Theory of Reinforcement Learning

Csaba Szepesvri

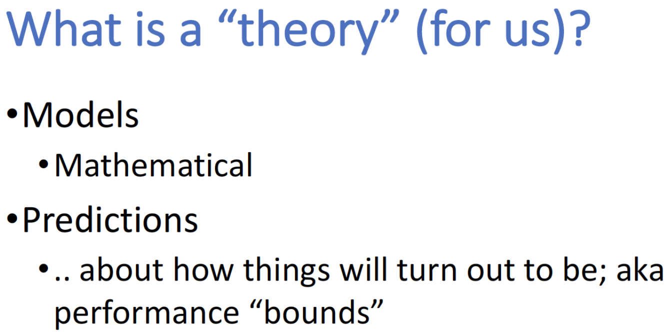
University of Alberta

Reinforcement Learning Summer School 2017
Presenter: Chao Jiang

Take home message of the video

- Mathematical Model + Predictions = Theory
- Theory can help practice, empirical work, inspires theory work
- RL $\neq$ Supervised Learning
  - Batch learning
  - When you have a simulator: fitted value iteration
  - Online: Bandit
- VC dimension

Theory



What is a “theory” (for us)?

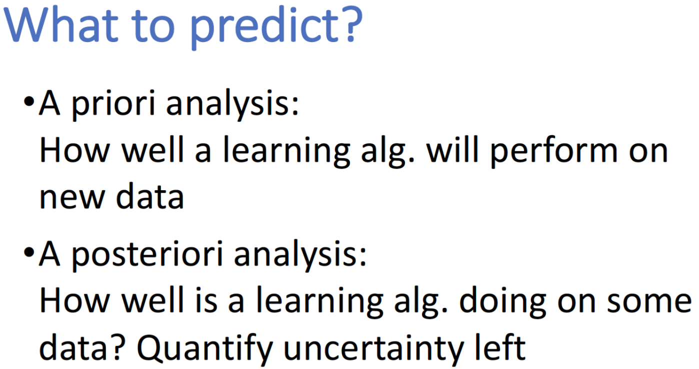
- Models
  - Mathematical
- Predictions
  - .. about how things will turn out to be; aka performance “bounds”

Theory

Statistical learning theory: ingredients

- Distributions $P \in \mathcal{P}$
- i.i.d. samples $S_n \sim P^n$
- Learning algorithms $A: S_n \mapsto h$
- Predictors $\rotatebox{90}{$\in$}\ \mathcal{H}$
- Loss functions $\ell: \mathcal{H} \to [0, \infty)$

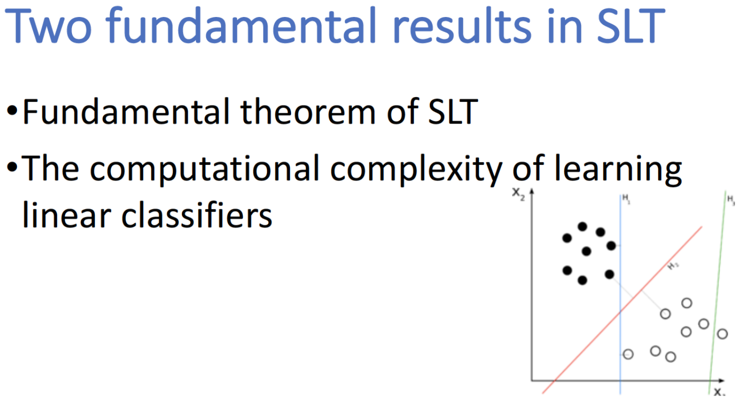
Theory



What to predict?

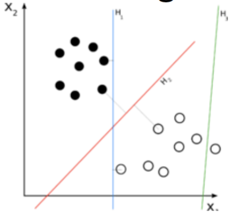
- A priori analysis:
  How well a learning alg. will perform on new data
- A posteriori analysis:
  How well is a learning alg. doing on some data? Quantify uncertainty left

Theory

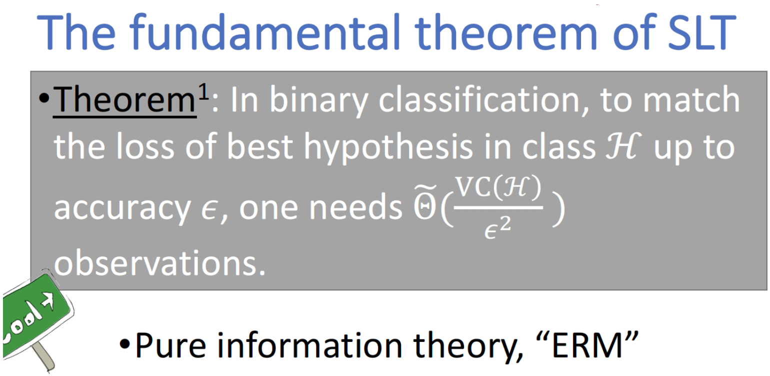


Two fundamental results in SLT

- Fundamental theorem of SLT
- The computational complexity of learning linear classifiers



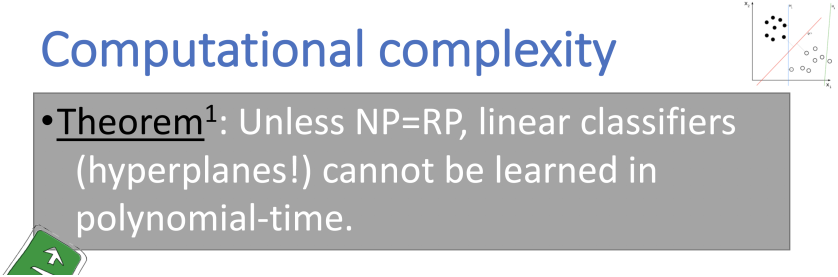
Theory



The fundamental theorem of SLT

- Theorem[1]: In binary classification, to match the loss of best hypothesis in class $\mathcal{H}$ up to accuracy ϵ, one needs $\widetilde{\Theta}(\frac{VC(\mathcal{H})}{\epsilon^2})$ observations.

- Pure information theory, “ERM”



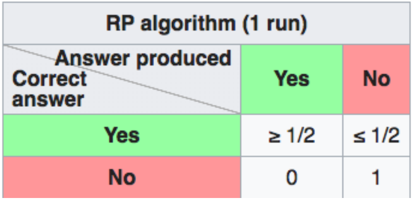
Computational complexity

- Theorem[1]: Unless NP=RP, linear classifiers (hyperplanes!) cannot be learned in polynomial-time.

RP (complexity)

In computational complexity theory, randomized polynomial time (RP) is the complexity class of problems for which a probabilistic Turing machine exists with these properties:

- It always runs in polynomial time in the input size
- If the correct answer is NO, it always returns NO
- If the correct answer is YES, then it returns YES with probability at least $1/2$ (otherwise, it returns NO).



RP algorithm (1 run)		
Answer produced / Correct answer	Yes	No
Yes	≥ 1/2	≤ 1/2
No	0	1

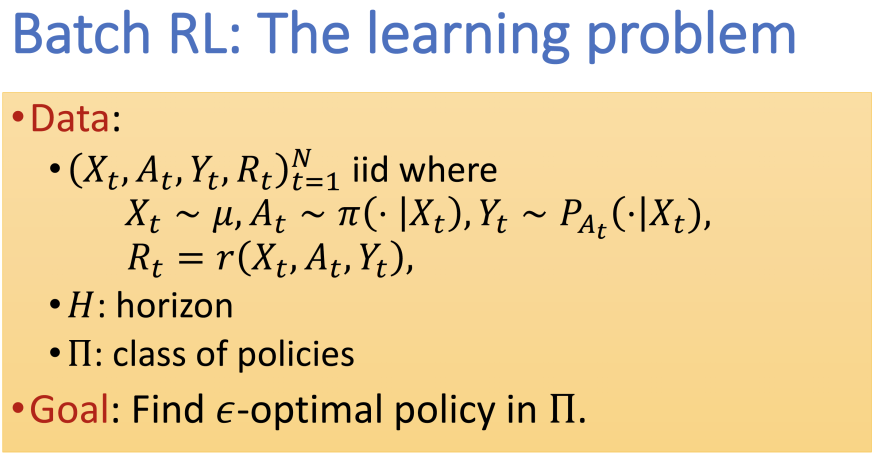
P is a subset of RP, which is a subset of NP

Batch Rineforcement Learning

- differing from the RL, in the batch learning problem the agent itself is not allowed to interact with the system during learning.
- Instead of observing a state s, trying an action a and adapting its policy according to the subsequent following state s' and reward r, the learner only receives a set $F = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, ..., p\}$ of p transitions (s, a, r, s') sampled from the environment.
- In the most general case of this batch reinforcement learning problem the learner cannot make any assumptions on the sampling procedure of the transitions. They may be sampled by an arbitraryeven purely randompolicy; they not even be sampled along connected trajectories.

Batch Reinforcement Learning

- During this application phase the policy is fixed and not further improved as new observations come in.
- Since the learner itself is not allowed to interact with the environment, and the given set of transitions is usually finite, the learner cannot be expected to always come up with an optimal policy.
- The objective has therefore been changed from learning an optimal policyas in the general reinforcement learning caseto deriving the best possible policy from the given data.



Batch RL: The learning problem

- Data:
  - $(X_t, A_t, Y_t, R_t)_{t=1}^N$ iid where
    $X_t \sim \mu, A_t \sim \pi(\cdot\,|X_t), Y_t \sim P_{A_t}(\cdot|X_t),$
    $R_t = r(X_t, A_t, Y_t),$
  - H: horizon
  - Π: class of policies
- Goal: Find ϵ-optimal policy in Π.

Theory

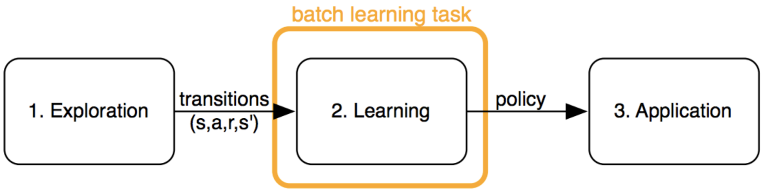


Fig. 1 The three distinct phases of the batch reinforcement learning process: 1: Collecting transitions with an arbitrary sampling strategy. 2: Application of (batch) reinforcement learning algorithms in order to learn the best possible policy from the set of transitions. 3: Application of the learned policy. Exploration is not part of the batch learning task. During the application phase, that isn't part of the learning task either, policies stay fixed and are not improved further.

Batch RL and supervised learning

- Corollary 1: For $H = 0$, batch RL is "cost sensitive classification" (CSS) with cost $-r(x, a)$ at input x and "label" a and "hypothesis class" Π.
- Corollary 2: The "Batch RL" learning problem is at least as hard as CSS

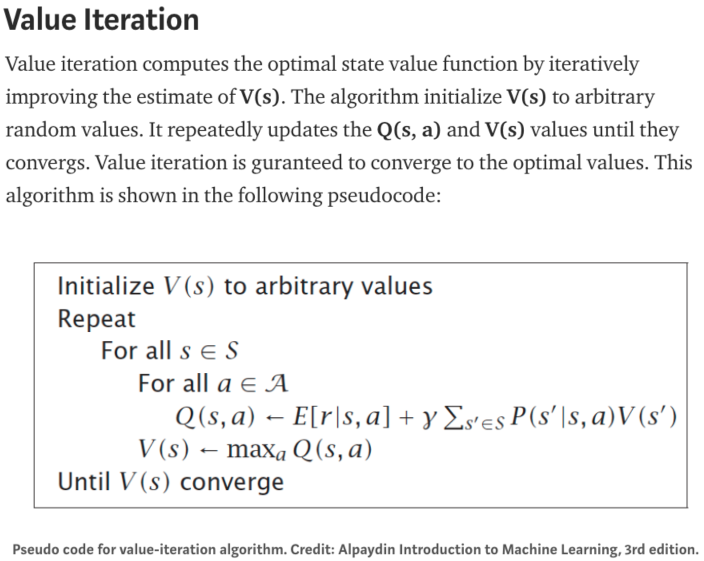
- Obviously, the distribution of transitions in the provided batch must resemble the true transition probabilities of the system in order to allow the derivation of good policies.
- The easiest way to achieve this is to sample the training examples from the system itself, by simply interacting with it.
- But when sampling from the real system, another aspect becomes important: the covering of the state space by the transitions used for learning.
- If important regions, e.g. states close to the goal stateare not covered by any samples, then it is obviously not possible to learn a good policy from the data, since important information is missing.

Fitted Value Iteration

..when you have a simulator

..anyone wants to play Atari games?

Fitted Value Iteration



Value Iteration

Value iteration computes the optimal state value function by iteratively improving the estimate of **V(s)**. The algorithm initialize **V(s)** to arbitrary random values. It repeatedly updates the **Q(s, a)** and **V(s)** values until they convergs. Value iteration is guranteed to converge to the optimal values. This algorithm is shown in the following pseudocode:

```
Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge
```

Pseudo code for value-iteration algorithm. Credit: Alpaydin Introduction to Machine Learning, 3rd edition.

Fitted Value Iteration

- The value function represent how good is a state for an agent to be in.
- Value Iteration works well if the number of states is small. The reason for this is that we have to be able to represent our approximation $\hat{V}$ as a single n $\times$ 1 vector, where n is the number of states. However, the number of states is large or infinite, making this infeasible or impossible.

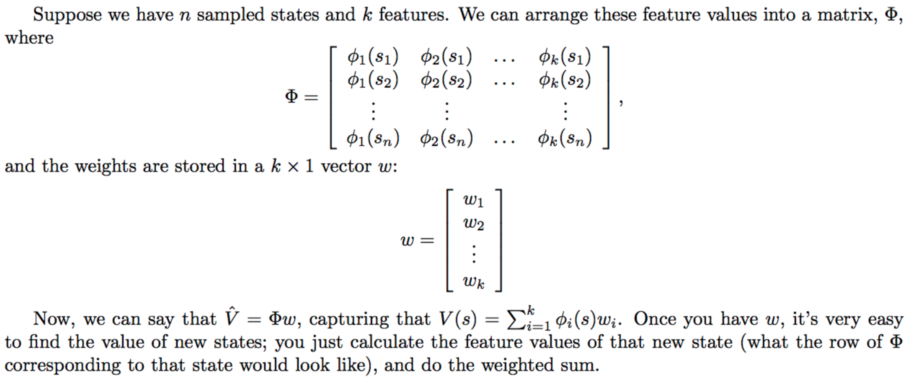
Fitted Value Iteration

In this case, we need a different way of representing our approximation $\hat{V}$, which meets the following requirements:

- The approximation should be able to be computed given a small subset of the total number of states.
- The approximation should be easily extended to new states which were not part of the training set.
- For many applications, we need this extension to be quickly computable.

Fitted Value Iteration

One approach to this is to use a linear approximation, in which $\hat{V}(s)$ is set equal to a weighted sum of features.



Suppose we have n sampled states and k features. We can arrange these feature values into a matrix, Φ, where

$$\Phi = \begin{bmatrix} \phi_1(s_1) & \phi_2(s_1) & \dots & \phi_k(s_1) \\ \phi_1(s_2) & \phi_2(s_2) & \dots & \phi_k(s_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(s_n) & \phi_2(s_n) & \dots & \phi_k(s_n) \end{bmatrix},$$

and the weights are stored in a $k \times 1$ vector w:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

Now, we can say that $\hat{V} = \Phi w$, capturing that $V(s) = \sum_{i=1}^{k} \phi_i(s) w_i$. Once you have w, it's very easy to find the value of new states; you just calculate the feature values of that new state (what the row of Φ corresponding to that state would look like), and do the weighted sum.

Fitted Value Iteration

But how do we find w? Suppose we have a vector b that we would like to approximate using a linear approximation, so that $\Phi w \approx b$. Consider the following steps:

$$\begin{aligned} \Phi w &= b \\ \underbrace{\left(\Phi^T \Phi\right)^{-1} \Phi^T \Phi}_{=I}\, w &= \left(\Phi^T \Phi\right)^{-1} \Phi^T b \\ w &= \underbrace{\left(\Phi^T \Phi\right)^{-1} \Phi^T}_{\Pi}\, b \end{aligned}$$

You'll notice on the second step, the underbraced portion is noted as equal to I, the multiplicative identity matrix. This is because $A^{-1}A = I$ for any invertible matrix A.

You'll notice on the third step that $\left(\Phi^T \Phi\right)^{-1} \Phi^T$ is labelled as Π, or the projection operator. This is because it projects b into the space defined by Φ. This is one way to calculate w.

Theory

Input: $\mathcal{F}$ – function space, N, M, K integers, μ – distribution over the state space.

Algorithm (stage k):

1. Sample "basis points": $X_1, \dots, X_N \in \mathcal{X}$, $X_i \sim \mu$
2. For each action $a \in \mathcal{A}$ and state X_i, sample next states and rewards: $Y_j^{X_i,a} \sim P(\cdot|X_i, a)$, $R_j^{X_i,a} \sim S(\cdot|X_i, a)$, $j = 1, \dots, M$
3. Calculate the Monte-Carlo approximation of backed up values:

$$v_i = \max_{a \in \mathcal{A}} \frac{1}{M} \sum_{j=1}^{M} \left[R_j^{X_i,a} + \gamma V_k(Y_j^{X_i,a}) \right], \quad i = 1, 2, \dots, N.$$

4. Solve the least-squares problem: $V_{k+1} = \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - v_i)^2$

Fitted Value Iteration

..no simulator, no pain..? Uh..no..

When things became “real”

Fitted Value Iteration

Defining online learning

- Interact with “real” system
- Collect as much reward as possible!
- Performance metric:
  - Total reward collected, or..
  - Regret: Difference to baseline (normalizing)
- PAC-MDP: not covered

Fitted Value Iteration

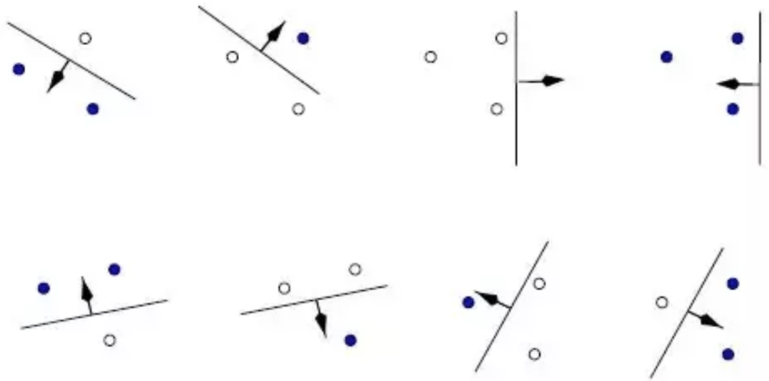
Why should you care?

- **Alternative**: Model-based RL
  - Learn a model & use planning (see previous part)
- Problems with model-based RL:
  - Models can be too expensive to build
  - Uncontrolled model inaccuracies may lead to poor behavior
- **Opportunity**: Online learning can be cheaper

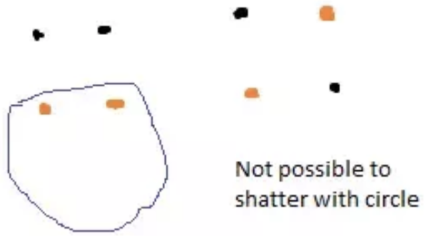
VC dimension

- In VapnikChervonenkis theory, the VC dimension is a measure of the capacity (complexity, expressive power, richness, or flexibility) of a space of functions that can be learned by a statistical classification algorithm.
- Given a classifier X , if X can classify a pattern with n numbers of points in space with all possible labels(2^n , for (+ and -) two label) then n is VC dimension of X.

VC dimension



VC dimension



Are larger VC dimensions good?

- So we would want a function family with a high VC dimension, right?
- What we eventually want is that our model predicts well. This eventual performance is denoted by a quantity called Risk, and it is bounded thus:
- $Risk = EmpiricalRisk + f(h)$
- The Empirical Risk is the classification error you obtain on our training set. The second term f(h), is a function that increases with the VC dimension.