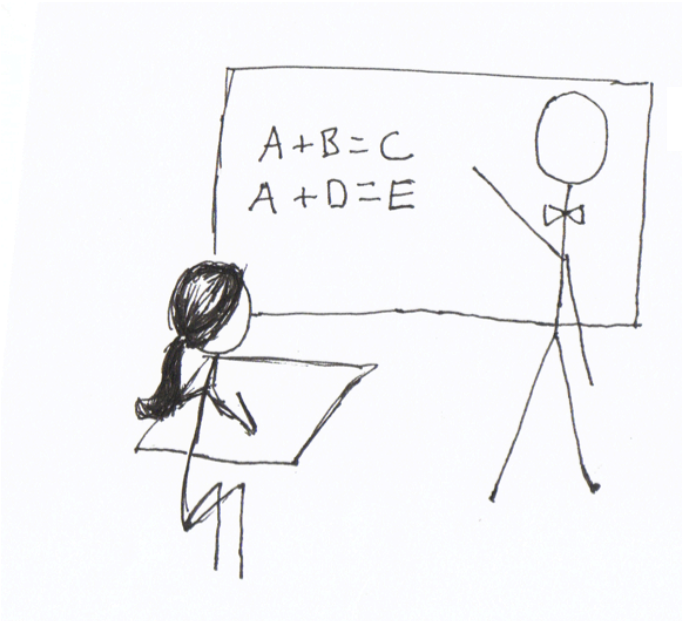
Day 9: Reinforcement Learning

Presenter: Auralee Edelen

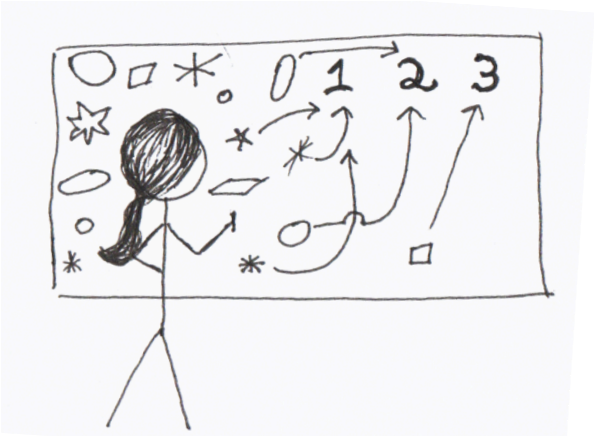
Day 9

Major Learning Paradigms



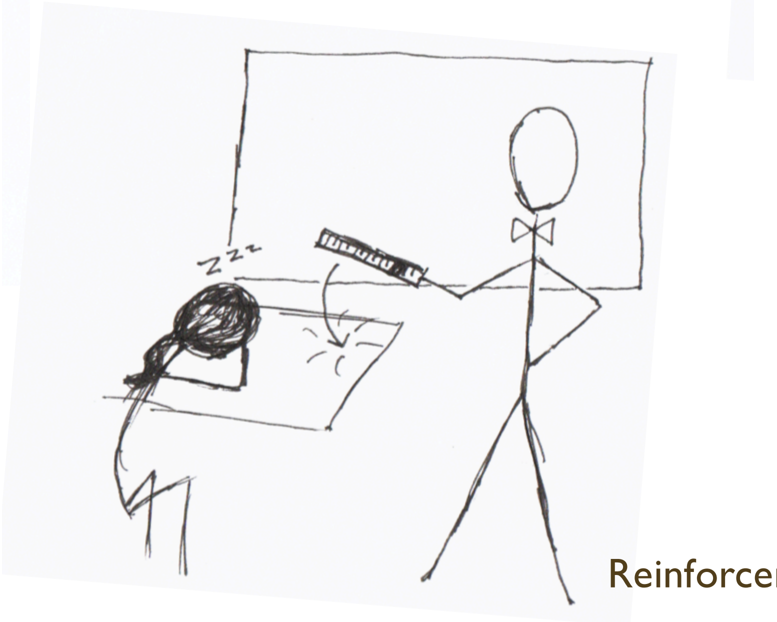
Supervised Learning

learn known input/output pairs



Unsupervised Learning

no labeled data → infer structure



Reinforcement Learning

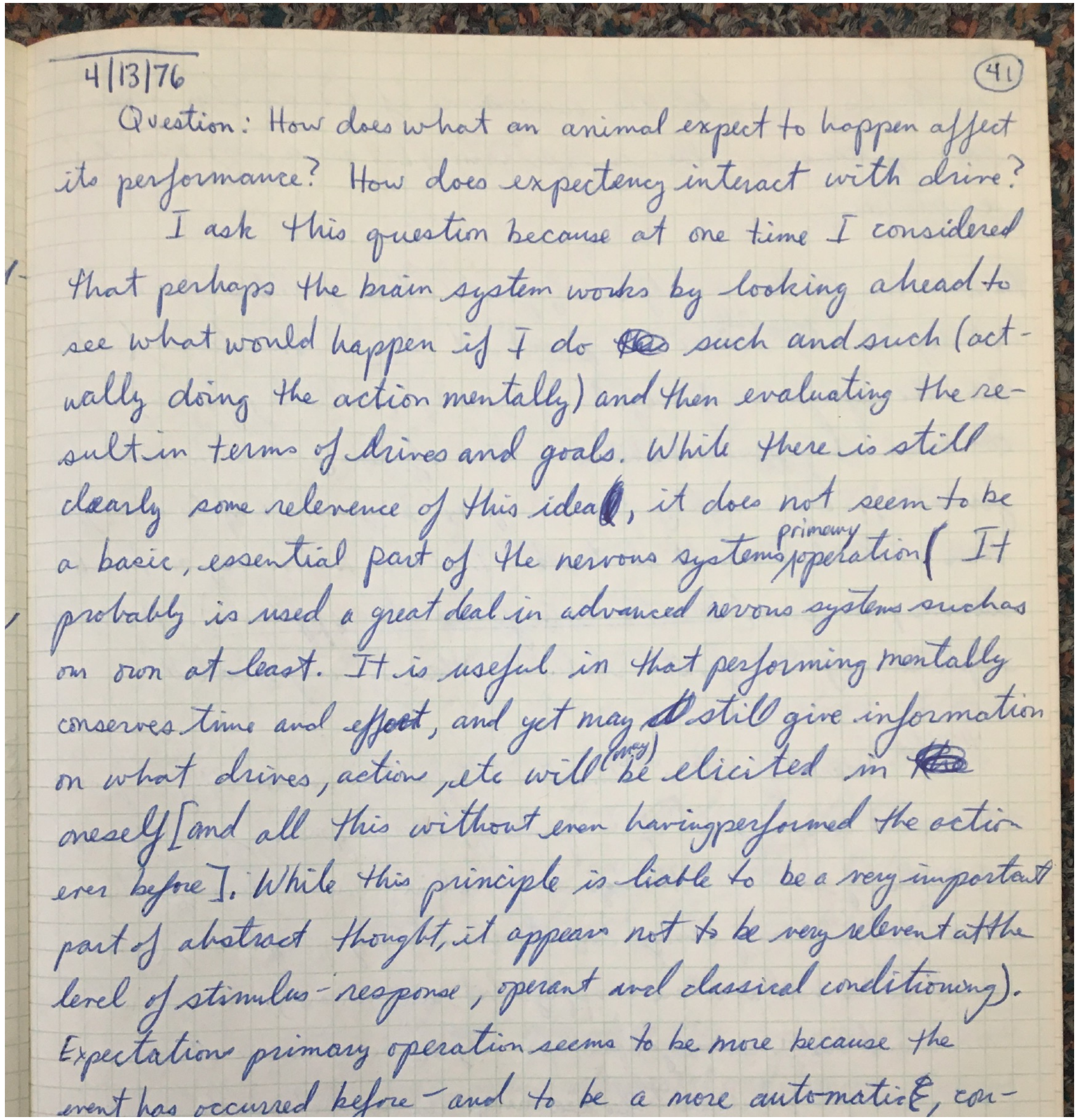
interact with the environment → adjust behavior based on reaction

Very Brief Reinforcement Learning History

- **Came out of trying to understand animal and human behavior, and in turn design systems capable of learning like animals and humans**

- Many parallels to **classical / optimal control** and Bayesian optimization *(developed in different communities)*

- Some major milestones in deep (i.e w/ NNs) RL:
  - 1992: TD-Gammon, human-level backgammon via self-play
  - 2013: Atari games, comparable to a human game tester; used deep Q-networks and CNNs (analyze state of the board)
  - 2015: AlphaGo beats Go champions; used initial supervised learning to imitate expert players; monte-carlo tree search with value network and policy network
  - 2017: AlphaZero beats Go champions without any human examples; counter-intuitive solutions studied

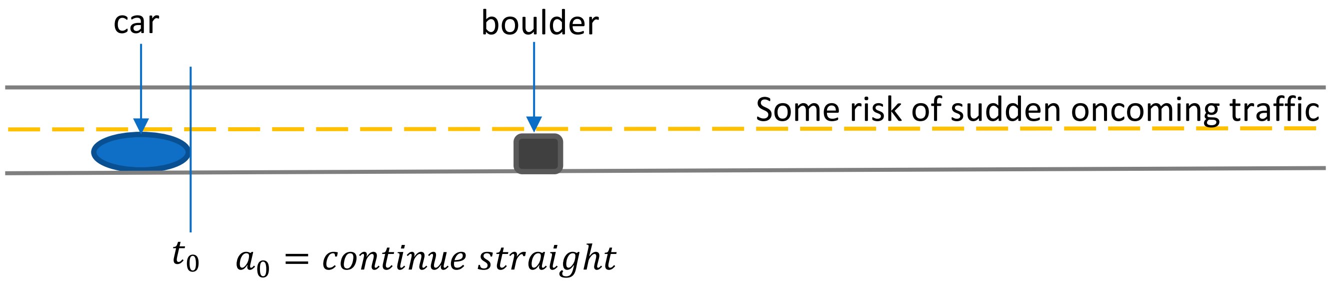

4/13/76 (41)

Question: How does what an animal expect to happen affect its performance? How does expectancy interact with drive?

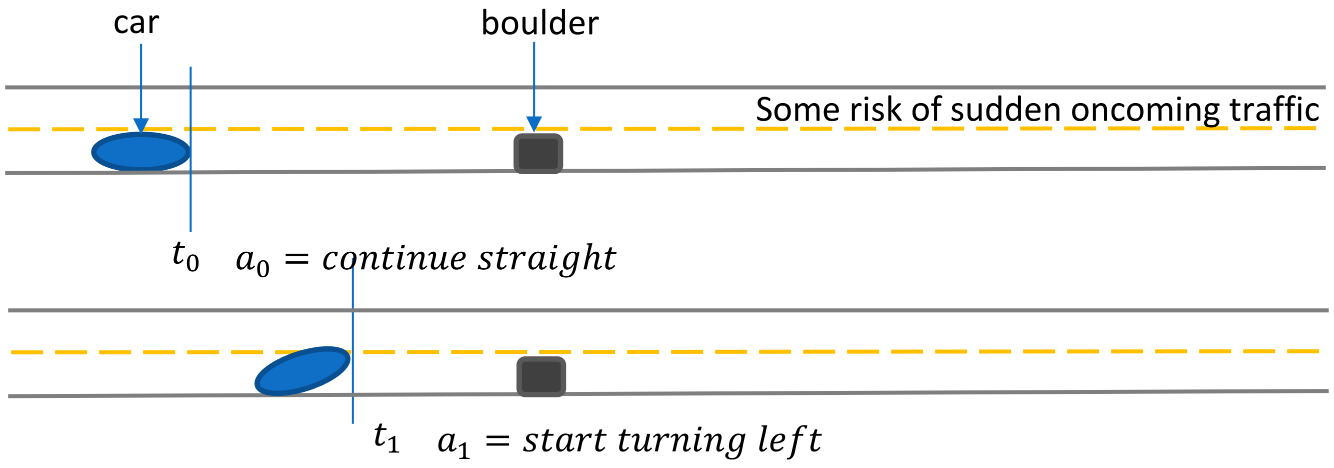
I ask this question because at one time I considered that perhaps the brain system works by looking ahead to see what would happen if I do such and such (actually doing the action mentally) and then evaluating the result in terms of drives and goals. While there is still clearly some relevence of this idea, it does not seem to be a basic, essential part of the nervous systems primary operation (It probably is used a great deal in advanced nervous systems such as our own at least. It is useful in that performing mentally conserves time and effort, and yet may still give information on what drives, action, etc will (may) be elicited in oneself [and all this without even having performed the action ever before]. While this principle is liable to be a very important part of abstract thought, it appears not to be very relevent at the level of stimulus-response, operant and classical conditioning).
Expectation primary operation seems to be more because the event has occurred before – and to be a more automatic, con-

example from a 1976 entry of Richard Sutton's research notebook

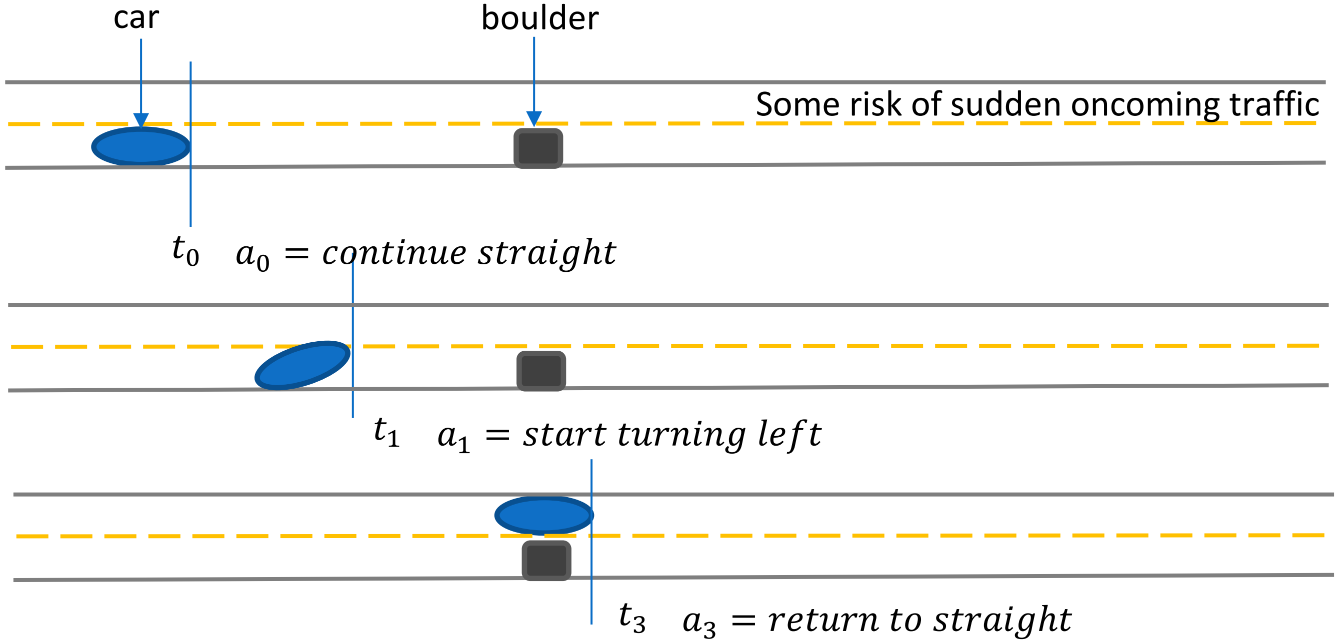
Example: Driving + Obstacle Avoidance



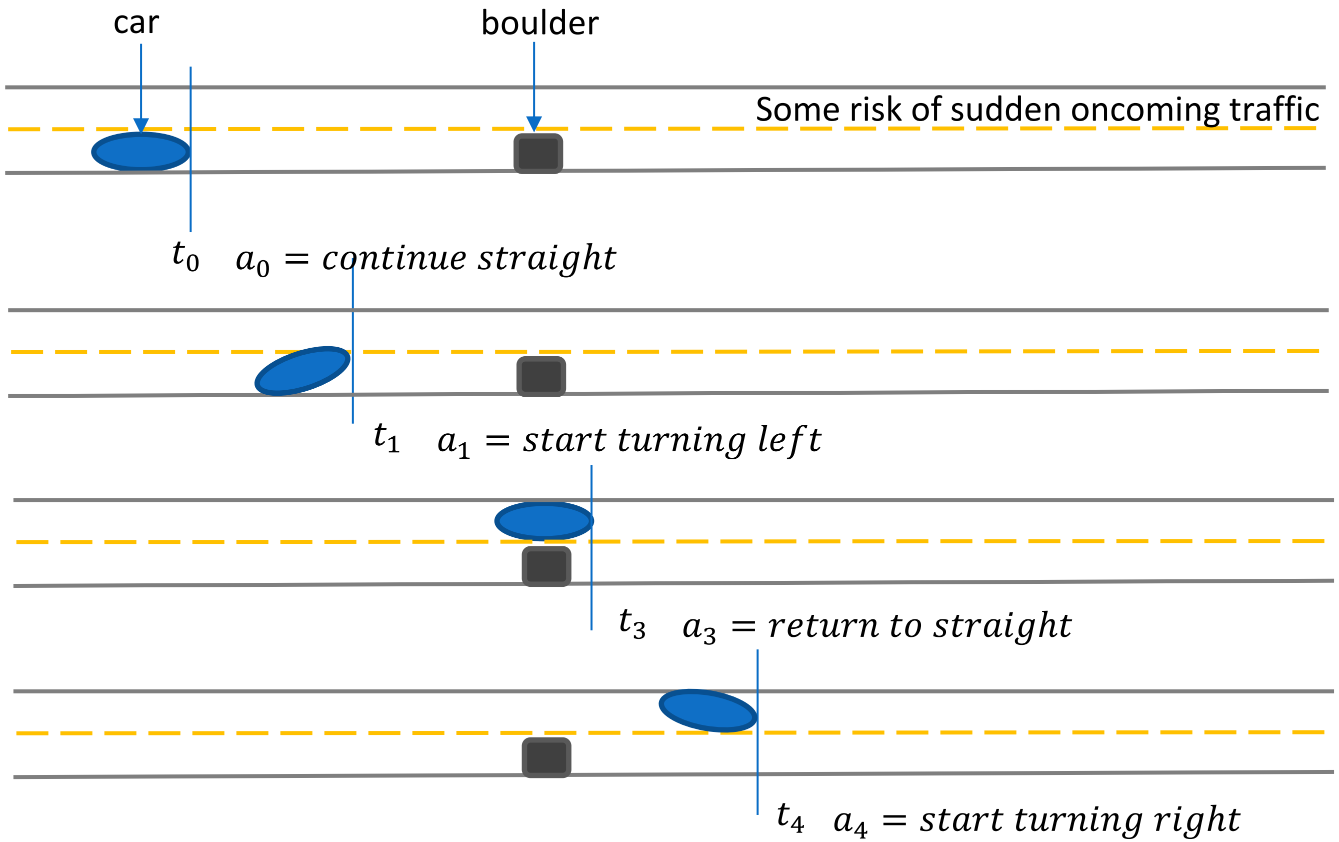
Example: Driving + Obstacle Avoidance



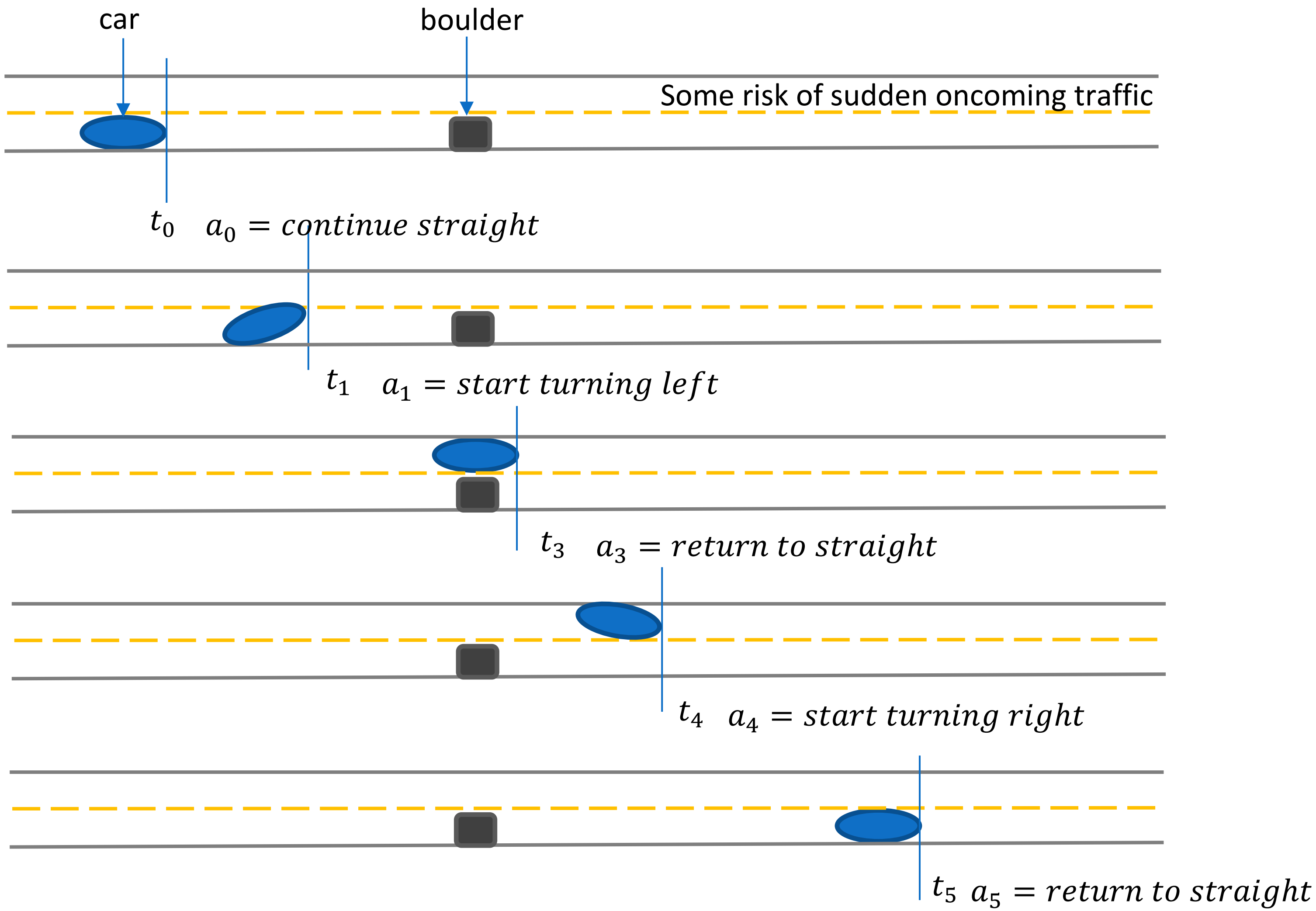
Example: Driving + Obstacle Avoidance



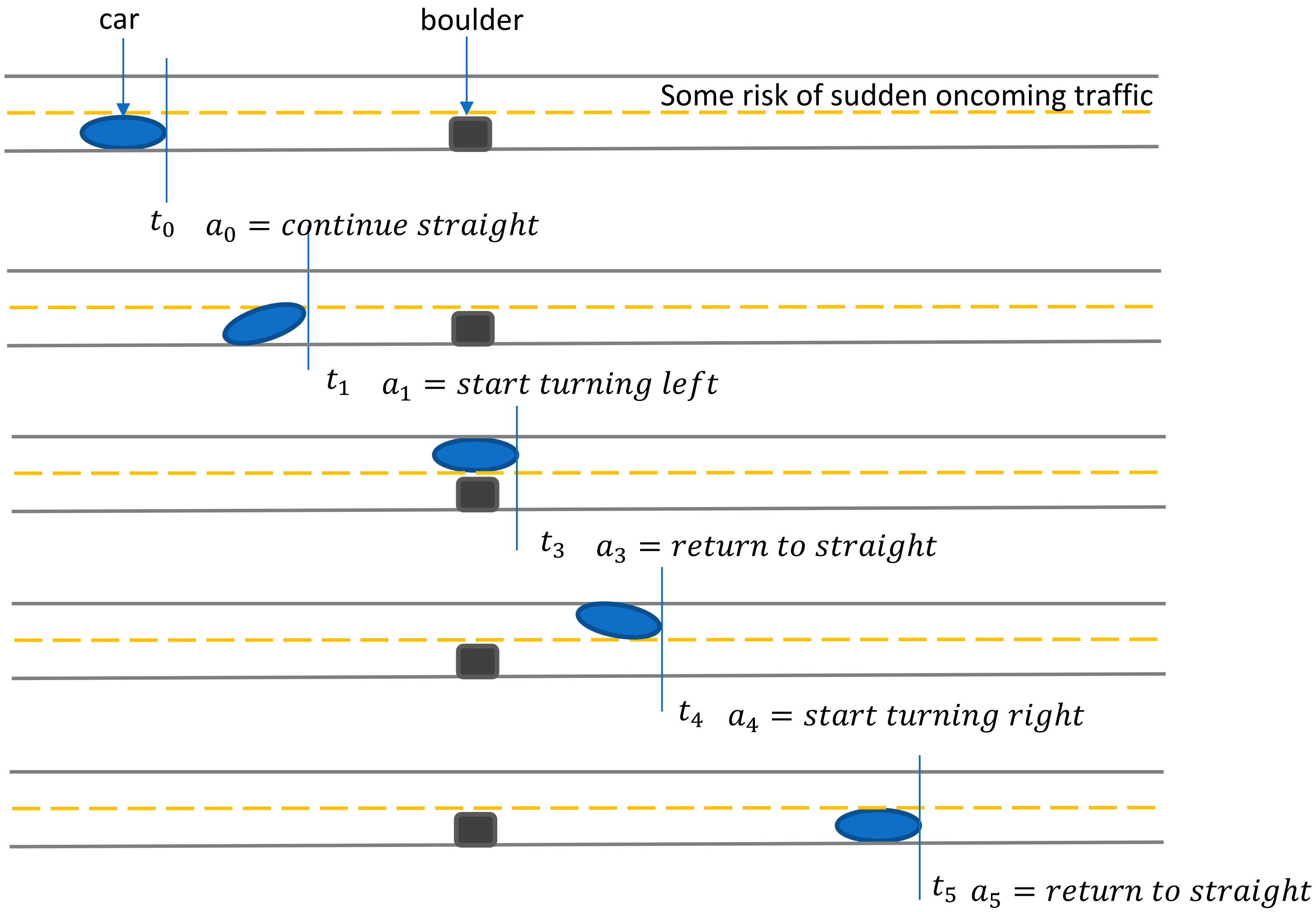
Example: Driving + Obstacle Avoidance



Example: Driving + Obstacle Avoidance



Example: Driving + Obstacle Avoidance



State

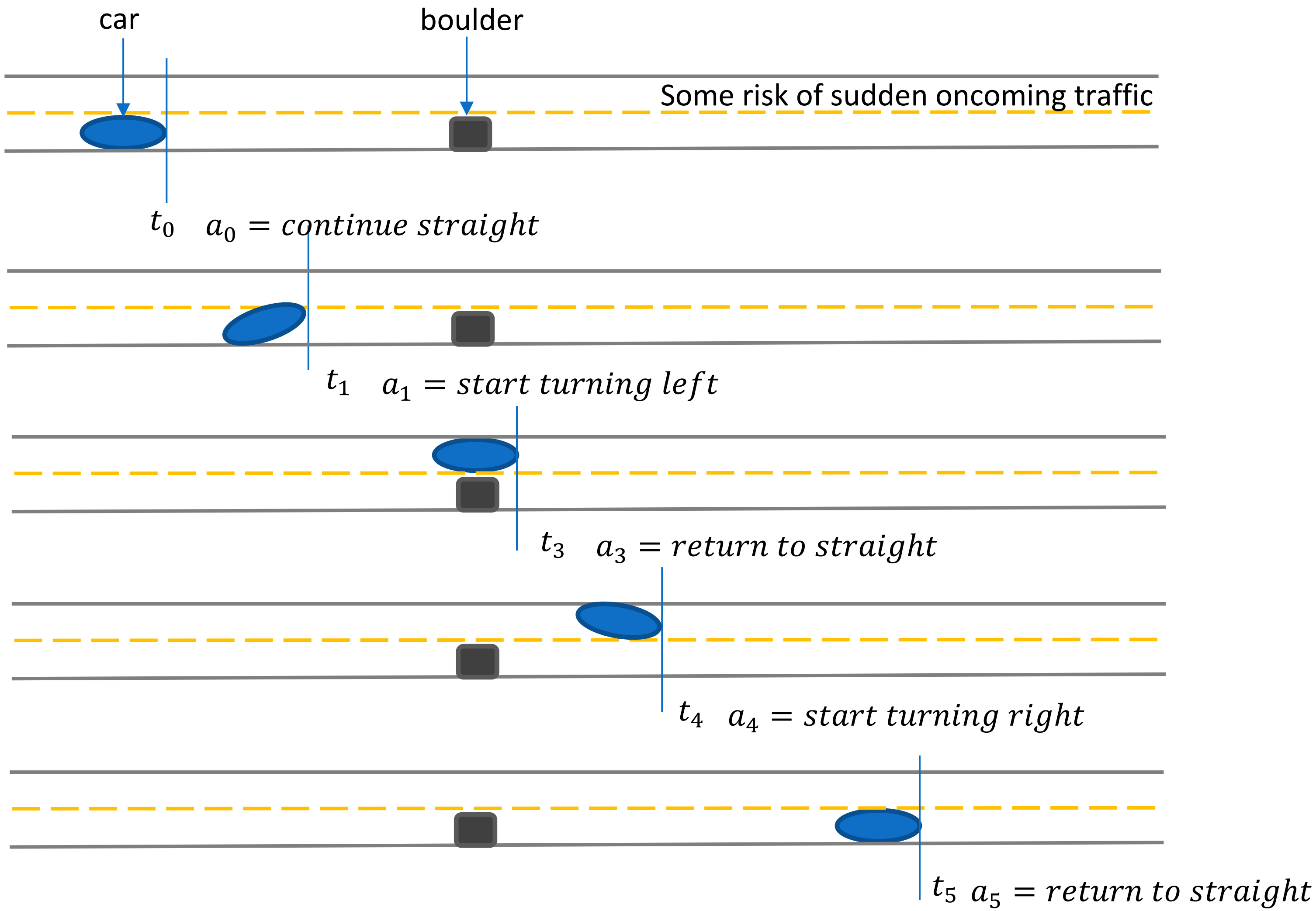
– view of surroundings

– current direction and speed

Example: Driving + Obstacle Avoidance

State
- view of surroundings
- current direction and speed

Actions
- turn the wheel left or right by n degrees
- depress gas or break by n degrees

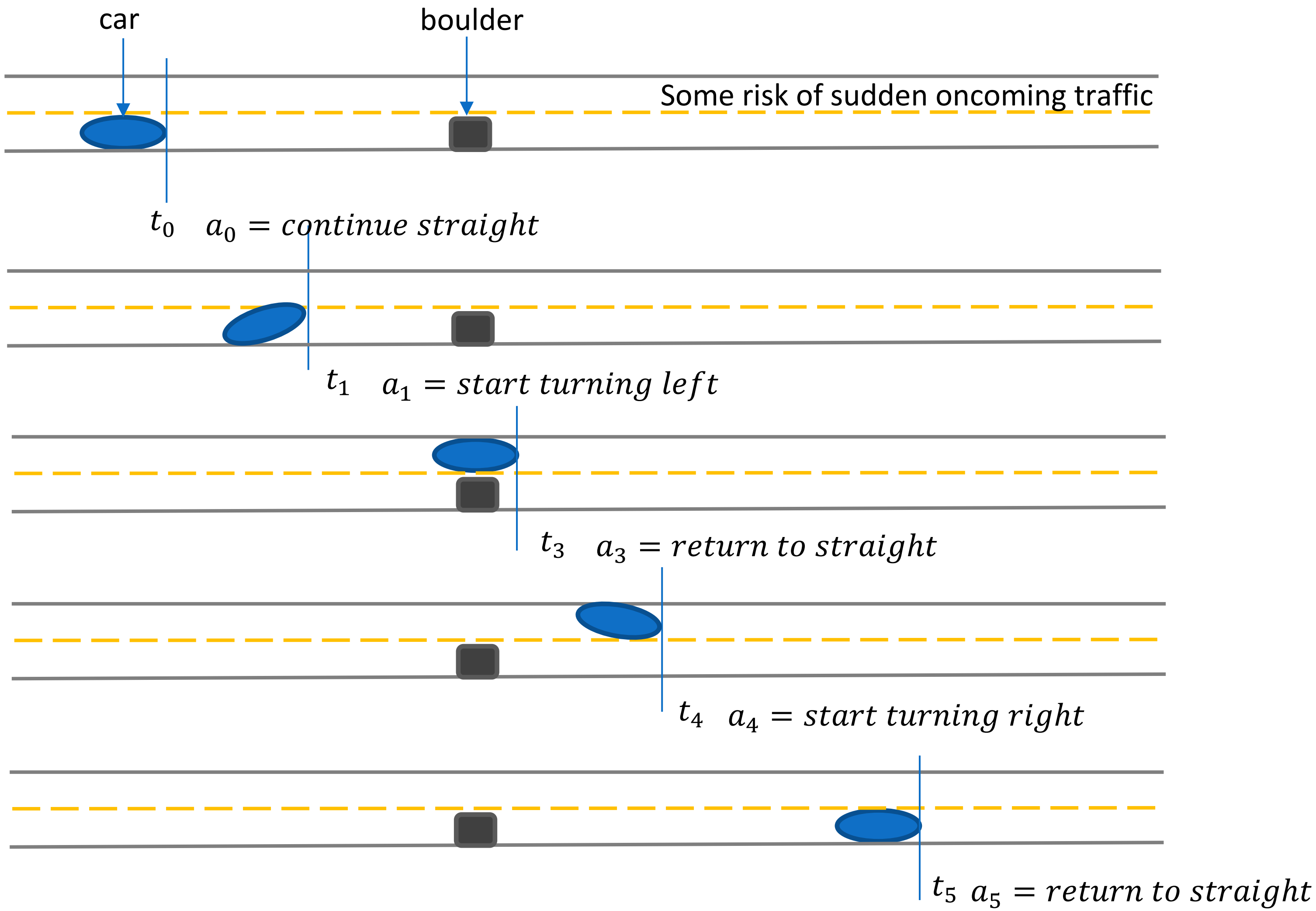


Example: Driving + Obstacle Avoidance

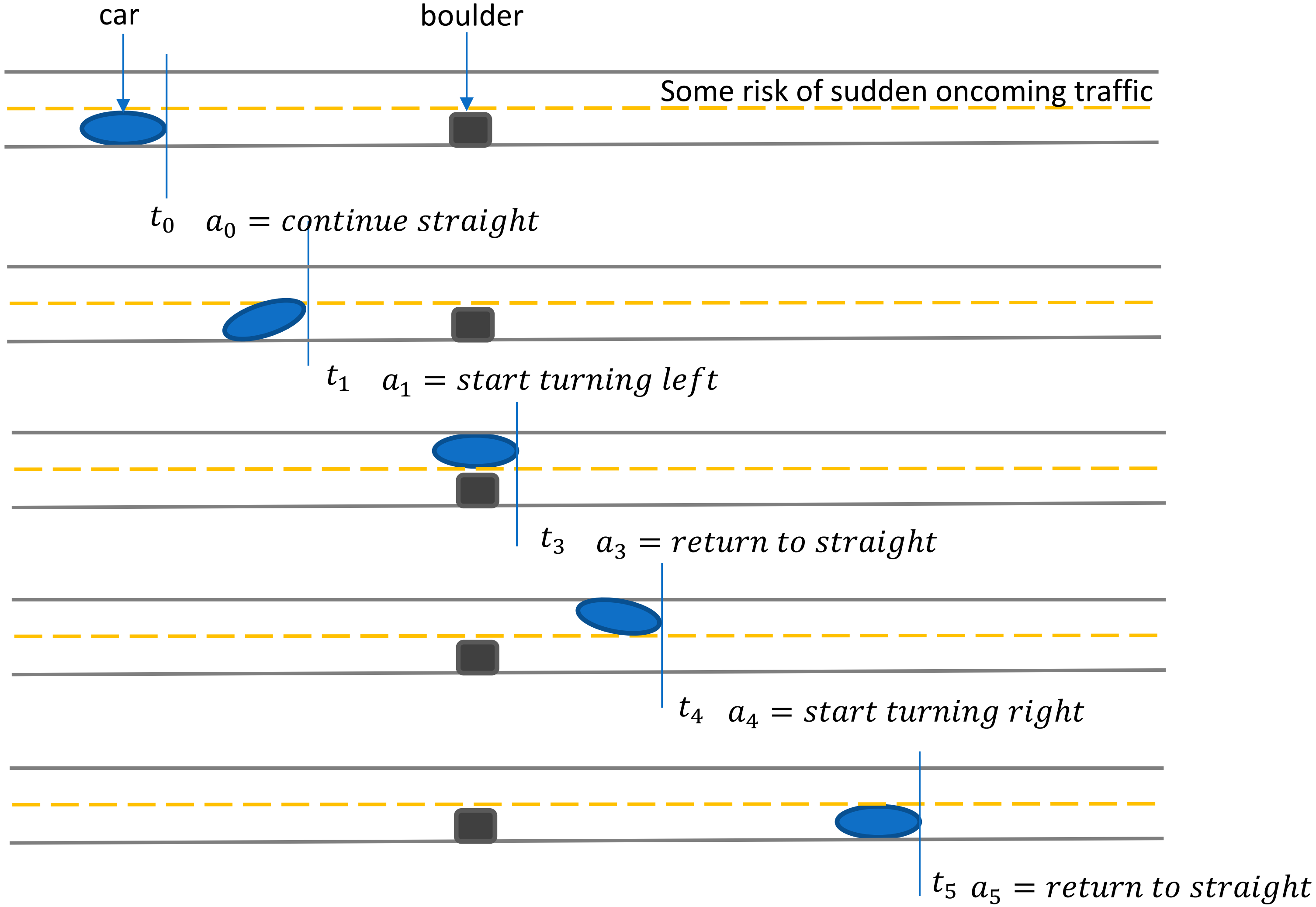
State
– view of surroundings
– current direction and speed

Actions
– turn the wheel left or right by n degrees
– depress gas or break by n degrees

Driving algorithm (a control policy)
– given observed state, decide on an action to take
– *looking forward in time + planning wrt impact of actions*



Example: Driving + Obstacle Avoidance



State
– view of surroundings
– current direction and speed

Actions
– turn the wheel left or right by n degrees
– depress gas or break by n degrees

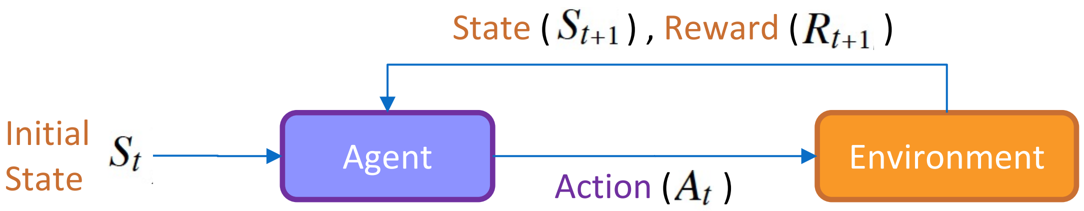
Driving algorithm (a control policy)
– given observed state, decide on an action to take
– *looking forward in time + planning wrt impact of actions*

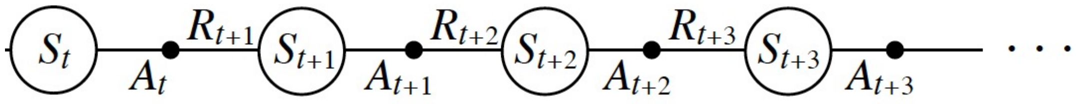
How did you learn to drive?

(1) watching and imitating others, (2) having an instructor watch and evaluate / correct your driving, (3) solo experience over time

→ There are analogies for each of these in the field of reinforcement learning

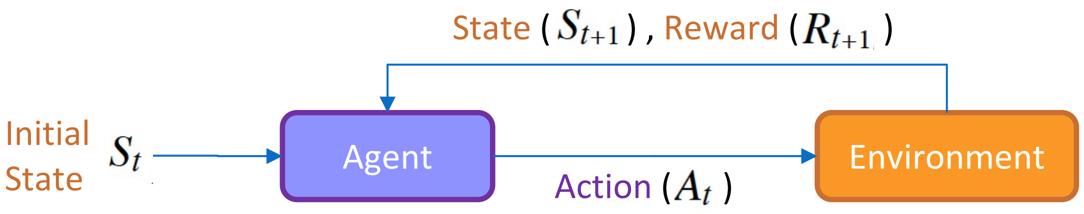
Basic Setup for an RL Problem

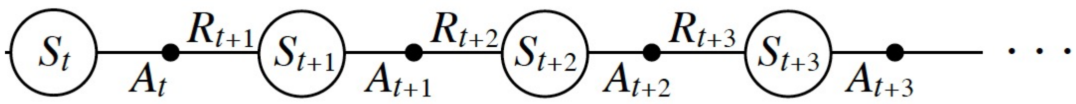




RL agent interacts with an **environment** over time → *goal is to maximize total returned reward*

Basic Setup for an RL Problem





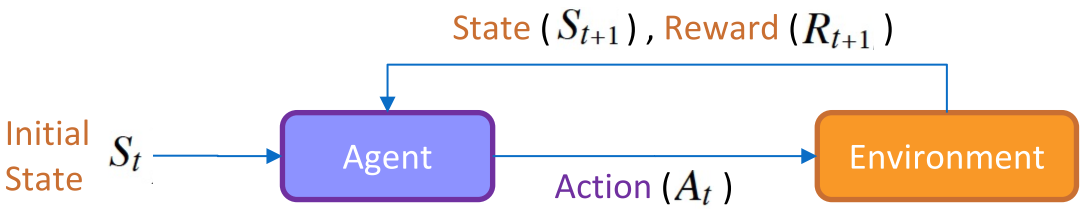
RL agent interacts with an **environment** over time → *goal is to maximize total returned reward*

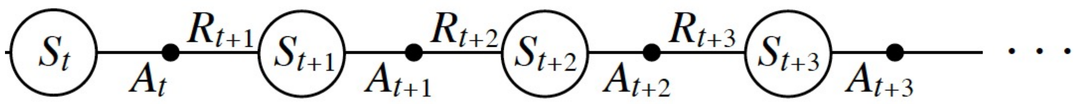
State – system information at present time

Action – a change the agent can make to the environment

Reward – scalar return from the environment at present time

Basic Setup for an RL Problem





RL agent interacts with an **environment** over time → *goal is to maximize total returned reward*

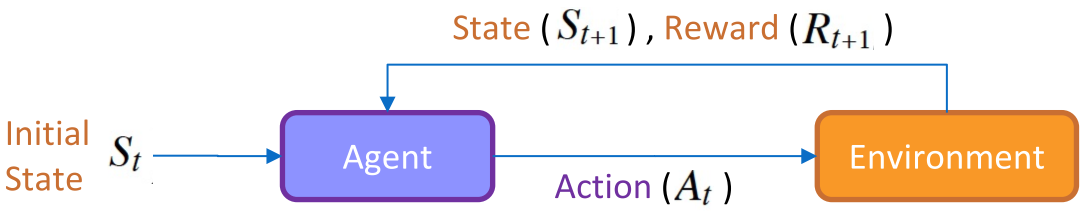
State – system information at present time

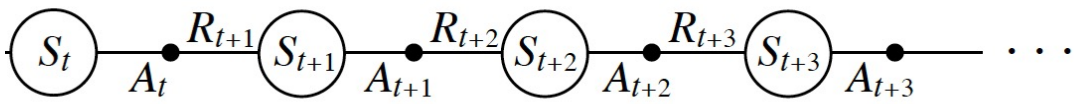
Action – a change the agent can make to the environment

Reward – scalar return from the environment at present time

Episode – sequences of (state1 → action1 → state2 + reward2); ends on some terminal condition

Basic Setup for an RL Problem





RL agent interacts with an **environment** over time → *goal is to maximize total returned reward*

State – system information at present time

Action – a change the agent can make to the environment

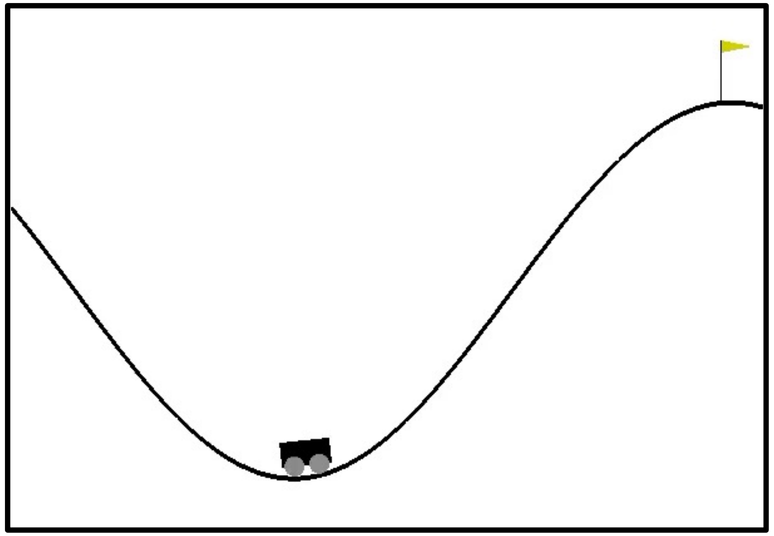
Reward – scalar return from the environment at present time

Episode – sequences of (state1 → action1 → state2 + reward2); ends on some terminal condition

Agent acts according to a **policy** (π) – determines actions to take based on observed state

Illustrative Example: Mountain Car Problem

Goal is to get to top of hill, but in an under-powered car!



Illustrative Example: Mountain Car Problem

Goal is to get to top of hill, but in an under-powered car!

State:

- car position and velocity
- bounded [(-1.2, 0.6), (-0.07, 0.07)]
- initialized randomly at [(-0.6, -0.4), 0]

Action:

- accelerate left [-1]
- accelerate right [+1]
- don't accelerate [0]

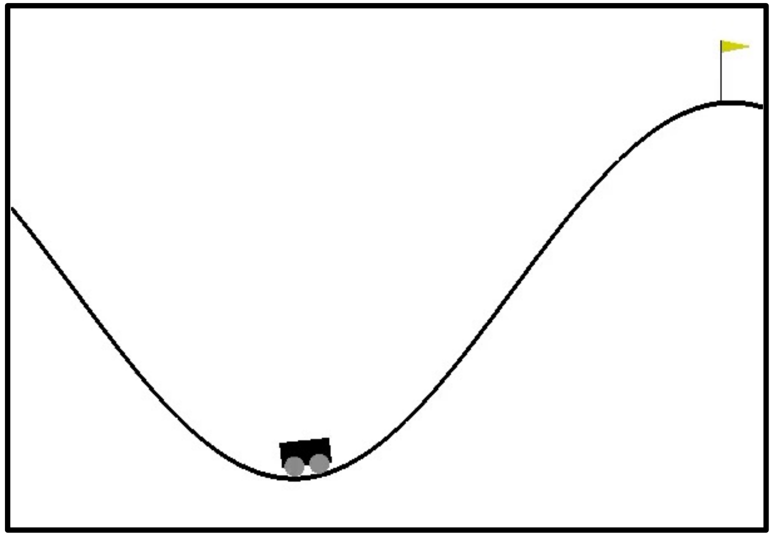
Reward:

- 0 if reach the top (position = 0.05)
- -1 if position is < 0.5

Episode:

- Ends if position > 0.5 or episode length > 200

$$Velocity = Velocity + Action * 0.001 + \cos(3 * position) * (-0.0025)$$
$$Position = Position + Velocity$$



Problem formulated in: A Moore, Efficient Memory-Based Learning for Robot Control, PhD thesis, University of Cambridge, 1990.

Illustrative Example: Mountain Car Problem

Goal is to get to top of hill, but in an under-powered car!

State:

- car position and velocity
- bounded [(-1.2, 0.6), (-0.07, 0.07)]
- initialized randomly at [(-0.6, -0.4), 0]

Action:

- accelerate left [-1]
- accelerate right [+1]
- don't accelerate [0]

Reward:

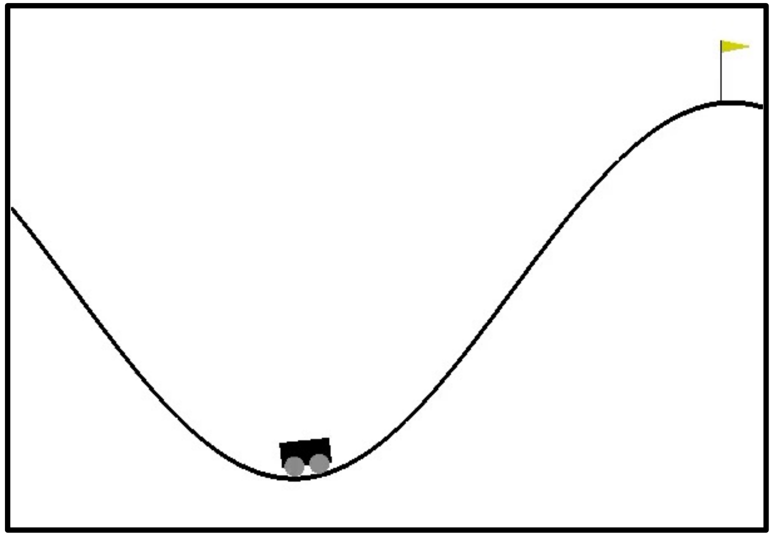
- 0 if reach the top (position = 0.05)
- -1 if position is < 0.5

Episode:

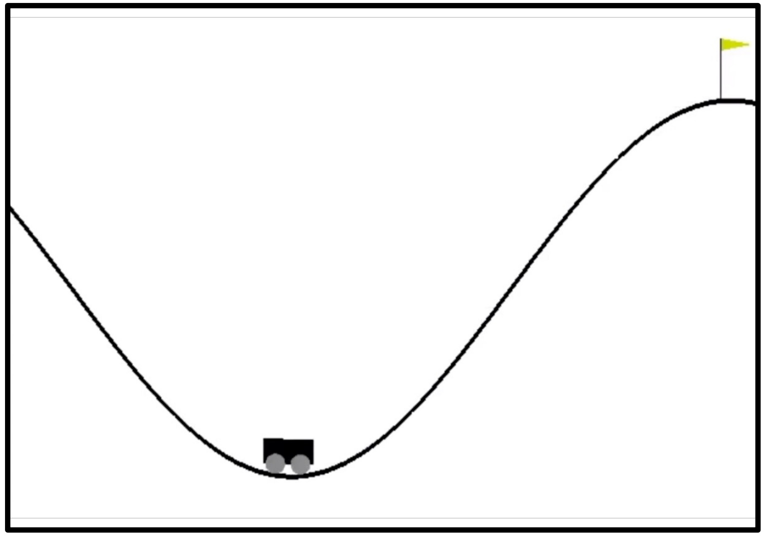
- Ends if position > 0.5 or episode length > 200

$$Velocity = Velocity + Action * 0.001 + \cos(3 * position) * (-0.0025)$$
$$Position = Position + Velocity$$

Before Training



After Training



Problem formulated in: A Moore, Efficient Memory-Based Learning for Robot Control, PhD thesis, University of Cambridge, 1990.

Illustrative Example: Mountain Car Problem

Goal is to get to top of hill, but in an under-powered car!

State:

- car position and velocity
- bounded [(-1.2, 0.6), (-0.07, 0.07)]
- initialized randomly at [(-0.6, -0.4), 0]

Action:

- accelerate left [-1]
- accelerate right [+1]
- don’t accelerate [0]

Reward:

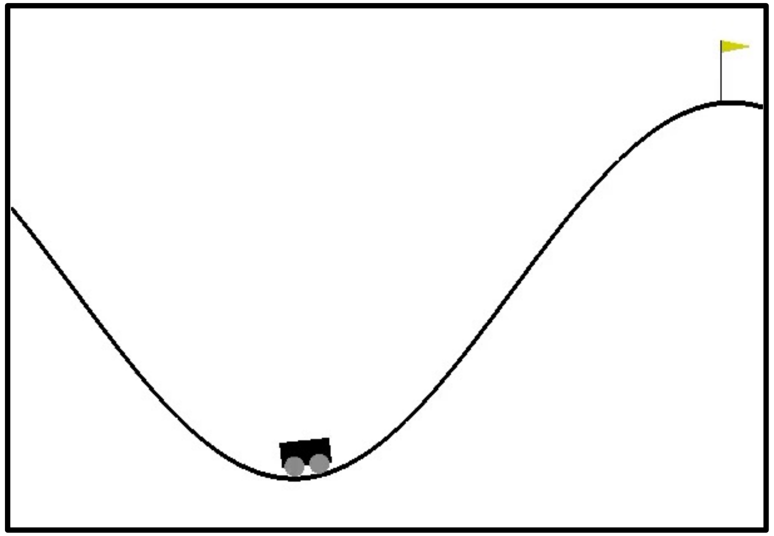
- 0 if reach the top (position = 0.05)
- -1 if position is < 0.5

Episode:

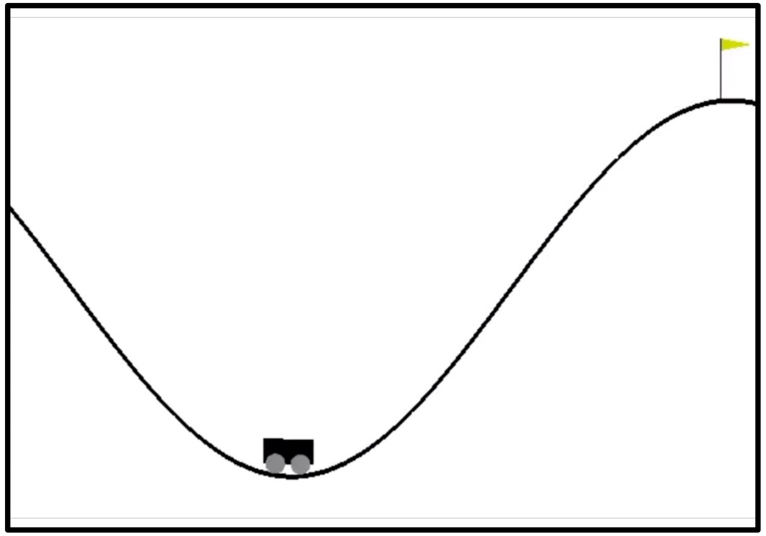
- Ends if position > 0.5 or episode length > 200

$$Velocity = Velocity + Action * 0.001 + \cos(3 * position) * (-0.0025)$$
$$Position = Position + Velocity$$

Before Training



After Training



Good example regarding **exploration vs. exploitation**
→ *need to explore seemingly sub-optimal actions to discover how to get enough momentum*

Problem formulated in: A Moore, Efficient Memory-Based Learning for Robot Control, PhD thesis, University of Cambridge, 1990.

Illustrative Example: Mountain Car Problem

Goal is to get to top of hill, but in an under-powered car!

State:

- car position and velocity
- bounded [(-1.2, 0.6), (-0.07, 0.07)]
- initialized randomly at [(-0.6, -0.4), 0]

Action:

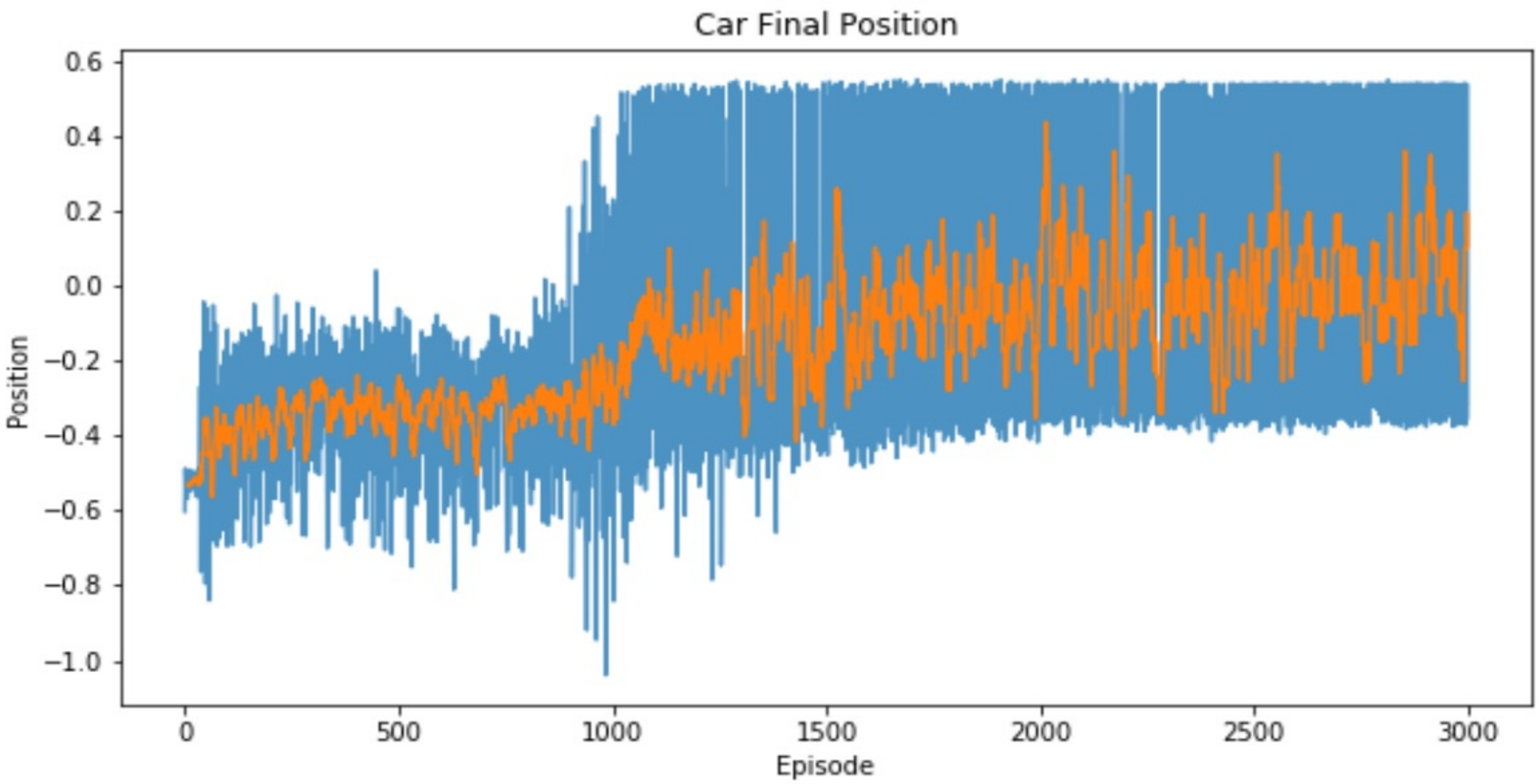
- accelerate left [-1]
- accelerate right [+1]
- don’t accelerate [0]

Reward:

- 0 if reach the top (position = 0.05)
- -1 if position is < 0.5

Episode:

- Ends if position > 0.5 or episode length > 200



Good example regarding **exploration vs. exploitation**

→ *need to explore seemingly sub-optimal actions to discover how to get enough momentum*

Problem formulated in: A Moore, Efficient Memory-Based Learning for Robot Control, PhD thesis, University of Cambridge, 1990.

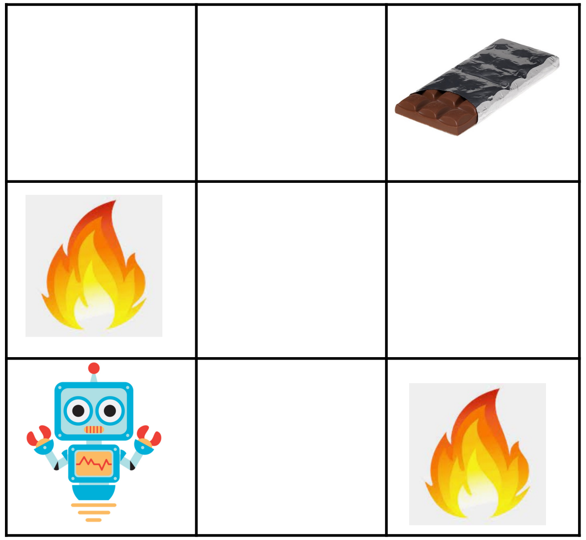
Continuous vs. Discrete State and Action Spaces

State and action spaces can be **discrete** or **continuous**

Clip art from wikipedia

Continuous vs. Discrete State and Action Spaces

State and action spaces can be **discrete** or **continuous**

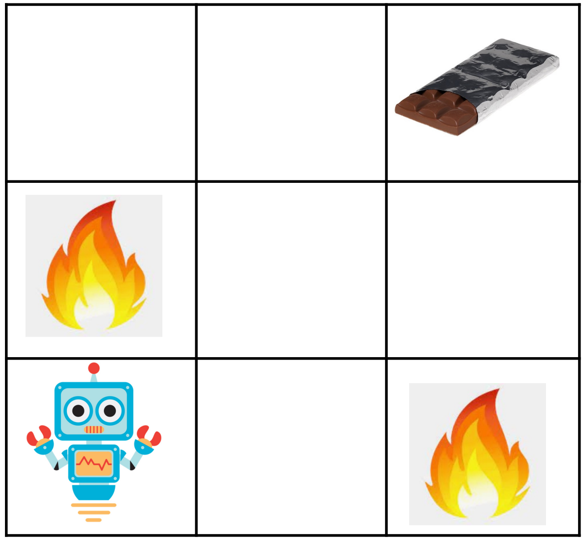


Actions: discrete move one square up, down, left, right
State: discrete position on board

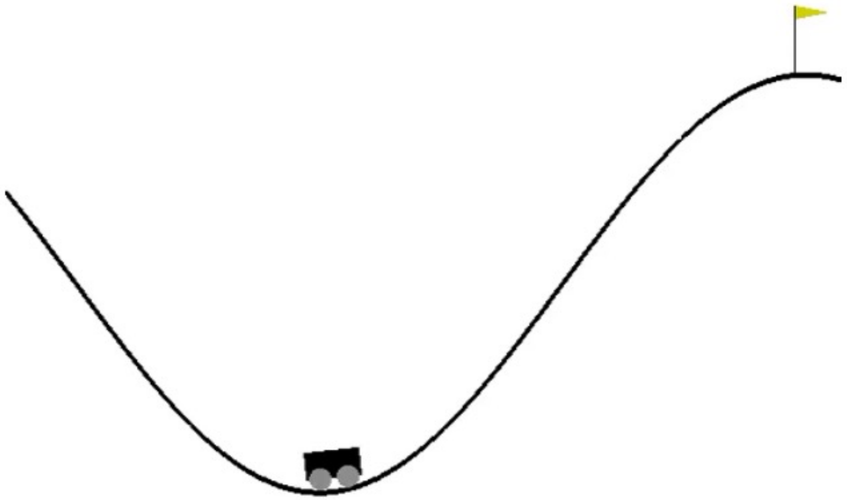
Clip art from wikipedia

Continuous vs. Discrete State and Action Spaces

State and action spaces can be **discrete** or **continuous**



Actions: discrete move one square up, down, left, right
State: discrete position on board

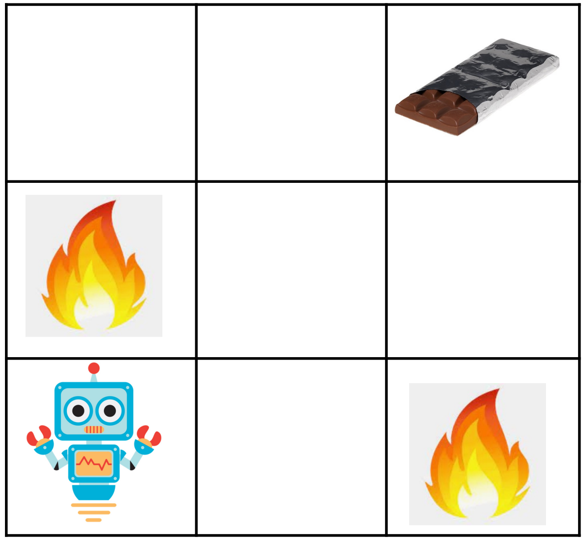


Actions: discrete acceleration factor [-1, 1, 0]
State: continuous position and velocity

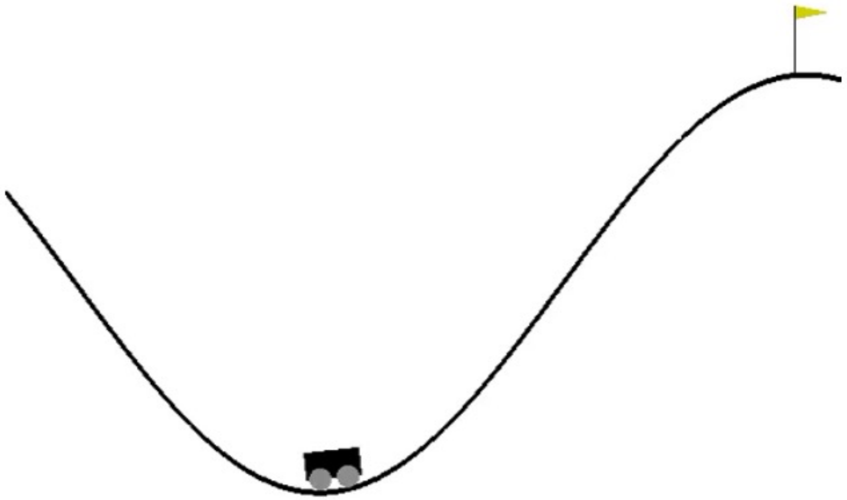
Clip art from wikipedia

Continuous vs. Discrete State and Action Spaces

State and action spaces can be **discrete** or **continuous**



Actions: discrete move one square up, down, left, right
State: discrete position on board



Actions: discrete acceleration factor [-1, 1, 0]
State: continuous position and velocity

→ Usually in accelerators we are dealing with continuous state and action spaces

Clip art from wikipedia

Returns and Episodes

Trying to maximize total estimated return → *how much should we care about near-term vs. long-term rewards?*

Total expected return

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Discount factor, $0 \leq \gamma \leq 1$

$\gamma = 0$ → *prioritize near-term rewards*

Can re-write in a form that will be useful in trying to *learn* an estimate of total reward:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Value Functions

Useful to estimate the expected long term reward at time t → encoded as **value functions**

Can be based on the present **state**, or a **state-action pair**

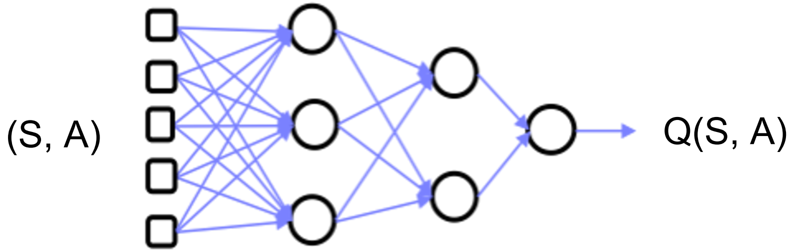
$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Best method for encoding the value function depends on the **size of the state and action space** (and whether it is continuous or discrete)

	A1	A2	A3
S1	Q(S1, A1)	Q(S1, A2)	Q(S1, A3)
S2	Q(S2, A1)	Q(S2, A2)	Q(S2, A3)
S3	Q(S3, A1)	Q(S3, A2)	Q(S3, A3)

Tabular Q-function

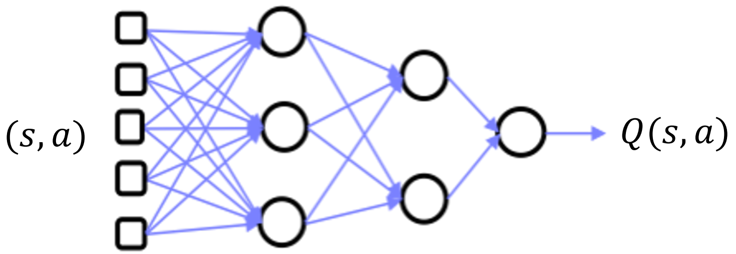


Parameterized Q-function

Policies

Policy — *how we decide to take certain actions given an observed state*

Option 1: Estimate value function, then use it to choose a_t



$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

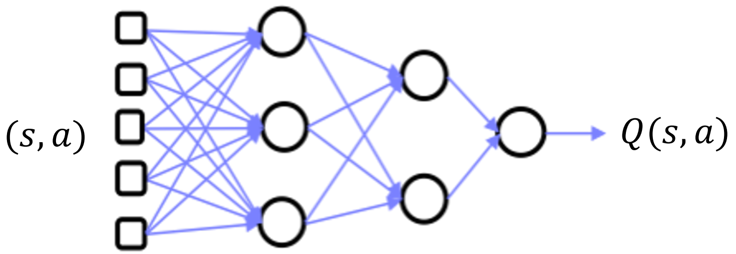
Policies

Policy — *how we decide to take certain actions given an observed state*

Option 1: Estimate value function, then use it to choose a_t

- Greedy policy – always choose the best action

- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)



$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

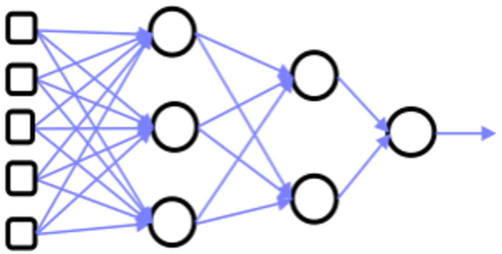
Policies

Policy — *how we decide to take certain actions given an observed state*

Option 1: Estimate value function, then use it to choose a_t

- Greedy policy – always choose the best action
- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)

(s, a)



$Q(s, a)$

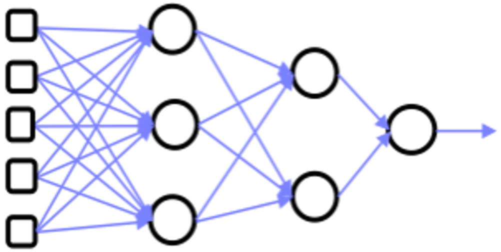
$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

Option 2: Parameterize the policy directly $\pi_\theta(a|s)$

- Mapping states directly to best actions
- Try to improve the policy by adjusting θ

s_t



a_t

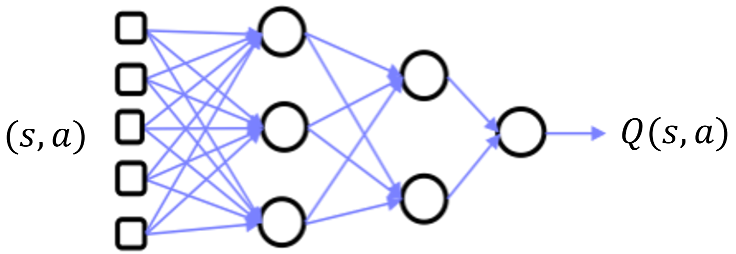
$\pi_\theta(a|s)$

Policies

Policy — *how we decide to take certain actions given an observed state*

Option 1: Estimate value function, then use it to choose a_t

- Greedy policy – always choose the best action
- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)

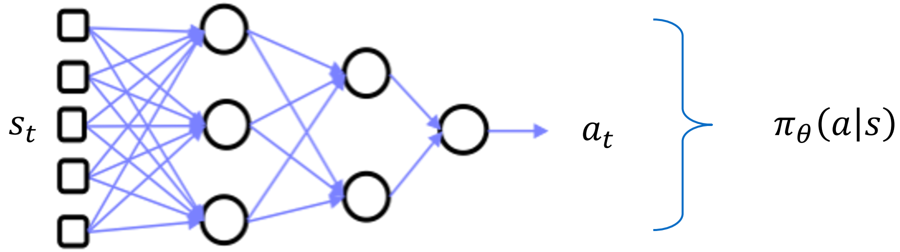


$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

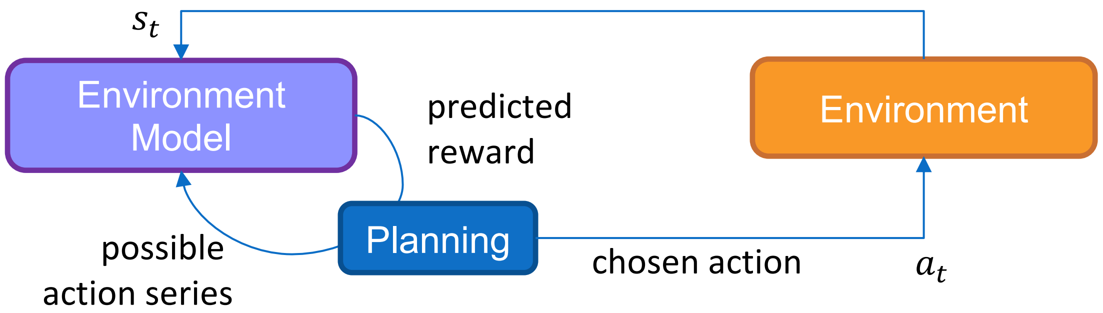
Option 2: Parameterize the policy directly $\pi_\theta(a|s)$

- Mapping states directly to best actions
- Try to improve the policy by adjusting θ



Option 3: Have a world model (for state transitions $s_t \rightarrow s_{t+1}$)?

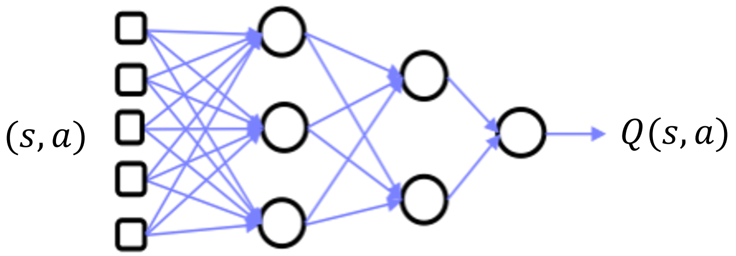
- Can explicitly plan with model to choose the best action
- Examples: LQR, Model Predictive Control
- Model could be analytic or learned (e.g. GPs, NNs, GMM, etc)
- Can backpropagate through model to learn policy



Policies

Policy — *how we decide to take certain actions given an observed state*

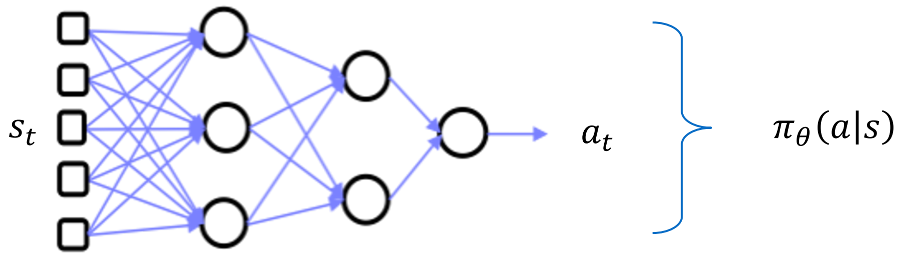
Option 1: Estimate value function, then use it to choose a_t
- Greedy policy – always choose the best action
- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)



$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

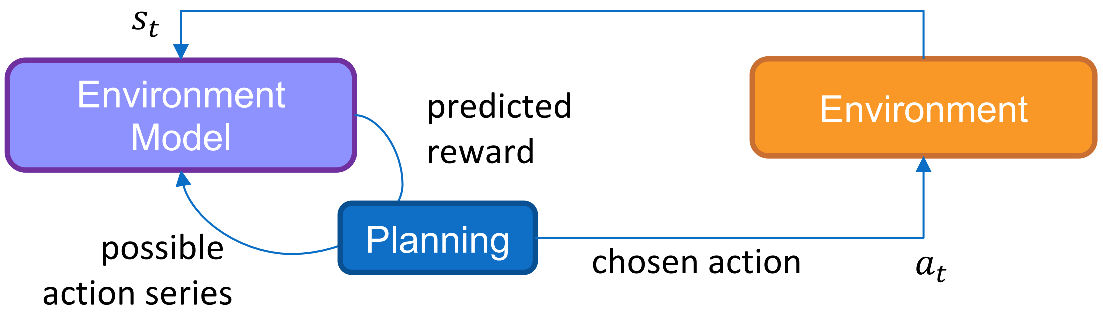
Option 2: Parameterize the policy directly $\pi_\theta(a|s)$
- Mapping states directly to best actions
- Try to improve the policy by adjusting θ



Model-free RL

Model-based RL

Option 3: Have a world model (for state transitions $s_t \rightarrow s_{t+1}$)?
- Can explicitly plan with model to choose the best action
- Examples: LQR, Model Predictive Control
- Model could be analytic or learned (e.g. GPs, NNs, GMM, etc)
- Can backpropagate through model to learn policy

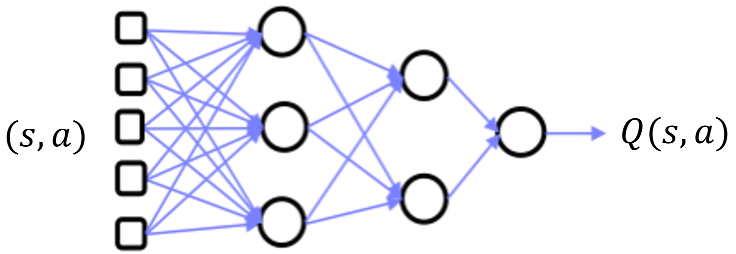


Policies

Policy — *how we decide to take certain actions given an observed state*

Value Based

Option 1: Estimate value function, then use it to choose a_t
- Greedy policy – always choose the best action
- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)

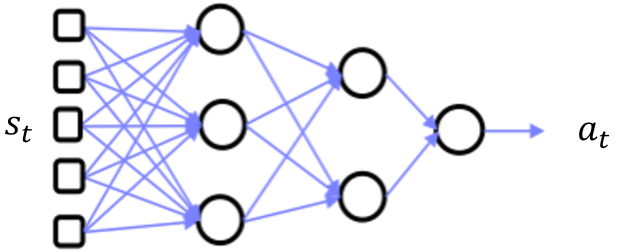


$a_t = \max_a Q(s_t, a)$

Could be costly if have big or continuous action space!

Policy Gradients

Option 2: Parameterize the policy directly $\pi_\theta(a|s)$
- Mapping states directly to best actions
- Try to improve the policy by adjusting θ

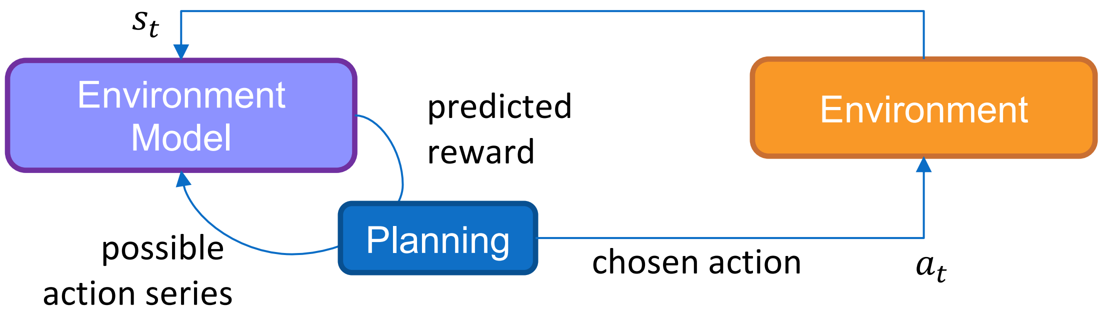


$\pi_\theta(a|s)$

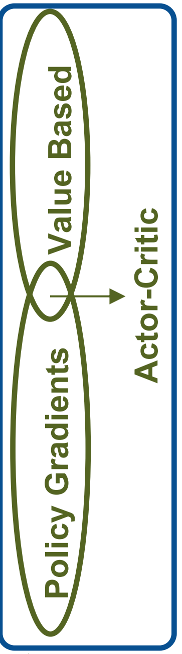
Model-free RL

Model-based RL

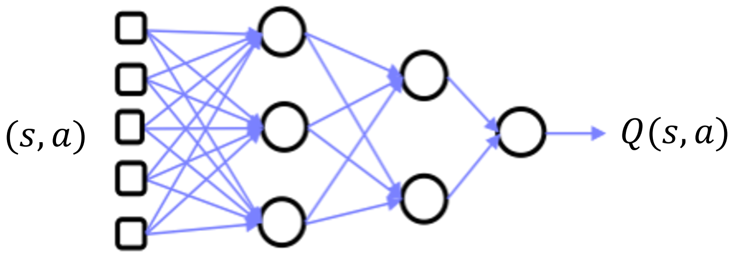
Option 3: Have a world model (for state transitions $s_t \rightarrow s_{t+1}$)?
- Can explicitly plan with model to choose the best action
- Examples: LQR, Model Predictive Control
- Model could be analytic or learned (e.g. GPs, NNs, GMM, etc)
- Can backpropagate through model to learn policy



Policy — *how we decide to take certain actions given an observed state*



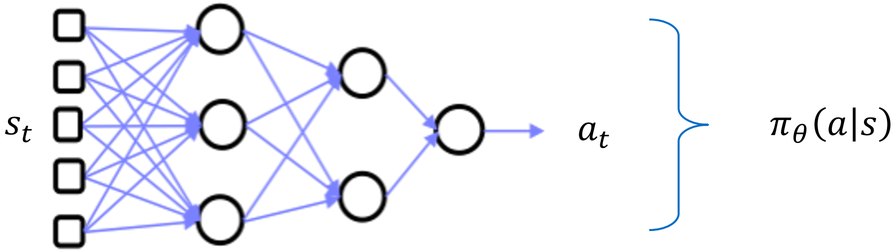
Option 1: Estimate value function, then use it to choose a_t
- Greedy policy – always choose the best action
- Ɛ-greedy policy – take random action with probability Ɛ, otherwise take the greedy action (adds exploration)



$$a_t = \max_a Q(s_t, a)$$

Could be costly if have big or continuous action space!

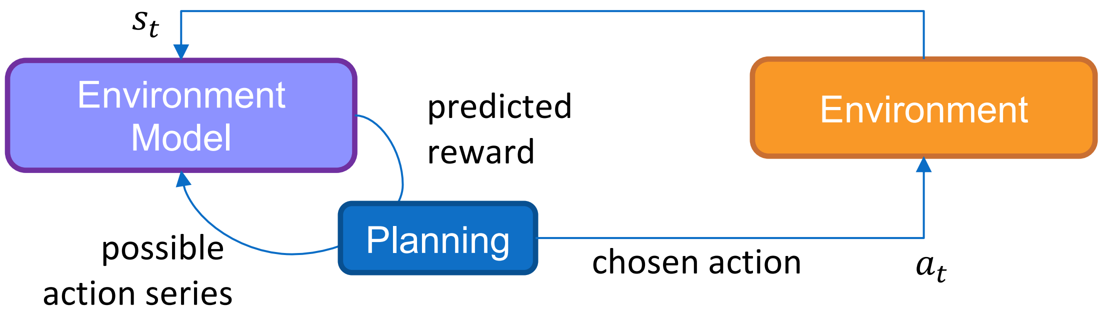
Option 2: Parameterize the policy directly $\pi_\theta(a|s)$
- Mapping states directly to best actions
- Try to improve the policy by adjusting θ



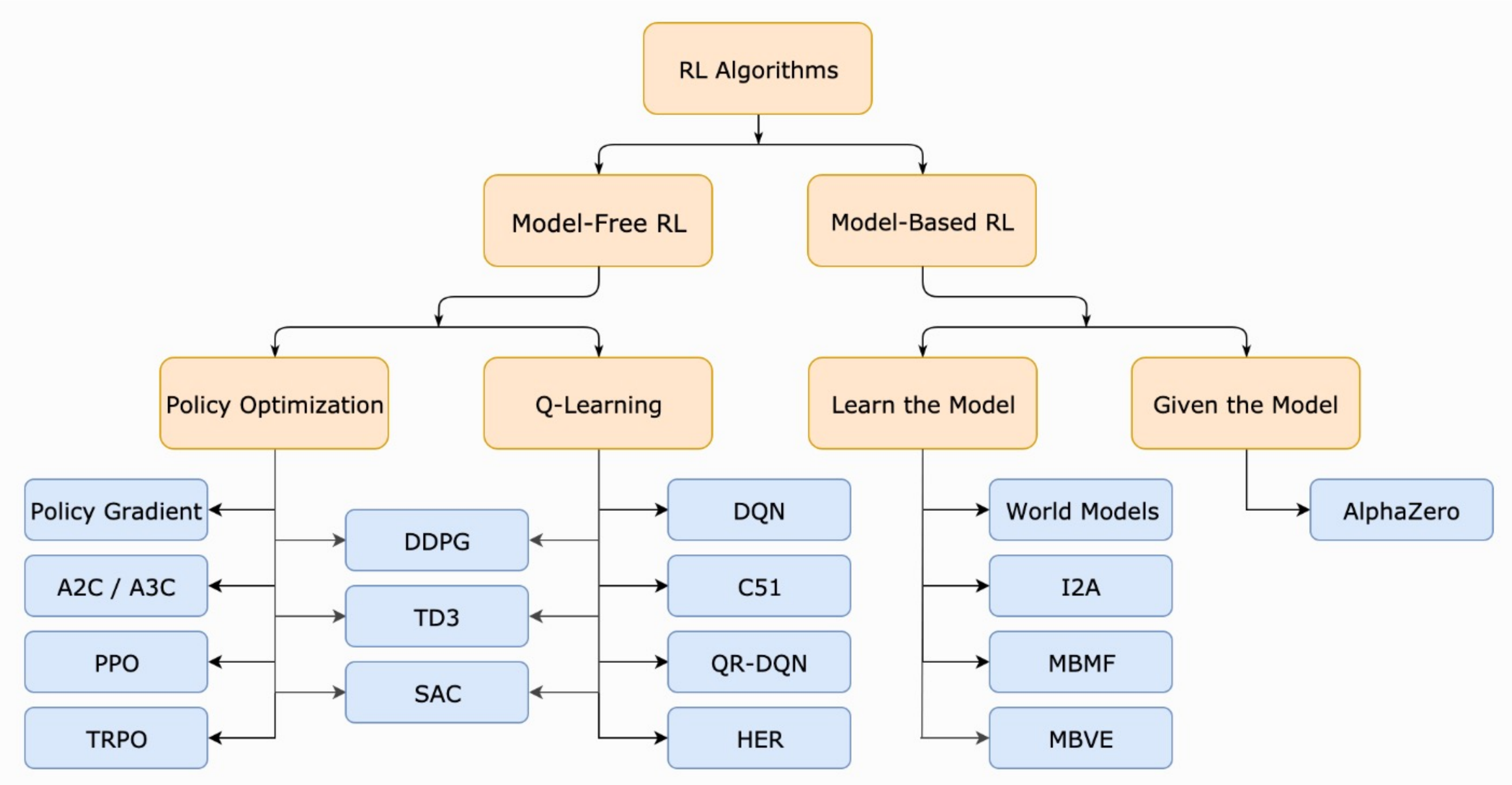
Model-free RL

Model-based RL

Option 3: Have a world model (for state transitions $s_t \rightarrow s_{t+1}$)?
- Can explicitly plan with model to choose the best action
- Examples: LQR, Model Predictive Control
- Model could be analytic or learned (e.g. GPs, NNs, GMM, etc)
- Can backpropagate through model to learn policy



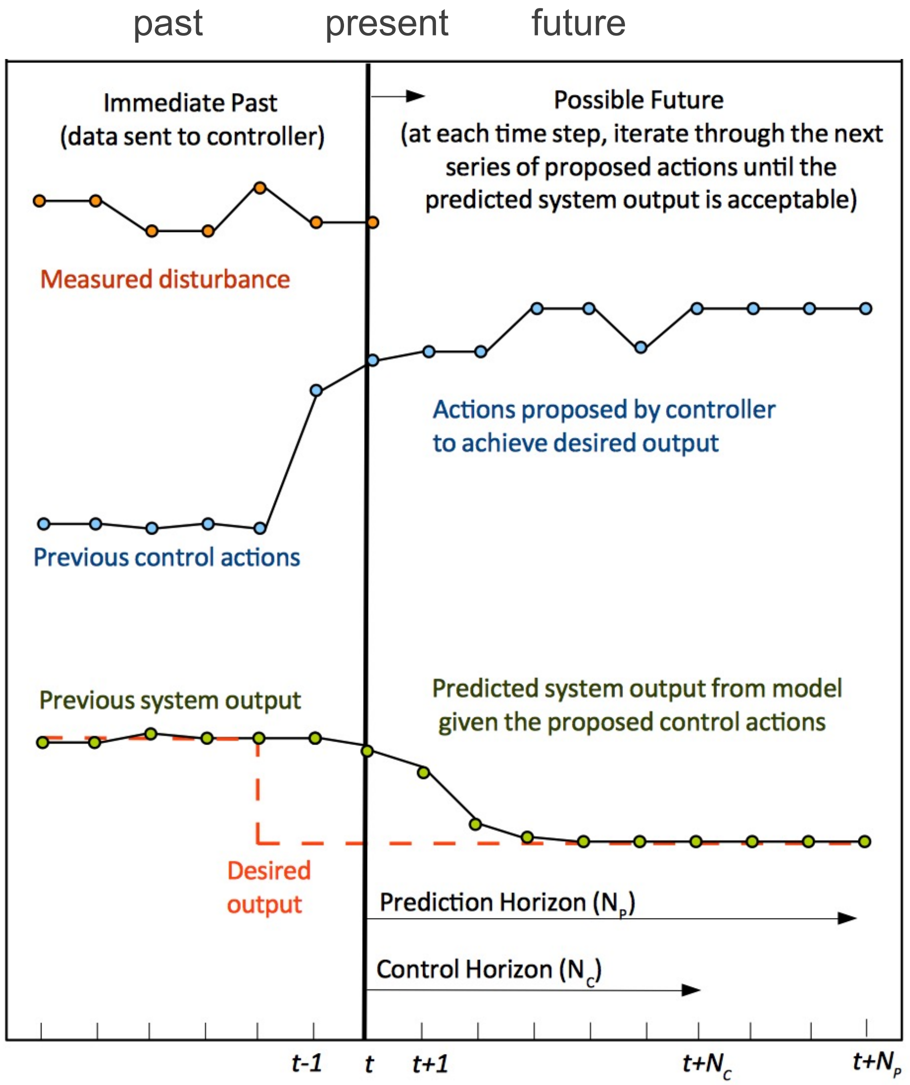
Wide Variety of RL Algorithms…



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

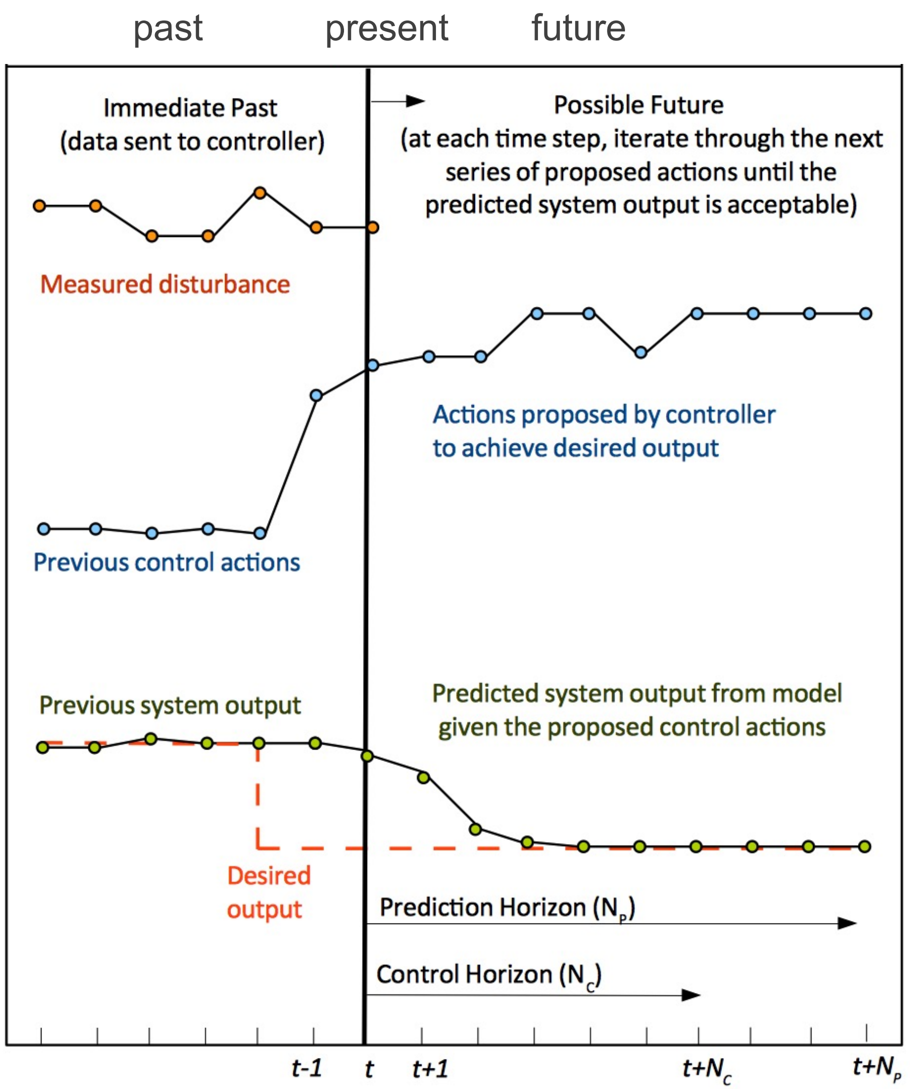
Model Predictive Control

For accelerator physicists, it is conceptually useful to think about **model predictive control** first:



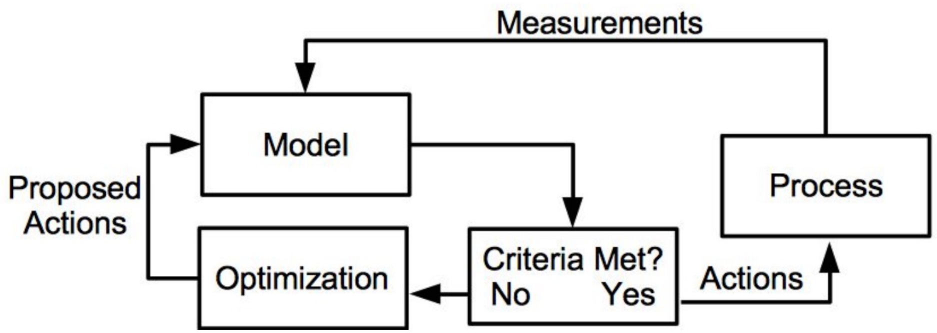
Model Predictive Control

For accelerator physicists, it is conceptually useful to think about **model predictive control** first:

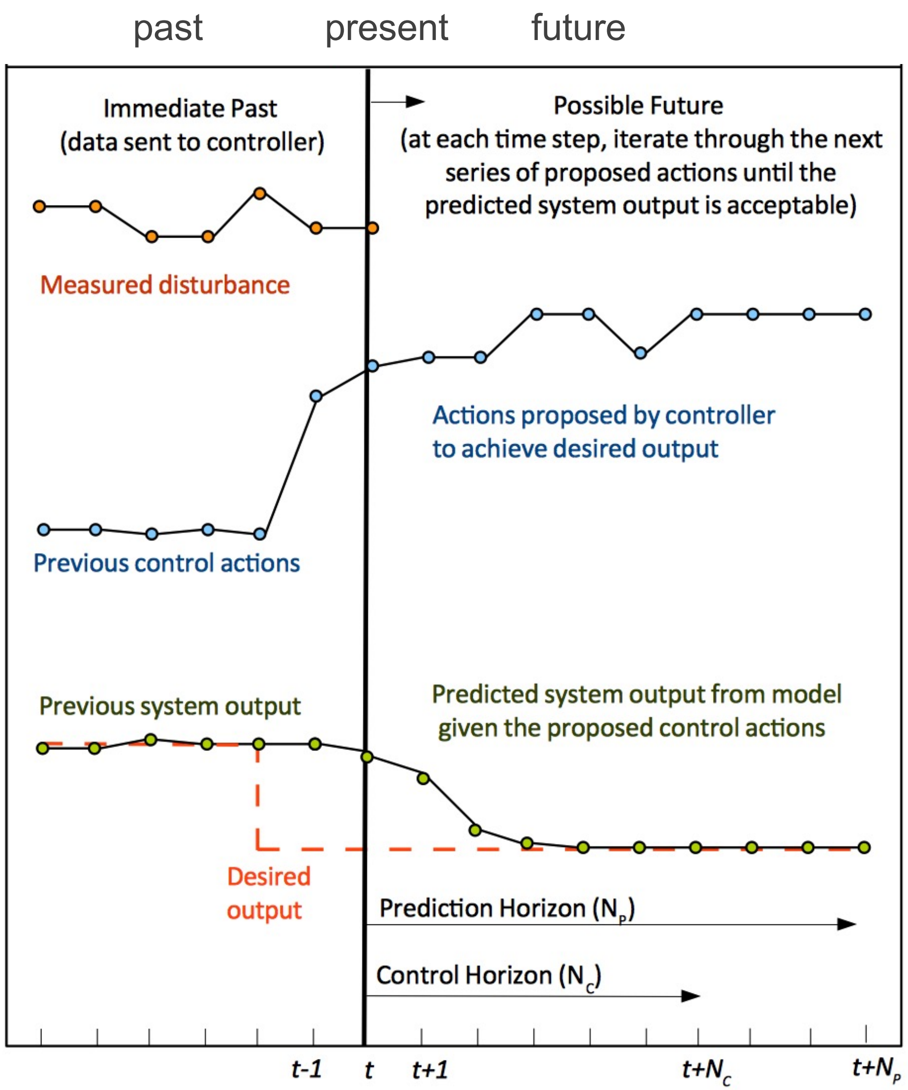


Basic concept:

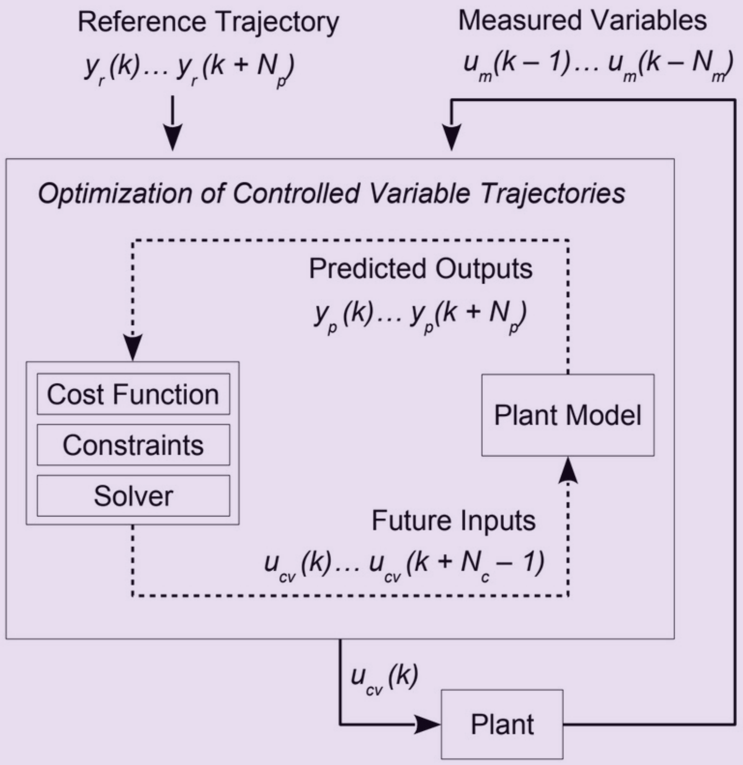
1. Use a predictive model to assess the outcome of possible future actions
2. Choose the best series of actions by optimizing a set of planned actions, with respect to a cost function over a set time horizon
3. Execute the first action
4. Gather next time step of data
5. Repeat



Model Predictive Control



Predicted cost is calculated over future time horizon:



N_m previous measurements

N_p future time steps predicted

N_c future time steps controlled

$$\sum_{i=1}^{N_p}\{w_y\,[y_r(k+i)-y_p(k+i)]\}^2$$
(output variable targets)

$$\sum_{j=1}^{ncv}\sum_{i=0}^{N_p-1}\{w_{u,j}\,[u_j(k+i)-u_{j,ref}(k+i)]\}^2$$
(controllable variable targets)

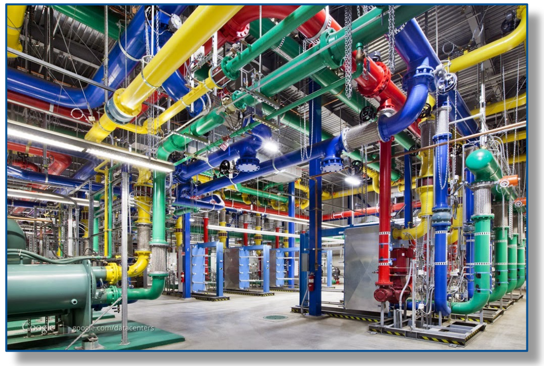
$$\sum_{j=1}^{ncv}\sum_{i=0}^{N_p-1}\{w_{\Delta u,j}\,[u_j(k+i)-u_j(k+i-1)]\}^2$$
(movement size)

Note: "process" and "plant" come from classic control → it's the system being controlled

Where does this apply?

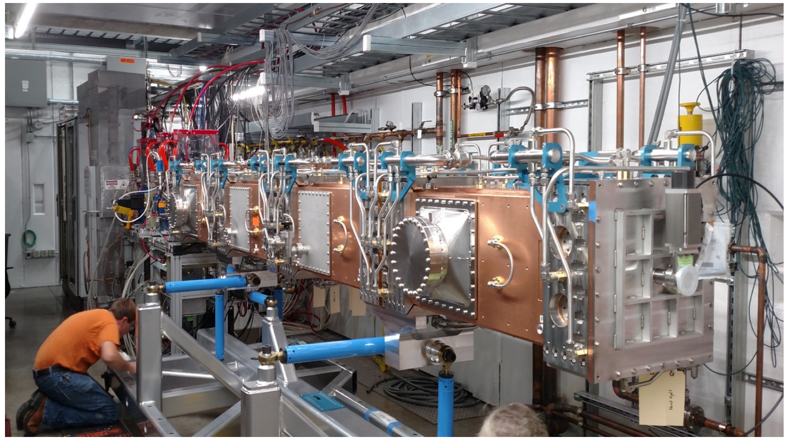
Many of the problems we discussed so far in the class are **singe timestep input, single-timestep output problems**
→ when control actions are taken at a faster rate than the system dynamics, we need to take into account time-evolution of system

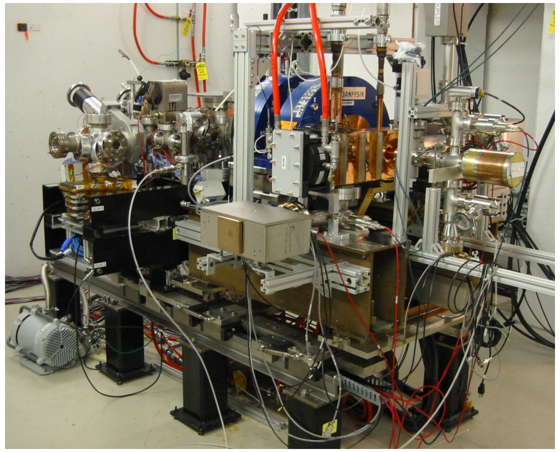
Transport delays, variable heat load
Efficient servers were not enough
--> needed better control of cooling system



https://googleblog.blogspot.com

RF accelerating cavities (e.g. resonance control)





Transport delays, variable heat load, complex dynamics

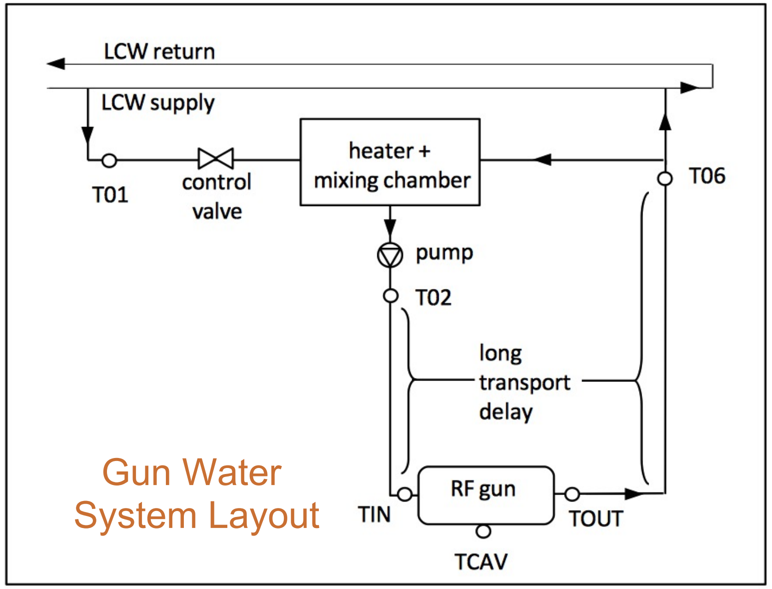
Cryogenic systems

Example: Resonant Frequency Control at FAST

Resonant frequency controlled via temperature
- *Long transport delays and thermal responses*
- *Two controllable variables: heater power + flow valve aperture*

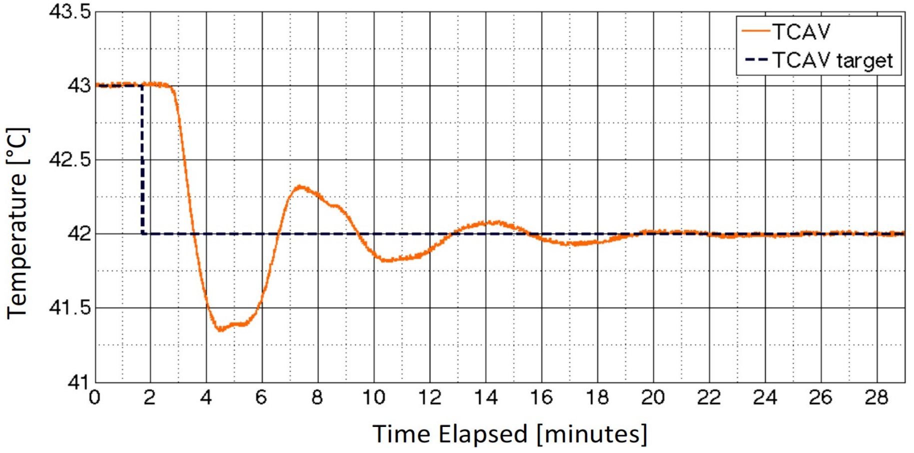
Applied model predictive control with a neural network model trained on measured data

~ 5x faster settling time + no large overshoot

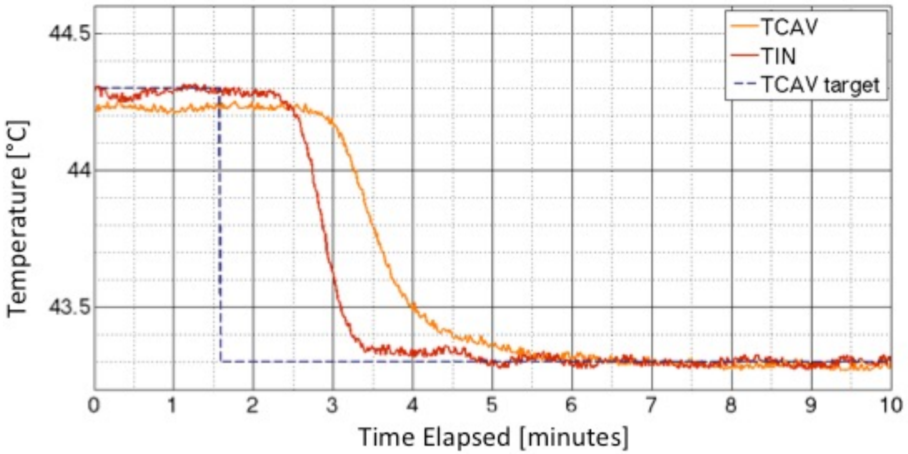


Gun Water System Layout

Existing Feedforward/PID Controller



Model Predictive Controller

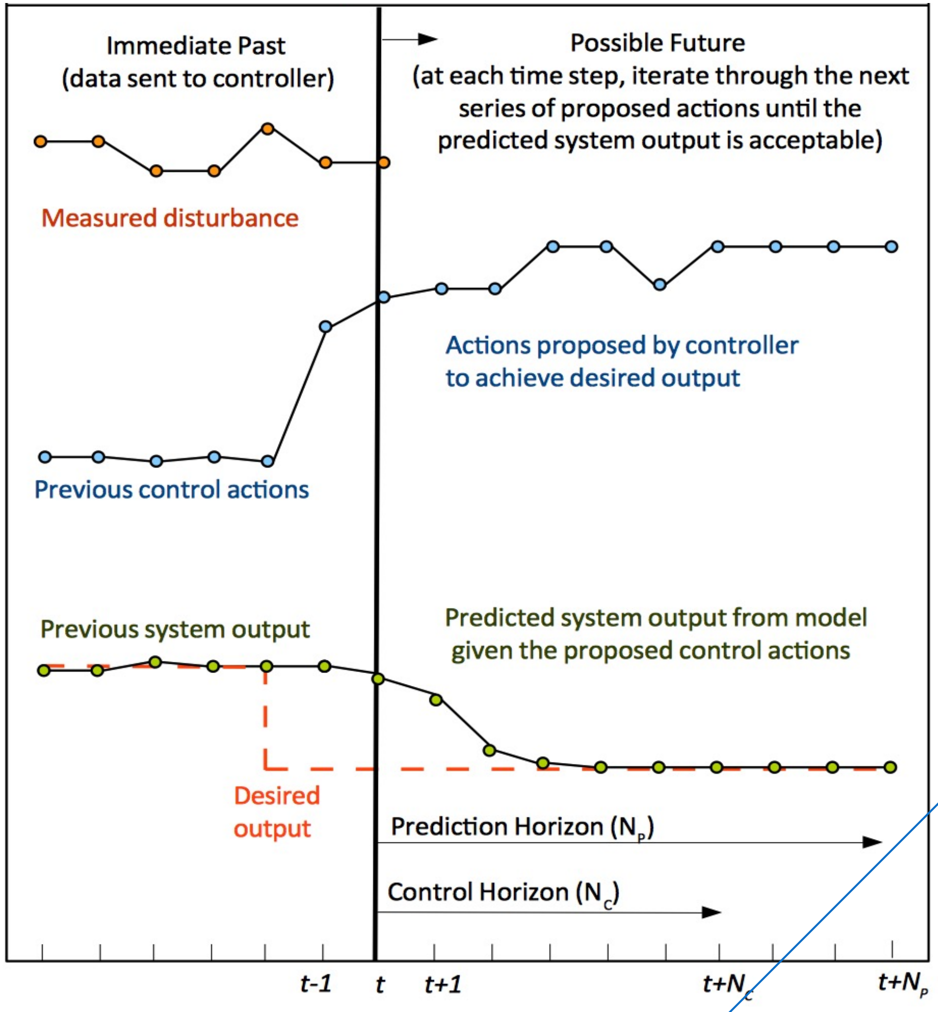


Note differences in scales!

Note that the oscillations are largely due to the transport delays and water recirculation, rather than PID gains

Edelen, IPAC'15 ; Edelen, TNS, 2016

Model Predictive Control: Analogies to Model-free RL



MPC: explicitly calculating the future time horizon and optimizing actions over it, given present state

Instead, model-free RL methods try to estimate aspects of this

Estimate total future reward (cost over prediction horizon) given s_t a_t
(value function)

and/or

Find a map between s_t and first optimal action a_t (skip optimization)
(policy gradient)

Where in accelerators might one want to use RL?

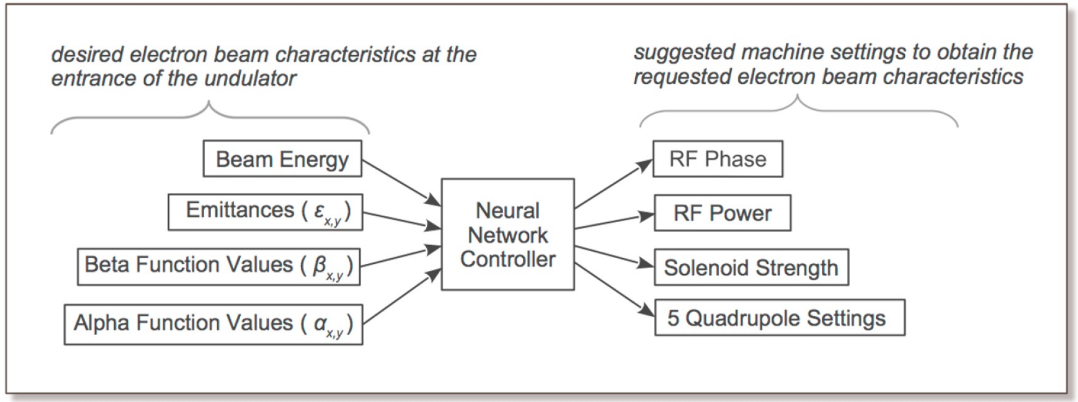
- Cases where time dependencies matter relative to control actions *(e.g. rf control, slow time delays etc)*

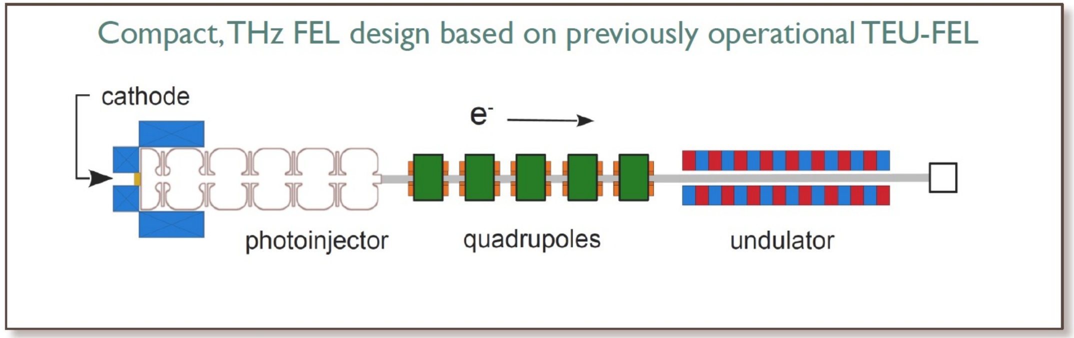
- Learning an optimization algorithm
  - *episode length becomes number of steps allowed*

- Control / fast switching between setups:
  - *e.g. trajectory control*
  - *e.g. phase space shaping inverse model → add fine tuning with RL*

Example: Backprop through model to learn policy

Goal: Rapid switching between energies (with appropriate match into undulator) for a compact THz FEL



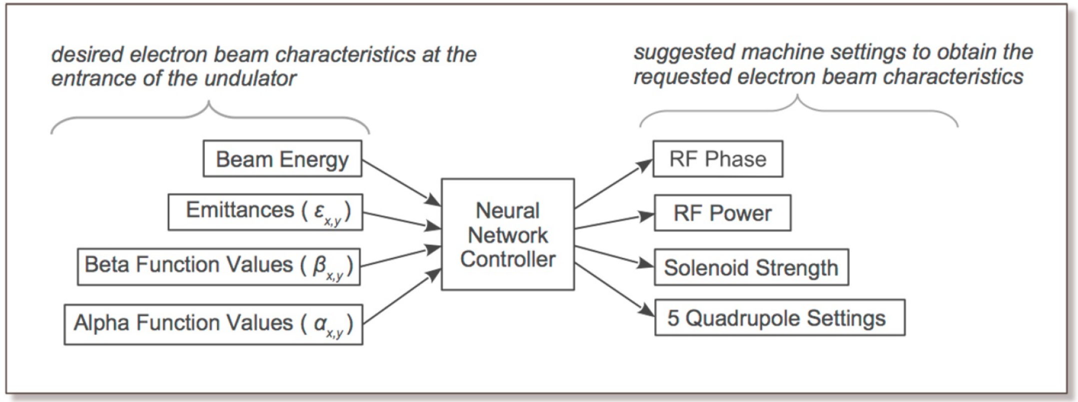


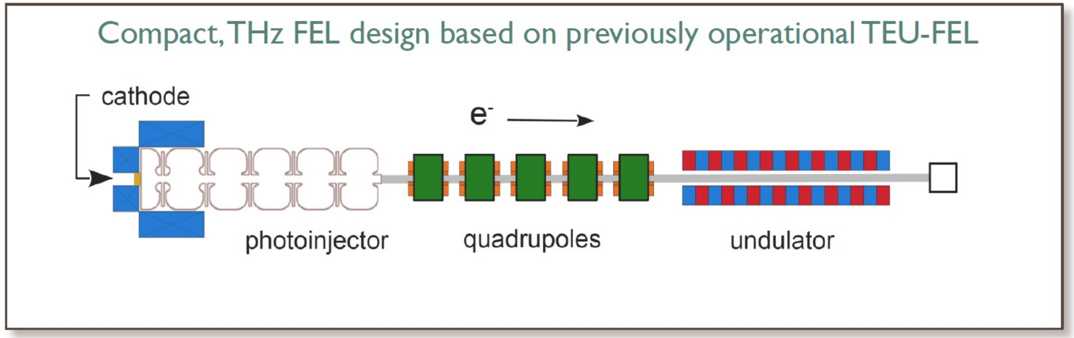
Variable Setting
$Q1$
$Q2$
$Q3$
$Q4$
$Q5$
Gun Solenoid Strength
Gun RF Phase
Gun RF Power

Edelen et al., NeurIPS 2017; IPAC'18

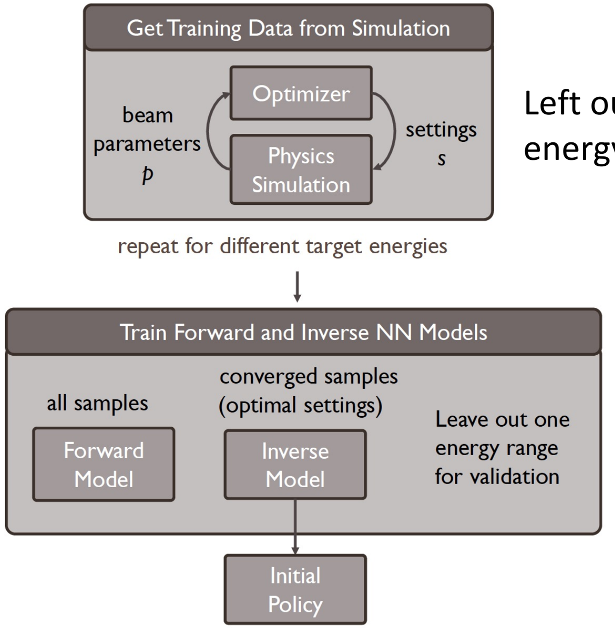
Example: Backprop through model to learn policy

Goal: Rapid switching between energies (with appropriate match into undulator) for a compact THz FEL





Variable Setting
$Q1$
$Q2$
$Q3$
$Q4$
$Q5$
Gun Solenoid Strength
Gun RF Phase
Gun RF Power



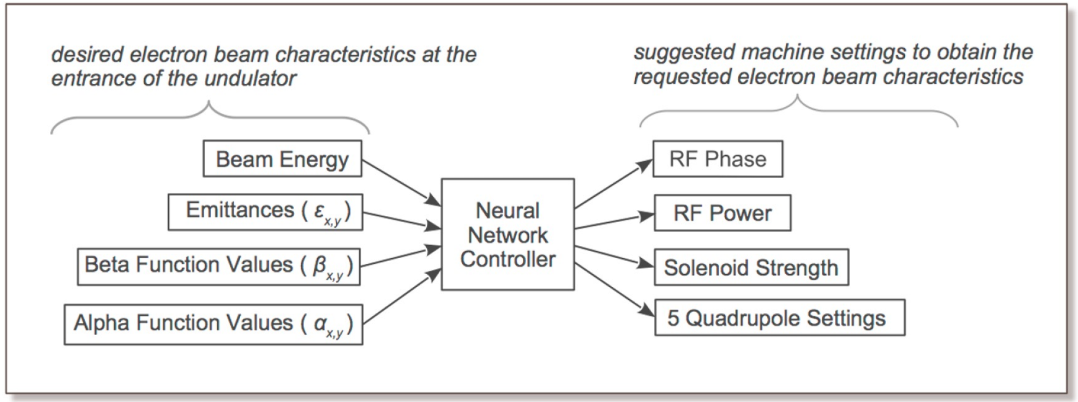
Left out some energy ranges

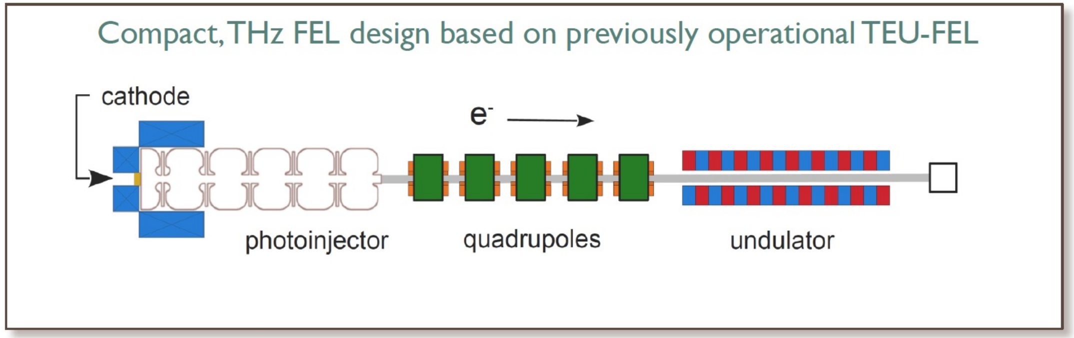
Supervised pre-training

Edelen et al., NeurIPS 2017; IPAC'18

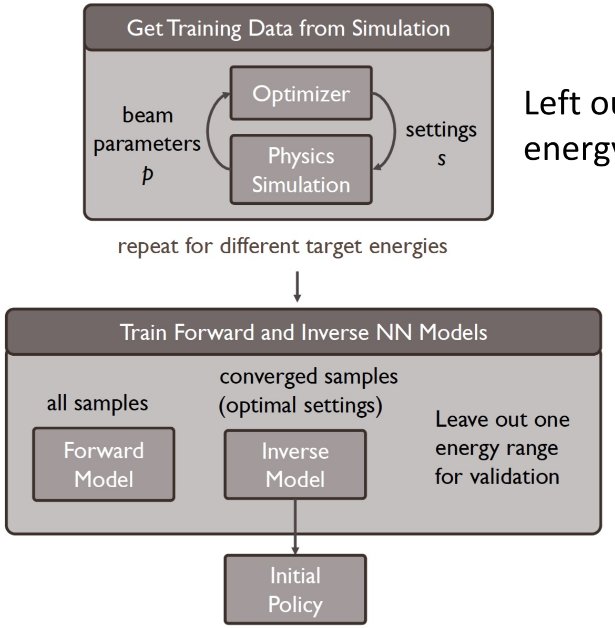
Example: Backprop through model to learn policy

Goal: Rapid switching between energies (with appropriate match into undulator) for a compact THz FEL





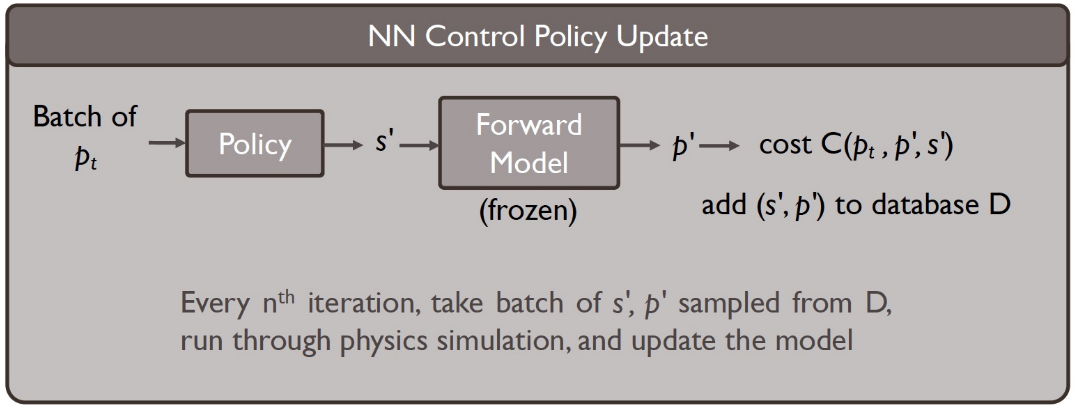
Variable Setting
$Q1$
$Q2$
$Q3$
$Q4$
$Q5$
Gun Solenoid Strength
Gun RF Phase
Gun RF Power



Left out some energy ranges

Supervised pre-training

Use backprop through model while exploring new regions of parameter space
→ periodically update model



p_t – target beam parameters

s' – predicted optimal settings

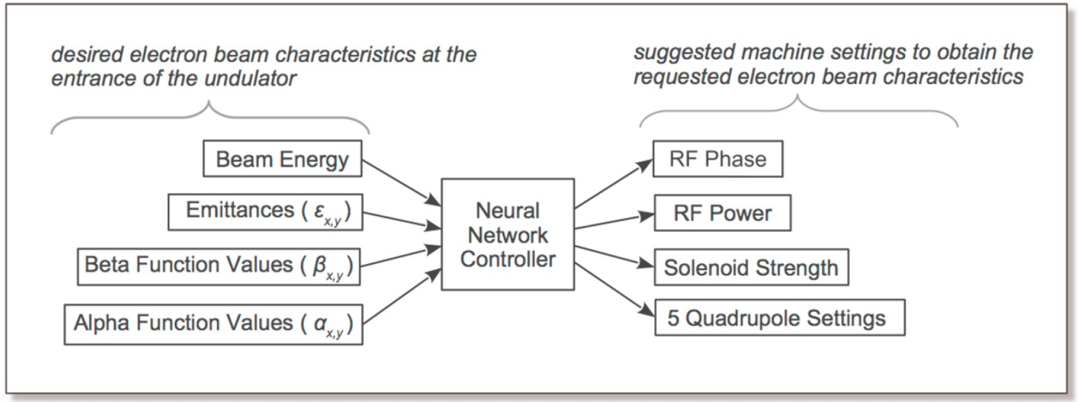
p' – predicted beam parameters

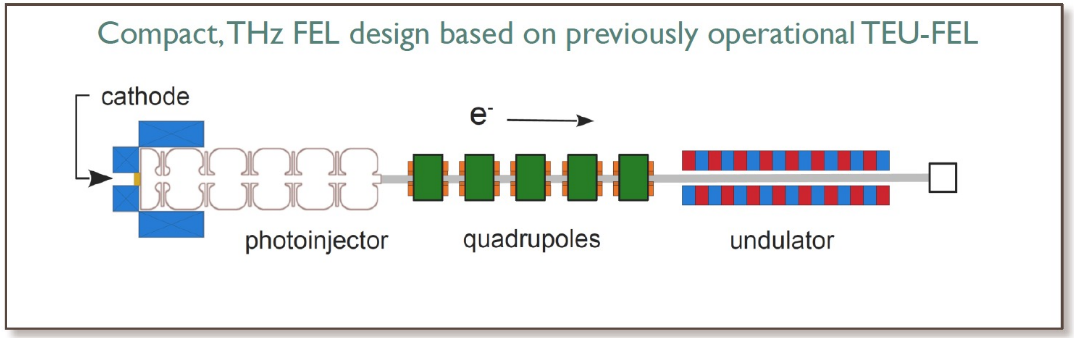
Cost:
difference between p' and p_t
penalize loss of transmission
penalize higher magnet settings

Edelen et al., NeurIPS 2017; IPAC'18

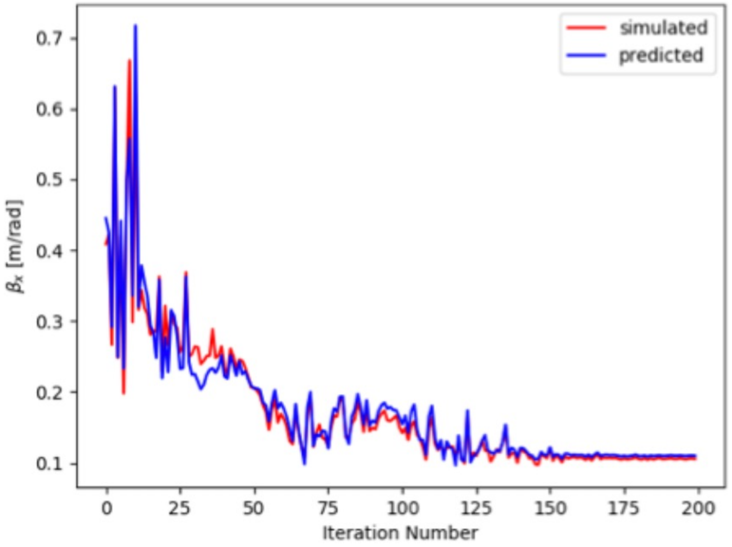
Example: Backprop through model to learn policy

Goal: Rapid switching between energies (with appropriate match into undulator) for a compact THz FEL





Variable Setting
$Q1$
$Q2$
$Q3$
$Q4$
$Q5$
Gun Solenoid Strength
Gun RF Phase
Gun RF Power



Example from running simplex on the simulation

→ ~ 170 iterations to converge for new energy target

NN policy can reach $\alpha_{x,y}$ = 0 $\beta_{x,y}$ = 0.106

in one iteration for new target energies

Parameter	Train MAE	Train STD
α_x [rad]	0.012	0.075
α_y [rad]	0.013	0.079
β_x [m/rad]	0.008	0.004
β_y [m/rad]	0.014	0.011

Edelen et al., NeurIPS 2017 ; IPAC'18

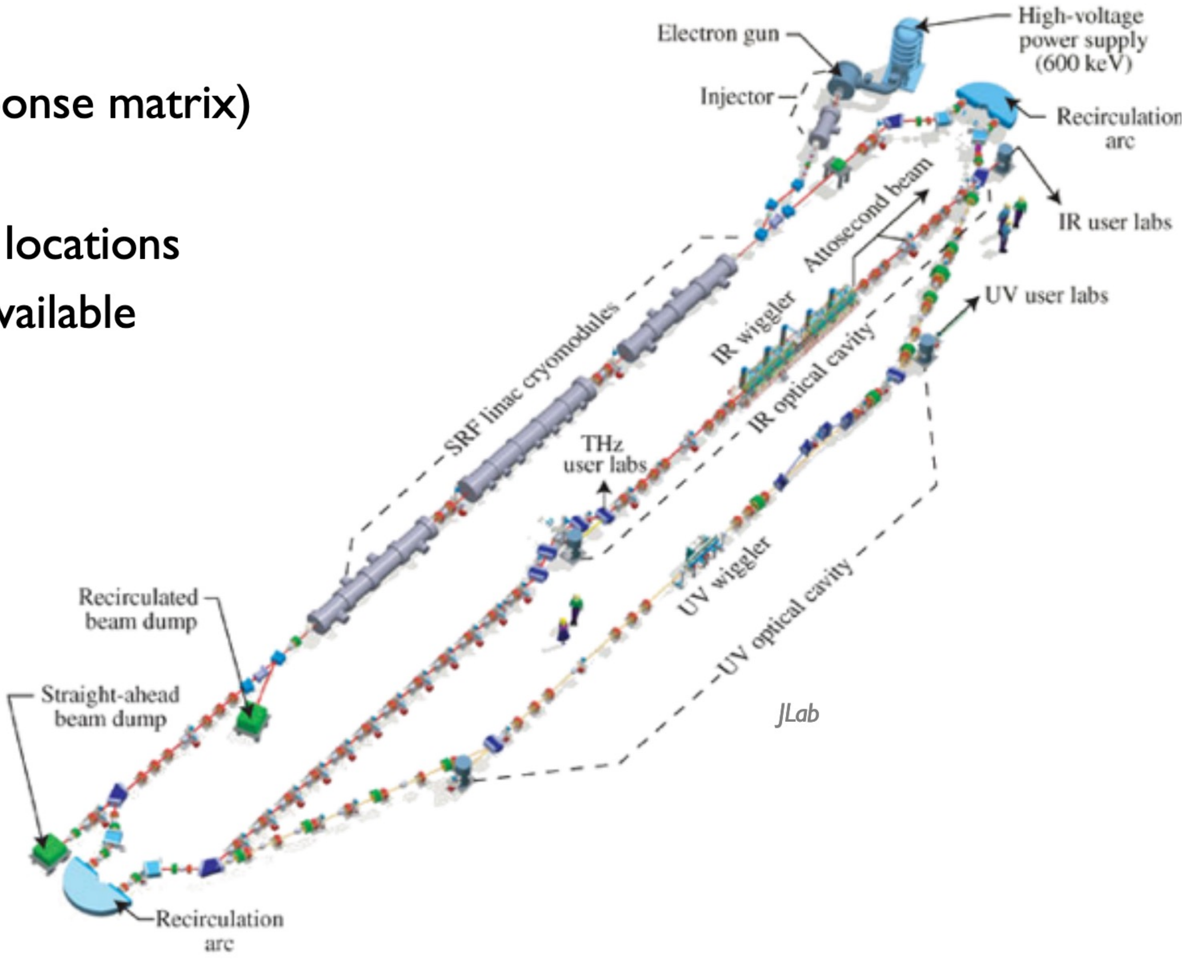
Fast Switching Between Trajectories

Work with C. Tennant and D. Douglas, JLab

- 76 BPMs, 57 dipoles, 53 quadrupoles
- Traditional approach has never worked (linear response matrix)
- Rely on a few experts for steering tune-up
- Want to specify small offsets in trajectory at some locations
- Didn't initially have an up-to-date machine model available

Learn responses (NN model) from tune-up data and dedicated study time:
dipole + quadrupole settings → predict BPMs + transmission

Train controller (NN policy) offline using NN model:
desired trajectory → dipole settings
(and penalize losses + large magnet settings)



JLab

Example: Trajectory Control

Fast Switching Between Trajectories

Main anticipated advantage of NN over standard approach:

Adaptive control policy → adjust without interfering with operation for response measurements as often?

Handling of trajectories away from BPM center (nonlinear)

But, need to quantify this …

Learn responses (NN model) from tune-up data and dedicated study time:
dipole + quadrupole settings → predict BPMs + transmission

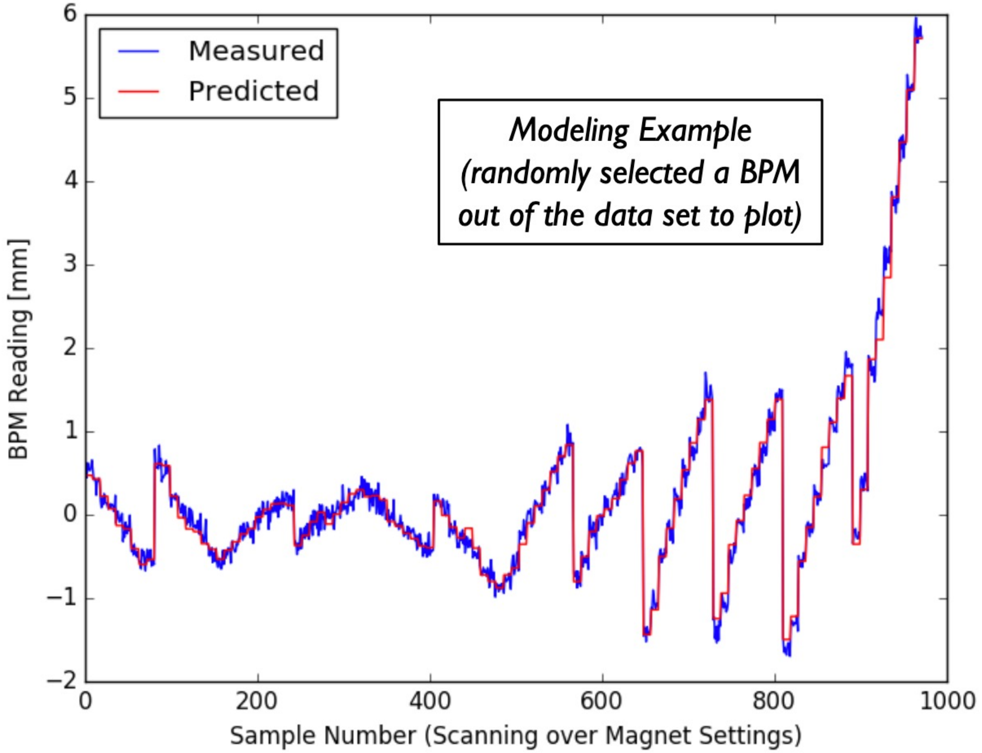
Train controller (NN policy) offline using NN model:
desired trajectory → dipole settings
(and penalize losses + large magnet settings)

Preliminary Results:

Model Errors for BPMs:

Training Set:	0.07 mm MAE	0.09 mm STD
Validation Set:	0.08 mm MAE	0.07 mm STD
Test Set:	0.08 mm MAE	0.03 mm STD

Controller: random initial states → on average within 0.2 mm of center immediately



Limitations of Model-Based RL

- Need a model!
  - May not have one
  - Can be harder to learn than policy
- Model setup
  - How expressive?
  - How fast?
- Model errors → how to handle where model is confident but wrong
- Need a good model, but a good model does not guarantee a good policy!

Easy policy, difficult model

Value-Based Methods: Temporal Difference Learning

How to learn a value function from experience [i.e. (state, action, reward) tuples]?

Update value function according to gradient descent at the end of an episode: $V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$

Value-Based Methods: Temporal Difference Learning

How to learn a value function from experience [i.e. (state, action, reward) tuples]?

Update value function according to gradient descent at the end of an episode: $V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$

Bootstrap by using the next estimate of the value function as an approximation for G_t:

$$\left.\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big] \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big] \end{aligned}\right\} \quad \textbf{Temporal Difference Equation TD(0)}$$

Value-Based Methods: Temporal Difference Learning

How to learn a value function from experience [i.e. (state, action, reward) tuples]?

Update value function according to gradient descent at the end of an episode: $V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$

Bootstrap by using the next estimate of the value function as an approximation for G_t:

$$\left.\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \end{aligned}\right\}$$

Temporal Difference Equation TD(0)

The error estimate is known as the TD error:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Value-Based Methods: Temporal Difference Learning

How to learn a value function from experience [i.e. (state, action, reward) tuples]?

Update value function according to gradient descent at the end of an episode: $V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$

Bootstrap by using the next estimate of the value function as an approximation for G_t:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Temporal Difference Equation TD(0)

The error estimate is known as the TD error:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

The value function can be written as:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned}$$

Value-Based Methods: Temporal Difference Learning

How to learn a value function from experience [i.e. (state, action, reward) tuples]?

Update value function according to gradient descent at the end of an episode: $V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$

Bootstrap by using the next estimate of the value function as an approximation for G_t:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Temporal Difference Equation TD(0)

The error estimate is known as the TD error:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

The value function can be written as:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned}$$

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Sutton, 1998

On-policy vs. Off-policy Learning

On-policy — need new samples / retrain whenever policy is changed

(e.g. policy gradients)

Off-policy — can improve policy without obtaining new samples from that policy

(e.g. Q-learning)

Example for value based methods: SARSA vs. Q-Learning

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

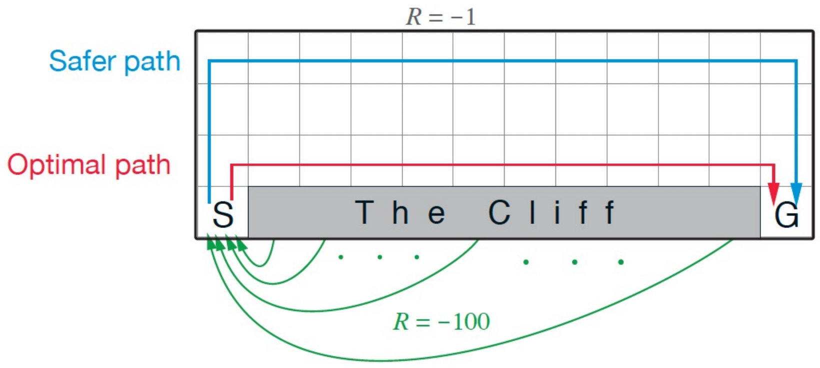
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

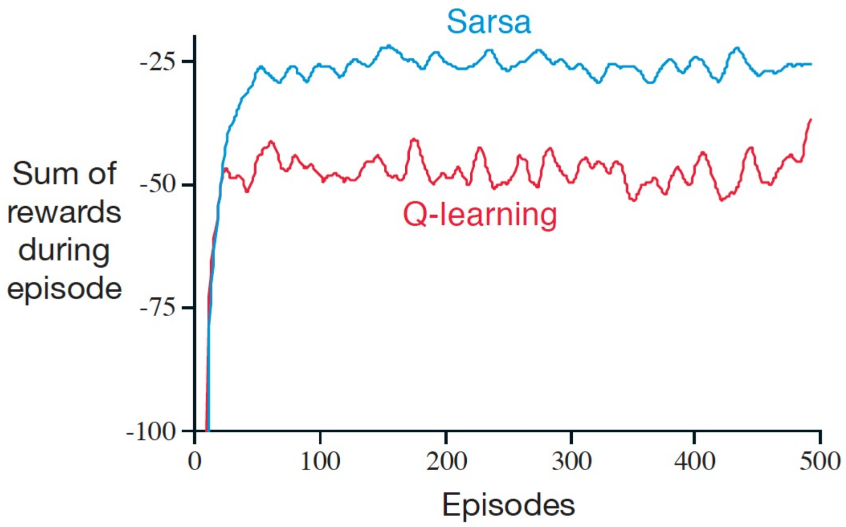
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal



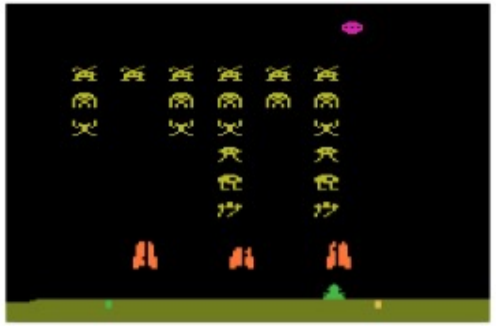


Q learning will converge to the optimal policy, but falls off the cliff a lot in the process

Figures from Sutton & Barto, 1998

Deep Q-Learning (DQN)

- Mnih et al., Playing Atari with Deep Reinforcement Learning(2013) https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
- Ɛ-greedy policy + Q-learning
- Experience replay
- CNN layers in Q-function to analyze the board



	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

HNeat –hand-engineered features

Breakout, Enduro and Pong > human
Others require policies over long timescales

Deep Q-Learning Tips

- Can be difficult to stabilize → best to test on simple problems first
- Use large replay buffers to help stabilize learning
- Takes time to converge → will look random for awhile
- If using $\mathcal{E}$-greedy policy, start with high ε

Deep Deterministic Policy Gradients (DDPG)

Lillicrap et al., Continuous Control with Deep Reinforcement Learning, (2016)
https://arxiv.org/pdf/1509.02971v6.pdf

Silver et al., Deterministic Policy Gradient Algorithms, (2014) http://proceedings.mlr.press/v32/silver14.pdf

Main elements:

- Learn Q values through experience replay buffer
- Update policy via Q function estimate + backprop
- Use target networks to stabilize learning
  → *time-delayed versions of each network*
- Ornstein-Uhlenbeck process to add noise to the action output for exploration
  (Uhlenbeck & Ornstein, 1930)

$$
\begin{aligned}
\theta^Q &: \text{Q network} \\
\theta^\mu &: \text{Deterministic policy function} \\
\theta^{Q'} &: \text{target Q network} \\
\theta^{\mu'} &: \text{target policy network}
\end{aligned}
$$

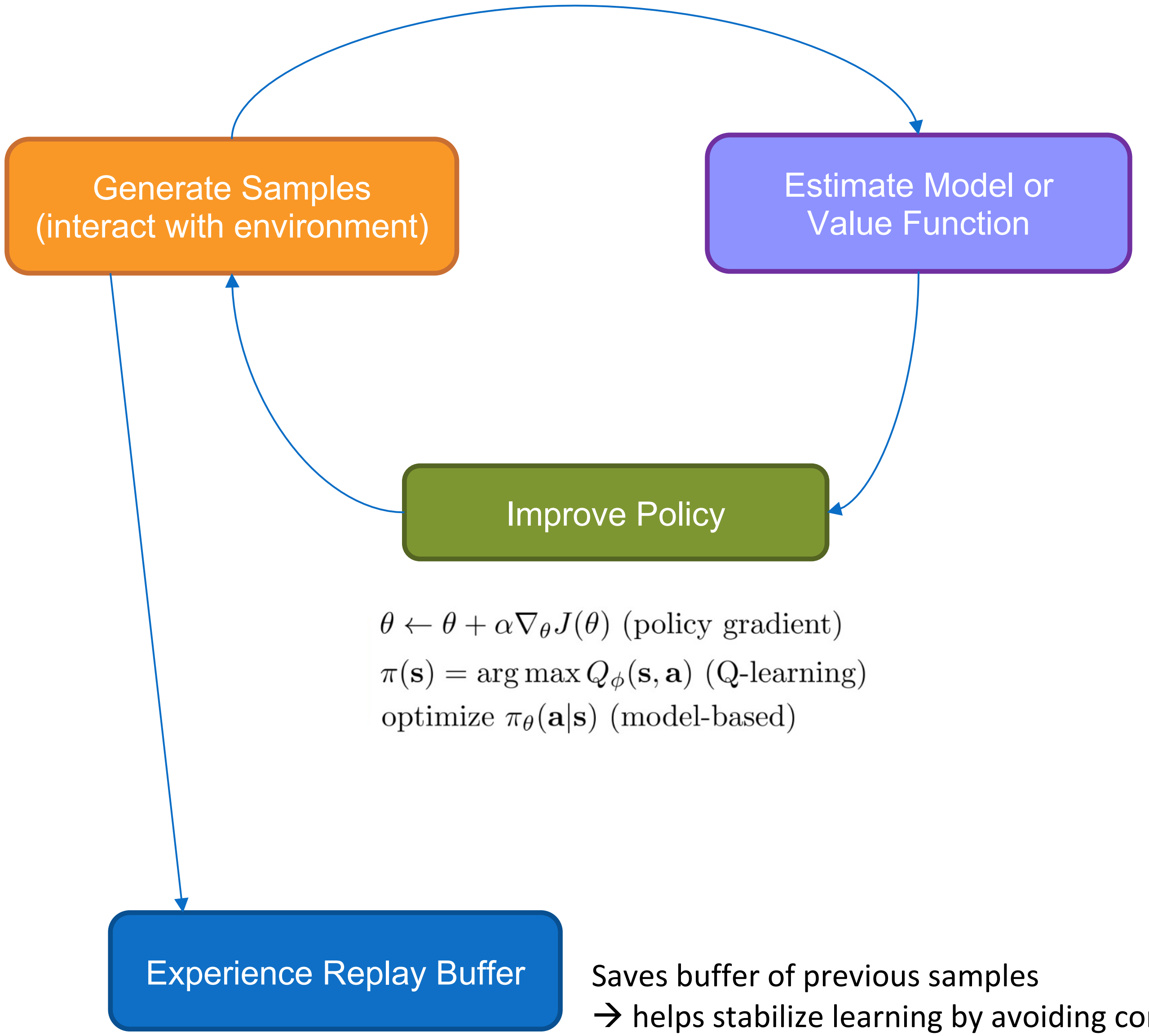
$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta^\mu} \mu(s|\theta^\mu)$$

$$
\begin{aligned}
\theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\
\theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}
\end{aligned}
$$

$$\text{where} \quad \tau \ll 1$$

$$\mu'(s_t) = \mu(s_t|\theta^\mu_t) + \mathcal{N}$$

Recap: High-Level View



fit $Q_\phi(\mathbf{s}, \mathbf{a})$ (actor-critic, Q-learning)
estimate $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)
$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ (Q-learning)
optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

Saves buffer of previous samples
→ helps stabilize learning by avoiding correlated sequences

Partially adapted from S. Levine

Sample Efficiency

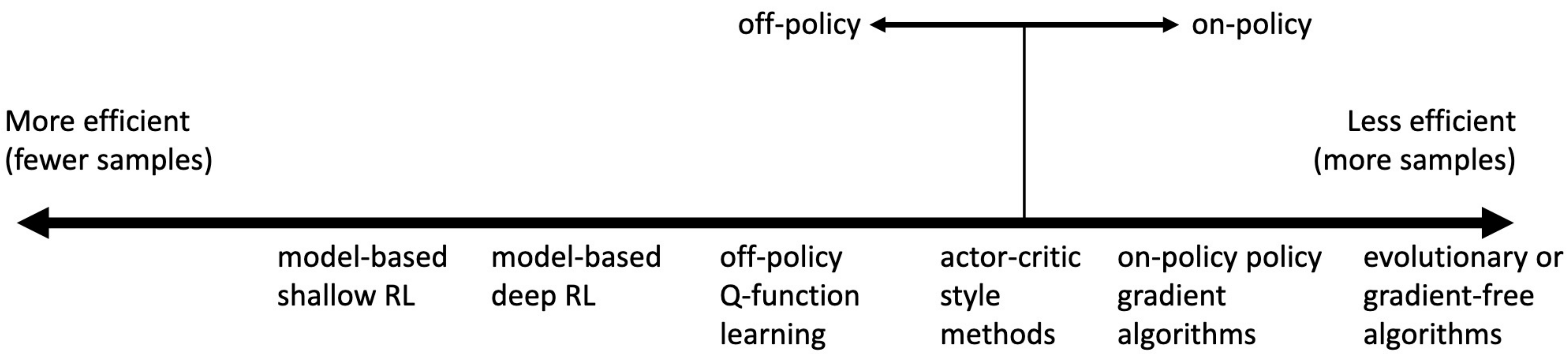


Chart from S. Levine

Sample Efficiency

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

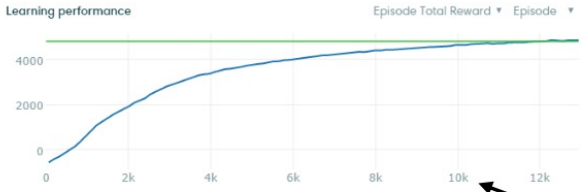
10x

model-based "shallow" RL
(e.g. PILCO)

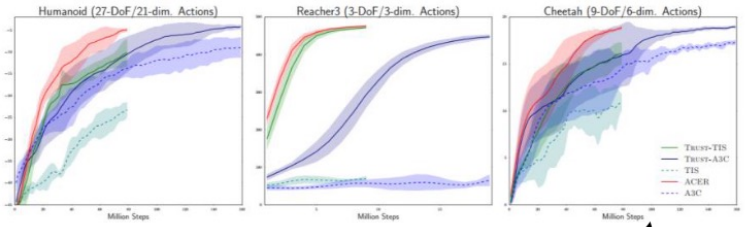
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans [1] Jonathan Ho [1] Xi Chen [1] Ilya Sutskever [1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

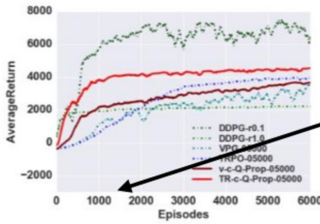


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

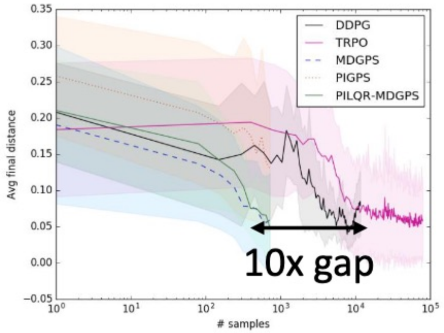
half-cheetah



Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)

	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	≈ 20 s	≈ 60 s–90 s	≈ 20 s–30 s
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$

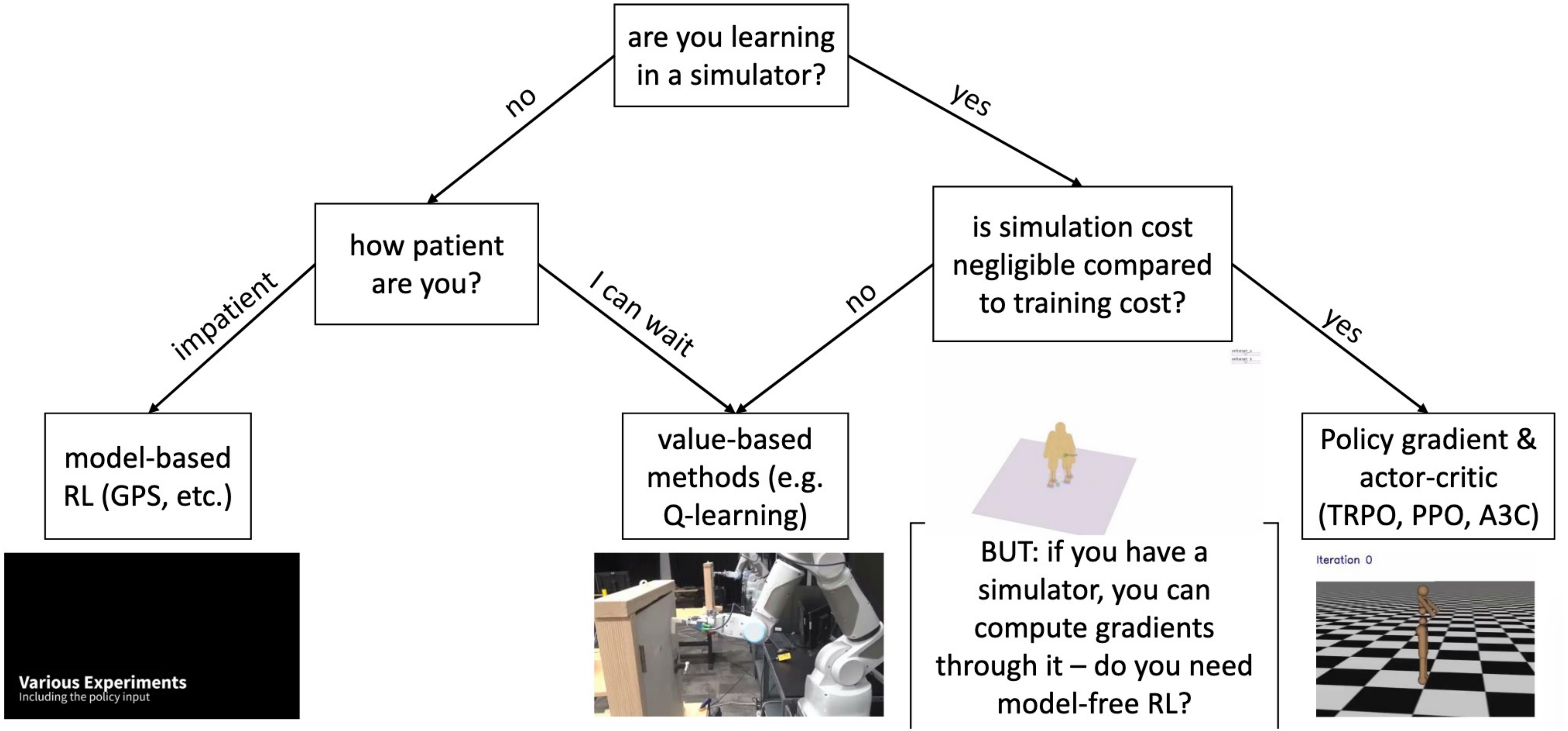


Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Slide from S. Levine

Choosing different RL methods

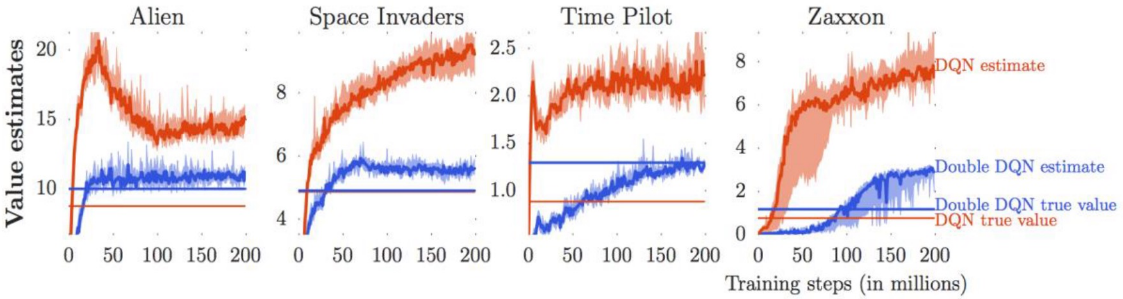


Choosing different methods

Tradeoffs:
Sample-efficiency
User-friendliness/stability

Assumptions about environment:
Continuous/discrete
Stochastic/deterministic

Where the main difficulty is:
Estimating model?
Estimating policy?
Obtaining samples?



Model-based RL

- Possible transfer between tasks
- Model can be harder to learn than policy
- Don’t directly optimize for the task at hand; no guarantee that better model will translate to better policy
- Typically more sample-efficient

Policy Gradient

- Directly optimizing task at hand
- Not sample efficient

Value Functions

- Minimize error, may not accurately represent real expected reward
- No convergence guarantees
- Can be quite sample efficient

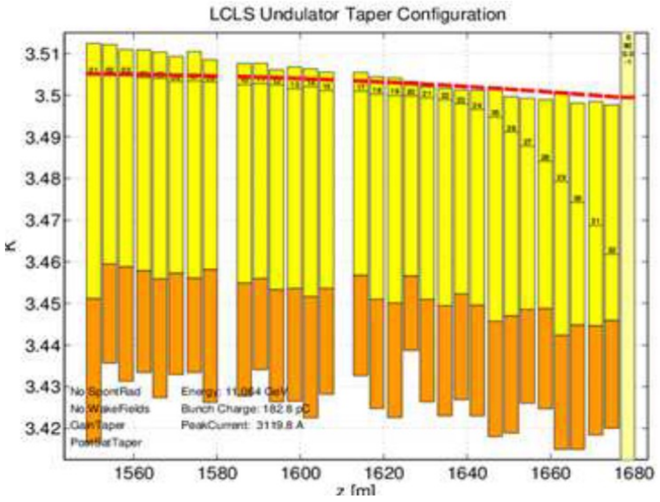
Easy policy, difficult model

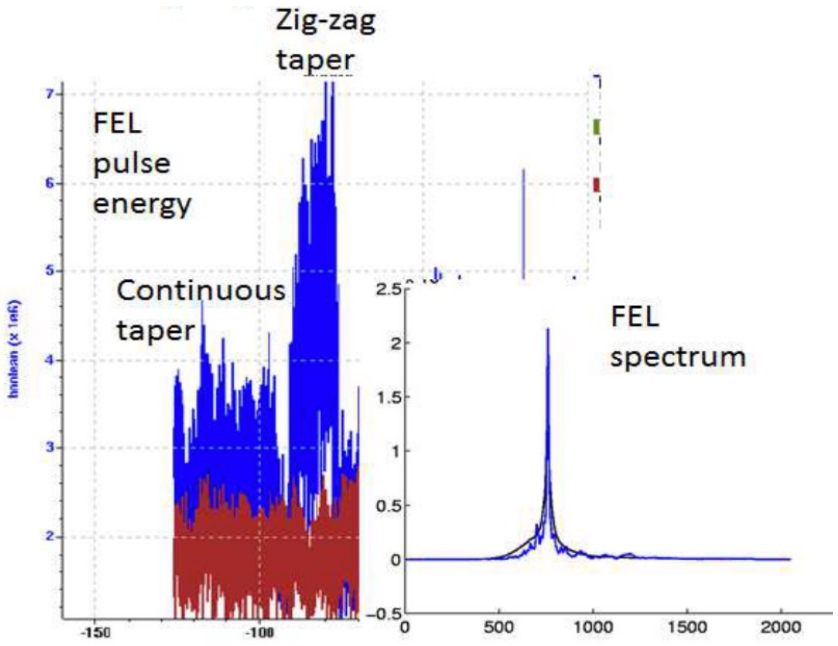
Example: FEL taper optimization

Wu et al., Recent Online Taper Optimization at LCLS, FEL’17
https://accelconf.web.cern.ch/fel2017/papers/tub04.pdf

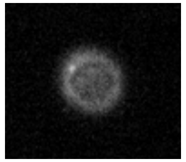
Compared a variety of optimization methods, including policy gradient RL

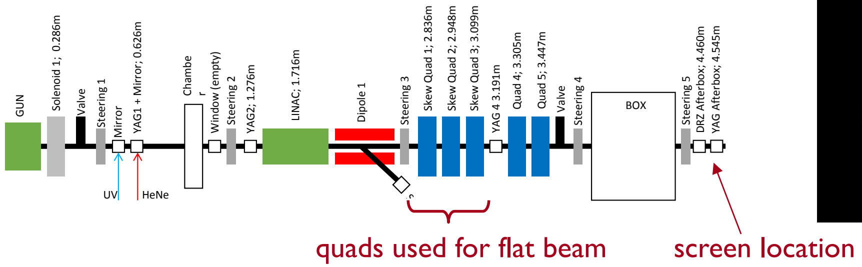
- Variables: taper magnets
- Target: FEL pulse energy
- RL found a “zig-zag” taper profile that had 2x pulse energy

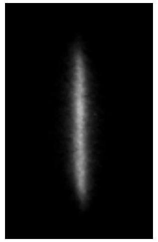




Example: offline training with a model





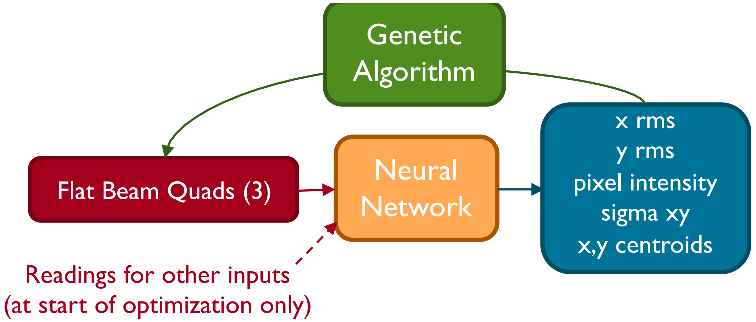


Expert hand-tuning:
10 – 20 minutes

Round-to-flat beam transforms are challenging to optimize

Took measured scan data at UCLA Pegasus beamline → trained neural network model to predict fits to beam image

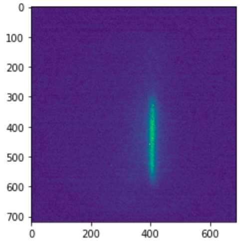
Tested online multi-objective optimization over model (3 quad settings) given present readings of other inputs



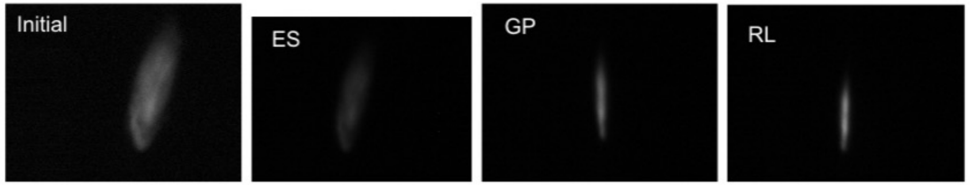
Also applied reinforcement learning (DDPG):

→ Trained offline using learned model

→ Transferred to machine for retraining (6 months later)



Beam result for one full day after last training data

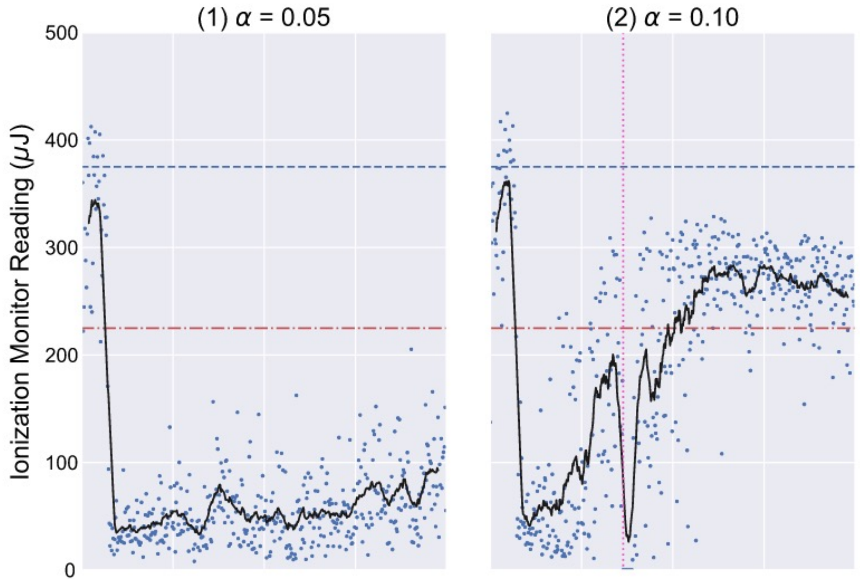


SLAC + UCLA collab: Cropp, in preparation

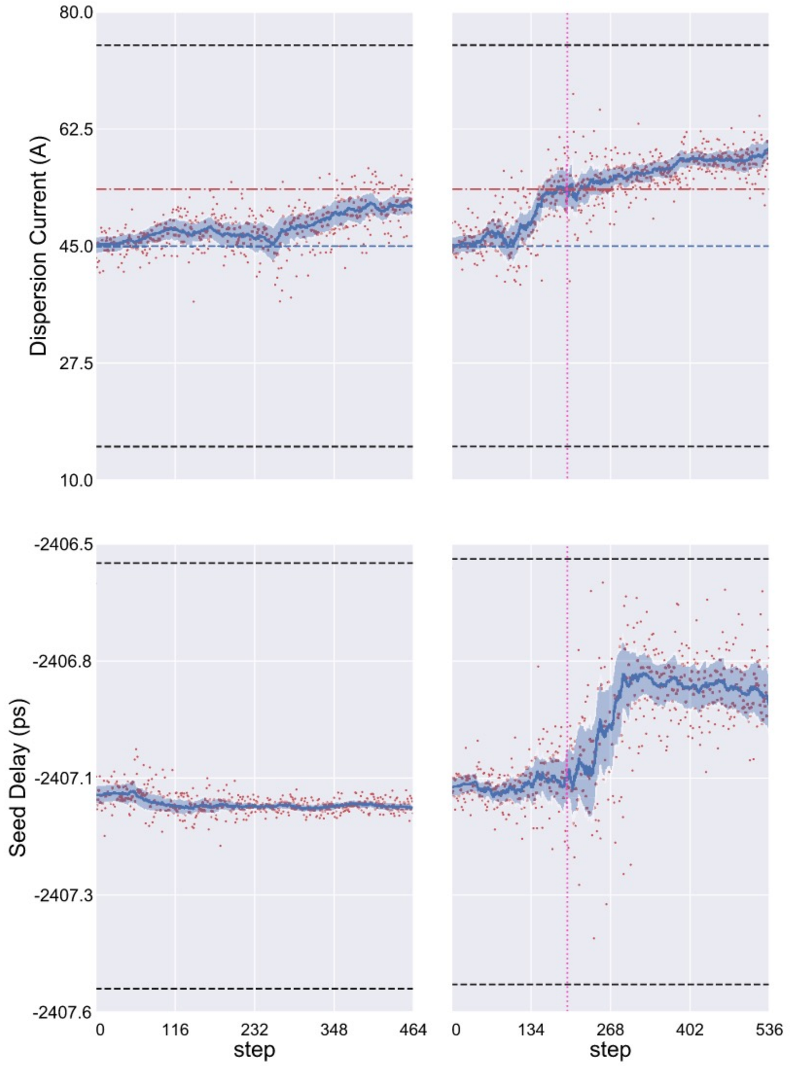
Example: HGHG FEL Optimization

F. O'Shea et al., Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the FERMI free-electron laser at Elettra, (2020)
https://journals.aps.org/prab/abstract/10.1103/PhysRevAccelBeams.23.122802

- Compared a variety of policy gradient methods for optimization and stabilization at FERMI for two tasks
- Settings: three kinds of magnets, piezo motors for laser alignment, and a mechanical delay stage for a seed laser
- Targets: the output energy of an HGHG FEL and the amount of Terahertz radiation produced
- Used same agent for the two different tasks



FEL energy



dispersive strength + laser delay
→ black lines are human settings

Example: Trajectory Control at CERN

Kain et al., Sample-efficient reinforcement learning for CERN accelerator control (2020)
https://journals.aps.org/prab/pdf/10.1103/PhysRevAccelBeams.23.124801

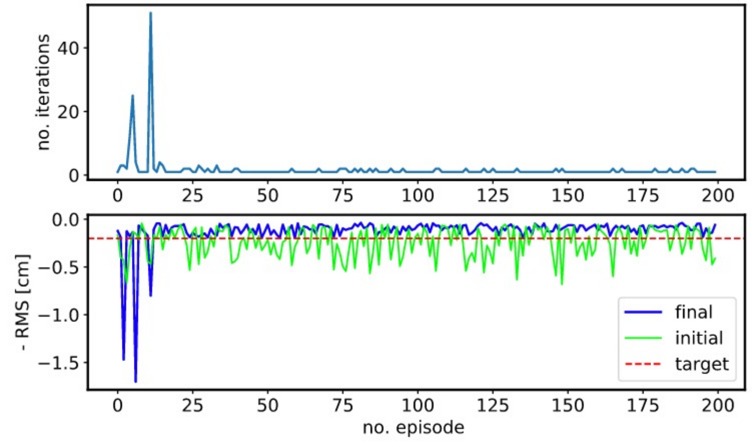
Aim: trajectory control for AWAKE and LINAC4

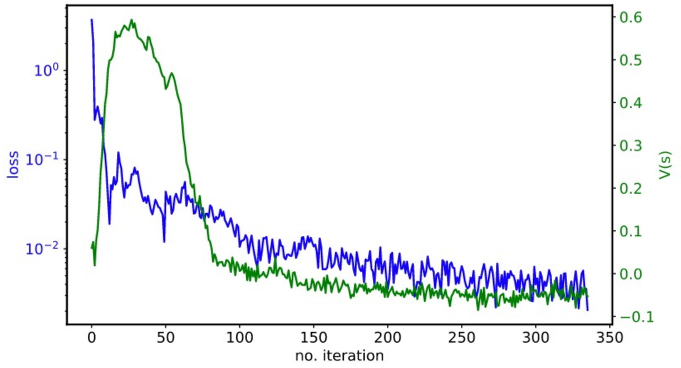
Used Normalized Advantage Function (Q-learning variant)

Setup for AWAKE :

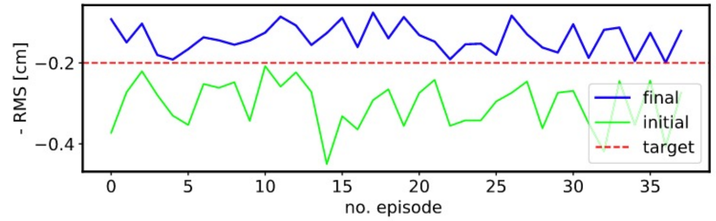
- 30 minutes training for 11 degrees of freedom 350 iterations
- Reset to random position at start of episode (no more than 7mm RMS offset – 2-3 x above normal)
- Limited corrector step size to 300urad

Tested agent 3 months later and still had good performance





online learning



3 months after last training

A Note on Reward Functions

- Reward functions may not account for un-intuitive behavior or implicit values
  - Classic example: *reduce office paper consumption* → *solution is to kill all humans*
- Big concern in AI safety, see https://openai.com/blog/concrete-ai-safety-problems/

https://youtu.be/tlOIHko8ySg

" **We assumed the score the player earned would reflect the informal goal of finishing the race**, so we included the game in an internal benchmark designed to measure the performance of reinforcement learning systems on racing games. **However, it turned out that the targets were laid out in such a way that the reinforcement learning agent could gain a high score without having to finish the course.** This led to some unexpected behavior when we trained an RL agent to play the game.

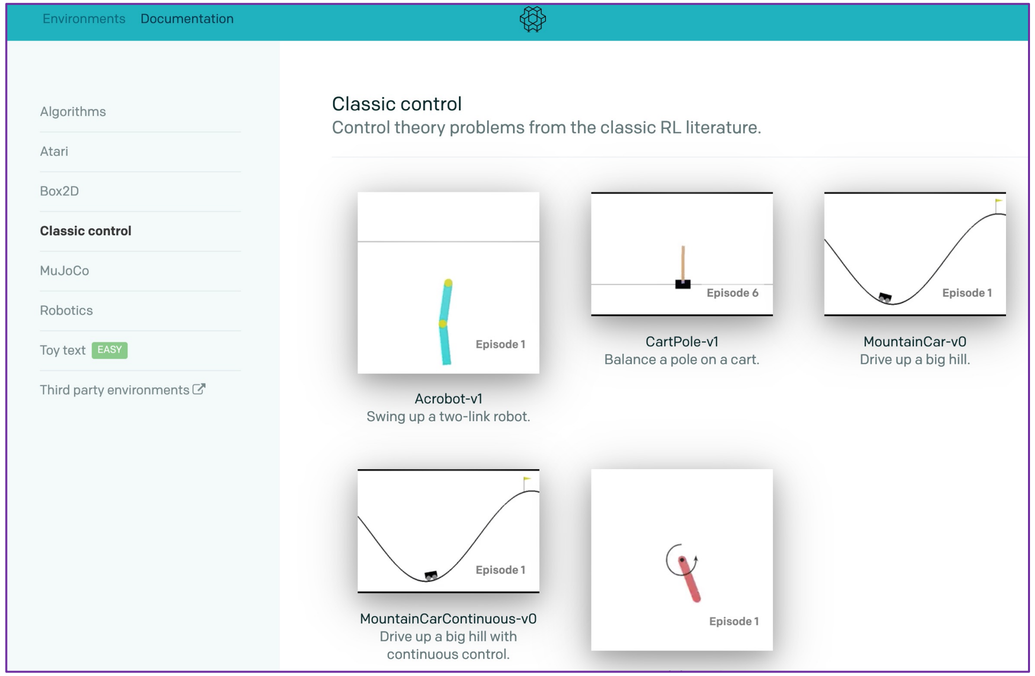
The RL agent finds an isolated lagoon where it can turn in a large circle and repeatedly knock over three targets, timing its movement so as to always knock over the targets just as they repopulate. **Despite repeatedly catching on fire, crashing into other boats, and going the wrong way on the track, our agent manages to achieve a higher score using this strategy than is possible by completing the course in the normal way**. Our agent achieves a score on average 20 percent higher than that achieved by human players."

https://openai.com/blog/faulty-reward-functions/

OpenAI gym

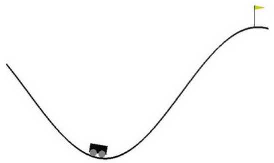
OpenAI gym has standards for interfacing with different environments and makes it easy to build your own environment: https://gym.openai.com/

Also has leaderboards with writeups of different solutions



MountainCarContinuous-v0

A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. Here, the reward is greater if you spend less energy to reach the goal Here, this is the continuous version.



- Environment details
- *MountainCarContinuous-v0 defines "solving" as getting average reward of 90.0 over 100 consecutive trials.*
- *This problem was first described by Andrew Moore in his PhD thesis [Moore90].*

User	Episodes before solve	Write-up	Video
Zhiqing Xiao	0 (use close-form preset policy)	writeup	
Ashioto	1	writeup	
Nextgrid.ai	9	writeup	Video
Keavnn	11	writeup	
camigord	18	writeup	

https://github.com/openai/gym/wiki/Leaderboard

Questions?

Classic Textbooks

- Miller, Werbos, Sutton, Neural Networks for Control, https://mitpress.mit.edu/books/neural-networks-control (1990)
- Bertsekas and Tsitsiklis, Neuro-dynamic Programming, http://athenasc.com/ndpbook.html (1996)
- Sutton and Barto, Reinforcement Learning: An Introduction, http://incompleteideas.net/book/the-book-2nd.html (1996, 2018)