# Why you should work on reinforcement learning


Csaba Szepesvari

UNIVERSITY OF ALBERTA

DeepMind

# Contents

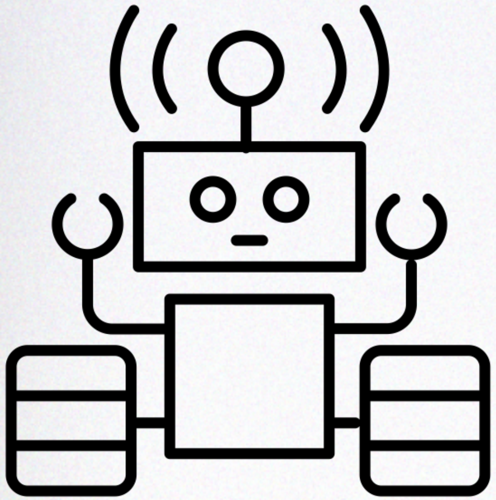
- Recent successes
- How is it done? Core ideas
- Learn cheaply: Exploration
- Conclusions

# Recent successes

# Reinforcement Learning (RL)

$A_t$

$\theta \in \Theta$ unknown



$$X_{t+1} = f_\theta(X_t, A_t, W_t)$$
$$Y_{t+1} = g_\theta(X_t, A_t, W_t)$$
$$R_{t+1} = r_\theta(X_t, A_t, W_t)$$

$Y_{t+1}, R_{t+1}$

Goal: maximize

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

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


DeepMind

Free pdf: https://goo.gl/ftTfS4

# Motto in machine learning & RL:

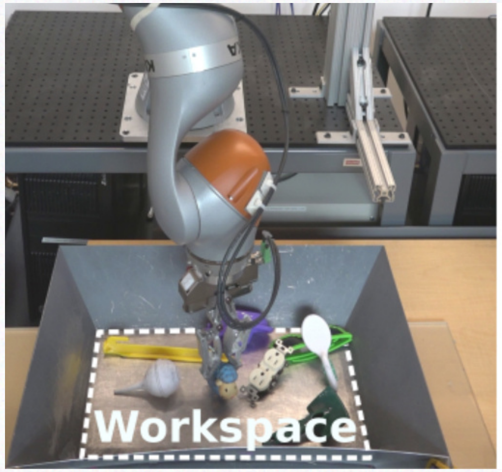
minimal modeling

maximum computation

# Recent successes

- Atari
- AlphaGo/Alpha Zero

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

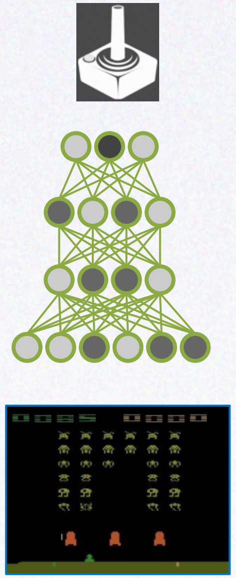


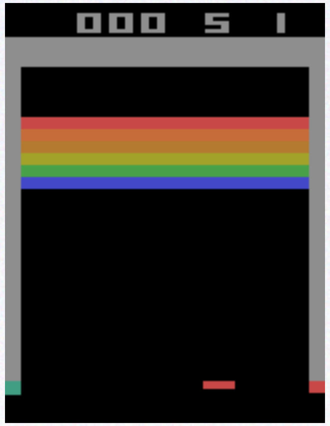

# Look ma! No preprocessing!

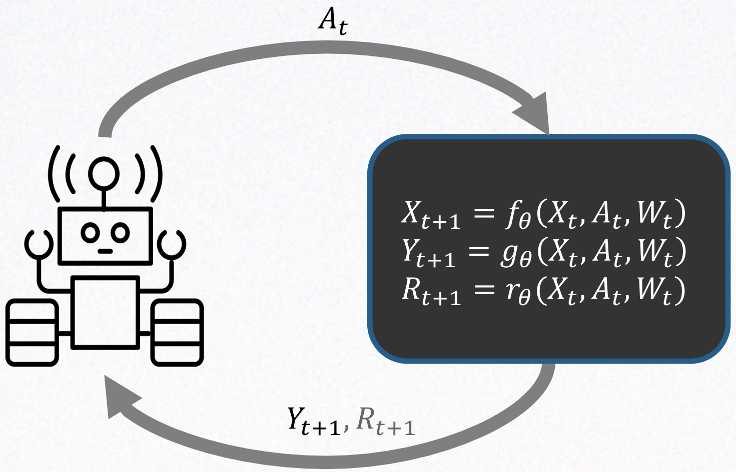

nature
Human-level control through deep reinforcement learning
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare

V. Mnih et al., 2015

- $Y_t$: last 4 frames, so $X_t = Y_t$
- $A_t$: joystick + button
- $R_t$: sign of score change
- Episodes: Life loss/minutes






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

- RL + Vision can be made to work using deep convnets
- Minimal prior, large compute is powerful

DeepMind

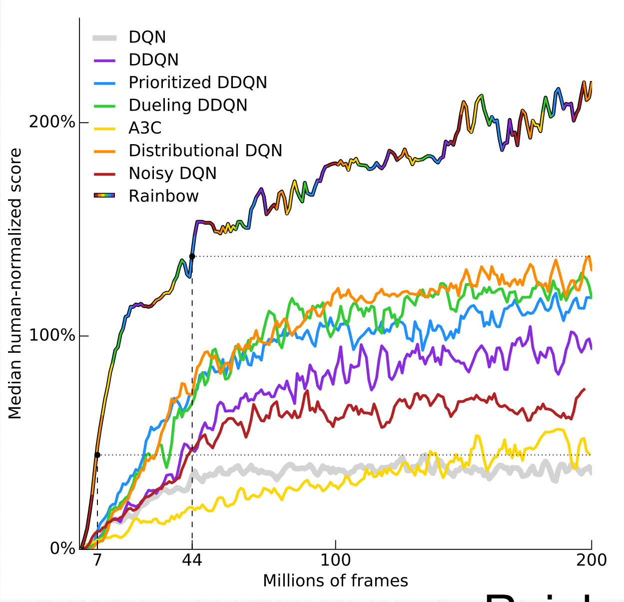
[https://deepmind.com/research/dqn/](https://deepmind.com/research/dqn/)

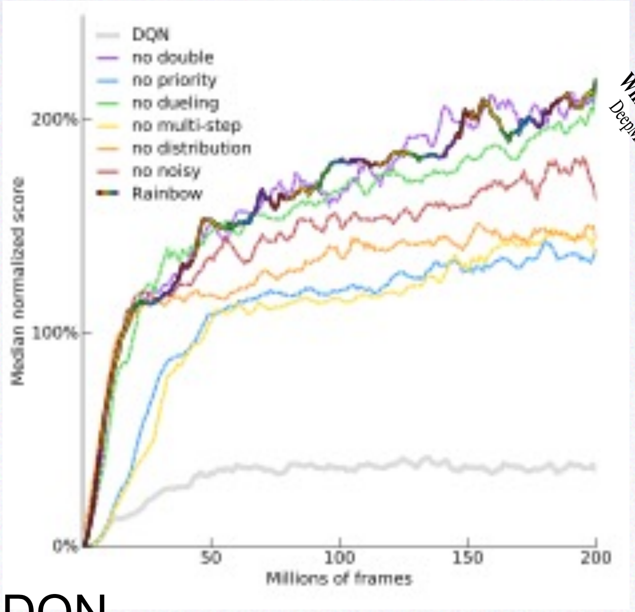
# » 2018

**Observe and Look Further:**
**Achieving Consistent Performance on Atari**

Tobias Pohlen[1], Bilal Piot[1], Todd Hester[1], Mohammad Gheshlaghi Azar[1], Dan Horgan[1], David Budden[1], Gabriel Barth-Maron[1], Hado van Hasselt[1], John Quan[1], Mel Večerík[1], Matteo Hessel[1], Rémi Munos[1], and Olivier Pietquin[2]

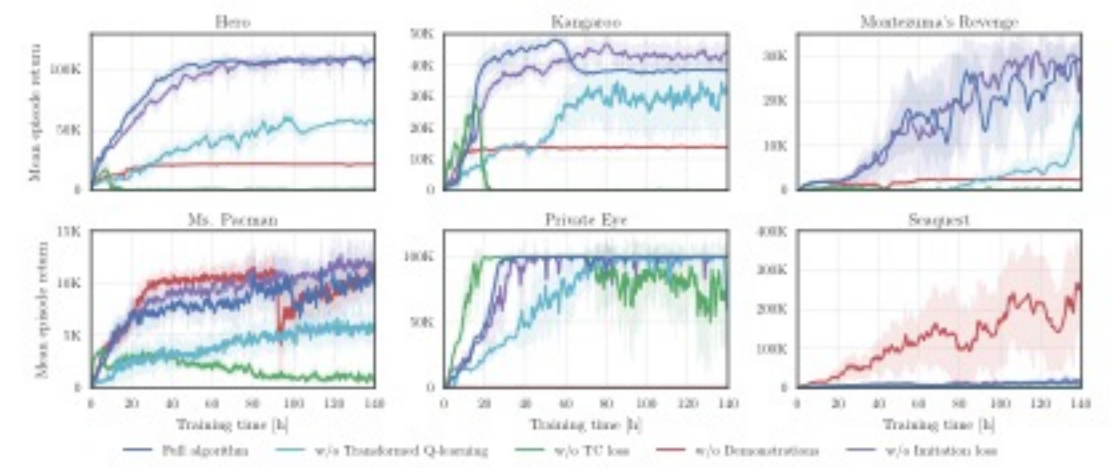
Ape-X DQfD
with expert data




Rainbow DQN


| Algorithm | Rainbow | DQfD | Ape-X DQN | Ape-X DQfD |
|---|---|---|---|---|
| Rainbow DQN | – | 31 / 42 | 9 / 42 | 10 / 42 |
| DQfD | 11 / 42 | – | 7 / 42 | 11 / 42 |
| Ape-X DQN | 34 / 42 | 35 / 42 | – | 28 / 42 |
| Ape-X DQfD | 32 / 42 | 39 / 42 | 15 / 42 | – |
| Ape-X DQfD (deeper) | **36 / 42** | **40 / 42** | **28 / 42** | **33 / 42** |



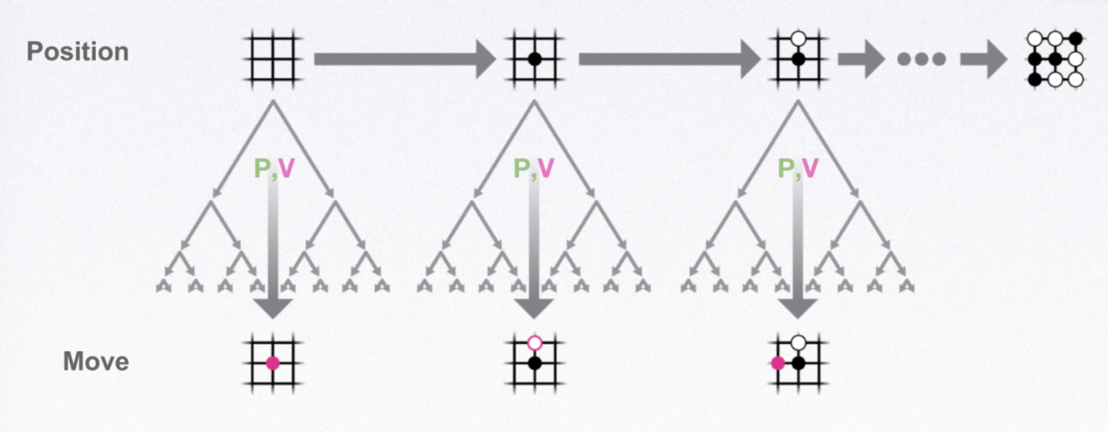

DeepMind

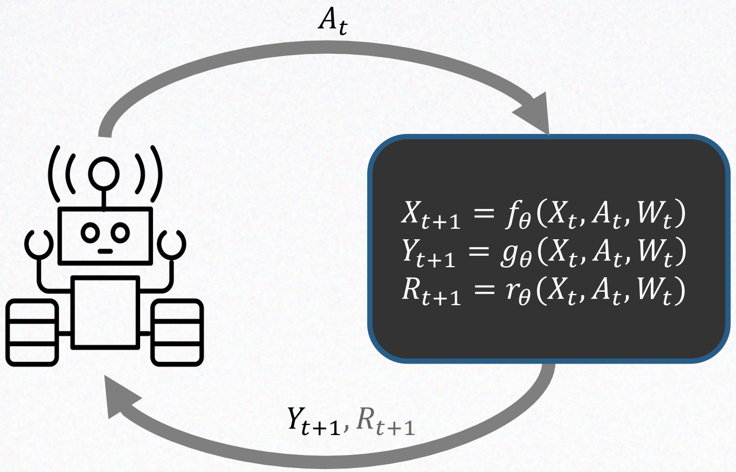
https://arxiv.org/pdf/1710.02298.pdf

https://arxiv.org/abs/1805.11593

# AlphaGo, AlphaGo Zero, AlphaZero

- $Y_t$= board position, turn, so $X_t = Y_t$
- $A_t$: what move
- $R_t$: 0 until end, when $R_t \in \{-1,0,1\}$
- Episodes: ~150 moves




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

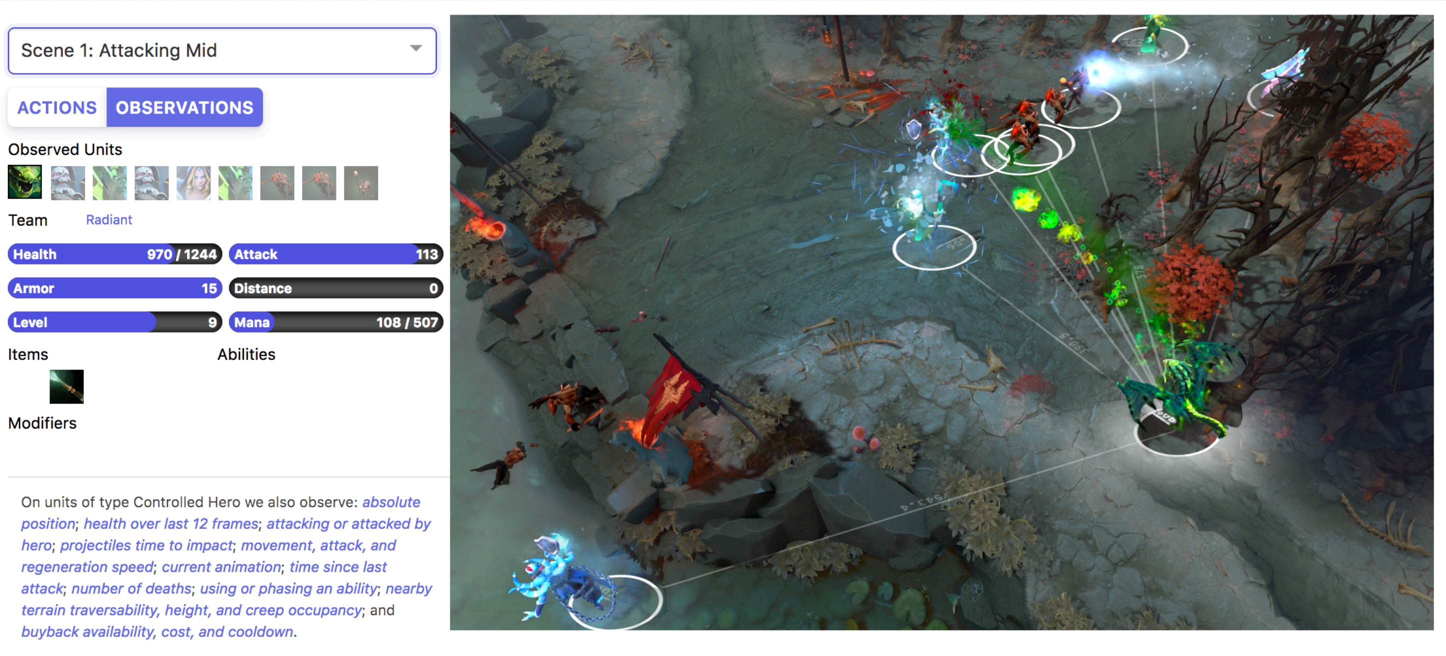
- Humbling experience for us, humans
- Power of neural nets, large compute

DeepMind

D. Silver et al., Nature (2016, 2017), arXiv (2017) [https://arxiv.org/abs/1712.01815](https://arxiv.org/abs/1712.01815) https://goo.gl/H8dKZ9

# OpenAI Dota-2

- $Y_t$: structured, 20000 dimensional $\neq X_t$
- $A_t$: structured, 8-dim
- $R_t$: shaped
- Episodes: 45 mins!





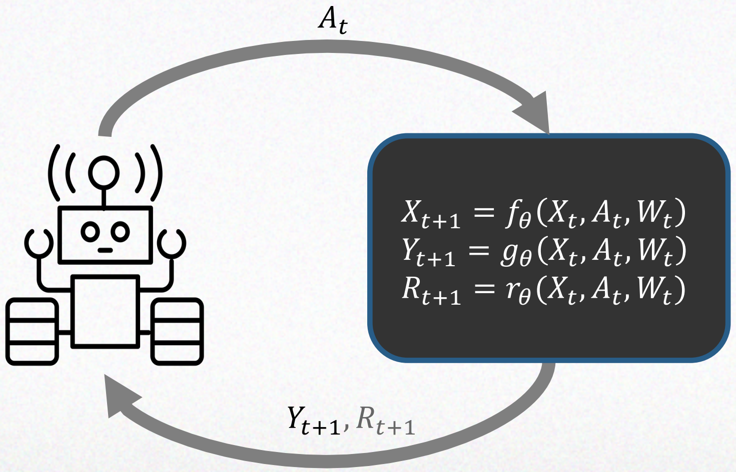
## Defeating amateur human teams in Dota-2

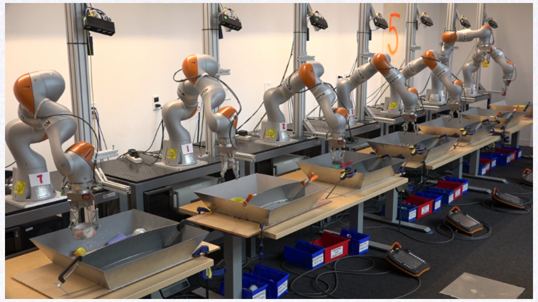
- Complexity, time horizon, $Y_t \neq X_t$
- Humans do care about this game ($40M annual prize pool)

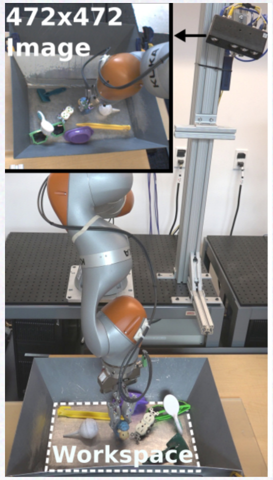
# Vision-based grasping

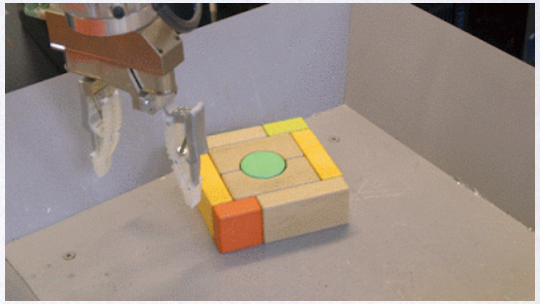
https://goo.gl/kTMcCb
Kalashnikov et al. (arXiv, 2018)

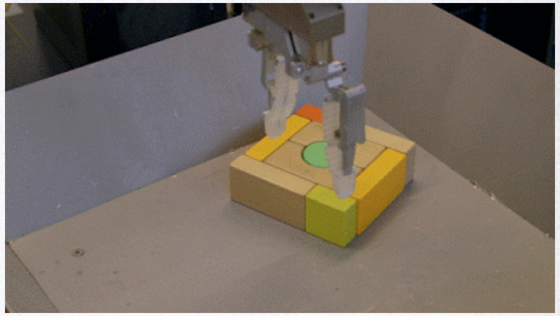
- $Y_t$: 472x472 RGB images, gripper state, height above ground, $Y_t \neq X_t$
- $A_t$: 3D gripper displacement, 2D rotation, gripper open/close, termination (7D)
- $R_t$: success or failure at the “end”, fixed cost per time step
- Episodes: 20 steps, learned







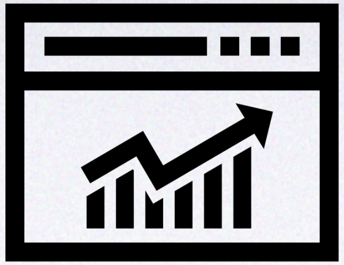
## Autonomous learning of vision-based grasping

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

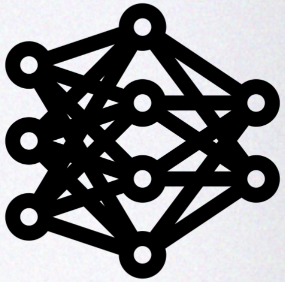
# When to use off-the-shelf ML/RL?

- 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

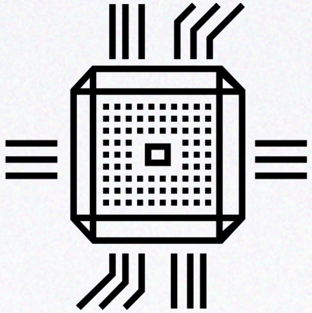
# Key enablers



- Reduce everything to (some form of) optimization; DP!

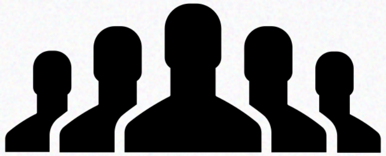


- Flexible models:
  - Deep neural networks, ReLu, LSTM, ConvNet, ..



- Large scale computation (GPU, TPU, Cloud, ..)

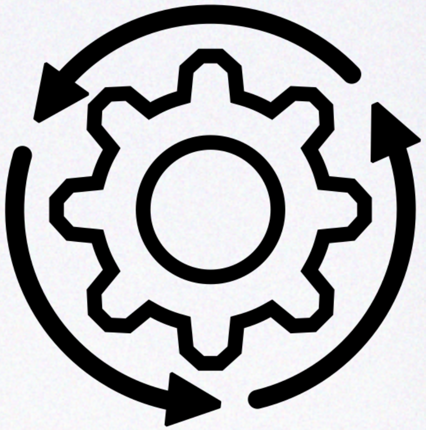
- Software frameworks, SGD!



- Rapidly growing, very active community
- Commercial interest, funding

# What works, what's hard?

- Works:
  - Lots of data, lots of compute, patience
  - Optimizing against a model we can simulate
  - Reactive agents with simple memory
- Hard:
  - Learning complex behaviors in the physical world
  - "Sim2real" problem
  - Combinatorics: big, multiscale worlds, large horizon with long-range dependencies (spatial, temporal, ..)
  - Learning from sparse reward, intelligent exploration
  - Learning and using models in an effective manner



# Core ideas in RL

# Core RL algorithms

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

How?

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

   $\Leftarrow$ all the methods in previous examples are based on this!

2. **Direct policy search**: $k^{\text{th}}$-order optimization, $0 \leq k \leq 2$
   - 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..

[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: $\forall(x,a) \in \mathcal{X}\times\mathcal{A}$:

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

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

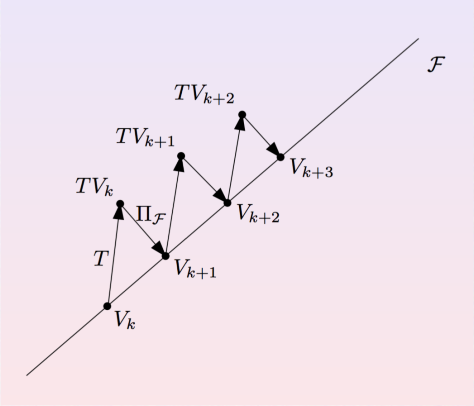
$$Q^* = TQ^*$$

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

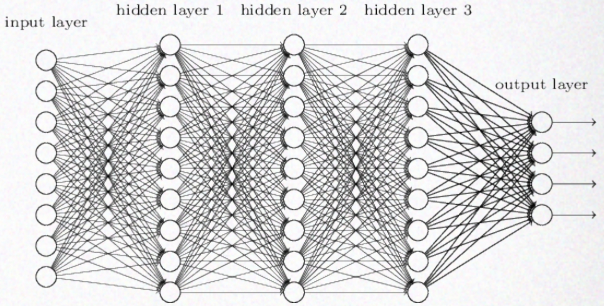
Richard E. Bellman
(1920-1984)

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

# Throw in that neural net!




- 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 $(X_i, A_i) \sim \mu$,
  $Y_i = r_\theta(X_i, A_i, W_i) + \gamma \max_{a'} Q(f_\theta(X_i, A_i, W_i), a')$
  $i = 1,2,\ldots,n$



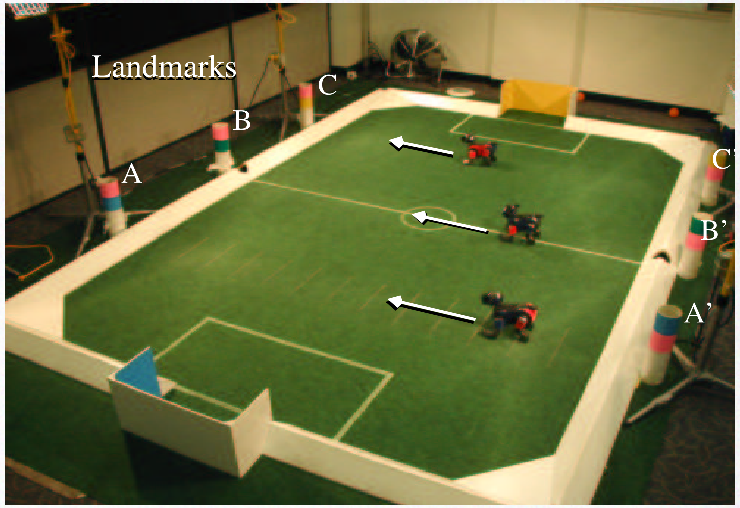
# Variations

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

# What did we cook?

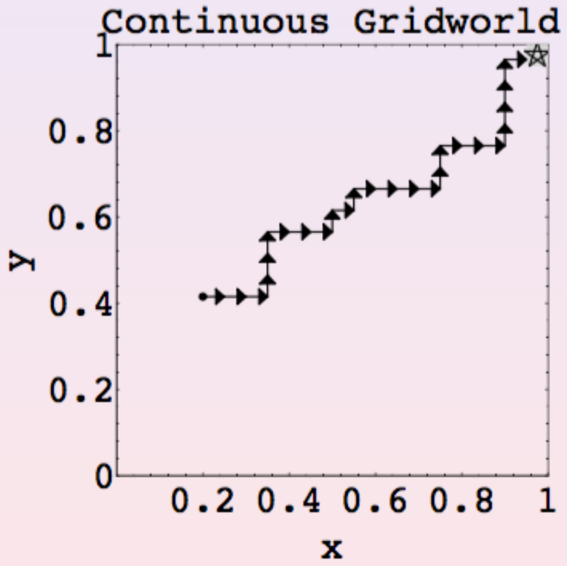
# Early successes

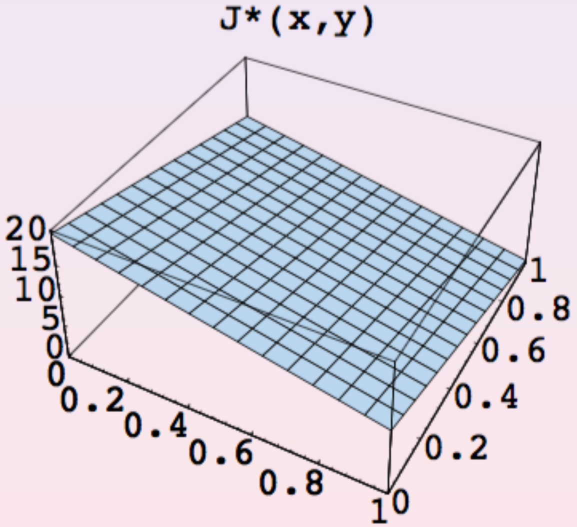
- TD-Gammon (Tesauro, 1992-95)
- Job-shop scheduling (Zhang & Dietterich, 1995)
- Dialogue management (Singh et al., 2002)
- Robocup (Kohl & Stone, 2004)
- Helicopter acrobatics (Ng et al., 2006)

# Clouds on the sky

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



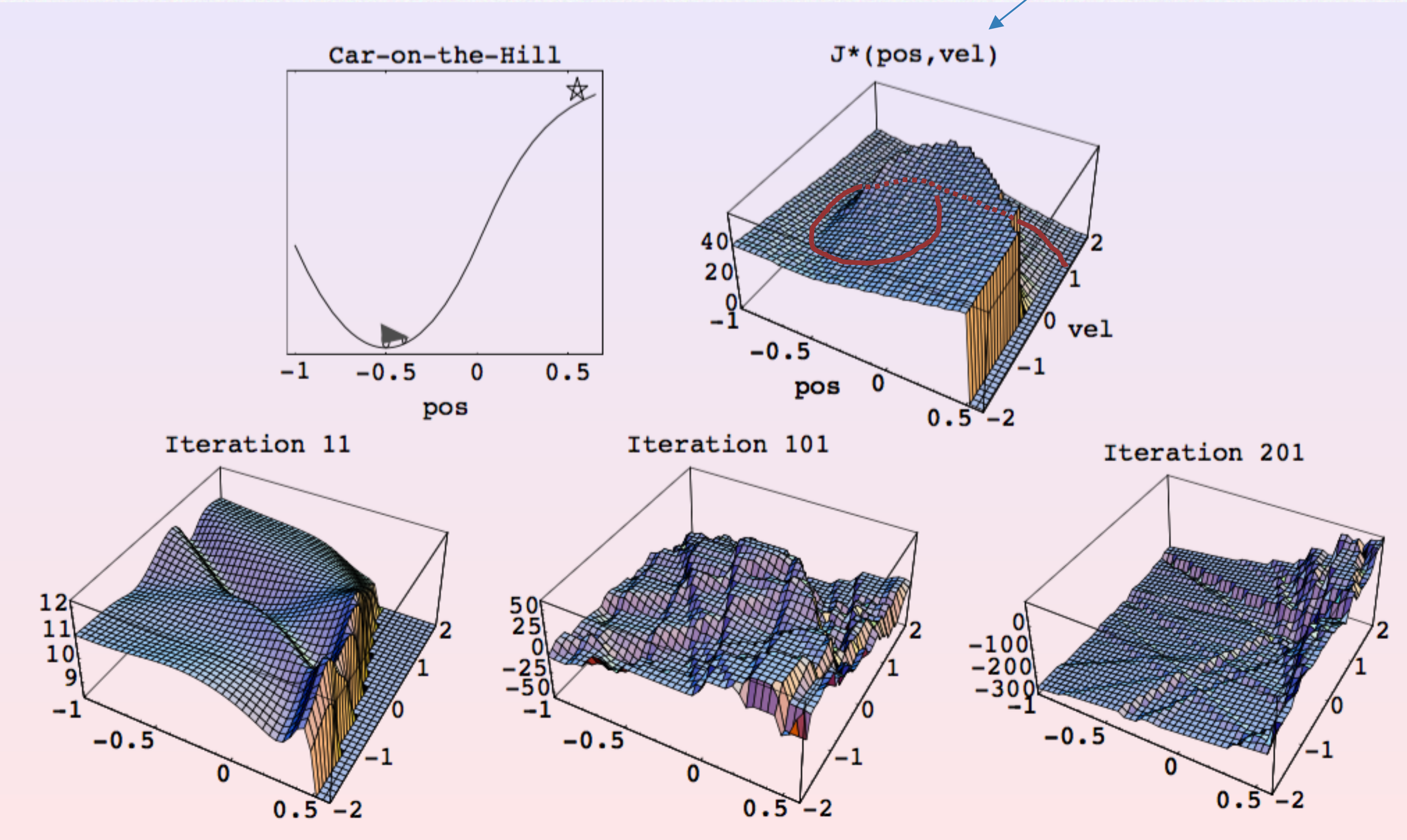




[1] With thanks to Justin Boyan

$\mu$ is the uniform distribution, quadratic polynomials used for value-function approximation

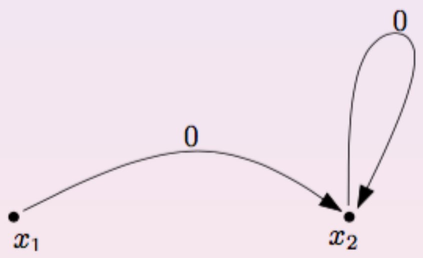
# ..add neural nets..

Optimal cost-to-go (-rewards)



# Disaster strikes

- 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:
$\mathcal{F} = \{\theta\phi \mid \theta \in \mathbb{R}\}$,

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

Iteration:

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

$\mu$ is the uniform distribution

# Poor outlook for ADP

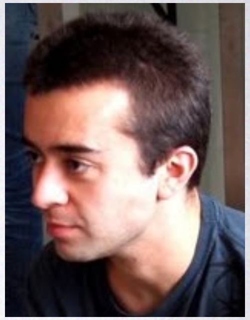
- *"In light of these experiments, we conclude that the straightforward combination of DP and function approximation is not robust."* (Boyan & Moore, NIPS-7, 1995)
- *Unfortunately, many popular functions approximators, such as neural nets and linear regression, do not fall in this[2] class (and in fact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

But how about TD-gammon, job-shop scheduling and other (early) successes???

# Explaining the bill


R. Munos


A.m. Farahmand


B.A. Pires

**Theorem** (Sz., Munos, 2005):

Covariate-shift price

Approximation error

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

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

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

Iteration cost

Estimation error

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

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


Uplifting András Antos

# Lesson: How to make ADP work?

Covariate shift, or off-policy problem

Need to control all terms!

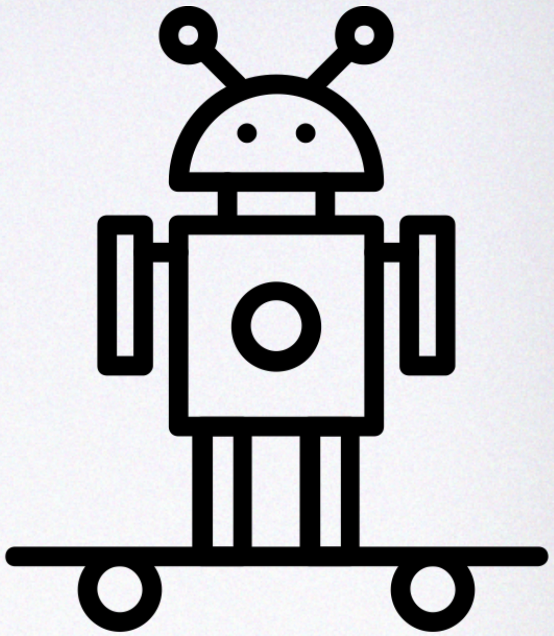
- $C(\rho, \mu)$: Sampling distr. $\mu$ should dominate $\rho \sum_{t=0}^{\infty} \gamma^t P_{\pi_K}^t$
  - Change $\mu$ as you go, change policies slowly, …
- Make approximation error $d(T\mathcal{F}, \mathcal{F})$ small:
  - Deep neural nets, LSTM, convnets, …
- Make sample size large to control estimation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

# How was this done?

| 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 |

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

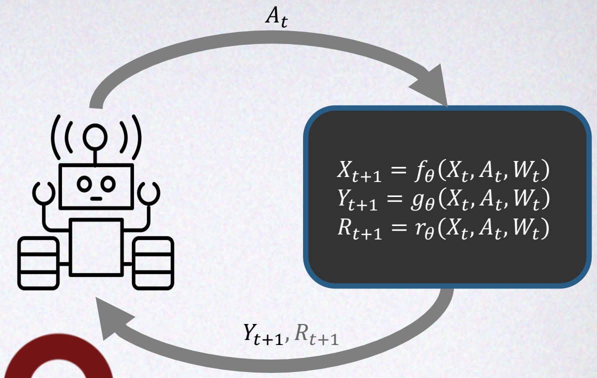
# Learning cheaply, online



- 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

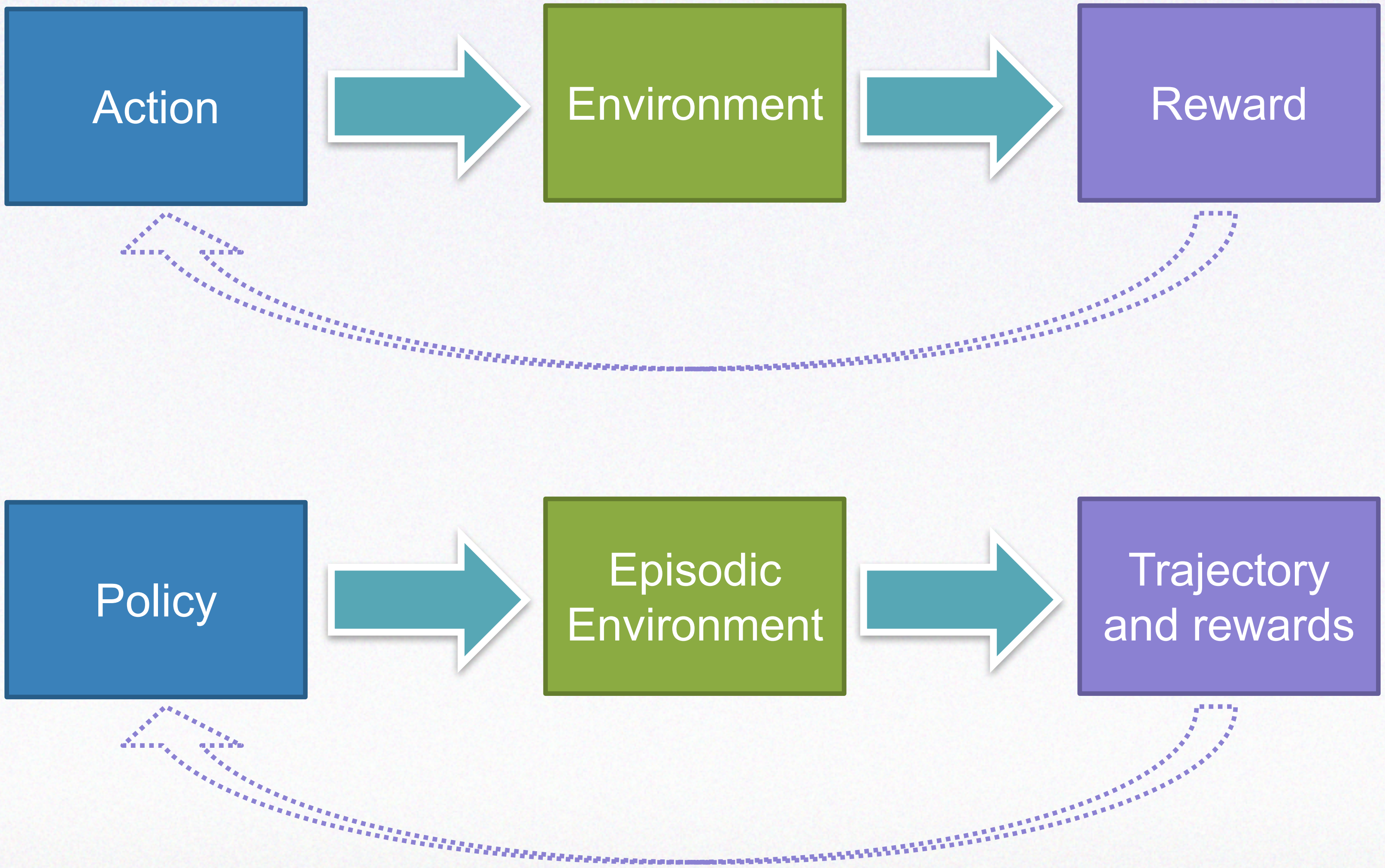



$\mathbb{P}$(payoff=1)=0.1 $\mathbb{P}$(payoff=1)=0.5 $\mathbb{P}$(payoff=1)=0.2

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

Regret = $n \max_a \mathbb{E}[r(a, W)] - \sum_{t=0}^{n-1} R_t = 0.5\, n \; - \sum_{t=0}^{n-1} R_t$

# Bandits vs. (episodic) MDPs



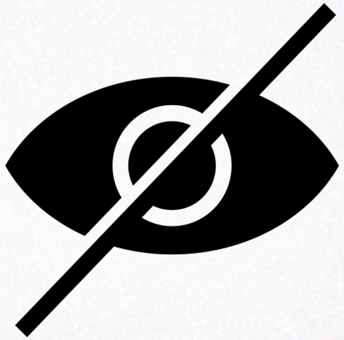
# $\epsilon$-greedy and friends

Action 1

Success = 6/10

Action 2

Success = 2/8



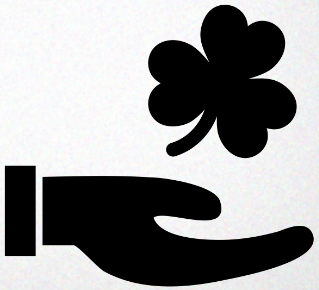
$\epsilon = 0.1$ greedy: Choose best looking action with probability $1 - \epsilon = 0.9$,
otherwise choose an action at random

# Optimism?

Success = 6/10


Chernoff's method: w.p. $1-\delta,\ \mu \le \hat{\mu}_t + \sqrt{\frac{\log(1/\delta)}{2t}}$

Choose $\delta = \frac{1}{n}$

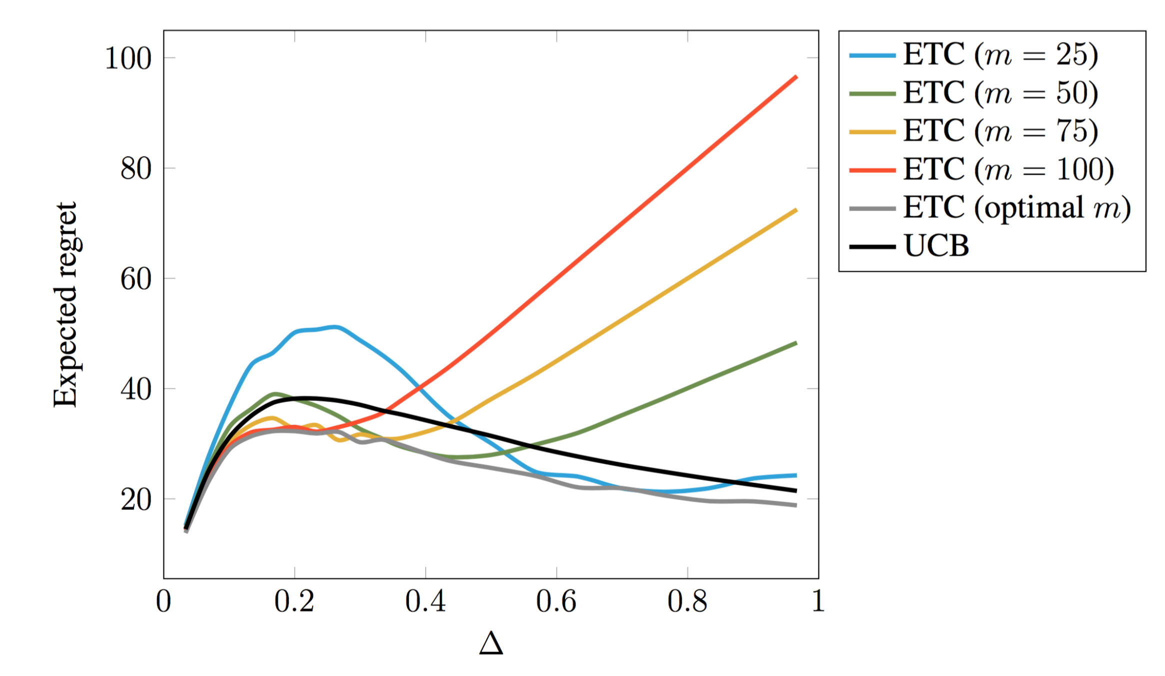


Actions tried fewer times will get a bigger boost: "optimism"

# Bandits on one slide

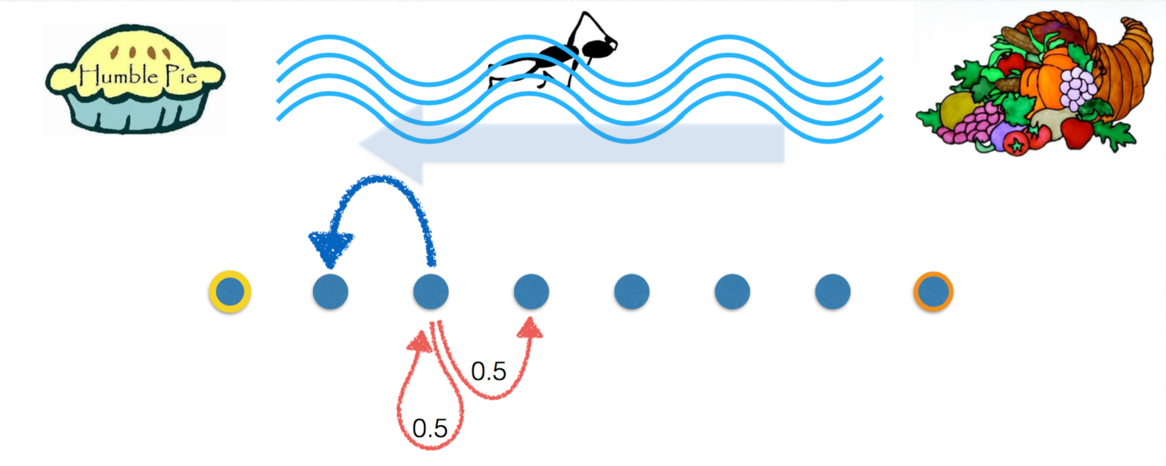
New book!
http://banditalgs.com

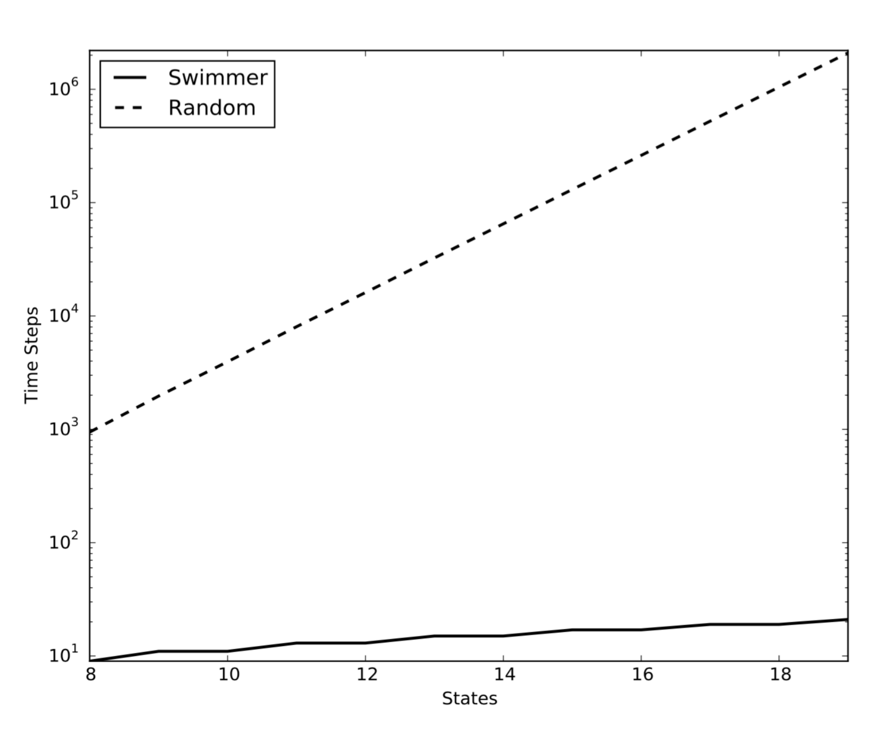
- Ad-hoc exploration: Good on some instances, bad on others
  - Explore-then-commit (ETC)
  - $\epsilon$-greedy, Boltzmann/Gibbs
- Planned exploration reaches optimal regret for all instances
  - UCB, posterior sampling a.k.a. Thompson sampling, …




2 arms, unit variance Gaussian rewards with means 0 and $-\Delta$, horizon 1000

# The challenge



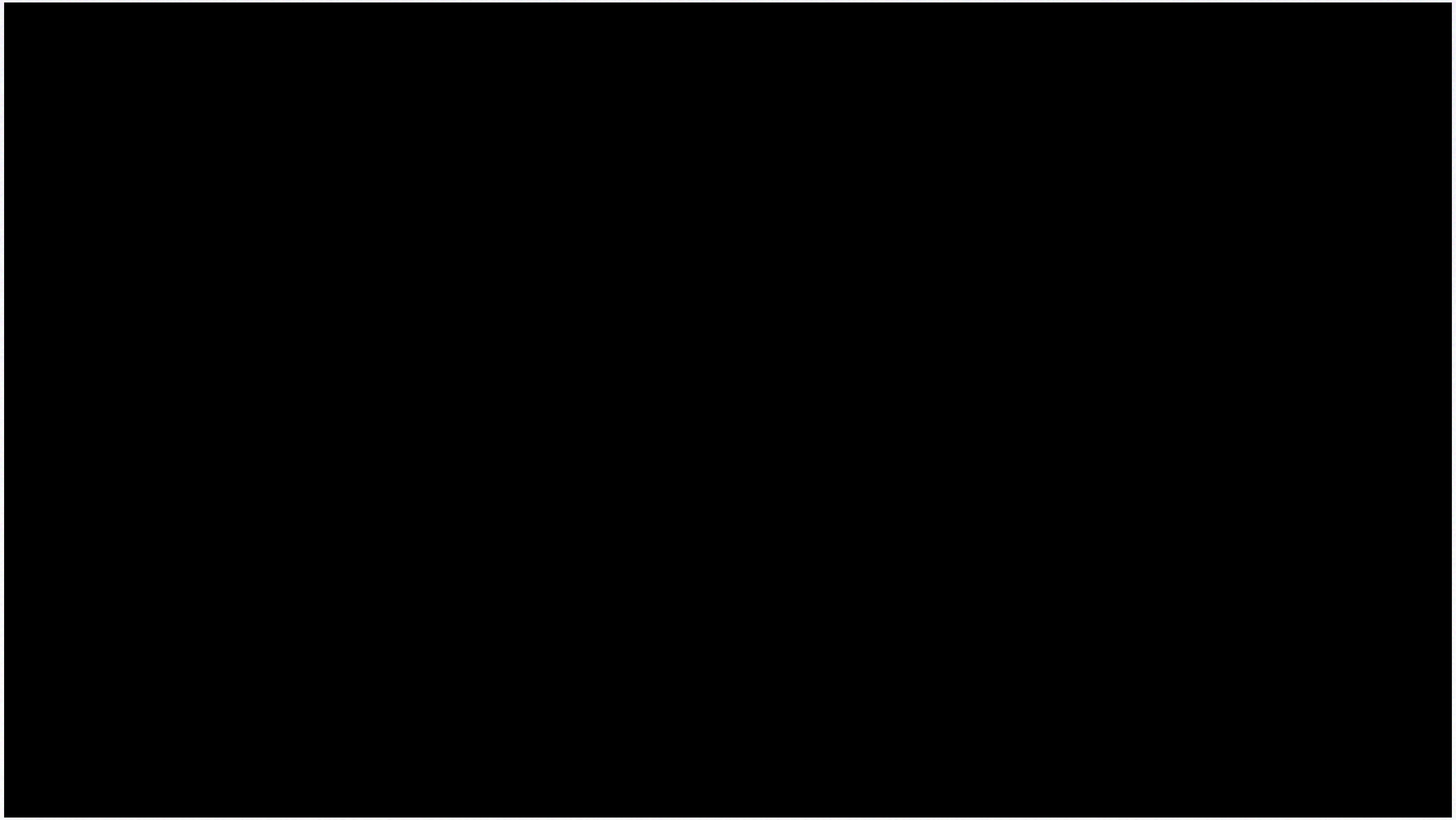




- First and last states are absorbing
- First state: small reward, last state: big reward
- Each state except the first and last have two actions
- Red action moves towards right, but is noisy
- Blue action moves towards left and is deterministic

# time steps before bounty found using random and “swimmer” policies

**Learner needs to plan to learn!**

# Beyond bandits



Video: courtesy of Ian Osband

[Jaksch-Ortner-Auer,'10]

# Exploration in finite MDPs

***S*** states, ***A*** actions, rewards in [0,1].

**<u>Definition:</u>** Diameter := maximum of best travel times between pairs of states. River swim: ***D = S***

- **<u>Theorem:</u>** The regret of an OFU learner satisfies

$$R_T = \tilde{O}(DS\sqrt{AT})$$

- **<u>Theorem:</u>** For any algorithm,

$$R_T = \Omega(\sqrt{DSAT})$$

# Optimism all the way?

Optimism is insufficient when an action can inform the learner about the reward of some other action

- Lattimore, Sz, “End of Optimism?” AISTATS’17
- Wu, György, Sz, ICML’15

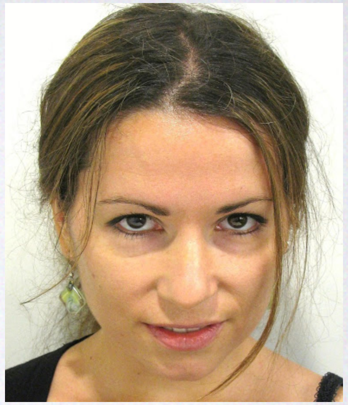
# Beyond finite MDPs

- 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)

Y. Abbasi-Yadkori



N. Lazic

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

Goal: minimize

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

$A, B$ 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 lots of data
    - Access to huge-scale compute

# Should we learn “everything”?

- 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’s 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 reward, intelligent exploration
    - Same problem as learning good models?

# Questions?