UDACITY

Reinforcement Learning

Michael Littman & Charles Isabel, Georgia Tech

Lectures

Contents

Summary

What is RL

- **RL = make decisions that maximize rewards over time**.
- Basic model of the world – **MDP: (S,A,T,R)**
  - States of world; actions that can be made in each state; Markovian transition to the next state T(s,a,s'); reward of the last transition/action R(s,a,s') or R(s,a); 1 time-unit per action.
- Sources of **uncertainty** (not all of them are included in every RL problem):
  - **Stochasticity**: action a in state s may result in non-deterministic:
    - s' (modeled through T(s,a,s'))
    - R(s,a) (see *stochasticity of rewards* below)
  - **Unknown world**: the states and transitions of the MDP are unknown – see *exploration*.
  - **Partial observations**: the current state is unknown – only a partial observation of it is available (see *POMDP* below).
  - Note: uncertainty may be converted from one type to another by different modeling.

Relations to other problems

- **Supervised learning**:
  - RL maximizes rewards rather than approximates a desired known output.
  - RL lives in time-oriented world – sequential problems which depend on each other through transitions over time.
- **Search / Planning**: RL needs exploration to reveal unknown parts of the world (in the same time of collecting rewards).
- **Markov Chains (or Markov Processes, or HMM)**: RL needs to choose actions and collect rewards in addition to learning the world.

Finding an optimal policy

- **Model-based** approach – learn the model of the world and use it to find an optimal policy:
  - Explore the world: learn transitions and rewards to form the corresponding MDP.
  - Solve the MDP: for every state s, estimate $V(s)$ which summarizes both immediate and future rewards.
  - Choose the policy: $\pi(s) \coloneqq \underset{a}{\operatorname{argmax}}\, R(s,a) + E[V(s_{next})|s,a] = \underset{a}{\operatorname{argmax}}\, Q(s,a)$.
- **Model-free** approach – don't bother to learn transitions & rewards explicitly:
  - Learn the values of states and/or actions while exploring the world with some policy.
  - Update the policy from time to time according to the values.
- **The tradeoff**: model requires more work, model-free requires more (typically simulative) data.

Model-free solutions

- **Temporal Difference** (**TD**): update states values while applying some policy (repeatedly in ending MDP or continuously in non-ending one).
  - **TD(0)**: at every step t, re-evaluate $V(s_{t-1})$ using $V(s_t)$:
    $$V(s_{t-1}) \coloneqq V(s_{t-1}) + \alpha_t[r_t + \gamma V(s_t) - V(s_{t-1})]$$
  - **TD(1)**: at every step t, re-evaluate all the states which preceded s_t.

        - This essentially leads to evaluation of any $V(s_{t-1})$ by $V(s_\infty)$, which is more direct (hence less sensitive to Markov assumption) but applies weaker exploitation of the data (since $V(s_{t-1})$ does not rely on the futures of s_t from previous visits).
    - It is also possible to update K-steps back (rather than 1 step as in TD(0) or ∞ steps as in TD(1)). Usually a weighted mixture of K's is used, where the weighting is controlled through a parameter $0 \leq \lambda \leq 1$, yielding $\boldsymbol{TD(\lambda)}$.
    - Note: learning may be sensitive to the (either constant or dynamic) policy used.
    - More generally, the **TD approach** allows online evaluation of future-dependent quantities in any Monte-Carlo-sampled process: at any step t we re-evaluate according to available data, which is helpful when multiple dependent quantities ($\{V(s)\}_s$) are evaluated simultaneously.
- **Q-learning**: learn $Q(s,a)$ directly rather than $V(s)$.
    - Can be seen as a variant of the general TD approach.
    - Very simple to implement, and guaranteeing (possibly slow) convergence to Q^*.

Model-based solutions

- **State Evaluation**: Immediate reward + future rewards.
- Formalized as a recursive equation – **Bellman equation** – with reduction γ.
    - $\boldsymbol{V(s) = \max_a \sum_{s'} T(s,a,s')\left(R(s,a,s') + \gamma V(s')\right)}$
    - Alternative formulation uses $Q(s,a)$ (defined by $V(s) = \max_a Q(s,a)$) instead of $V(s)$.
- [Method I] Direct **Value Iteration**: $\forall s$: $V_{t+1}(s) \coloneqq \max_a \sum_{s'} T\left(R + \gamma V_t(s')\right)$ (iterate over t)
- [Method II] **Linear Programming** (less common): rewrite Bellman eq. as a set of linear constraints $V(s) \geq \cdots$ (instead of "V(s)=max…") and minimize V(s) under these constraints (leading back to equality).
- [Method III] **Policy Iteration** – improve policy iteratively:
    - Policy evaluation: solve simplified Bellman eq. (with $a \coloneqq \pi(s)$ instead of $\max_a$) to find $\{V(s)\}_s$ wrt the current policy – either with VI (shorter without the $\max_a$) or by solving linear system (one linear eq. per state).
    - Policy improvement: update policy greedily.

Exploration

- **Random exploration**: in step t, take a random action with probability p_t.
    - **Exploration-exploitation tradeoff**: $p_t \coloneqq 1/t \to 0$ guarantees convergence of policy, while $\sum p_t \to \infty$ guarantees unbounded exploration.
- $\boldsymbol{R_{max}}$: always maximize value (no random actions), but assume that unexplored states/actions have high rewards.
- **Stochasticity of rewards** can be handled by repeating exploration of the same states/actions, and estimating rewards using the Central-Limit Theorem.
    - In R_{max} with stochastic rewards, until a state is sampled enough times (according to CLT), the missing samples can be replaced with high rewards.

Additional tips

Setting rewards

- Domain-knowledge.
- **Potential-based rewards reshaping** may help accelerate learning.

Generalization and Scale-up

- **Components to generalize: value functions, policies and transition-models.**
- **Value function generalization**: Instead of learning $Q(s,a)$ separately for any (s,a) – Learn $\boldsymbol{Q}\big(\boldsymbol{f}(\boldsymbol{s})\big)$ for features of the states.
  - The learned $Q\big(f(s)\big)$ is often linear ($\sum_i w_i f_i(s)$) or a Neural Network.
    - Note: GD updates of $Q\big(f(s)\big)$ may diverge, especially if the features poorly distinct the various states, or if the initial Q's are too big (compared to the rewards).
    - Averaging generalization Q-function (e.g. hard/soft KNN) can prevent divergence.
  - That's actually supervised-learning which generalizes the value-assignment according to observations made by the exploration of the RL.
  - Equivalently (and more elegantly), the MDP may be re-defined in terms of the features.
- **Options**: abstract actions which consist of (possibly conditional) sequences of actions, and last variant amount of time (in contrast to the 1-time-unit actions of MDP).
  - **Semi-MDP**: MDP with options. Bellman equation can be generalized.
- **Modular RL** (**goal abstraction**): define multiple goals, define rewards according to each goal separately, learn Q-values for each goal separately, and aggregate all Q-values using voting.
- **Monte-Carlo Tree Search**: local approximation of the Q-value of an MDP, which is based on simulating several short futures from any encountered state, and choosing action accordingly.

Extended model: limited observations (current state is unknown to the agent)

- Partially-observable model:
  - **POMDP**: the model includes observations z with probabilities O(s,z).
  - **Belief state**: $b \coloneqq (P(s))_{s\in S}$ – allows Bayesian State-Estimation (**SE**) every step according to the observations.
- **EM**: alternatively refine the belief state and the transitions-model.
- **Bayesian RL**: find mixed (random) action which maximizes the expected value over all possible states (with weights b(s)).
- **Predictive State Representation** (**PSR**): redefine states by the distribution of the future observations & rewards.
  - Naïve state representation – redefining states according to available observations (i.e. $s \coloneqq z$) – makes the Markov assumptions too strong (the current observation isn't informative enough to predict future rewards).

[II,III] Basics

- **MDP**, rewards, q-values, policies, **Bellman Equation** (variants with V,Q,C).

[IV,V] TD(λ) and Convergence

- **Temporal Difference** (**TD**): update states values while applying some policy (repeatedly in ending MDP or continuously in non-ending one).
  - **TD(0)**: at every step t, re-evaluate $V(s_{t-1})$ using $V(s_t)$:
    $$V(s_{t-1}) \coloneqq V(s_{t-1}) + \alpha_t[r_t + \gamma V(s_t) - V(s_{t-1})]$$
  - **TD(1)**: at every step t, re-evaluate all the states which preceded s_t.
    - This essentially leads to evaluation of any $V(s_{t-1})$ by $V(s_\infty)$, which is more direct (hence less sensitive to Markov assumption) but applies weaker exploitation of the data (since $V(s_{t-1})$ does not rely on the futures of s_t from previous visits).
  - It is also possible to update K-steps back (rather than 1 step as in TD(0) or ∞ steps as in TD(1)). Usually a weighted mixture of K's is used, where the weighting is controlled through a parameter $0 \leq \lambda \leq 1$, yielding $\boldsymbol{TD(\lambda)}$.
  - Note: learning may be sensitive to the (either constant or dynamic) policy used.
  - More generally, the **TD approach** allows online evaluation of future-dependent quantities in any Monte-Carlo-sampled process: at any step t we re-evaluate according to available data, which is helpful when multiple dependent quantities ($\{V(s)\}_s$) are evaluated simultaneously.
  - **Q-learning**: learn $Q(s,a)$ directly rather than $V(s)$.
    - Can be seen as a variant of the general TD approach.
    - Very simple to implement.
    - **Convergence** (possibly slow) to the optimal Q^* is guaranteed.
      - Proven by the fact that the update in Q-learning is a contraction operator over Q-functions space.

[VI] Value-Iteration, Linear-Programming and Policy-Iteration

- **Value Iteration**: given a policy, assign values to states iteratively (starting from the rewards and updating according to the policy, the following states and the discount factor gamma).
- **Linear Programming** for the MDP problem:
  - Original representation – find max vs which equals the following: $\forall s: v_s = \max_a(R + \gamma E[v_{s'}])$ – constraint is not linear!
  - New representation – find minimal vs which is larger than the following: $min \sum v_s$ such that $v_s \geq R + \gamma E[v_{s'}]$ – that's solvable through linear programming! Though
  - Dual form of the LP problem – variables and constraints are swapped – derives a different interpretation of MDPs (maximize rewards such that the "policy-flow" that enters each state equals the flow that leaves it).
  - LP usually isn't very practical for MDPs.
- **Policy Iteration** (Ron Howard): while policy is not optimal (in sense of Bellman Equation): (1) assign values to states (e.g. using Value Iteration wrt current policy); (2) update policy to maximize the currently-assigned values.
  - Policy Iteration works – converges without local optima.

[VII] Rewards Reshaping

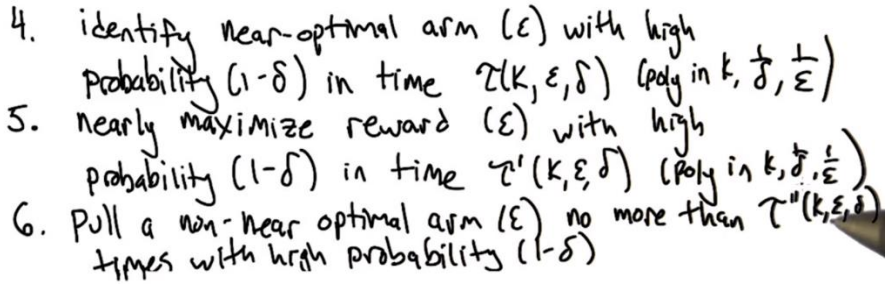
- **Rewards can be redefined in ways that don't change the optimal policy, but may allow a learning algorithm to learn that policy more quickly**.
  - E.g. if a robot gets a reward for moving towards a ball, hitting it, and making it reach the goal – then the robot will practically never encounter the rewarded states enough to understand what it needs to do. However, if it is slightly rewarded for getting closer to the ball and moving it towards the direction of the goal, then it will naturally explore more relevant states and actions.
  - Note: **reshaping rewards** expresses our **Domain Knowledge**.
- Adding a constant scalar to all rewards or multiplying by a positive scalar don't change the optimal policy, though they're not very useful either.
- **Potential-based reward reshaping**:
  - Theorem: if we assign values for states F(s) in advance, and redefine the rewards as **R'(s,a,s') := F(s')-F(s)+R(s,a,s')** (i.e. the actual reward + the value of the new state – the value of the old state), then the optimal policy doesn't change.
  - Theorem [E. Rewarf]: Q-learning with potential function F(s) ≡ Q-learning with Q-values initialized to Q0(s):=F(s), i.e. **potential-based rewards reshaping is equivalent to initialization of the values**.
    - **In particular, randomization of the initial Q-values is equivalent to randomized potential function, thus is not recommended**.

[VIII] Exploration

- **K-armed Bandits problem** (מכונת מזל עם כמה ידיות):
  - Can be represented as a degenerated MDP (e.g. single state with K actions and randomized rewards).
  - Basically learn the distribution of each bandit and then choose the best bandit repeatedly.
  - If we're interested in maximizing the expected reward, then by the Central Limit Theorem, after n samples of a certain bandit we know that $\mu \in I_{\delta,n}$ with probability $1-\delta$, where $I_{\delta,n}$ is:

$$\left[\hat{\mu} - \frac{z_\delta}{\sqrt{n}}, \hat{\mu} + \frac{z_\delta}{\sqrt{n}}\right] \quad z_\delta = \sqrt{\frac{1}{2}\ln\frac{2}{\delta}}$$

  - By **setting $N_{K,\delta,\epsilon}$ such that $\frac{z_{\delta/K}}{\sqrt{N}} < \epsilon$**, we receive an **$\epsilon$-approximation of μ (hence we find ϵ-near optimal arm) with probability $\geq 1-\delta$ within $K \cdot N$ steps**.
    - $N_{K,\delta,\epsilon} = \frac{2}{\epsilon^2} ln\left(\frac{2K}{\delta}\right)$, hence **accuracy is much more expensive than certainty**.
  - Note: it turns out that all the **following criteria are equivalent** for algorithms of the bandits problem, hence all of them are achieved by the algorithm described above:



- Exploring deterministic MDPs (transitions and rewards are deterministic but unknown) – $\boldsymbol{R_{max}}$ exploration method:
  - Set some global R_{max} (intended to be larger than any actual reward), and for any unknown reward R(s,a,s') assume that $R(s,a,s') = R_{max}$.
  - Each time a new reward is observed, the new approximation of the MDP needs to be solved (i.e. the policy needs to be recomputed).
  - When the exploration is done, the solution of the current approximation of the MDP is optimal for the actual MDP.
    - Note: **due to rewards discount, the exploration may be done** (in sense of stopping observing new rewards) **without exploring all the possible rewards**: for aggressive discount γ, far states may remain unobserved –in cases where even the optimistic reward $\gamma^{\#steps} \cdot R_{max}$ isn't high enough.
  - Due to assumption of determinism, the **exploration can be done within $\boldsymbol{n^2k}$ steps** (nk state-action pairs to observe, and up to n states to get to a new observation), which also **bounds the number of non-optimal actions** used for exploration.
  - Note: assuming optimistic scores for unexplored states is similar to the idea of all Breath-based graph search algorithms (BFS, Dijkstra, **A***), in which the frontier states receive optimistic heuristic scores $score(s) \coloneqq d(s_0, s) + d_{optimistic}(s, s_f)$.
- **<u>Exploring stochastic MDPs</u>** (with n states and k actions) using **General $\boldsymbol{R_{max}}$**:
  - For any (s,a), **use the mean of the existing observations as estimate of R(s,a)** (that's also the expected value and the ML of R(s,a) given the observations).
    - **If the number of observations if too small yet ($< N_{n \cdot k, \delta, \epsilon}$ as defined above), then define the missing observations to be $\boldsymbol{R_{max}}$.**
  - Note: non-optimality may be caused by inaccurate transitions model in addition to inaccurate rewards model, hence both have to be referred to as part of errors analysis.

[IX] Generalization

- Since practical problems have too many states for either modeling or exploring, practical algorithms must **generalize their knowledge to "similar" states**.
- Solving Q explicitly:
  - Using a set of features of states f(s), we limit the Q-function to be some (possibly linear) function of the features and of some to-be-learned parameters w: $\boldsymbol{Q(s,a) = F\big(w, f(s)\big) = \sum_i w_i f_i(s)}$.
  - The Q-learning now updates Q(s,a) only through $\{w_i\}$ (and not directly $Q(s,a) \coloneqq \cdots$ as before):

$$\Delta w_i := \alpha\left(\left(R + \gamma \max_{a'} Q(s', a')\right) - Q(s, a)\right)\frac{dQ(s,a)}{dw_i}$$

i.e. w_i is changed such that Q(s,a) is moved towards its Bellman-Equation-desired-value, similarly to **Gradient Descent**.

- This turns out to do quite well in robotics and in certain games' bots.

- Baird's counterexample: explicit Q-learning using GD wrt features **may cause wrong and diverging learning**, in particular if the initial Q's are very dominant compared to the states rewards, or if the states share the same weights (i.e. there're features which are similar over various states).
- **To avoid divergence** of weights (and respectively divergence of values), one can use an **averaging generalization** function – one that assigns to any unobserved state a value which is some convex combination of the observed values $v(s_{new}) = \sum_{base\ states\ /\ observed\ states} w(s, s_{new})v(s)$.
  - A possible supervised learner which never extrapolates out of the range of the y-values of the data is **K-Nearest-Neighbors** ($w(s, s_{new})$ =1/K if s is a nearest neighbor else 0).
  - Another one is distance-weighted function ("**soft-KNN**") – $w \sim 1/d(s, s_{new})$.
  - Note: averaging generalization **cannot learn a trend and extrapolate** to states which are more radical than the observed ones, even though such states may be of much interest.

- Solving Q through a new MDP:
  - Features-based generalization of states of an MDP yields a new MDP:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s')\right)$$
$$= \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') \sum_{s_b \in B} w(s', s_b) \cdot V(s_b)\right)$$
$$= \max_a \left(R(s,a) + \gamma \sum_{s_b \in B} \left(\sum_{s'} T(s,a,s') w(s', s_b)\right) V(s_b)\right)$$
$$\max_a \left(R(s,a) + \gamma \sum_{s_b \in B} T'(s,a,s_b) V(s_b)\right)$$

MDP!

  - In particular, this solves the too-large-world exploration issue, allows us to approximate values of states as weighted average of some basis states, and brings us back to a smaller MDP problem over the observed states, which is solvable by standard Q-learning or value-iteration.

- While this section mainly dealt with value function approximation, policies and transition-models can be generalized as well.
- Further buzz-algorithms: least-squares policy-iteration (LSPI), Q-fitted iteration, GTD & GTD2.

[X] Partially-Observable MPDs

- **POMDP** (Partially-Observable MDP): MDP in which the state s (and possibly the reward r) is unknown to the agent, who only knows some observation z, whose relation to s is defined by the observation function $\boldsymbol{O(s,z) \coloneqq P(observe\ z|s)}$.
- State estimation: **belief state** = $b(s) \coloneqq P(s_t = s|z_1 \dots z_t)$.
  - Instead of current state, we hold at any point of time the vector of probabilities of states, which can be updated directly by P(we've been in s', and moved to s, and observed z):

$$\boldsymbol{b_{t+1}(s) = SE(b_t, a_t, z_{t+1}) \coloneqq \sum_{s'} b_t(s')T(s', a_t, s)O(s, z_{t+1})}.$$

- POMDPs are **undecidable** (unclear in the lectures but seems that the optimal solution can only be approximated or something).
- Value-iteration: $v_t(b) = \max_a\left(\sum_s b(s)R(s,a) + \gamma \sum_z P(z|b,a)v_{t-1}\left(SE(b,a,z)\right)\right)$.
  - It looks like the value-iteration needs to iterate over infinity belief-states b, but since v_{t+1} can be represented as max over finite (though exponential in t) number of linear functions of b, then it's essentially just piecewise-linear function in |{S}|-dimensional space, thus it is unnecessary to compute the value v_t for any specific b.
  - Of course, value-iteration only helps us after we know the transitions model T.
- **Model-based reinforcement-learning** of POMDP (i.e. learn the transitions model and use it):

  - As implied by the table, similarly to HMM, POMDP can be solved using **Expectation-Maximization**: starting from some initial transitions model, we iteratively compute the sequential belief-states $b_1 \dots b_t$ from the model, and then refine the model according to $b_t, z_t, O()$.
- **Model-free RL** of POMDP (i.e. evaluate pairs (state,action) without explicit transitions model):
  - Since we don't know s but only z, then model-free learning in POMDP actually evaluates Q(z,a). Such a memoryless policy in partially-observable world is problematic since we may keep doing a bad action in a certain state just because it looks the same (i.e. same z) as other states. To avoid that, random actions ($\{P(a)\}_a$) (as mixed strategies in Games Theory) are used.
  - **Bayesian RL**: one way to apply random action is to model the uncertainty as "we're within one of several possible worlds, and we need to figure out which one is that while gathering rewards".
    - That can be modeled as belief states in duplications of the world.
    - In this approach we assign initial probability to each possible world, thus every action has various possible outcomes according to which world we're in, and choosing the best action (i.e. highest expected value according to VI in the various worlds) implicitly applies both exploration and exploitation.

    - Since all the uncertainty is theoretically modeled in advance, then the RL looks more like **planning**. One algorithm that solves such planning problem is **Bayesian Exploration-Exploitation Tradeoff in LEarning** (**BEETLE**).
    - In complex worlds, such solutions still tend to be impractical compared to simple **Q-learning**, however they clarify the point that **utopic RL would just be planning**.
- **Predictive State Representation** (**PSR**):
    - If the states aren't directly measurable, then they're essentially not real but part of some fictional model. Instead, **a state will be defined by the distribution of its future measurements**.
        - E.g. an illness can be defined by the future risks it applies rather than by some internal, unmeasurable state of the body.
    - PSR Theorem: **PSR representation ≡ belief-state representation**: any n-state POMDP (i.e. belief-state of size n) can be represented by a PSR with up to n future-measurements ("tests", e.g. "if I go right will I measure X?") of up to n steps long each.
    - PSR representations turn to be **mainly useful in continuous-world problems**, in which there are א-infinity states, but tests-oriented representation apparently helps the learning.
    - Some more modern predictive methods combines PSR with **Least-Squares** rather than Probabilistic learning, mainly when very much data is available.

[XI] Scale-up

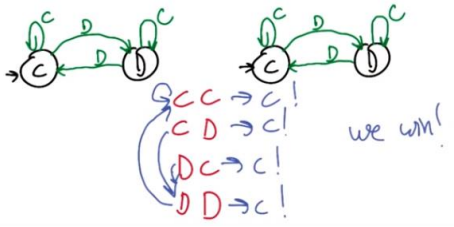
- **Temporal Abstraction**: add abstract actions which combine sequences of the basic actions to apply high-level activity.
    - **Option** = high-level action = $(I;\pi;\beta)$ = (valid states for beginning the option $I \subseteq S$; (possibly nondeterministic) policy $\pi: S \times A \to [0,1]$; distribution of in which state the option will terminate $F: S \to [0,1]$).
    - Options allow abstraction of actions which causes jumps in time (according to the option's length). This mechanism is called **Semi Markov Decision Process** (**SMDP**).
    - In addition to simplifying the world to high-level actions, the definition of options may exploit our **domain knowledge** to focus on the interesting actions that lead to the interesting states, rather than explore lots of irrelevant states (e.g. going to arbitrary empty corners of rooms). It also allows to focus on the interesting points of times – where decisions are taken – rather than deciding from scratch at every unit of time.
        - As always, using
    - Bellman equation for SMDP: $v_{t+1}(s) = \max_o (R(s,o) + \sum_{s'} F(s,o,s') v_t(s'))$.
        - R(s,o) is actually the expected reward during the option $\sum \gamma^{i-1} R(s_i, a_i)$.
        - F(s,o,s') is the probability of ending the option in s' given we began in s.
- **Goal Abstraction** – **Modular RL** – **Voting**: define several goals (e.g. in Pacman: eat dots & avoid ghosts), define states rewards wrt each goal, solve Q-value wrt each goal, aggregate all Q-values ("voting") and choose option (abstract action) accordingly.
    - Methods of voting over goals:
        - **Greatest-mass Q-learning**: $Q(s,a) \coloneqq \sum Q_i(s,a)$
        - Top Q-learning – "optimistic": $Q(s,a) \coloneqq max Q_i(s,a)$

        - Negotiated w-learning – "loss-hater": Q-value is weighted sum of the Q-values, with larger weights for goals with larger negative values.
    - Problem with voting over goals: **Arrow's impossibility theorem**.
        - I'm not sure it's relevant at all, since arrows theorem assumes ranking of goals (1,2,3…) rather than scoring, so sum over Q-values, for example, sounds fine.
        - The actual claim is more about being **scale-sensitive** – the rewards of the various goals must be scaled carefully so that the goals are correctly comparable.
- **Monte-Carlo Tree Search**:
    - Finding the best policy can be represented as tree search, where the leaves (either terminal states or just the practical horizon we're looking forward) are the aggregate rewards over the tree path.
    - Solving the Bellman equation for Q-value using VI can be seen as expanding the tree:
        - Select a leaf state in the tree, expand it by simulating several possible actions, and propagate (back-up) the achieved value back through the tree.
        - **Sampling**: to expand a state s accurately, $V(s) = \max_a Q(s,a)$ must be calculated over all possible actions. However, when numerous actions are possible, one may simulate just a sample of actions to get an estimation of V(s).
        - **A*-like search**: the current estimates of the Q-values along the tree can be used as heuristic for selection new states to expand – such that the horizon of the tree will be expanded in the paths which yield the best values.
            - Note: such greedy expanding of the tree may be far from optimal, since the heuristic of $\hat{Q}$ does not satisfy the optimism property required for A*.
            - This may still work if all rewards are negative or by using kind of R_{max} assumptions for non-expanded states. Otherwise, greedy expansion of the tree may be bad idea.
        - **Rolling out** the expansion: simulating the outcome of an action several steps forward is problematic since at each step it involves many new possible actions. Simplest way to expand a (state,action) pair is to continue the next steps randomly, though it may significantly underestimate the value of the (state,action). Another way is to expand as long as some constraints hold (e.g. for Pacman: simulate forward 10 steps as long as not eaten by a ghost). Possibly reasonable way is to use options rather than actions also in the rollout.
            - All these ideas sound quite foolish.
    - Ideally, the search along the tree (up to some horizon) would be simulated (assuming some simulation tool is available) from scratch after each real step.
    - Essentially, **MCTS** applies **local approximation of the Q-value of an MDP**, which allows us to **search for a local policy and take a step without solving the whole (typically huge) MDP**.

[XII,XIII,XIV] Game Theory

- Most of the section is contained in any basic Game Theory course.
- *Game Theory is the mathematics of conflict* (Charles Isabel).
- Game = generalization of MDP to more than one player.
- Policies of players are called **strategies**.
- **Tree representation**: straight forward generalization from MDPs.
- **Matrix representation**: For d players, any set of strategies $S_1 \dots S_d$ determines a (possibly stochastic) outcome. Hence all the information of the game can be encoded into a d-dimensional matrix above the space of outcomes, where each dimension's size is the number of possible strategies of the corresponding player.
- Properties of a game:
  - Number of players (≥ 2)
  - Zero-sum vs. separate rewards
  - Deterministic vs. non-deterministic
  - Perfect information vs. hidden information
- Von-Neuman Theorem, minmax, Nash Equilibrium, pure & mixed strategies, prisoner's dilemma.
- **Repeating prisoner's dilemma**:
  - If the number of rounds is known, then by backwards induction the only rational strategy is to constantly defect.
  - **In repeating prisoner's dilemma with unknown number of rounds, however, defecting is not a dominant strategy**.
    - E.g. after each round the game continues with probability γ (i.e. $1/(1-\gamma)$ expected rounds).
      - Note: the game can be represented as a graph, but not as a finite tree (cycle-less) anymore.
  - **Tit-for-Tat** strategy – copy rival's behavior: $a_1 \coloneqq cooperate$, $a_{n+1} \coloneqq b_n$. Against TfT:
    - Constant cooperation is better than constant defection, as long as $\gamma > \gamma_0$.
    - **TfT vs. TfT result in constant cooperation**, and are optimal against each other (by solving the corresponding MDP), which also makes it a **Nash equilibrium**.
      - Note: cooperation vs. cooperation would result in the same behavior, yet it's not a Nash equilibrium since each player can benefit from changing.
    - **Grim strategy** – cooperate as long as the rival has never defected, otherwise defect forever – can also form a Nash equilibrium.
- **Subgame-perfect Nash equilibrium**:
  - Nash equilibrium may be based on "threats" regarding what one player would do if the other changed his strategy (as in TfT – if the rival changes from TfT and defects, then I'll defect as well). This may be problematic if these are **implausible threats** (as in Grim – where a single defection is supposed to lead to eternal counter-defection).
  - In both equilibriums above (TfT vs. TfT and Grim vs. Grim), the threats deal with states in the game that never occur unless a player changes his strategy. Thus, one may disbelieve those threats (i.e. disbelieve the strategy claimed by his rival), which ruins the Nash equilibrium.

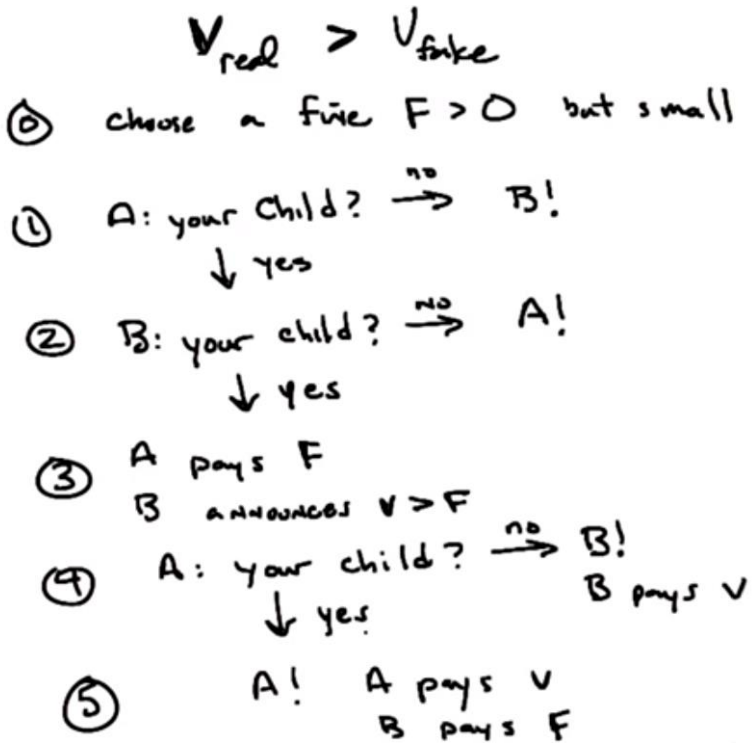
- A stronger equilibrium verifies that all the threats are plausible – i.e. even if the rival does some unplanned move, and we get to an unexpected state in the game – the strategies still form an equilibrium in this subgame.
- **Subgame perfect Nash equilibrium** = Nash equilibrium in every subgame of the game – including subgames that are not expected to ever occur.
- Both Grim & TfT Nash equilibriums are not subgame perfect, i.e. show implausible threats.
- **Pavlov strategy**: cooperate on first round, and change behavior whenever the rival defects ($a_{n+1} \coloneqq a_n$ if b_n =cooperate else ! a_n). It can be seen as "keep your behavior as long as it's winning".
- **Pavlov vs. Pavlov is subgame perfect Nash equilibrium!**
  - Turns out that the Pavlovian mechanism "punishes" a single exception from the opponent's strategy with two defections (rather than one as in TfT), which causes the opponent to get back to cooperation, and makes the threat plausible.



  - Pavlov in human behavior.
- **Computational Folk Theorem** (Michael Littman): for any 2-player game (in matrix representation), if there exist some cooperative (i.e. mutually beneficial) equilibrium then Pavlov-like equilibrium can be found, otherwise the game is zero-sum-like (or close to it) and corresponding equilibrium can be found – in both cases subgame perfect equilibrium and in polynomial time.

- **Stochastic games** – explicit **generalization of MDPs to multi-agent RL**:
  - States S, actions set Ai per player, transitions $T(s, a_1 \dots a_d, s')$, rewards Ri per player, discount γ.
    - The model is a mathematical generalization of MDPs, yet it was published by Shapely before MDPs were published by Bellman.
  - **Minmax Q**: in **zero-sum** stochastic games, Q-learning can be applied based on a generalization of Bellman equation with minmax instead of max.
    - VI works, minmax-Q converges, Q* solution is unique, policies of players can be computed independently and are derived directly from Q*.
  - **Nash Q**: in **general** stochastic games, Q-learning can be applied based on a generalization of Bellman equation with Nash equilibrium instead of max.
    - VI doesn't work, minmax-Q doesn't converge, Q* solution isn't unique, policies of players depend on each other, and the Nash equilibrium is hard to compute (non polynomial by current algorithms).
    - Other ideas are developed nowadays for this problem.
- **Chicken game**:
  - Nash equilibriums: (1) Dare-vs.-Chicken, and (2) a symmetric mixed strategy (whose P depends on the exact rewards of the game).

  - We would like to have some equilibrium which is both "fair" (symmetric) and "good" (0 probability for low-reward actions as Dare-vs.-Dare).
- **Correlated equilibrium**:
  - A **shared source of randomization** defines a set of pairs of strategies, randomizes one pair out of the set, and tells each player what strategy he should adopt.
  - Examples:
    - In chicken game – uniform distribution over (C,D), (D,C), (C,C) – even though (C,C) is not a Nash equilibrium.
    - Traffic light.
  - The external mechanism must be trustable (fair randomization) but doesn't have to be authorized (not forcing strategies upon players).
  - The mechanism forms a **correlated equilibrium** iff no player can do better than following the instructions.
  - A correlated equilibrium can be found in polynomial time.
  - Every convex combination of Nash equilibriums is also a correlated equilibrium.
  - The information sharing in correlated equilibrium can also help to avoid bad outcomes that arise in equilibriums of independent mixed strategies, and encourage good outcomes – as in the chicken game, where (C,C) is promoted and (D,D) is completely avoided.
- **Coco strategies** (**Cooperative-competitive**): strategies of 2 players that **maximize the joint reward** and divide it between the 2 players.
  - The rewards of the game are decomposed into joint rewards $R_{coop} \coloneqq (R_1 + R_2)/2$ and relative rewards $R_{comp} \coloneqq (R_1 - R_2)/2$.
  - The joint rewards define a cooperative game, where $maxmax R_{coop}$ yields the maximize total reward for the 2 players.
  - The relative rewards define a zero-sum game which is solvable by minmax (through LP?).
  - **Coco strategies** $\coloneqq \mathbf{argmax}_{\boldsymbol{S_1}} \mathbf{argmax}_{\boldsymbol{S_2}} \boldsymbol{R_{coop}}(\boldsymbol{S_1}, \boldsymbol{S_2})$, with rewards sharing of **Coco value**$\coloneqq \boldsymbol{\left(R_{coop} + R_{comp}\right)/2}$ to player 1 and $\left(R_{coop} - R_{comp}\right)/2$ to player 2.
    - Reward sharing requires **side payments**.
  - Coco is **NOT necessarily an equilibrium** – **just a "fair" way to maximize utility and dividing the reward** – hence both strategies and side-payments must be binding by trust or some external force.
  - Coco strategies and values can be found in polynomial time.
  - Generalization to $d > 2$ players is currently unknown.
- **Mechanism design**: design the rules of a game to encourage certain players behaviors (economics, studies incentives, etc.).
  - Bonus: **King Solomon's sentence**: the fake mother wouldn't really agree to cutting the baby (not reasonable behavior + indication to not being a mother). However, **the following mechanism would reveal the real mother** (assuming the real mother assigns larger value to having the baby):



- Note: I think this mechanism is actually just an **auction** with participation cost.

[XV] Coordinating, Communicating and Coaching

- **Decentralized POMDP**: generalization of POMDP to multiple agents who act simultaneously every time-step.
  - Communication between agents can be modeled as part of the observation function.
  - The problem is NEXP-complete (non-deterministic exponential complete, i.e. very hard).
- **Inverse RL**: Bayesianly deduce the rewards of states from the behavior of an agent.
  - Kind of **Reverse-Engineering** of RL which is useful to understand the values that people/animals/organizations assign to states.
  - Can be solved by **Maximum-Likelihood-IRL**.
- **Policy shaping**: use **interactive feedback from a "coach"** (e.g. a human who watches a bot playing a game and reports good and bad moves) to update policy.
  - The use of explicit feedback regarding the policy directly (rather than through states rewards) makes the learning simpler that classic RL.
  - If the feedback has limited reliability, then the limited reliability can be modeled by the probability of wrong feedback, and the feedback can be incorporated into the agent's knowledge through the Bayesian expression P(action was good | good feedback).
  - The interactive feedback needs to be combined by the agent with other environmental feedbacks (e.g. the standard exploration and states rewards).
- **Multiple sources** of feedback:
  - **ML**: choose the action whose likelihood to be the best (among possibilities) is maximal:

$$a_{chosen} \coloneqq \underset{a \in possible\ actions}{\operatorname{argmax}} \Pi_{s \in sources} P(feedback_s | a\ is\ best)$$

[XVI] Outroduction

- Code libraries and DBs:
  - BURLAP – well suited for RL, including simulative environment.
  - UCI & Weka – intended for supervised learning.
- Popular applications of RL:
  - Games with turns (e.g. backgammon).
  - Elevator control.
  - Helicopter control.