Some outstanding challenges in reinforcement learning

Csaba Szepesvári

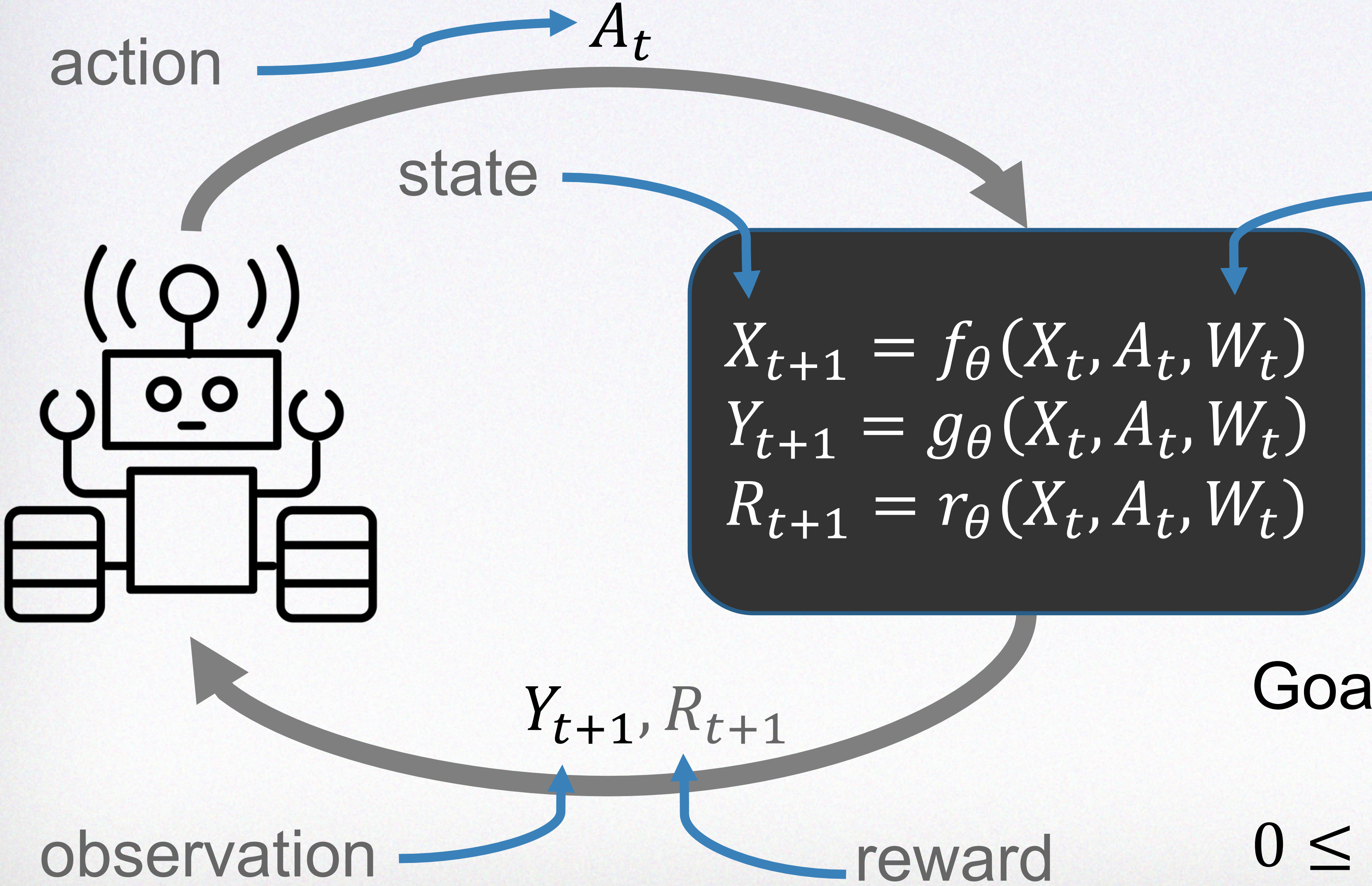
DeepMind

UNIVERSITY OF ALBERTA

Contents

- What is RL? How does ML work?
- Does it work? What makes it work?
- How is it done?
  - ADP
  - What is known about ADP?
  - Challenge #1: Efficient planning
- On the exploration problem
  - Strategic planning, optimism
  - Challenge #2: Efficient exploration
- Conclusions

Reinforcement Learning (RL)



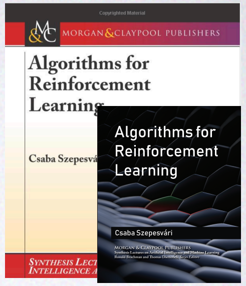
$\theta \in \Theta$: unknown "parameter"

Goal: maximize

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1}]$$

$0 \leq \gamma \leq 1$ fixed, known

RL= problems, $\neq$ techniques!!



https://goo.gl/ftTfS4

- **Offline learning**
  - Learn a good controller given some data collected from interacting with the system – **batch RL**
- **Online learning**
  - Interact with the system with the goal of finding a good controller with the least number of interactions – **pure exploration**
  - Interact with the system with the goal of collecting as much reward as possible – **the exploration problem**
- **Learn from a simulator**
  - Find a good controller/action for the simulated system (or beyond) with minimal computation – **planning (with a simulator)**

The modus operandi in RL
($\subseteq$ machine learning)

minimal modeling

maximum compute

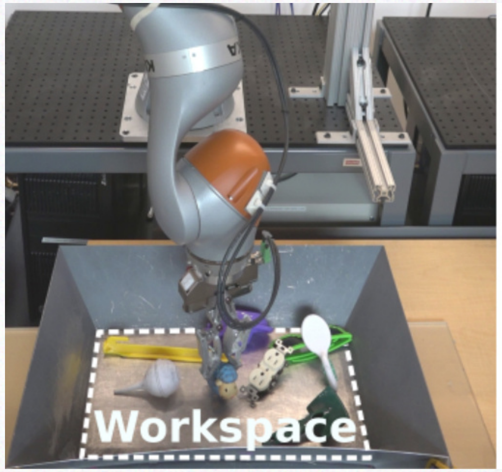
Does it work? Why (now)?

Some landmark results

- DeepMind:
  - Atari
  - AlphaGo/Alpha Zero
- Others:
  - OpenAI Five: Dota-2 agents
    - Capture the flag (Deepmind)
  - Google Brain & X: vision-based grasping

Single RL algorithm defeating world-champion in Go & best chess program

Single RL algorithm learning to play 49 Atari games @ human level or beyond



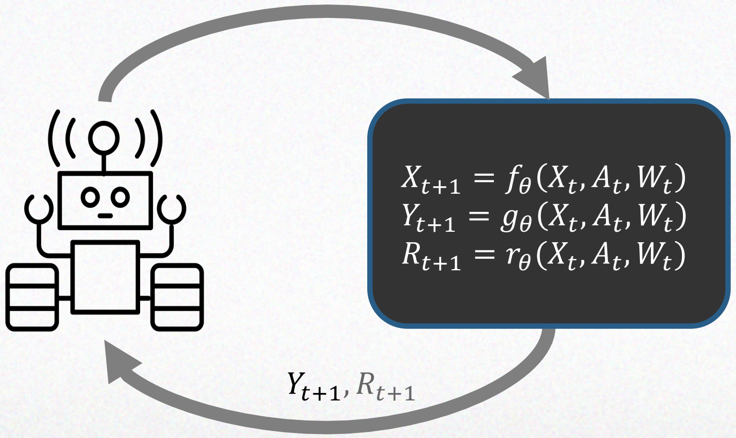
Autonomous learning of vision-based grasping

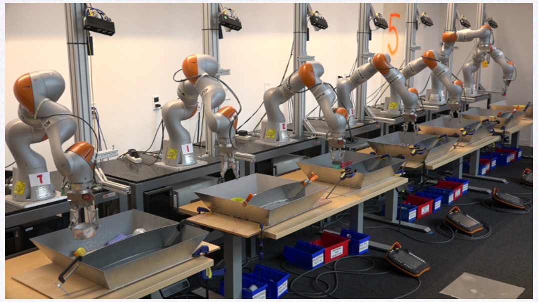
Defeating amateur human teams in Dota-2

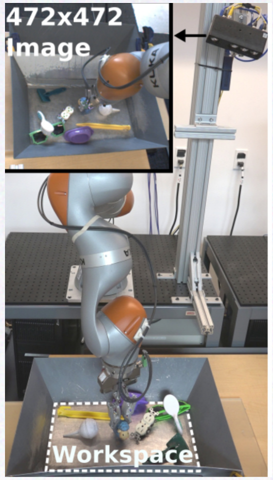
Vision-based grasping

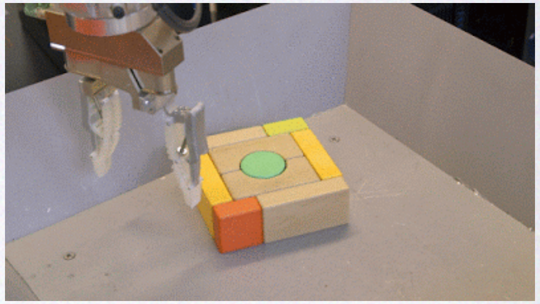
https://goo.gl/kTMcCb
Kalashnikov et al. (arXiv, 2018)

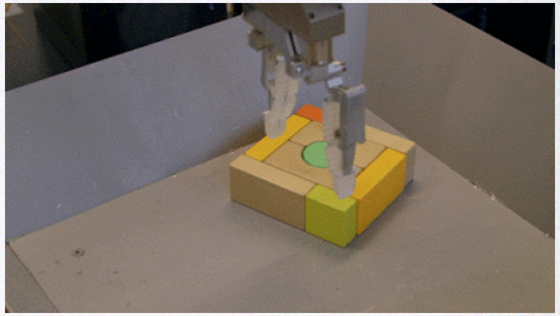
- Y_t: 472x472 RGB images, gripper state, height above ground, $Y_t \neq X_t$
- A_t: 3D gripper displacement, 2D rotation, gripper open/close, termination (7D)
- R_t: success or failure at the “end”, fixed cost per time step
- Episodes: 20 steps, learned stopping









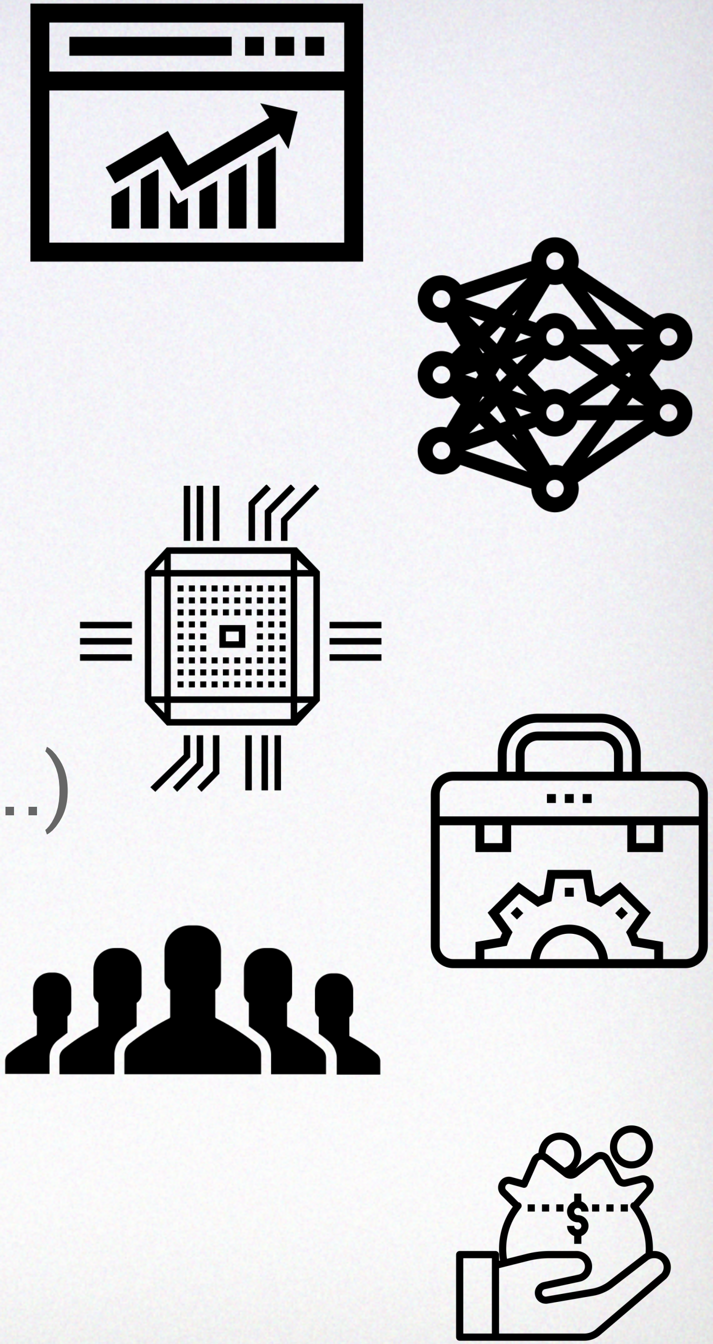


Autonomous learning of vision-based grasping

- RL on a physical system
- High success rate (78%→ 96%)
- Intelligent, robust, closed-loop behavior

Why now?

- Reduce everything to (some form of) optimization: DP (=use value functions)
- Flexible models:
  - Deep neural networks, ReLu, LSTM, ConvNet, ..
- Large scale computation (GPU, TPU, Cloud, ..)
- Software frameworks, SGD!
- Rapidly growing, very active community
- Commercial interest, funding



When to use off-the-shelf ML/RL?

- Mathematical modeling is painful to impossible
  - E.g., complex observations (vision, text, ...)
- Task can be specified as an optimization/constraint satisfaction problem
- Access to lots of data
  - High-fidelity simulator can be built
  - High throughput experimentation
- Access to huge-scale compute
- A priori verifiability is not a major concern
  - Simulator can be trusted
  - Physical experiments/online learning are feasible and sufficient

The core ideas

How RL works (~1990s)

Incrementally produce policies[1] $\pi_1, \pi_2, \ldots$

How?

1. **Value-based policy search** a.k.a. approximate **dynamic programming** (ADP)

    $\Leftarrow$ all the methods in "success stories" are based on ADP!

2. **Direct policy search**: k^{th}-order optimization, $0 \le k \le 2$
    - FDSA, SPSA, Monte-Carlo ($k = 0$),
    - SGD=REINFORCE ($k = 1$), Adam, momentum, Batchnorm, ...
    - LBFGS, K-FAC, .. ($k = 2$)
    - Name of the game: Variance reduction

Models?
Not really.. Could be.. Should be!

[1]policy = feedback controller, static or dynamic

Dynamic programming
(optimal control)

$\int h(y)P(dy|x,a) = \mathbb{E}[h(f(x,a,W))]$

- Value functions: $Q^{\pi}(x,a) = \mathbb{E}_{\pi,A_0=a,X_0=x}[\sum_{t=0}^{\infty} \gamma^t \; R_t]$
- Bellman optimality equation: $\forall(x,a) \in \mathcal{X}\times\mathcal{A}$:

$$Q^*(x,a) = \underbrace{r(x,a) + \gamma \int P(dy|x,a) \max_{a'} Q^*(y,a')}_{(TQ^*)(x,a)}$$

- $T: \mathbb{R}^{X\times A} \to \mathbb{R}^{X\times A}$

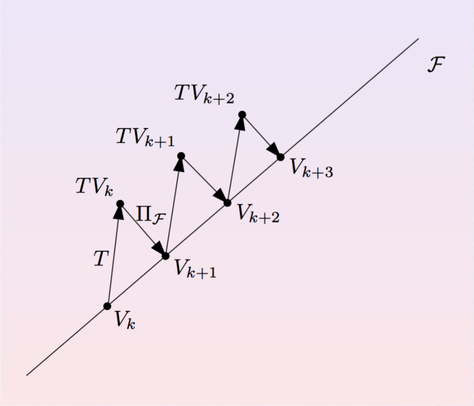
$$Q^* = TQ^*$$

- Optimal policy: $\pi^*(x) = \arg\max_a Q^*(x,a)$
- Classic DP: Compute Q^*, use greedy policy
- Methods: Value-iteration, policy iteration, linear programming

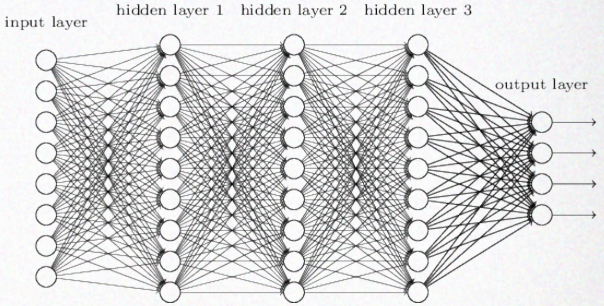
Richard E. Bellman
(1920-1984)

No state aliasing!
$X_t = Y_t$,
or some known
function of it..

Function approximation



- Value iteration: $Q_{k+1} = TQ_k \to Q^*$
  - Converges geometrically
- TQ_k is intractable:
  - $(TQ)(x,a) = r(x,a) + \gamma \int P(dy|x,a) \max_{a'} Q(y,a')$
- Set up regression problem to "learn" TQ_k using eg neural net!
- Sample $(X_i, A_i) \sim \mu$,
  $Y_i = r_\theta(X_i, A_i, W_i) + \gamma \max_{a'} Q(f_\theta(X_i, A_i, W_i), a')$
  $i = 1,2, \ldots, n$



Variations

Alpha Zero!

- Between value and policy iteration:
  - $\pi_{k+1}(x) = \text{argmax}_a (T^p Q_k)(x,a), p \geq 0$ ⇒"classification"
  - $Q_{k+1} = T^q_{\pi_{k+1}} Q_k, q \in \{1,2,\ldots,\infty\}$ ⇒"regression"
- Use incremental learning methods ("recursive updates", "stochastic approximation", **TD-learning**, ...)
- Modify the operators involved: λ-update, entropy regularization, approximate greedification, ...
- Recycle data ("replay"); importance weighting
- Optimize data collection, parallelize computation

..does this work?

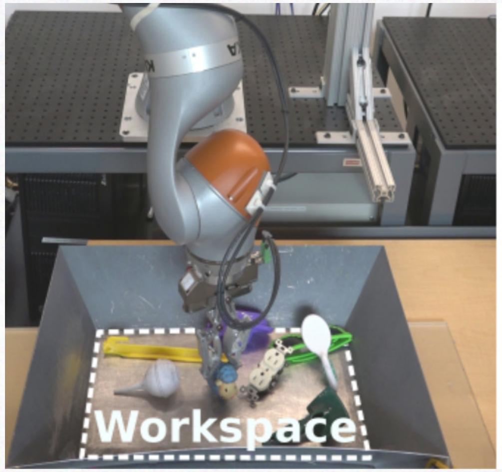
Some landmark results

- DeepMind:
  - Atari
  - AlphaGo/Alpha Zero

Single RL algorithm defeating world-champion in Go & best chess program

Single RL algorithm learning to play 49 Atari games @ human level or beyond

- Others:
  - OpenAI Five: Dota-2 agents
    - Capture the flag (Deepmind)
  - Google Brain & X: vision-based grasping

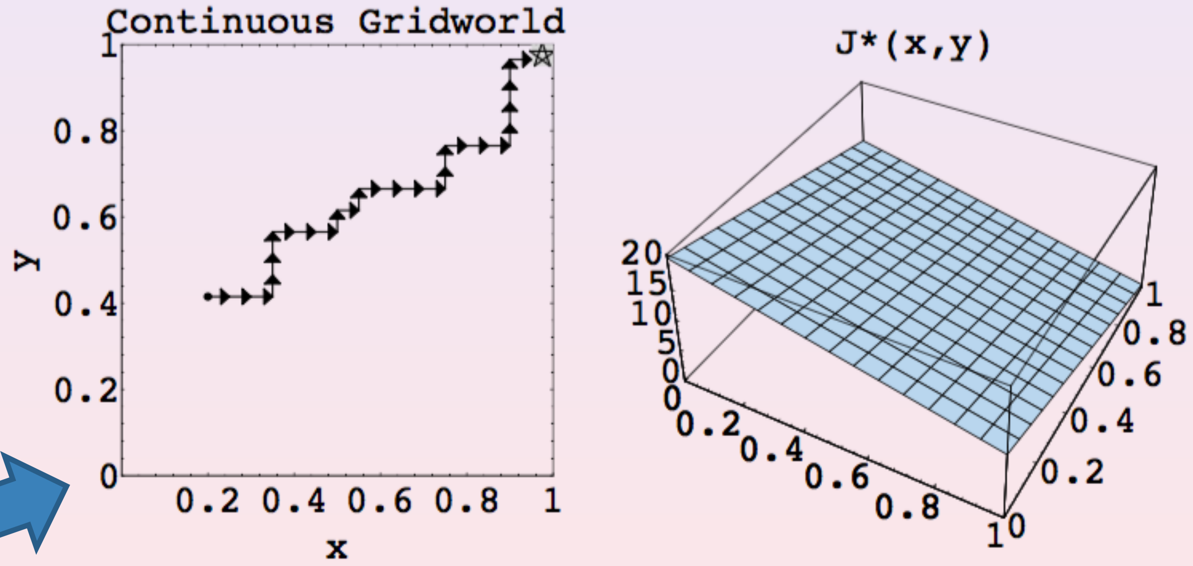


Autonomous learning of vision-based grasping

Defeating amateur human teams in Dota-2

..and failures..

From: Boyan & Moore: "Generalization in Reinforcement Learning: Safely Approximating the Value Function", *NIPS-7*, 1995.



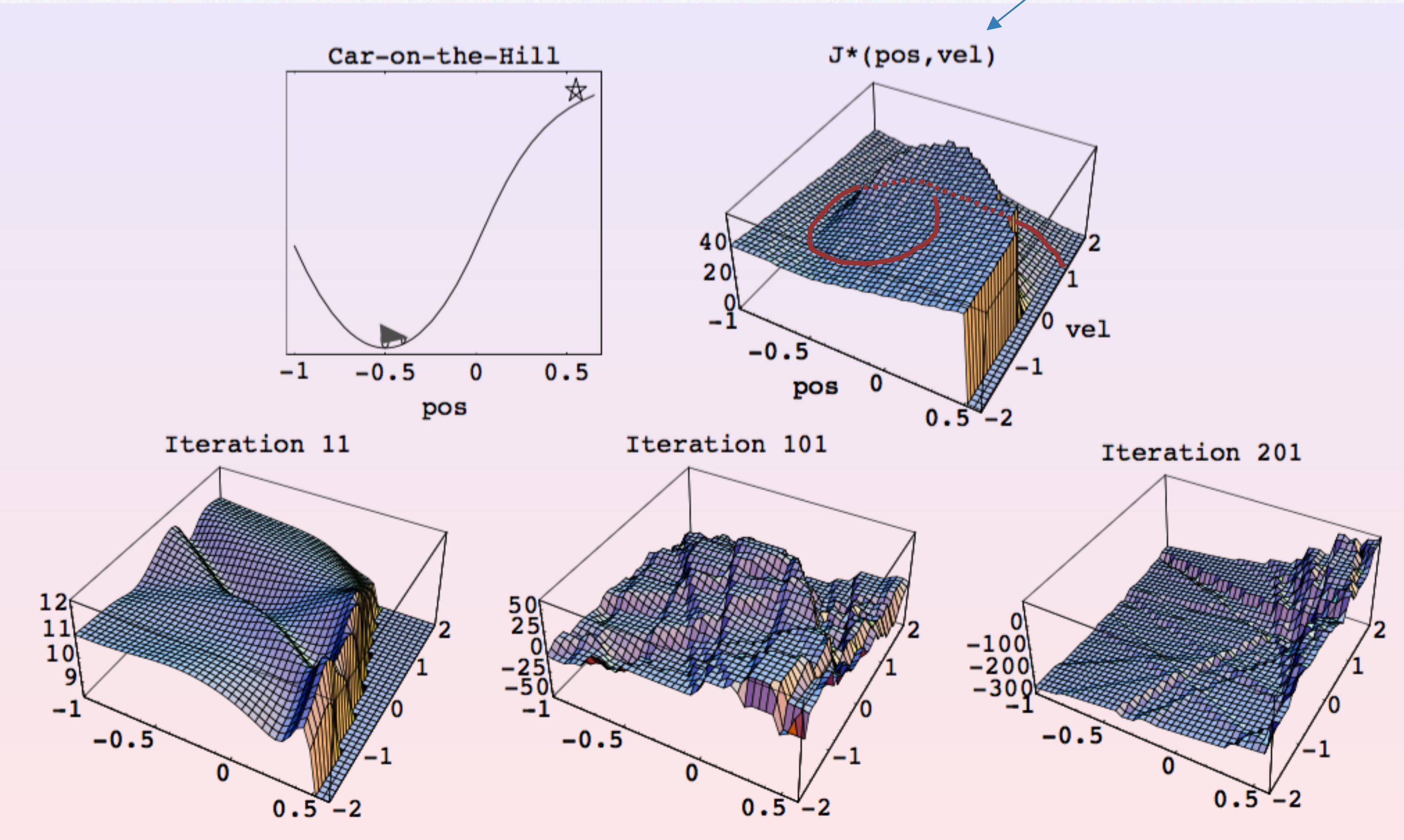
Goal position

[1] With thanks to Justin Boyan

μ is the uniform distribution, **quadratic polynomials** used for value-function approximation

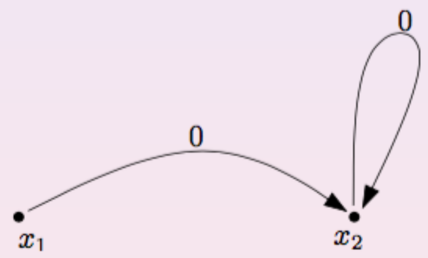
..add neural nets..

Optimal cost-to-go (-rewards)



..or trivial function approximation..

- Tsitsiklis & Van Roy (1996)
- State space: $\mathcal{X} = \{x_1, x_2\}$
- Dynamics:



- Bellman operator:

$$\begin{aligned}(TV)(x_1) &= 0 + \gamma V(x_2) \\ (TV)(x_2) &= 0 + \gamma V(x_2).\end{aligned}$$

- Function-space:
  $\mathcal{F} = \{\theta\phi \,|\, \theta \in \mathbb{R}\}$,

$$\phi(x_1) = 1,\ \phi(x_2) = 2.$$

Iteration:

$$\begin{aligned}\theta_{t+1} &= \text{argmin}_\theta \|\theta\phi - T(\theta_t\phi)\|_2 \\ &= \text{argmin}_\theta (\theta - \gamma 2\theta_t)^2 + (2\theta - \gamma 2\theta_t)^2 = (6/5\gamma)\theta_t \to +\infty\end{aligned}$$

μ is the uniform distribution

Poor outlook for ADP

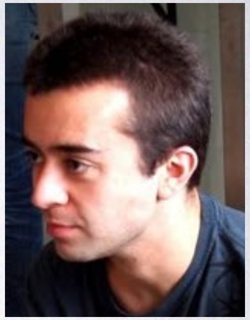
- *"In light of these experiments, we conclude that the straightforward combination of DP and function approximation is not robust."* (Boyan & Moore, NIPS-7, 1995)
- *Unfortunately, many popular functions approximators, such as neural nets and linear regression, do not fall in this*[2] *class (and in fact can diverge).* (G. Gordon, ICML, 1995).

But then why does it work for the "landmark results"?

~~Why~~ When does it work?


R. Munos


A.m. Farahmand


B.A. Pires

Theorem (Sz., Munos, 2005):

Covariate-shift price

Approximation error

$$\|V^* - V^{\pi_K}\|_{p,\rho} \le \frac{2\gamma}{(1-\gamma)^2}\left\{C(\rho,\mu)^{1/p}\,\epsilon_1 + \epsilon_2\right\}$$

$$\epsilon_1 = d(T\mathcal{F},\mathcal{F}) + \mathrm{poly}\left(\frac{\log(N)}{N}, \frac{\log(N|\mathcal{A}|)}{M}, \log(K), \dim(\mathcal{F})\right)$$

$$\epsilon_2 = \mathrm{const} \times \gamma^K$$

Iteration cost

Estimation error

Range of $V^* \sim \frac{1}{1-\gamma}$. We need both $\epsilon_1, \epsilon_2 \ll 1-\gamma$

Extensions (2005-2010): Single sample path, $|\mathcal{A}| = \infty$, regularization, classification, ...


We made it work!
(with A. Antos)

Lesson: How to make ADP work?

Covariate shift, or off-policy problem

Need to control all terms!

- $C(\rho, \mu)$: Sampling distr. μ should dominate $\rho \sum_{t=0}^{\infty} \gamma^t P_{\pi_K}^t$
  - Change μ as you go, change policies slowly, …
- Make approximation error $d(T\mathcal{F}, \mathcal{F})$ small:
  - Deep neural nets, LSTM, convnets, …
- Make sample size large to control estimation error
  - Large compute

..and in practice..

Work	Covariate shift	Approximation error	Estimation error	Computation platform
Atari2600 - DQN	Replay buffer	ConvNet, relatively shallow	50M frames, 38 days	GPUs
AlphaZero	Small learning rate	Deep convnet, residual blocks	700,000x4096=28 B	5000 TPUv1, 64 TPUv2
OpenAI Five	Penalize fast changes (PPO)	Large network, 1024 LSTM units	N*180 years, N = no. days	256 GPUs and 128,000 CPU
Vision-based grasping (QT-Opt)	Soft improvement in OPT, slowly mixing in new data	Deep convnet, 1.2 M params	580K offline grasps + 28K online grasps	1000 machines, 14K cores, 10 GPUs

Open problem #1

- Goal: Find a good policy/controller
- Setting: Access to a (stochastic) simulator
- **Assumption:**
    - Given a function approximator (linear, or not) that can represent/"learn" the optimal value function[1] with small error
- (When) can we do this in polynomial time? How good a policy can we find?
- Note: Assumption **much weaker** than used by above ADP result!

---

[1]And/or optimal policy/stationary distribution of optimal policy/..

A partial result

A Linearly Relaxed Approximate Linear Program for Markov Decision Processes

Chandrashekar Lakshminarayanan[†], Shalabh Bhatnagar[*], and Csaba Szepesvári[†]

IEEE TAC 63(4), 1185-1191, 2018

$$\min_{r \in \mathbb{R}^k} c^\top \Phi r$$

$$\text{s.t.} \sum_a W_a^\top \Phi r \geq \sum_a W_a^\top (g_a + \alpha P_a \Phi r)$$

$$c \geq 0, 1^\top c = 1$$
$$W_a \in [0,\infty)^{S \times m}, \psi \in [0,\infty)^S$$
$$\|J\|_{\infty,\psi} = \max_s \frac{|J(s)|}{\psi(s)}$$
$$\beta_\psi := \alpha \max_a \|P_a \psi\|_{\infty,\psi} < 1$$
$$\psi \in \text{span}(\Phi)$$

Theorem: Let $\epsilon = \inf_{r \in \mathbb{R}^k} \|J^* - \Phi r\|_{\infty,\psi}$, $J_{\text{LRA}} = \Phi r_{\text{LRA}}$, where r_{LRA} is the solution to the above LP. Then, under the said assumptions,

$$\|J^* - J_{\text{LRA}}\|_{1,c} \leq \frac{2c^\top \psi}{1 - \beta_\psi} \left(3\epsilon + \|J^*_{\text{ALP}} - J^*_{\text{LRA}}\|_{\infty,\psi}\right)$$

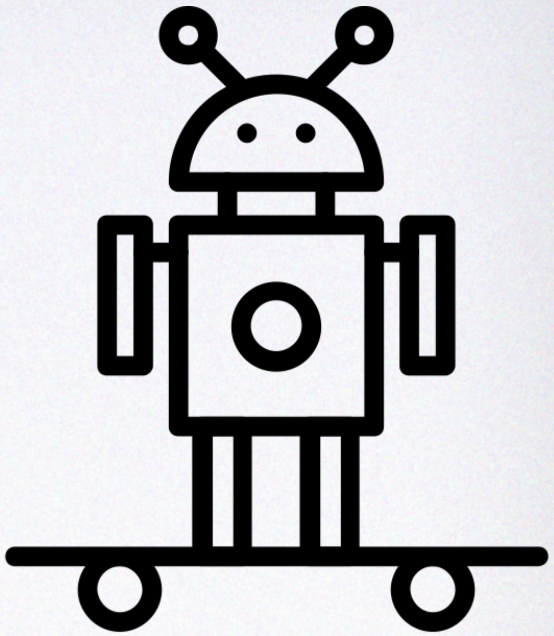
P. J. Schweitzer and A. Seidmann, "Generalized polynomial approximations in Markovian decision processes," *Journal of Mathematical Analysis and Applications*, vol. 110, pp. 568–582, 1985.

D. P. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, pp. 850–865, 2003.

——, "On constraint sampling in the linear programming approach to approximate dynamic programming," *Mathematics of Operations Research*, vol. 29, pp. 462–478, 2004.

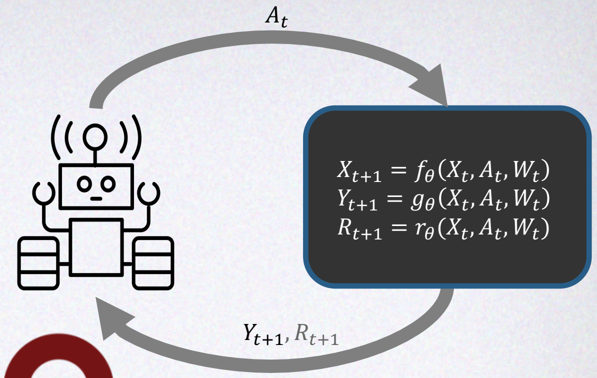
On the exploration problem

Learning cheaply, online



- Goal: Interact with a "real" system and collect as much reward as possible!
- Performance metric:
  - Total reward collected, or..
  - **Regret: Measure of learning speed**
    "Difference to a baseline"
    - Regret is invariant to shifting the rewards
    - Scale fixed: Algorithms can be compared across different environments

Bandit problems

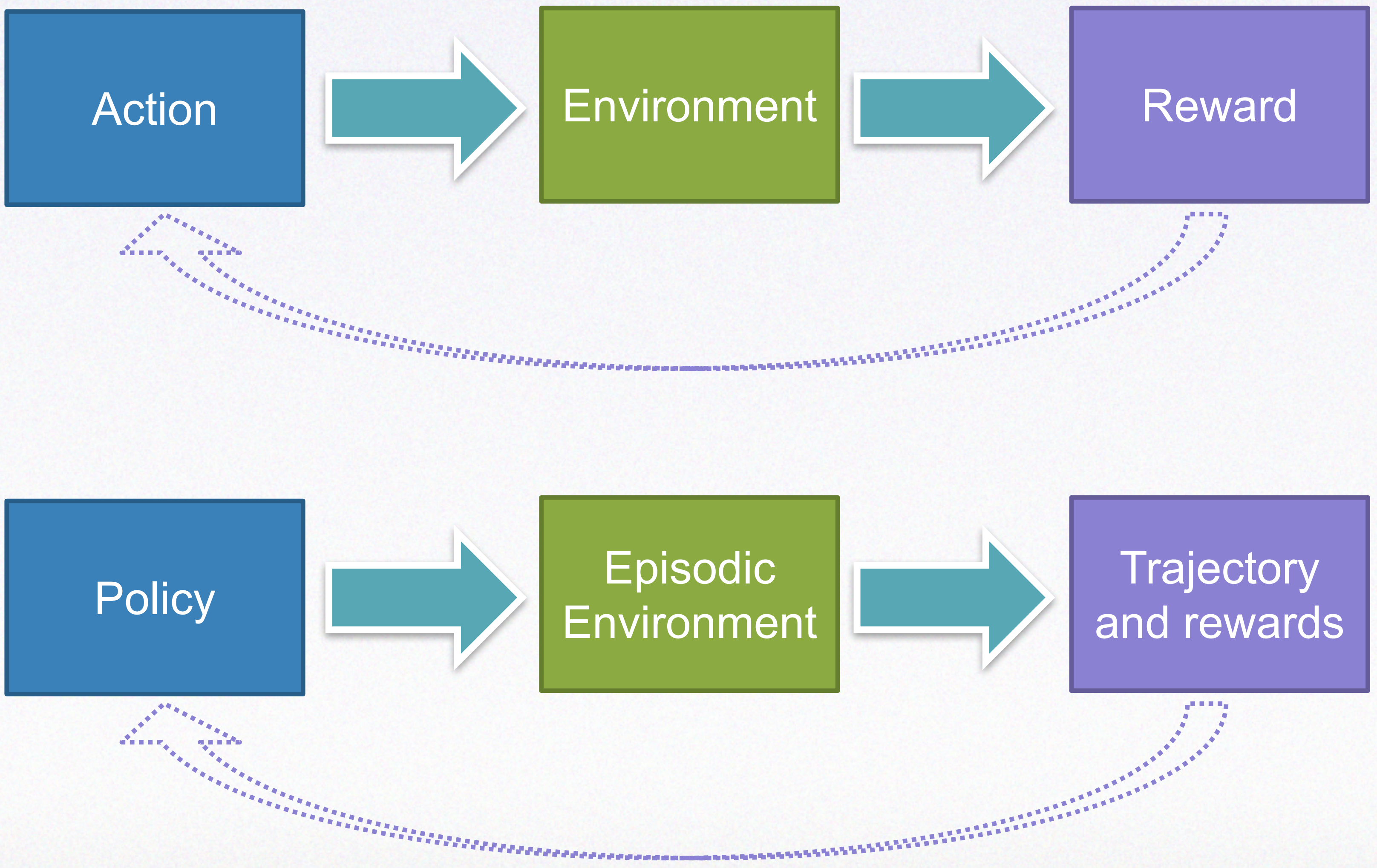


$\mathbb{P}$(payoff=1)=0.1 $\mathbb{P}$(payoff=1)=0.5 $\mathbb{P}$(payoff=1)=0.2

$$X_{t+1} = X_t,\ Y_{t+1} = R_{t+1},\ R_{t+1} = r(A_t, W_t)$$

Regret = $n \max_a \mathbb{E}[r(a, W)] - \sum_{t=0}^{n-1} R_t$

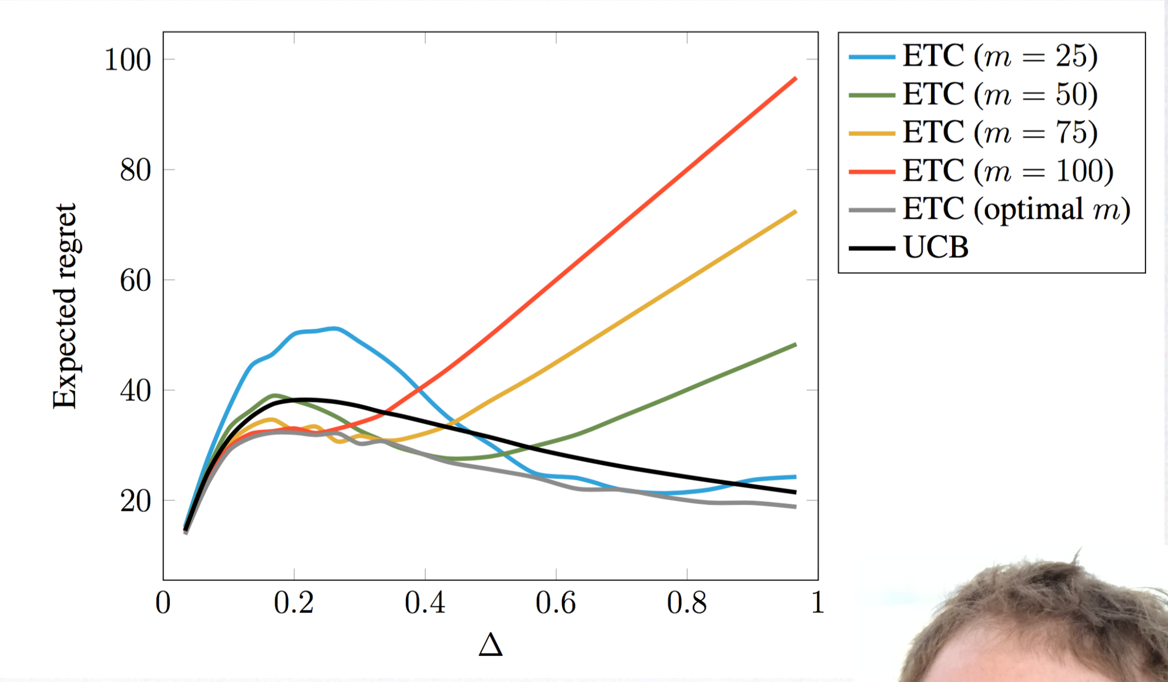
Bandits vs. (episodic) MDPs



Bandits on one slide

New book!
http://banditalgs.com

- Ad-hoc exploration: Good on some instances, bad on others
  - Explore-then-commit (ETC)
  - ϵ-greedy, Boltzmann/Gibbs
- Planned exploration reaches optimal regret for all instances
  - UCB, posterior sampling a.k.a. Thompson sampling, …



2 arms, unit variance Gaussian rewards with means 0 and –Δ, horizon 1000

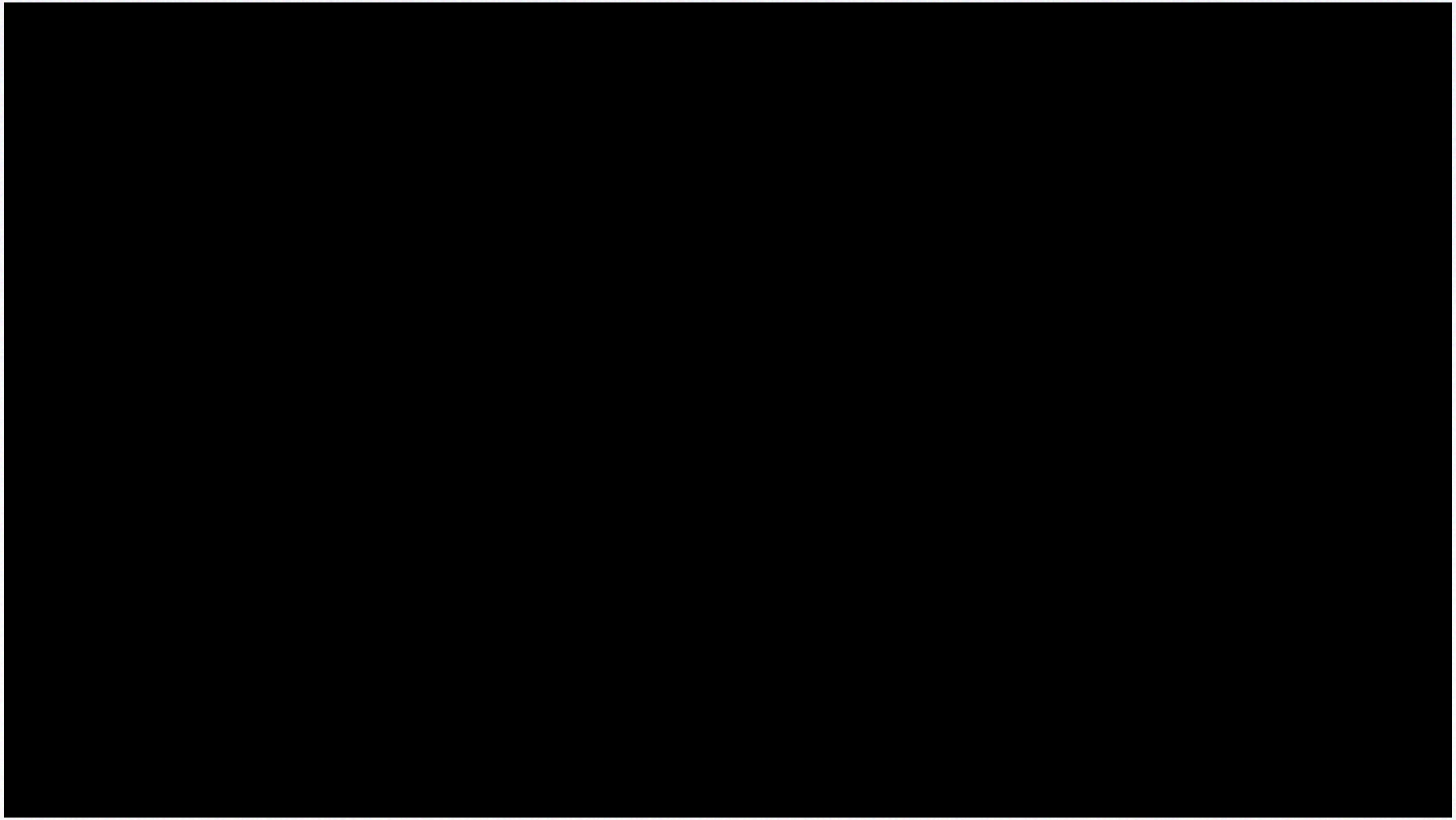
Open problem #2

- Goal: Collect as much reward as possible
- Setting: Interacting with an unknown environment
- **Assumption:**
  - Given a function approximator (linear, or not) that can represent/"learn" the optimal value function[1] with small error
- How big will be the regret? Can this be done with polynomial time computation? When?
- Note: Much harder than problem #1

---

[1]And/or optimal policy/stationary distribution of optimal policy/..

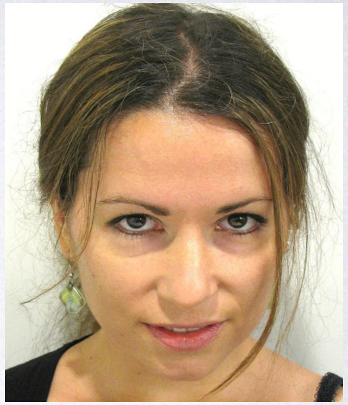
An illustration of the differences



Video: courtesy of Ian Osband

Partial results

Y. Abbasi-Yadkori



N. Lazic

- Linear Quadratic Regulation
- Optimism gives $\tilde{O}(\sqrt{T})$ regret

  (Abbasi-Yadkori, Sz., COLT'11)
- Current work/open
  - Computational efficiency
  - Regret efficiency
  - Non-asymptotic
  - Dependence on instance
  - **Model-free**, $O(T^{3/4})$ regret
    (Lazic, Abbasi-Yadkori, Sz., 2018)

$$X_{t+1} = AX_t + BU_t + W_{t+1}$$
$$Y_t = X_t$$
$$c_t = X_t^\top Q X_t + U_t^\top R U_t$$

Goal: minimize

$$\lim_{T\to\infty} \frac{1}{T} \mathbb{E}[\textstyle\sum_{t=0}^{T-1} c_t] ,$$

A, B are unknown, $W_t \sim N(0, I)$

Conclusions

Current approach in ML/RL

minimal modeling

maximum computation

Did it work?

- Yes, a few times..
- Requirements:
  - Task can be specified as an optimization/constraint satisfaction problem
  - Access to **loads of data**
  - Access to huge-scale compute

Can we overdo learning?

- Meta-learning, evolution, learning to plan, learning symbol manipulation, …
- Why?
  - Because it worked
  - Seamless integration with the rest of the architecture
- Why not?
  - Combinatorial explosion
  - Slow
  - Lack of understanding, transparency, verifiability, ..

What else is missing?

- Learning and using models in an effective manner
  - Learn “planner-friendly” models
  - Models that work despite complex sensory inputs
  - Multiscale problems (fine-coarse-huge)
- Learning from sparse/no-reward reward
  - Same problem as learning good models?

Questions?