:

$$V(b) = w_0 + w_1 \cdot bp(b) + w_2\, rp(b) + \dots$$

- $bp(b)$: number of black pieces on b
- $rp(b)$: number of red pieces on b
  ...
- $bt(b)$: number of red pieces threatened by black (i.e., which can be taken on black's next turn)

We DON'T know the WEIGHTS w_i, therefore learning V means estimating the weight's w_i.

We need an algorithm to estimate weights.

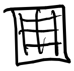
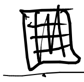
Obtaining Training examples :

Notation:

- $V(b)$: the true target function (always unknown)
- $\hat{V}(b)$: the learned function (approximation of $V(b)$ computed by the learning algorithm)
- $V_{train}(b)$: the training value obtained at $b \in$ training data

Dataset $D = \{(b_i, V_{train}(b_i))_{i=1}^{m}\}$

Dataset D (input)

V_{train}	
Board	Value
	7.223
	8.2
⋮	⋮
	6.5

Estimating training values:

- $V_{train}(b_i) \leftarrow$ human expert
- $V_{train}(b_i) \leftarrow$ set of games
- ...

The learning algorithm:

LMS (least mean square) Weight update rule

the algorithm depends on the choices done up to now

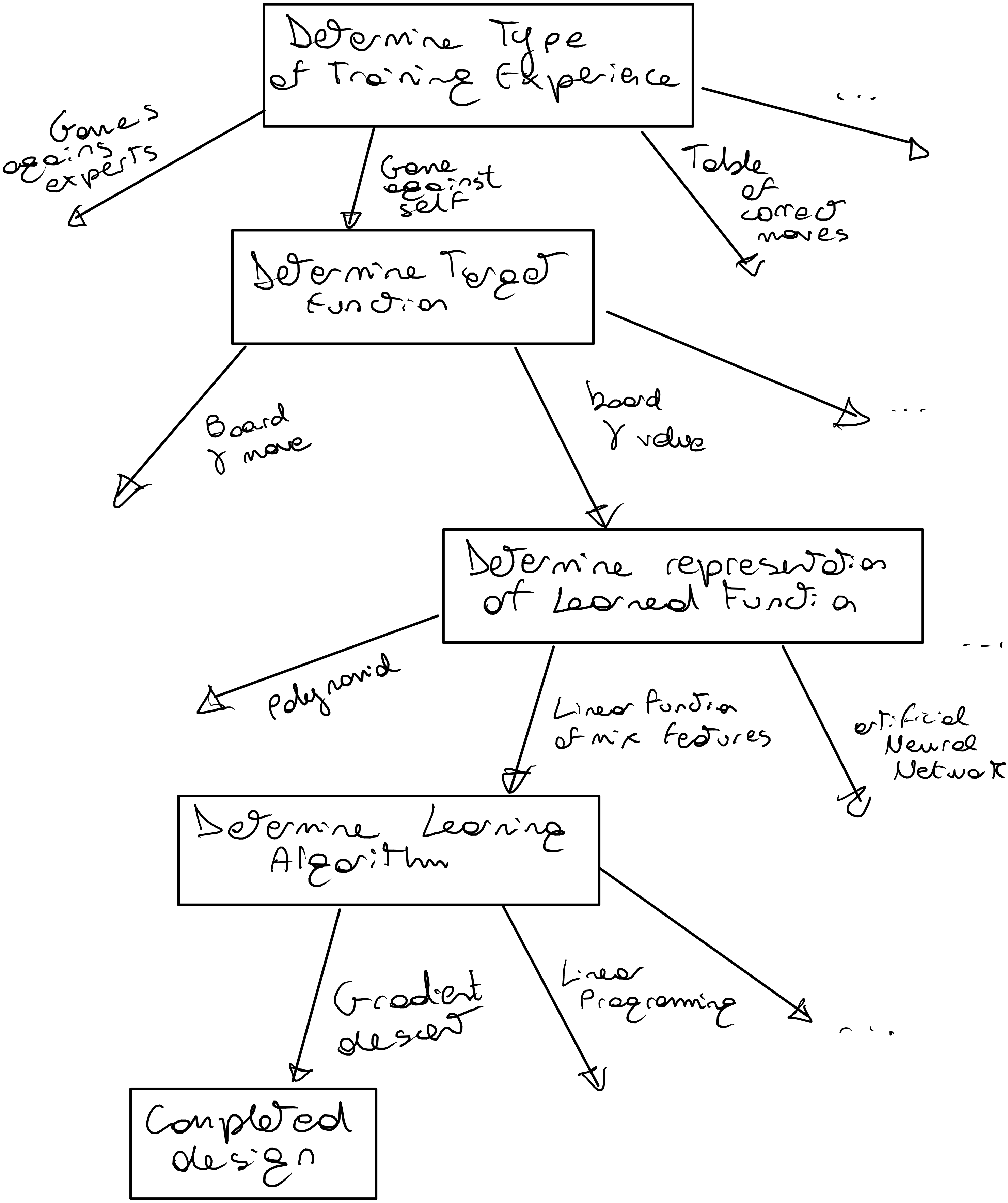
- Initialize w_i
- Do iteratively:
  - select a training example b
    compute $error(b) = V_{train}(b) - \hat{V}(b)$
  - for each feature f_i update the weight:

$$w_i \leftarrow w_i + c \cdot f_i \cdot error(b)$$

c is a small constant

Note: collecting experience is the most difficult part.

• Design choices



- ML learning problems

1. SUPERVISED LEARNING
   - Classification
   - Regression
2. UNSUPERVISED LEARNING
3. REINFORCEMENT LEARNING

The function to learn is very different

In general ML means learning a function.

Learning a function $f: X \mapsto Y$, given a dataset containing sampled information about f.

Learning a function means computing an approximated function $\hat{f}$ that returns values AS CLOSE AS POSSIBLE to f. ESPECIALLY FOR SAMPLES NOT IN D

D = training set, $X_D = \{x \mid x \in D\} \subset X$

$|X_D| << |X|$ (f is called the TRUE-FUNCTION)

- Learning a function $f: X \mapsto Y$, given:
  - $D = \{\langle x_i, y_i \rangle\}$ pairs of input-output

    for these instances I know the corresponding value.

    (SUPERVISED LEARNING)

  - $D = \{x_i\}$ I don't have any corresponding value

    (UNSUPERVISED LEARNING)

- Training a BEHAVIOUR FUNCTION $\pi: S \mapsto A$, given
  - $D = \{\langle a_i^{(1)}, \dots, a_i^{(n)}, r_i \rangle\}$

    (REINFORCEMENT LEARNING)

Reinforcement learning is applied to dynamic systems, they evolve over time.
The dataset is not a pair of input - output because is impossible to define the best action, so we have a <u>sequence of actions</u> that form an episode and a reward that tells how the episode was (won, lost, drawn).

SUPERVISED LEARNING

X: → Discrete: $A_1 \times \dots \times A_n$, A_i finite set
→ Continuous: $\mathbb{R}^n$

Y: → $\mathbb{R}^k$ Regression (II)
→ $\{C_1, \dots, C_k\}$ Classification (I)

Depending on the type of the output we define diff. problems

X - discrete (+) $Y: \{0, 1\}$ ⇒ Concept Learning

(I) Classification: Return the class to which a specific instance belongs.

Ex: face recognition, speech recognition, ...

(II) Regression: approximate a real value function

UNSUPERVISED LEARNING

Unsupervised learning is more tricky, since we have information only on the X. We can extract knowledge by using the clustering, this analysis is usually done in combination with a classification problem to SEGMENT THE INPUT IN CLUSTERS

REINFORCEMENT LEARNING

The agent learn how to behave: the right choice to take considering a particular situation (learn a policy)

In ML there are many open questions, some results are obtained in an empirical result.

- NOTATION (focus on concept learning)

c: target function, $c: X \mapsto \{0,1\}$

X: instance space, $x \in X$ is one instance

D: $\{<x_i, c(x_i)>_{i=1}^{n}\}$ training set

Evaluation can be done only is samples in D

H: HYPOTHESIS SPACE, $h \in H$ one hypothesis

H is the set of the possible functions I can compute on the possible assignments of the weights in a linear combination

$h(x)$ is the ESTIMATION of h over x.

Given a training set D, find the best approximation $h^* \in H$ of the target function.

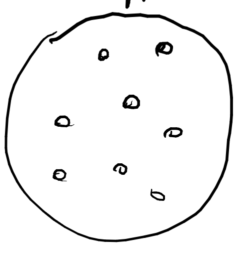
STEPS:

1. Define H
2. Define a performance metric to determine the best approximation
3. Define an appropriate algorithm.

Given a representation of H, SEARCH FOR THE BEST HYPOTHESIS $h^* \in H$, according to given performance measure.

Search problem

H



$h(x)$ can be computed for every $x \in X$ and $h(x_i) = c(x_i)$ can be verified only for $x_i \in D$.

Let's introduce the concept of consistency:

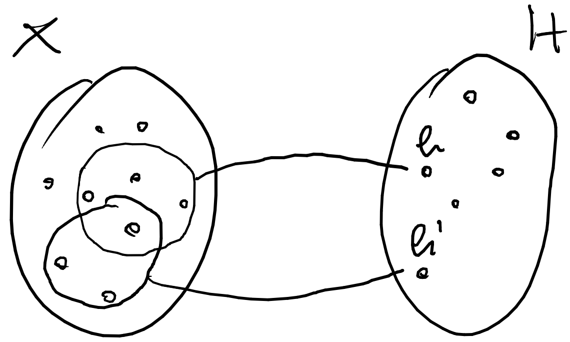
A hypothesis h is CONSISTENT with a set of training examples D of target concept c iff $h(x) = c(x)$ for each training sample in D

This is not enough because the real goal of ML system is to find the best h that predicts correct values for samples outside the dataset. (training set)

The hypothesis that does well on the dataset is likely to do well outside the dataset (TRUE IF THE DATASET IS REPRESENTATIVE OF THE PROBLEM).

INDUCTIVE LEARNING HYPOTHESIS:
Any hypothesis that approximates the target function well over a sufficiently large set of training examples will also approximate the target function well over unobserved examples.

In concept learning (i.e. binary classification) every hypothesis is associated to a set of instances (ALL INSTANCES THAT ARE CLASSIFIED BY SUCH HYPOTHESIS).



h maps to a subset of x, such that $\{x \in X \mid h(x) = 1\}$

There is a relation between $H \leftrightarrow 2^X$

Version space

The version space $VS_{H,D}$ is the subset of hypothesis that are consistent with D

$$VS_{H,D} = \{ h \in H \mid consistent(h, D) \}$$

List-then-eliminate Algorithm

$VS_{H,D} \leftarrow$ a list containing every $h \in H$

For each $\langle x, c(x) \rangle \in D$

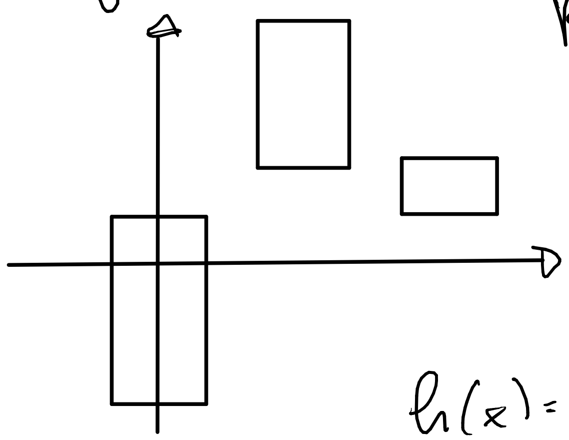
$$VS_{H,D} = VS_{H,D} - h \quad \text{s.t. } h(x) \neq c(x)$$

Output of the list $VS_{H,D}$:

The algorithm is not possible to execute since it enumerates all possible hypothesis, that could be infinite.

Let's consider an example.

- Instance space: X integer points in a 2D plane
- Hypothesis H: set of rectangles with edges parallel to the axis.



- D: positive & negative examples of points belonging to a rectangle

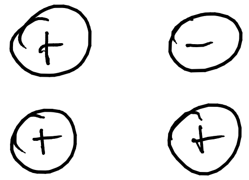
$$h(x) = \begin{cases} + & \text{if } x \in \text{Rectangle} \\ - & \text{if } x \notin \text{Rectangle} \end{cases}$$

A consistent hypothesis is one of the rectangles

Now let's consider a different hypothesis space:

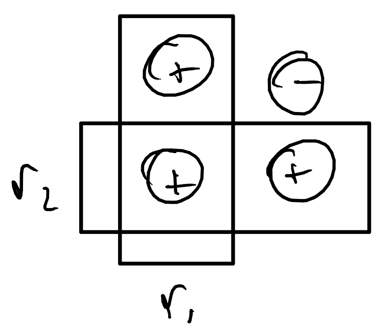
- Hypothesis H': SETS of RECTANGLES in the 2D plane with edges parallel to the axes

Let's consider the following dataset D^*:



VS_{H,D^*} is EMPTY! there is no rectangle containing + samples

VS_{H',D^*} is not empty, we have many consistent hypothesis:



H' IS MORE POWERFUL wrt H.

When you have a problem you can represent hypothesis space with different representation power (can represent more subset of X).

When we consider the relation between h and x, given a subset in X is not true that there exists h that represents the subset (the case between D^* and H). In the case of H' this is true.

I have 3 cases now to the algorithm:

1. return all consistent h, considering H'
2. return all consistent h, considering H
3. return one particular h

Let's consider a new instance $x' \notin D$ and how this will be classified by the hypothesis. We don't know if the hypothesis will be consistent since we don't have $c(x')$

If I have a set of hypotheses consistent with D, we have no guarantee that all hypotheses will agree

You may have half of hypothesis that they claim + and other half −.

If you consider a solution, for a ML problem, a $VS_{H,D}$ with H that can represent all possible subset of X, then for EVERY $x' \notin D$ WE HAVE HALF of h VOTING for + and the other for −

unuseful for ML!

To choose which is the best solution, we have to consider the two issues:

- REPRESENTATION POWER (language bias)

  constraint on all possible solutions, we have to constrain the algorithm (I want a rectangle with edges...)

- SEARCH BIAS

  Instead of returning all the solution, I want one solution considered the best