Artificial Intelligence

Reinforcement Learning

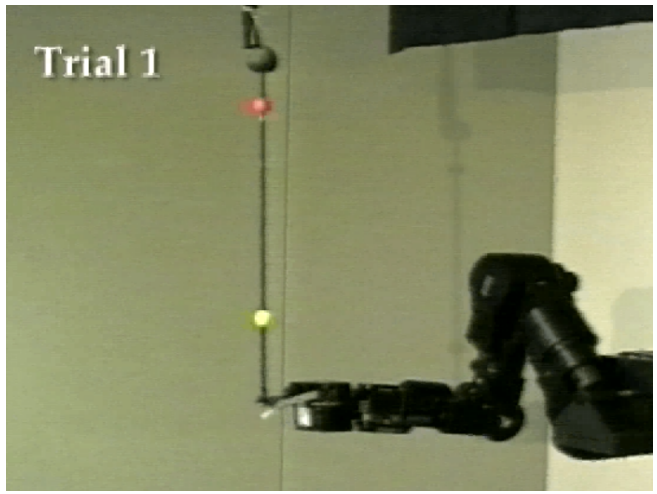
Marc Toussaint
University of Stuttgart
Winter 2018/19

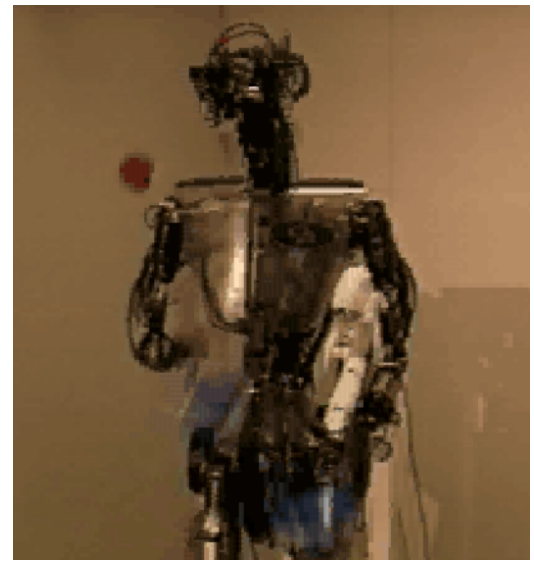
Motivation:

Reinforcement Learning means to learn to perform well in an previously unknown environment. So it naturally combines the problems of learning about the environment and decision making to receive rewards. In that sense, I think that the RL framework is a core of AI. (But one should also not overstate this: standard RL solvers typically address limited classes of MDPs—and therefore do not solve many other aspects AI.)

The notion of *state* is central in the framework that underlies Reinforcement Learning. One assumes that there is a 'world state' and decisions of the agent change the state. This process is formalized as Markov Decision Process (MDP), stating that a new state may only depend the previous state and decision. This formalization leads to a rich family of algorithms underlying both, planning in known environments as well as learning to act in unknown ones.

This lecture first introduces MDPs and standard Reinforcement Learning methods. We then briefly focus on the exploration problem—very much related to the exploration-exploitation problem represented by bandits. We end with a brief illustration of policy search, imitation and inverse RL without going into the full details of these. Especially inverse RL is really worth knowing about: the problem is to learn the underlying reward function from example demonstrations. That is, the agent tries to "understand" (human) demonstrations by trying to find a reward function consistent with them.





(around 2000, by Schaal, Atkeson, Vijayakumar)

(2007, Andrew Ng et al.)

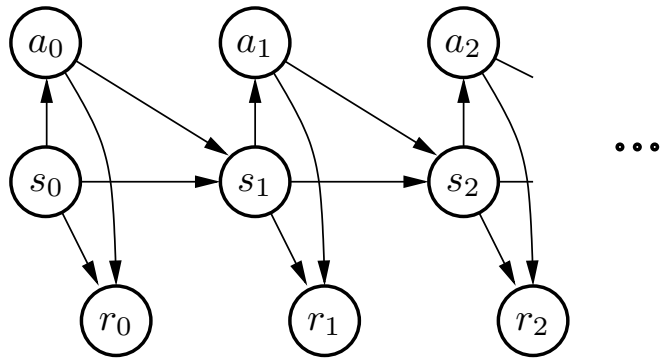
Long history of RL in AI

Idea of programming a computer to learn by trial and error (Turing, 1954)
SNARCs (Stochastic Neural-Analog Reinforcement Calculators) (Minsky, 54)
Checkers playing program (Samuel, 59)
Lots of RL in the 60s (e.g., Waltz & Fu 65; Mendel 66; Fu 70)
MENACE (Matchbox Educable Naughts and Crosses Engine (Mitchie, 63)
RL based Tic Tac Toe learner (GLEE) (Mitchie 68)
Classifier Systems (Holland, 75)
Adaptive Critics (Barto & Sutton, 81)
Temporal Differences (Sutton, 88)

from Satinder Singh's *Introduction to RL*, videolectures.com

- Long history in Psychology

Recall: Markov Decision Process



$$P(s_{0:T+1}, a_{0:T}, r_{0:T}; \pi) = P(s_0) \prod_{t=0}^{T} P(a_t|s_t;\pi)\, P(r_t|s_t,a_t)\, P(s_{t+1}|s_t,a_t)$$

- world's initial state distribution $P(s_0)$
- world's transition probabilities $P(s_{t+1} \mid s_t, a_t)$
- world's reward probabilities $P(r_t \mid s_t, a_t)$
- agent's *policy* $\pi(a_t \mid s_t) = P(a_0|s_0;\pi)$ (or deterministic $a_t = \pi(s_t)$)

- **Stationary MDP:**
  - We assume $P(s' \mid s, a)$ and $P(r|s,a)$ independent of time
  - We also define $R(s,a) := \mathsf{E}\{r|s,a\} = \int r\ P(r|s,a)\ dr$

Recall

- Bellman equations

$$V^*(s) = \max_a Q^*(s,a) = \max_a \Big[R(s,a) + \gamma \textstyle\sum_{s'} P(s' \,|\, s,a)\; V^*(s') \Big]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \,|\, s,a)\; \max_{a'} Q^*(s',a')$$

- Value-/Q-Iteration

$$\forall s:\; V_{k+1}(s) = \max_a \Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a)\; V_k(s') \Big]$$

$$\forall_{s,a}:\; Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)\; \max_{a'} Q_k(s',a')$$

Towards Learning

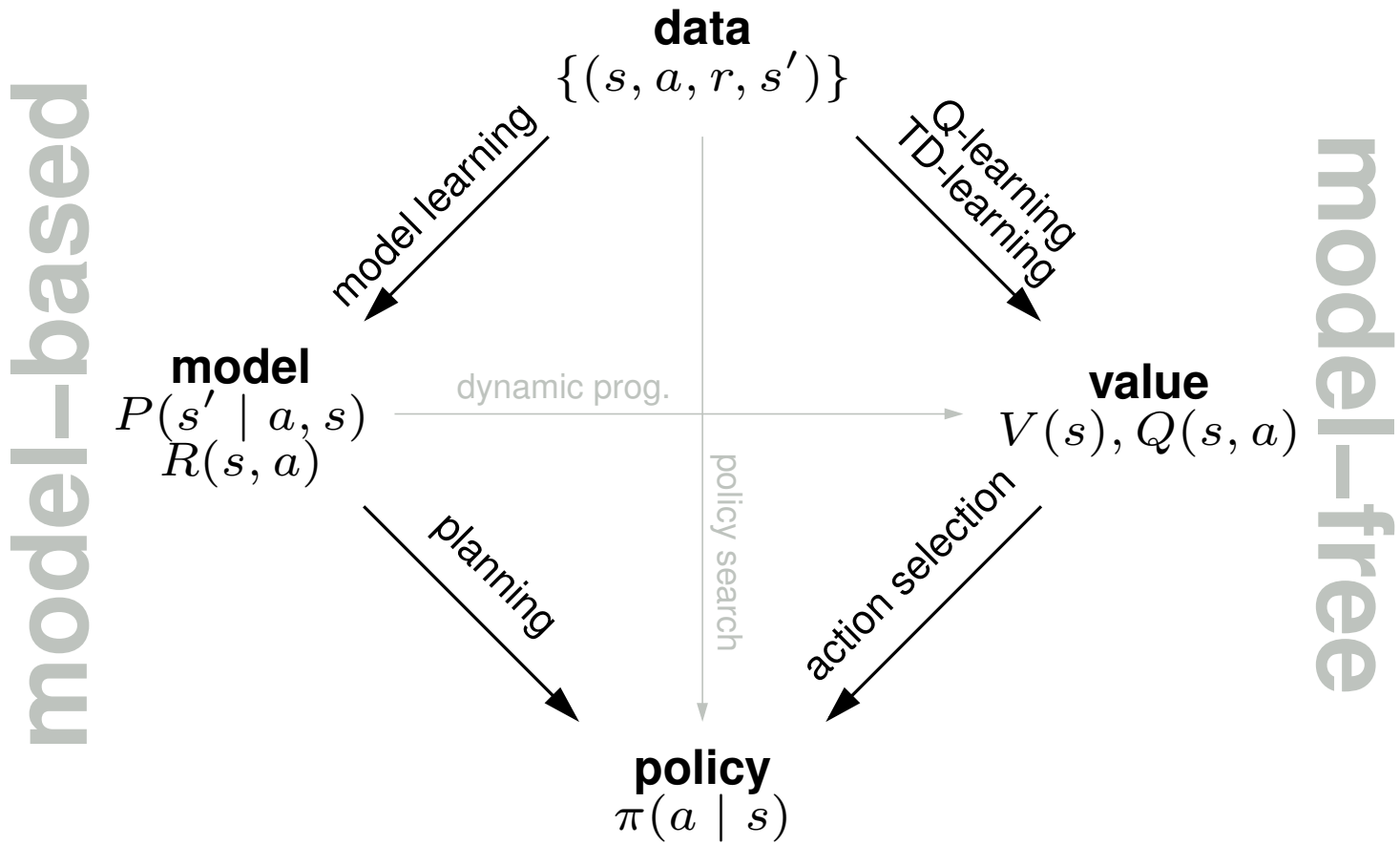
- From Sutton & Barto's *Reinforcement Learning* book:
  The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. DP provides an essential foundation for the understanding of the methods presented in the rest of this book. In fact, all of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

- So far, we introduced basic notions of an MDP and value functions and methods to compute optimal policies **assuming that we know the world** (know $P(s'|s,a)$ and $R(s,a)$)

  Value Iteration and Q-Iteration are instances of Dynamic Programming

- *Reinforcement Learning?*

Learning in MDPs


model–based
model–free
data
$\{(s, a, r, s')\}$
model learning
Q-learning
TD-learning
model
$P(s' \mid a, s)$
$R(s, a)$
dynamic prog.
value
$V(s), Q(s, a)$
planning
policy search
action selection
policy
$\pi(a \mid s)$

Learning in MDPs

- While interacting with the world, the agent collects data of the form
$$D = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{T}$$
(state, action, immediate reward, next state)

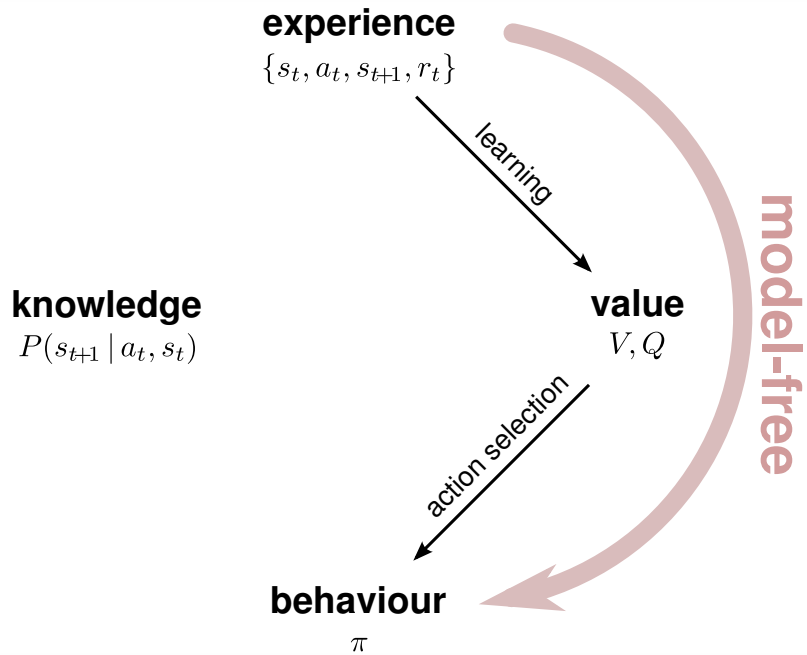
What could we learn from that?

- **Model-based RL:**
learn to predict next state: estimate $P(s'|s, a)$
learn to predict immediate reward: estimate $P(r|s, a)$

- **Model-free RL:**
learn to predict *value*: estimate $V(s)$ or $Q(s, a)$

- **Policy search:**
e.g., estimate the "policy gradient", or directly use black box (e.g. evolutionary) search

Let's introduce basic *model-free* methods first.



Q-learning: Temporal-Difference (TD) learning of Q^*

- Recall the Bellman optimality equation for the Q-function:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

- **Q-learning** (Watkins, 1988) Given a new experience (s,a,r,s')

$$\begin{aligned} Q_{\text{new}}(s,a) &= (1-\alpha)\, Q_{\text{old}}(s,a) + \alpha\, [r + \gamma \max_{a'} Q_{\text{old}}(s',a')] \\ &= Q_{\text{old}}(s,a) + \alpha\, \underbrace{[r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)]}_{\text{TD error}} \end{aligned}$$

- **Reinforcement:**
  - more reward than expected $(r > Q_{\text{old}}(s,a) - \gamma \max_{a'} Q_{\text{old}}(s',a'))$
    $\rightarrow$ increase $Q(s,a)$
  - less reward than expected $(r < Q_{\text{old}}(s,a) - \gamma \max_{a'} Q_{\text{old}}(s',a'))$
    $\rightarrow$ decrease $Q(s,a)$

Q-learning pseudo code

- Q-learning is called **off-policy:** We estimate Q^* while executing π

- **Q-learning:**

1: Initialize $Q(s, a) = 0$
2: **repeat** *// for each episode*
3: Initialize start state s
4: **repeat** *// for each step of episode*
5: Choose action $a \approx_\epsilon \operatorname{argmax}_a Q(s, a)$
6: Take action a, observe r, s'
7: $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
8: $s \leftarrow s'$
9: **until** end of episode
10: **until** happy

- **ϵ-greedy action selection:**

$$a \approx_\epsilon \operatorname*{argmax}_a Q(s, a) \quad \Longleftrightarrow \quad a = \begin{cases} \text{random} & \text{with prob. } \epsilon \\ \operatorname{argmax}_a Q(s, a) & \text{else} \end{cases}$$

Q-learning convergence with prob 1

- Q-learning is a stochastic approximation of Q-Iteration:

  Q-learning: $Q_{\text{new}}(s,a) = (1-\alpha)Q_{\text{old}}(s,a) + \alpha[r + \gamma \max_{a'} Q_{\text{old}}(s',a')]$
  Q-Iteration: $\forall_{s,a}: \; Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \; \max_{a'} Q_k(s',a')$

  We've shown convergence of Q-Iteration to Q^*

- Convergence of Q-learning:
  Q-Iteration is a deterministic update: $Q_{k+1} = T(Q_k)$
  Q-learning is a stochastic version: $Q_{k+1} = (1-\alpha)Q_k + \alpha[T(Q_k) + \eta_k]$
  η_k is zero mean!

Q-learning impact

- Q-Learning was the first provably convergent direct adaptive optimal control algorithm

- Great impact on the field of Reinforcement Learning in 80/90ies
  - "Smaller representation than models"
  - "Automatically focuses attention to where it is needed," i.e., no sweeps through state space
  - Can be made more efficient with eligibility traces

Variants: TD(λ), Sarsa(λ), $Q(\lambda)$

- TD(λ):
  $\forall s:\ V(s) \leftarrow V(s) + \alpha\ e(s)\ [r_t + \gamma V_{\text{old}}(s_{t+1}) - V_{\text{old}}(s_t)]$
- Sarsa(λ)
  $\forall_{s,a}:\ Q(s,a) \leftarrow Q(s,a) + \alpha\ e(s,a)\ [r + \gamma Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)]$
- $Q(\lambda)$
  $\forall_{s,a}:\ Q(s,a) \leftarrow Q(s,a) + \alpha\ e(s,a)\ [r + \gamma \max_{a'} Q_{\text{old}}(s',a') - Q_{\text{old}}(s,a)]$

- **On-policy vs. off-policy** learning:
  - Onpolicy: estimate Q^π while executing π (Sarsa, TD)
  - Offpolicy: estimate Q^* while executing π (Q-learning)

Eligibility traces

- Temporal Difference: based on single experience (s_0, r_0, s_1)

$$V_{\text{new}}(s_0) = V_{\text{old}}(s_0) + \alpha[r_0 + \gamma V_{\text{old}}(s_1) - V_{\text{old}}(s_0)]$$

- Longer experience sequence, e.g.: $(s_0, r_0, r_1, r_2, s_3)$

Eligibility traces

- Temporal Difference: based on single experience (s_0, r_0, s_1)

$$V_{\text{new}}(s_0) = V_{\text{old}}(s_0) + \alpha[r_0 + \gamma V_{\text{old}}(s_1) - V_{\text{old}}(s_0)]$$

- Longer experience sequence, e.g.: $(s_0, r_0, r_1, r_2, s_3)$
  Temporal credit assignment, think further backwards: receiving $r_{0:2}$ and ending up in s_3 also tells us something about $V(s_0)$

$$V_{\text{new}}(s_0) = V_{\text{old}}(s_0) + \alpha[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 V_{\text{old}}(s_3) - V_{\text{old}}(s_0)]$$

Eligibility traces

- Temporal Difference: based on single experience (s_0, r_0, s_1)

$$V_{\text{new}}(s_0) = V_{\text{old}}(s_0) + \alpha[r_0 + \gamma V_{\text{old}}(s_1) - V_{\text{old}}(s_0)]$$

- Longer experience sequence, e.g.: $(s_0, r_0, r_1, r_2, s_3)$
  Temporal credit assignment, think further backwards: receiving $r_{0:2}$ and ending up in s_3 also tells us something about $V(s_0)$

$$V_{\text{new}}(s_0) = V_{\text{old}}(s_0) + \alpha[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 V_{\text{old}}(s_3) - V_{\text{old}}(s_0)]$$

- **TD(λ)**: remember where you've been recently ("eligibility trace") and update those values as well:

$$\begin{aligned}
&e(s_t) \leftarrow e(s_t) + 1 \\
&\forall s: \; V_{\text{new}}(s) = V_{\text{old}}(s) + \alpha \, e(s) \, [r_t + \gamma V_{\text{old}}(s_{t+1}) - V_{\text{old}}(s_t)] \\
&\forall s: \; e(s) \leftarrow \gamma \lambda e(s)
\end{aligned}$$

- Core topic of Sutton & Barto book
  $\rightarrow$ great improvement of basic RL algorithms

Q(λ) pseude code

1: Initialize $Q(s,a) = 0$, $e(s,a) = 0$
2: **repeat** *// for each episode*
3: Initialize start state s
4: **repeat** *// for each step of episode*
5: Choose action $a \approx_\epsilon \operatorname{argmax}_a Q(s,a)$
6: Update eligibility: $e(s,a) \leftarrow 1$ or $e(s,a) \leftarrow e(s,a) + 1$
7: Take action a, observe r, s'
8: Compute TD-error $D = [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
9: $\forall_{\tilde{s},\tilde{a}}$ with $e(\tilde{s},\tilde{a}) > 0$: $Q(\tilde{s},\tilde{a}) \leftarrow Q(\tilde{s},\tilde{a}) + \alpha\, e(\tilde{s},\tilde{a})\, D$
10: Discount all eligibilities $\forall_{\tilde{s},\tilde{a}}: e(\tilde{s},\tilde{a}) \leftarrow \gamma\lambda e(\tilde{s},\tilde{a})$
11: $s \leftarrow s'$
12: **until** end of episode
13: **until** happy

- Analogously for TD(λ) and SARSA(λ)

Experience Replay

- In large state spaces, the Q-fucntion is represented using function approximation
  We cannot store a full table $e(s, a)$ and update $\forall_{\tilde{s},\tilde{a}}$
- Instead we store the full data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=o}^{t}$ up to now (time t), called the **replay buffer**
- Update the Q-function for a B subsample (of contant size) of D plus the most recent experience

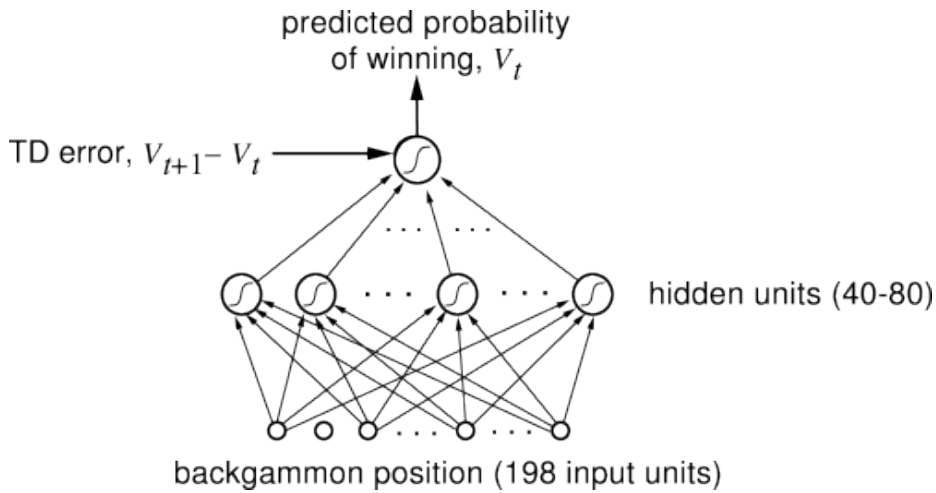
$$\forall (s, a, r, s') \in B : \; Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

(See paper “A Deeper Look at Experience Replay” (Zhang, Sutton))

TD-Gammon, by Gerald Tesauro**

(See section 11.1 in Sutton & Barto's book.)

- MLP to represent the value function $V(s)$



- Only reward given at end of game for win.
- **Self-play**: use the current policy to sample moves on *both* sides!
- random policies $\rightarrow$ games take up to thousands of steps. Skilled players $\sim 50 - 60$ steps.
- TD(λ) learning (gradient-based update of NN weights)

TD-Gammon notes**

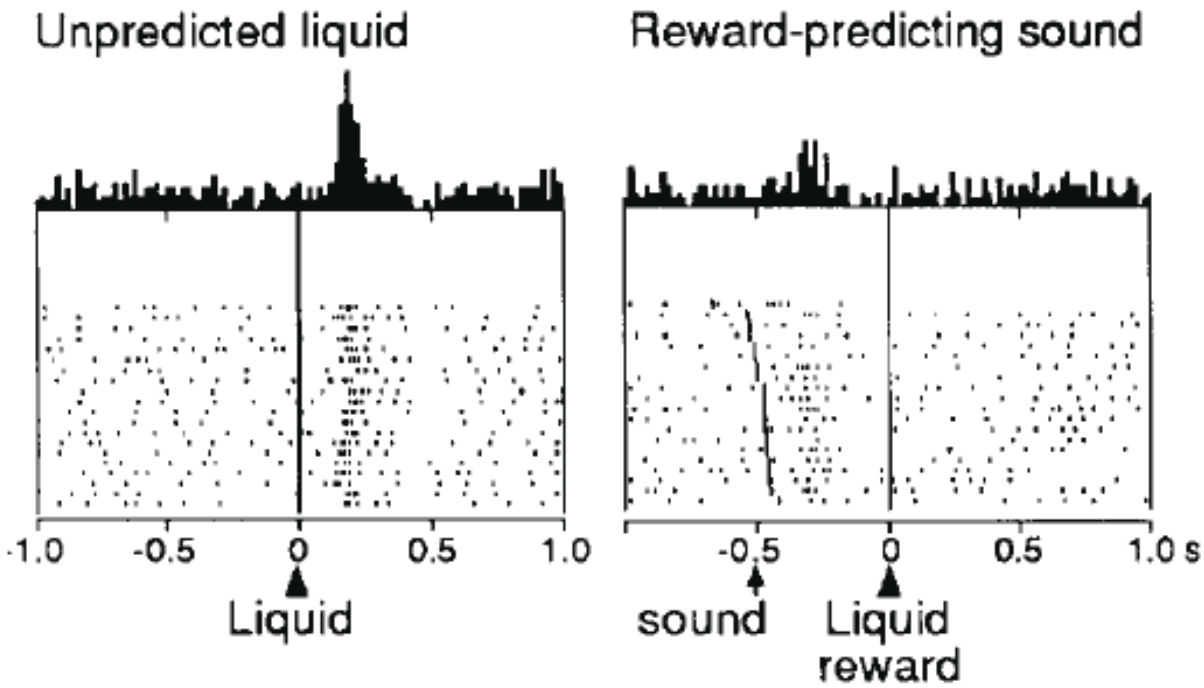
- Choose features as raw position inputs (number of pieces at each place)
  $\rightarrow$ as good as previous computer programs
- Using previous computer program's expert features
  $\rightarrow$ world-class player

- Kit Woolsey was world-class player back then:
  - TD-Gammon particularly good on vague positions
  - not so good on calculable/special positions
  - just the opposite to (old) chess programs

- See anotated matches: `http://www.bkgm.com/matches/woba.html`

- Good example for
  - value function approximation
  - game theory, self-play

Detour: Dopamine**



Montague, Dayan & Sejnowski: *A Framework for Mesencephalic Dopamine Systems based on Predictive Hebbian Learning.* Journal of Neuroscience, 16:1936-1947, 1996.

So what does that mean?

- We derived an algorithm from a general framework
- This algorithm involves a specific variable (reward residual)
- We find a neural correlate of exactly this variable

Great!

So what does that mean?

- We derived an algorithm from a general framework
- This algorithm involves a specific variable (reward residual)
- We find a neural correlate of exactly this variable

Great!

Devil's advocate:

- Does not proof that TD learning is going on
  Only that an expected reward is compared with a experienced reward

- Does not discriminate between model-based and model-free
  (Both can induce an expected reward)

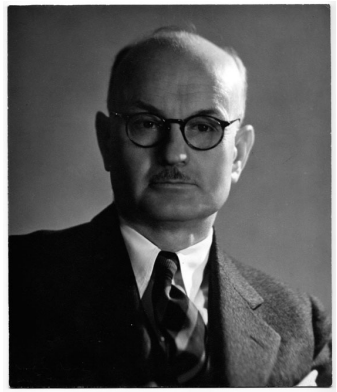
Limitations of the model-free view

- Given learnt values, behavior is a fixed SR (or state-action) mapping
- If the “goal” changes: need to re-learn values for every state in the world! all previous values are obsolete
- No general “knowledge”, only values
- No anticipation of general outcomes (s'), only of value
- No “planning”

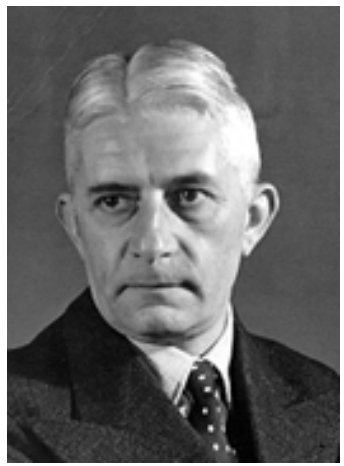
Wolfgang Köhler (1917)
Intelligenzprüfungen am Menschenaffen
The Mentality of Apes

model-free RL?
NO WAY!

Detour: Psychology**



Edward Tolman (1886 - 1959)



Wolfgang Köhler (1887–1967)

learn facts about the world that they could subsequently use in a flexible manner, rather than simply learning automatic responses

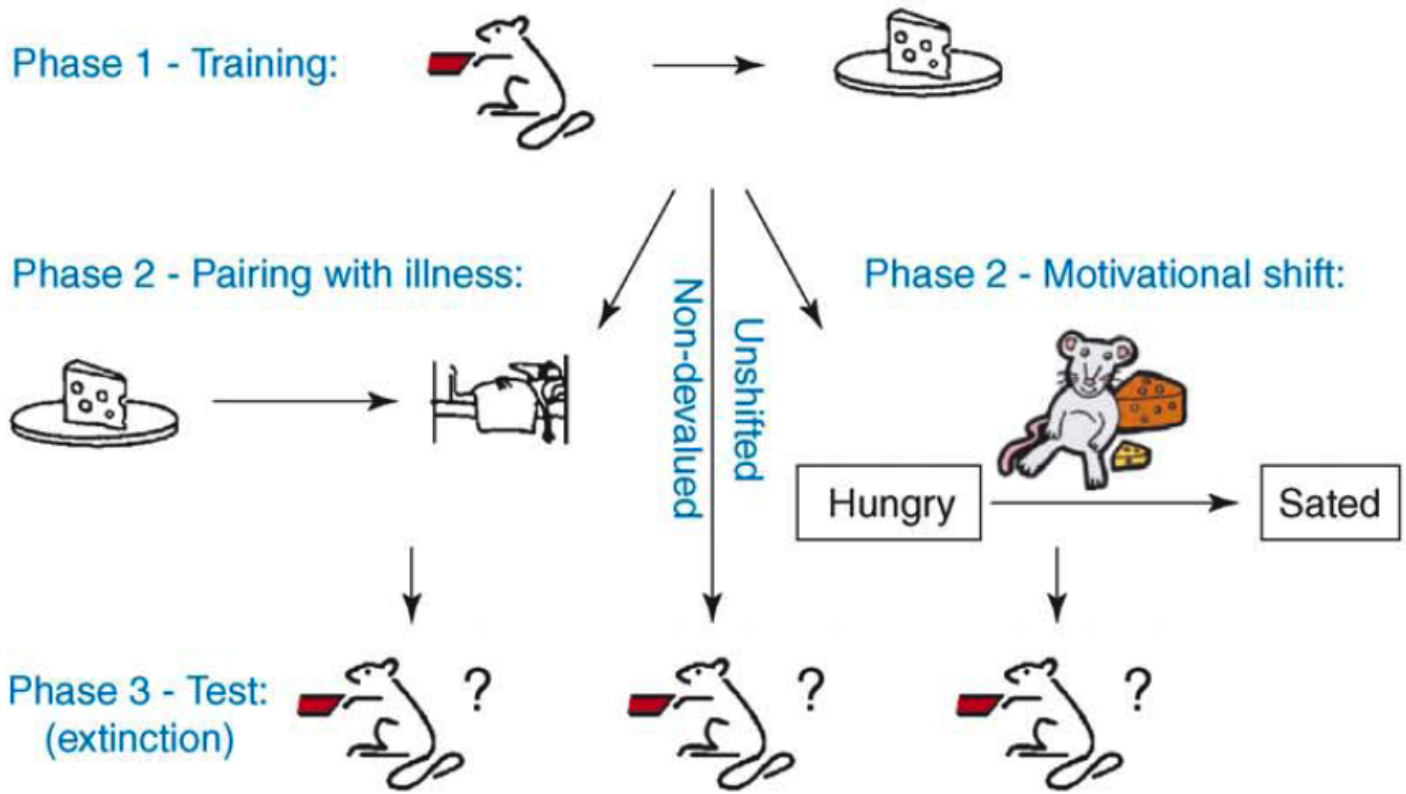
Clark Hull (1884 - 1952)

Principles of Behavior (1943)

learn stimulus-response mappings based on reinforcement

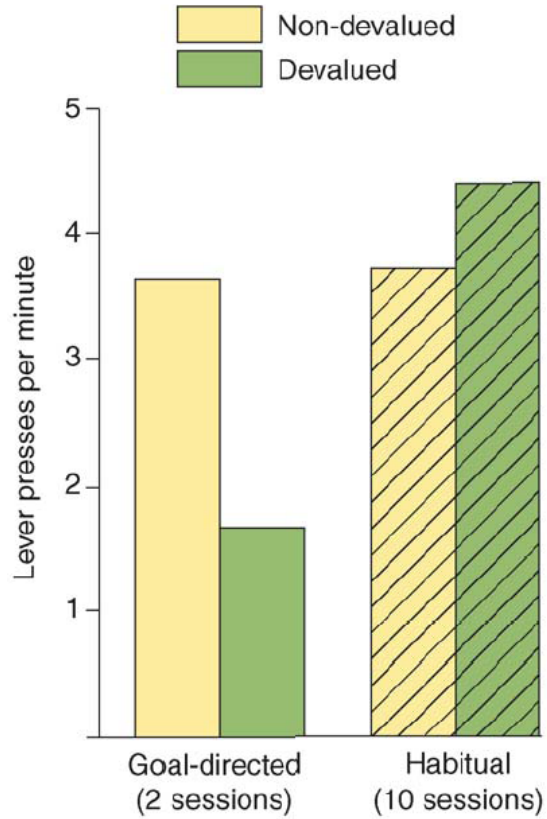
Goal-directed vs. habitual: Devaluation**

[skinner]



Niv, Joel & Dayan: *A normative perspective on motivation.* TICS, 10:375-381, 2006.

Goal-directed vs. habitual: Devaluation**

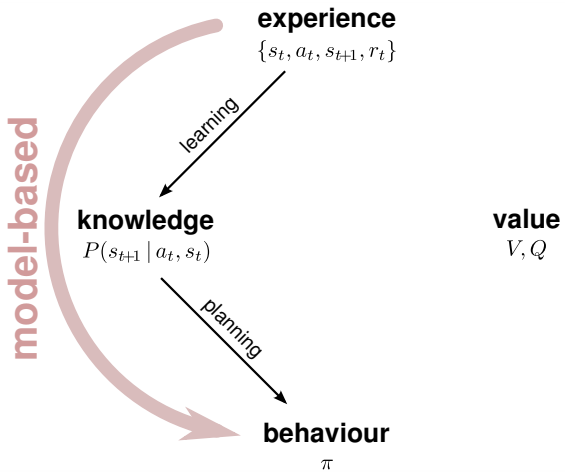


Niv, Joel & Dayan: *A normative perspective on motivation.* TICS, 10:375-381, 2006.

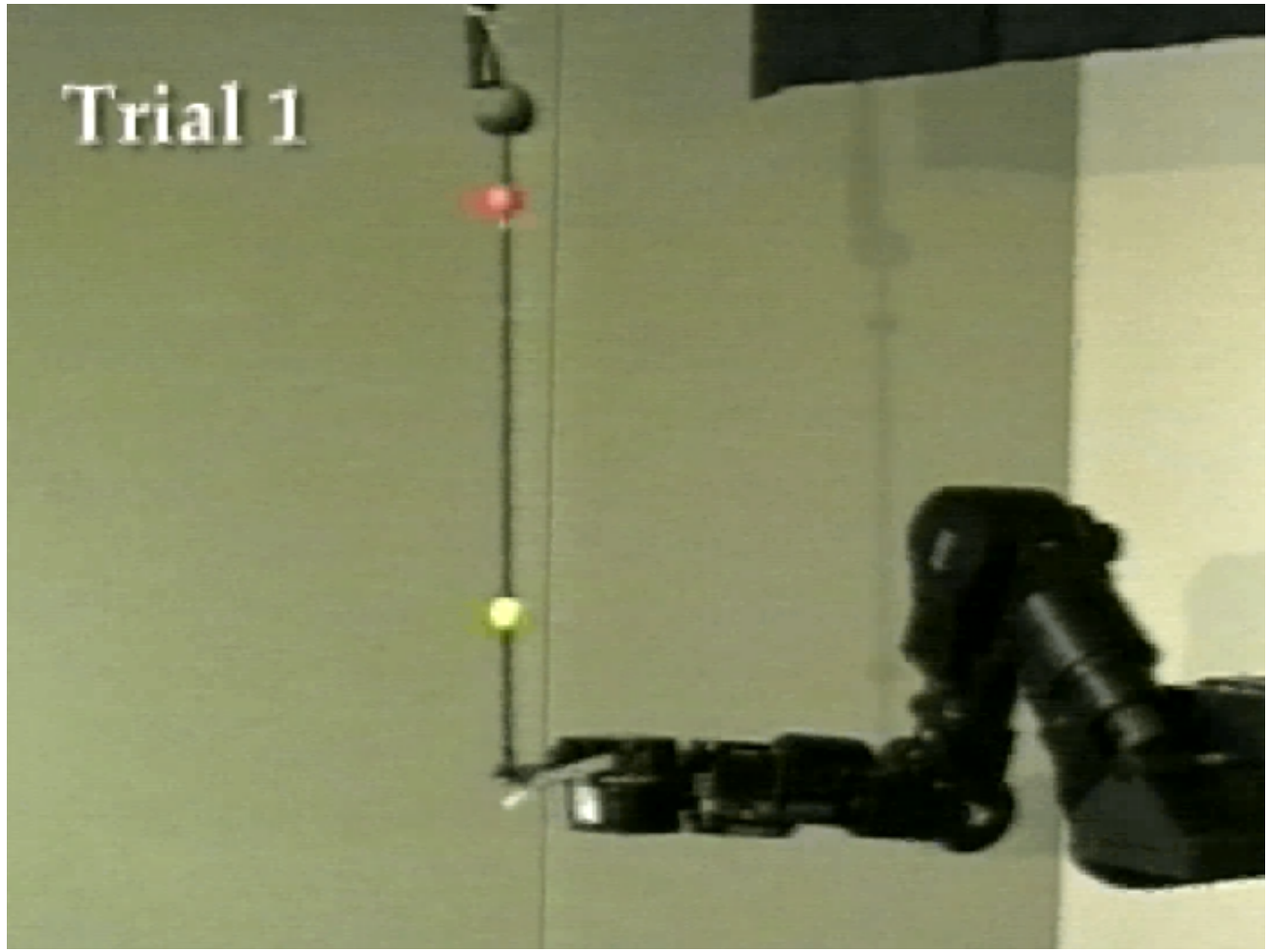
By definition, goal-directed behavior is performed to obtain a desired goal. Although all instrumental behavior is **instrumental** in achieving its contingent goals, it is not necessarily purposively **goal-directed**. Dickinson and Balleine [1,11] proposed that behavior is goal-directed if: (i) it is sensitive to the contingency between action and outcome, and (ii) the outcome is desired. Based on the second condition, motivational manipulations have been used to distinguish between two systems of action control: if an instrumental outcome is no longer a valued goal (for instance, food for a sated animal) and the behavior persists, it must not be goaldirected. Indeed, after moderate amounts of training, outcome revaluation brings about an appropriate change in instrumental actions (e.g. leverpressing) [43,44], but this is no longer the case for extensively trained responses ([30,31], but see [45]). That extensive training can render an instrumental action independent of the value of its consequent outcome has been regarded as the experimental parallel of the folk psychology maxim that wellperformed actions become **habitual** [9] (see Figure I).

Niv, Joel & Dayan: *A normative perspective on motivation.* TICS, 10:375-381, 2006.

Model-based RL



- **Model learning:** Given data $D = \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^T$ estimate $P(s'|s,a)$ and $R(s,a)$. For instance:
  - discrete state-action: $\hat{P}(s'|s,a) = \frac{\#(s',s,a)}{\#(s,a)}$
  - continuous state-action: $\hat{P}(s'|s,a) = \mathcal{N}(s' \mid \phi(s,a)^\top \beta, \Sigma)$ estimate parameters β (and perhaps Σ) as for regression (including non-linear features, regularization, cross-validation!)
- **Planning**, for instance:
  - discrete state-action: Value Iteration with the estimated model
  - continuous state-action: Least Squares Value Iteration Stochastic Optimal Control (Riccati, Differential Dynamic Prog.)



(around 2000, by Schaal, Atkeson, Vijayakumar)

- Use a simple regression method (locally weighted Linear Regression) to estimate

$$P(\dot{x}|u,x) \overset{\text{local}}{=} \mathcal{N}(\dot{x} \,|\, Ax + Bu, \sigma)$$

Exploration

ϵ-greedy exploration in Q-learning

1: Initialize $Q(s, a) = 0$
2: **repeat** *// for each episode*
3: Initialize start state s
4: **repeat** *// for each step of episode*
5: Choose action $a = \begin{cases} \text{random} & \text{with prob. } \epsilon \\ \text{argmax}_a \, Q(s, a) & \text{else} \end{cases}$
6: Take action a, observe r, s'
7: $Q_{\text{new}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha \left[r + \gamma \max_{a'} Q_{\text{old}}(s', a') - Q_{\text{old}}(s, a) \right]$
8: $s \leftarrow s'$
9: **until** end of episode
10: **until** happy

Optimistic initialization & UCB

- Initialize the Q function optimistically! E.g., if you know R_{max}, $Q(s,a) = 1/(1-\gamma)R_{max}$ (in practise, this is often too large..)
- UCB: If you can estimate a confidence bound $\sigma(s,a)$ for your Q-function (e.g., using bootstrap estimates when using function approximation), choose your action based on $Q(s,a) + \beta\sigma(s,a)$
- Generally, we need better ways to explore than ϵ-greedy!!

R-MAX

Brafman and Tennenholtz (2002)

- Model-based RL: We estimate $R(s,a)$ and $P(s'|s,a)$ on the fly
- Use an *optimistic* reward function: 1

$$R^{\text{R-MAX}}(s,a) = \begin{cases} R(s,a) & c(s,a) \geq m \text{ } (s,a \text{ known}) \\ R_{max} & c(s,a) < m \text{ } (s,a \text{ unknown}) \end{cases}$$

- Is PAC-MDP efficient
- Optimism in the face of uncertainty

KWIK-R-max**

(Li, Littman, Walsh, Strehl, 2011)

- Extension of R-MAX to *more general representations*
- Let's say the transition model $P(s' \mid s, a)$ is defined by n parameters
  Typically, $n \ll$ *number of states*!
- Efficient KWIK-learner L requires a number of samples which is polynomial in n to estimate approximately correct $\hat{P}(s' \mid s, a)$
  (KWIK = Knows-what-it-knows framework)
- KWIK-R-MAX using L is *PAC-MDP efficient in* n
  $\rightarrow$ polynomial in number of parameters of transition model!
  $\rightarrow$ more efficient than plain R-MAX by several orders of magnitude!

Bayesian RL**

- There exists an optimal solution to the exploration-exploitation trade-off: belief planning (see my tutorial "Bandits, Global Optimization, Active Learning, and Bayesian RL – understanding the common ground")

$$V^\pi(b,s) = R(s,\pi(b,s)) + \int_{b',s'} P(b',s' \mid b,s,\pi(b,s))\; V^\pi(b',s')$$

  - Agent maintains a *distribution (belief)* $b(m)$ *over MDP models* m
  - typically, MDP structure is fixed; belief over the parameters
  - belief updated after each observation (s,a,r,s'): $b \to b'$
  - only tractable for very simple problems

- *Bayes-optimal policy* $\pi^* = \operatorname{argmax}_\pi V^\pi(b,s)$
  - no other policy leads to more rewards in expectation w.r.t. prior distribution over MDPs
  - solves the exploration-exploitation tradeoff

Optimistic heuristics

- As with UCB, choose estimators for R^*, P^* that are optimistic/over-confident

$$V_t(s) = \max_a \left[R^* + \textstyle\sum_{s'} P^*(s'|s,a)\, V_{t+1}(s') \right]$$

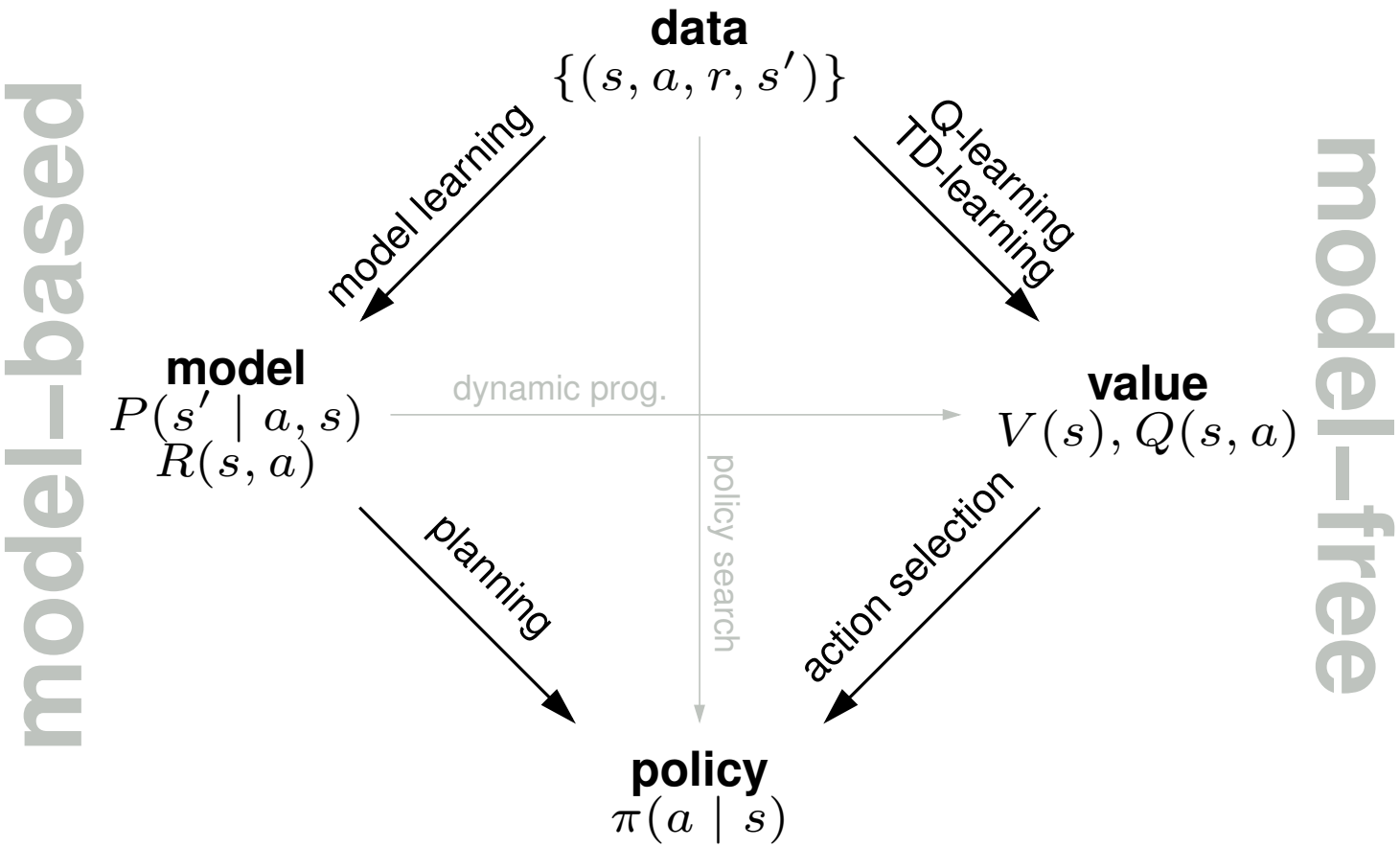
- Rmax:
  - $R^*(s,a) = \begin{cases} R_{\max} & \text{if } \#_{s,a} < n \\ \hat{\theta}_{rsa} & \text{otherwise} \end{cases}$, $P^*(s'|s,a) = \begin{cases} \delta_{s's*} & \text{if } \#_{s,a} < n \\ \hat{\theta}_{s'sa} & \text{otherwise} \end{cases}$
  - Guarantees over-estimation of values, polynomial PAC results!
  - Read about "KWIK-Rmax"! (Li, Littman, Walsh, Strehl, 2011)
- Bayesian Exploration Bonus (BEB), Kolter & Ng (ICML 2009)
  - Choose $P^*(s'|s,a) = P(s'|s,a,b)$ integrating over the current belief $b(\theta)$ (non-over-confident)
  - But choose $R^*(s,a) = \hat{\theta}_{rsa} + \frac{\beta}{1+\alpha_0(s,a)}$ with a hyperparameter $\alpha_0(s,a)$, over-estimating return
- Confidence intervals for V-/Q-function (Kealbling '93, Dearden et al. '99)

More ideas about exploration

- **Intrinsic rewards** for *learning progress*
  - "fun", "curiousity"
  - in addition to the external "standard" reward of the MDP
  - *"Curious agents are interested in learnable but yet unknown regularities, and get bored by both predictable and inherently unpredictable things."* (J. Schmidhuber)
  - Use of a meta-learning system which learns to predict the error that the learning machine makes in its predictions; meta-predictions measure the *potential interestingness of situations* (Oudeyer et al.)

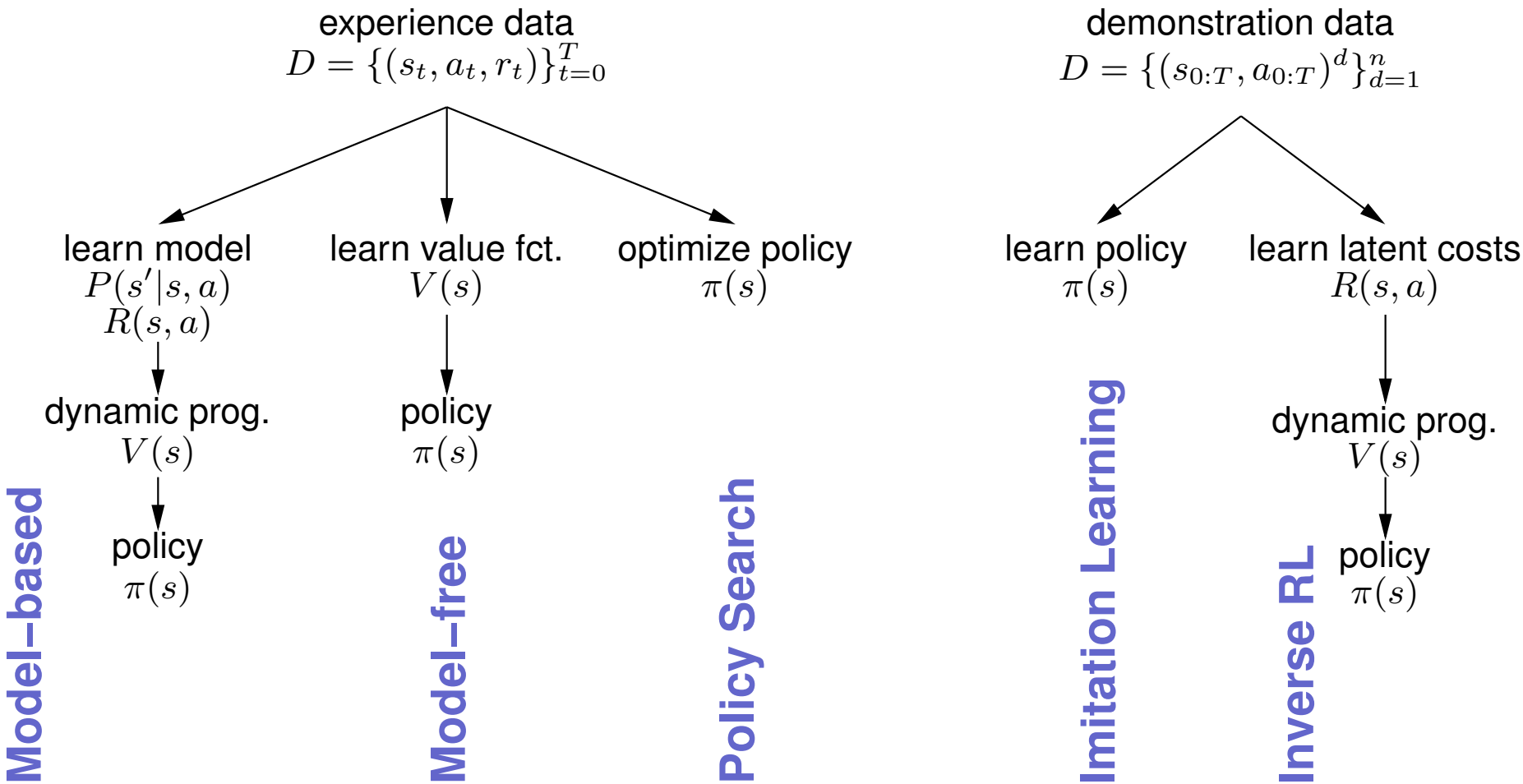
- *Dimensionality reduction* for model-based exploration in *continuous spaces*: low-dimensional representation of the transition function; focus exploration on relevant dimensions (A. Nouri, M. Littman)

Policy Search, Imitation, & Inverse RL**



– Policy gradients are one form of policy search.
– There are other, *direct* policy search methods
(e.g., plain stochastic search, “Covariance Matrix Adaptation”)

Five approaches to learning behavior**



Policy Gradients**

- In continuous state/action case, represent the policy as linear in arbitrary state features:

$$\pi(s) = \sum_{j=1}^{k} \phi_j(s)\beta_j = \phi(s)^\top \beta \qquad \text{(deterministic)}$$

$$\pi(a \,|\, s) = \mathcal{N}(a \,|\, \phi(s)^\top \beta, \Sigma) \qquad \text{(stochastic)}$$

  with k features ϕ_j.

- Basically, given an episode $\xi = (s_t, a_t, r_t)_{t=0}^{H}$, we want to estimate

$$\frac{\partial V(\beta)}{\partial \beta}$$

Policy Gradients**

- One approach is called REINFORCE:

$$\frac{\partial V(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta} \int P(\xi|\beta)\ R(\xi)\ d\xi = \int P(\xi|\beta) \frac{\partial}{\partial \beta} \log P(\xi|\beta) R(\xi) d\xi$$

$$= \mathsf{E}_{\xi|\beta}\{\frac{\partial}{\partial \beta} \log P(\xi|\beta) R(\xi)\} = \mathsf{E}_{\xi|\beta}\{\sum_{t=0}^{H} \gamma^t \frac{\partial \log \pi(a_t|s_t)}{\partial \beta} \underbrace{\sum_{t'=t}^{H} \gamma^{t'-t} r_{t'}}_{Q^\pi(s_t, a_t, t)}\}$$

- Another is PoWER, which requires $\frac{\partial V(\beta)}{\partial \beta} = 0$

$$\beta \leftarrow \beta + \frac{\mathsf{E}_{\xi|\beta}\{\sum_{t=0}^{H} \epsilon_t Q^\pi(s_t, a_t, t)\}}{\mathsf{E}_{\xi|\beta}\{\sum_{t=0}^{H} Q^\pi(s_t, a_t, t)\}}$$

See: Peters & Schaal (2008): *Reinforcement learning of motor skills with policy gradients*, Neural Networks.
Kober & Peters: *Policy Search for Motor Primitives in Robotics*, NIPS 2008.
Vlassis, Toussaint (2009): *Learning Model-free Robot Control by a Monte Carlo EM Algorithm.* Autonomous Robots 27, 123-130.

Kober & Peters: *Policy Search for Motor Primitives in Robotics*, NIPS 2008.

Imitation Learning**

$$D = \{(s_{0:T}, a_{0:T})^d\}_{d=1}^n \quad \overset{\text{learn/copy}}{\rightarrow} \quad \pi(s)$$

- Use ML to imitate demonstrated state trajectories $x_{0:T}$

  Literature:

  Atkeson & Schaal: Robot learning from demonstration (ICML 1997)

  Schaal, Ijspeert & Billard: Computational approaches to motor learning by imitation (Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences 2003)

  Grimes, Chalodhorn & Rao: Dynamic Imitation in a Humanoid Robot through Nonparametric Probabilistic Inference. (RSS 2006)

  Rüdiger Dillmann: Teaching and learning of robot tasks via observation of human performance (Robotics and Autonomous Systems, 2004)

Imitation Learning**

- There a many ways to imitate/copy the oberved policy:

  Learn a density model $P(a_t \mid s_t)P(s_t)$ (e.g., with mixture of Gaussians) from the observed data and use it as policy (Billard et al.)

  Or trace observed trajectories by minimizing perturbation costs (Atkeson & Schaal 1997)

Imitation Learning**

Atkeson & Schaal

Inverse RL**

$$D = \{(s_{0:T}, a_{0:T})^d\}_{d=1}^n \quad \overset{\text{learn}}{\rightarrow} \quad R(s,a) \quad \overset{\text{DP}}{\rightarrow} \quad V(s) \quad \rightarrow \quad \pi(s)$$

- Use ML to "uncover" the latent reward function in observed behavior

Literature:

Pieter Abbeel & Andrew Ng: Apprenticeship learning via inverse reinforcement learning (ICML 2004)

Andrew Ng & Stuart Russell: Algorithms for Inverse Reinforcement Learning (ICML 2000)

Nikolay Jetchev & Marc Toussaint: Task Space Retrieval Using Inverse Feedback Control (ICML 2011).

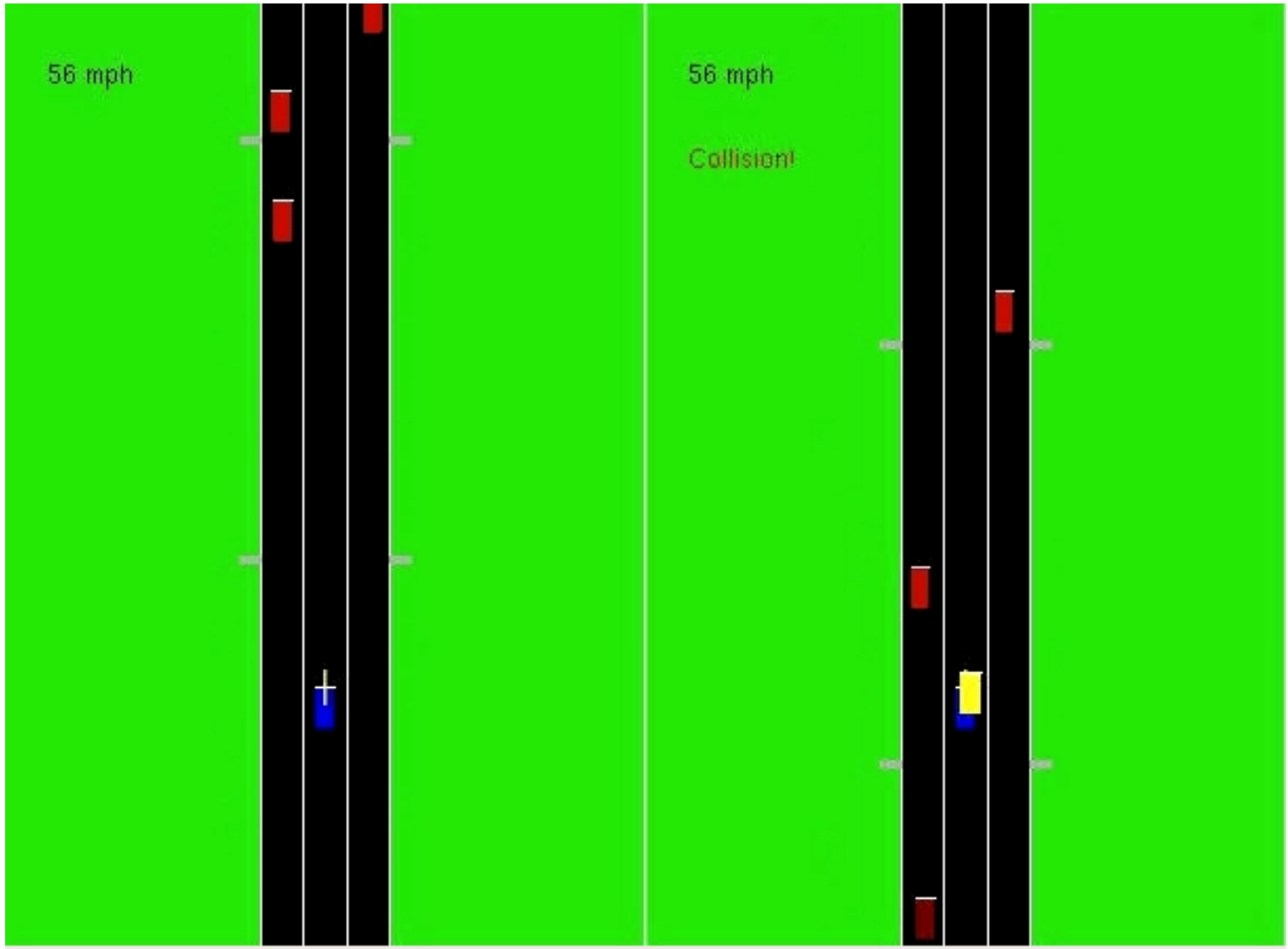
Inverse RL (Apprenticeship Learning)**

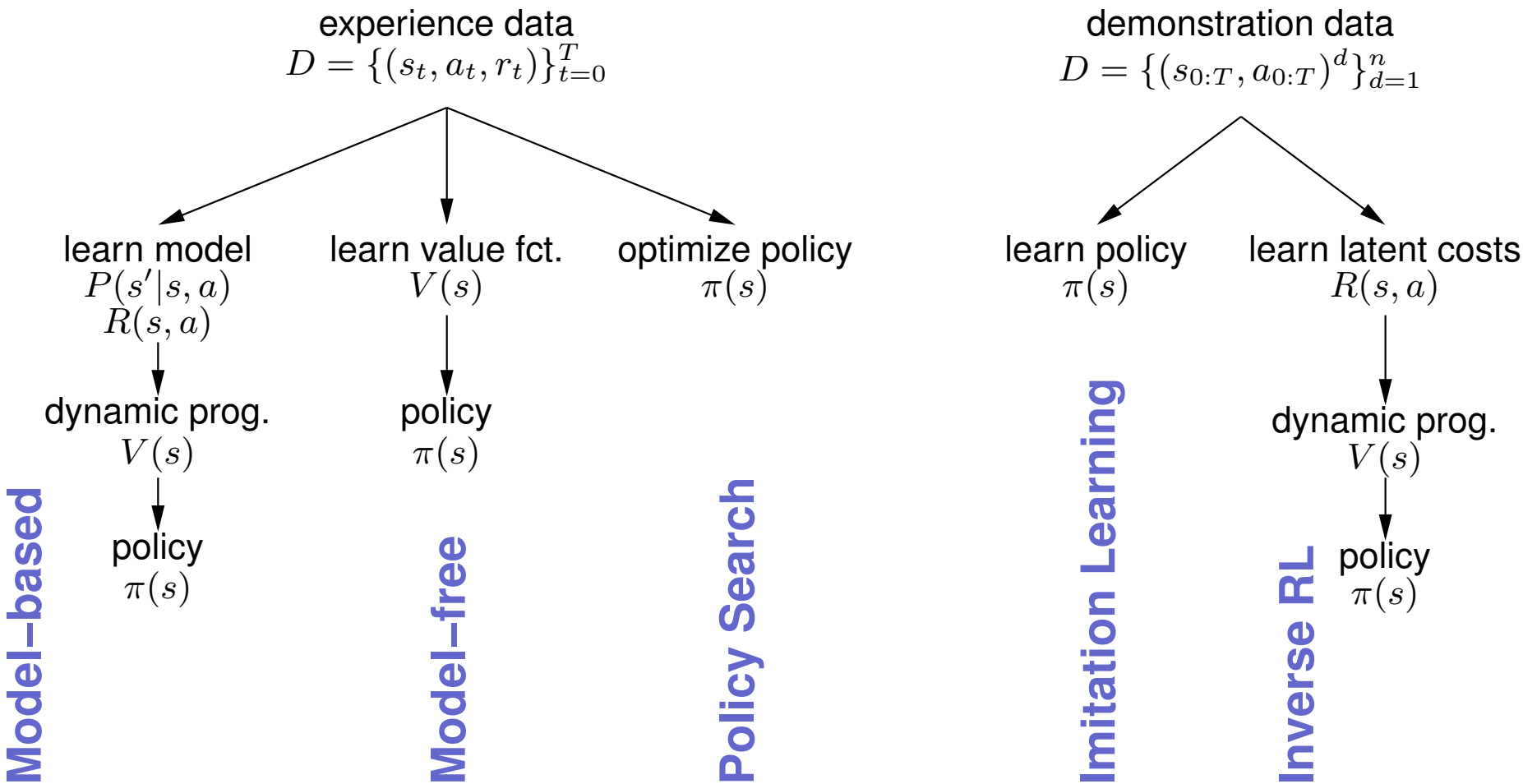
- Given: demonstrations $D = \{x^d_{0:T}\}_{d=1}^n$
- Try to find a reward function that **discriminates demonstrations from other policies**
  – Assume the reward function is linear in some features $R(x) = w^\top \phi(x)$
  – Iterate:
    1. Given a set of candidate policies $\{\pi_0, \pi_1, ..\}$
    2. Find weights w that maximize the value margin between teacher and all other candidates

$$\max_{w,\xi} \ \xi$$
$$\text{s.t. } \forall_{\pi_i}: \ \underbrace{w^\top \langle \phi \rangle_D}_{\text{value of demonstrations}} \geq \underbrace{w^\top \langle \phi \rangle_{\pi_i}}_{\text{value of } \pi_i} + \xi$$
$$\|w\|^2 \leq 1$$

    3. Compute a new candidate policy π_i that optimizes $R(x) = w^\top \phi(x)$ and add to candidate list.

(Abbeel & Ng, ICML 2004)


56 mph
56 mph
Collision!


experience data
$D = \{(s_t, a_t, r_t)\}_{t=0}^{T}$
learn model
$P(s'|s,a)$
$R(s,a)$
dynamic prog.
$V(s)$
policy
$\pi(s)$
learn value fct.
$V(s)$
policy
$\pi(s)$
optimize policy
$\pi(s)$
Model–based
Model–free
Policy Search
demonstration data
$D = \{(s_{0:T}, a_{0:T})^d\}_{d=1}^{n}$
learn policy
$\pi(s)$
learn latent costs
$R(s,a)$
dynamic prog.
$V(s)$
policy
$\pi(s)$
Imitation Learning
Inverse RL