Lecture 18: Reinforcement Learning

Administrative

- Grades: Midterm and A3 grades are out, and regrade portals will be open until June 1.
- Assignments
  - A5 due June 5. Required for Grad version.
  - Project report due June 8
  - Project poster: June 8 10:30am - 12:20pm at Allen Atrium

So far… Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

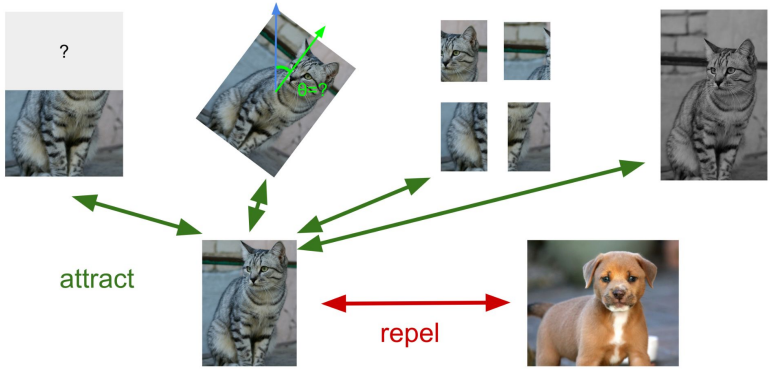
Classification

So far… Self-supervised Learning

Data: x
Just data, no labels!

Goal: Define proxy tasks to uncover useful representations from the data

Examples: SimClr, MoCo, Dino, etc.

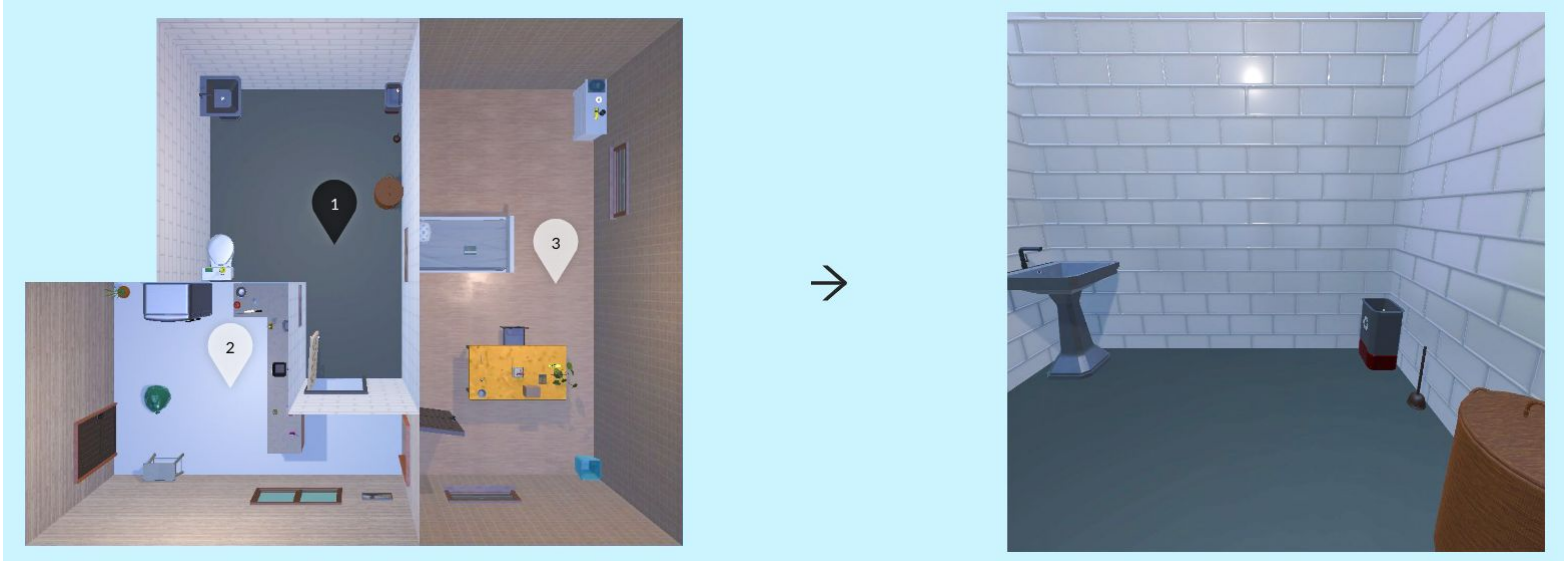


2-d density images left and right are CC0 public domain

Why Supervised Learning Is Not Enough

- **The environment is not differentiable**: robots, games, and other real systems are not differentiable functions, so we cannot train through them with backpropagation
- **There are no correct labels**: the learner is not told the right answer, only how good or bad an outcome was
- **Feedback is sparse and delayed**: success or failure is often only known after many actions, making it hard to tell which action mattered
- **Actions change what comes next**: each decision affects future inputs, so the data the learner sees depends on its own behavior
- **The learner must collect its own data**: there is no fixed dataset; the learner has to try things in the environment to find out what works

From passive images of active observations: Embodied AI

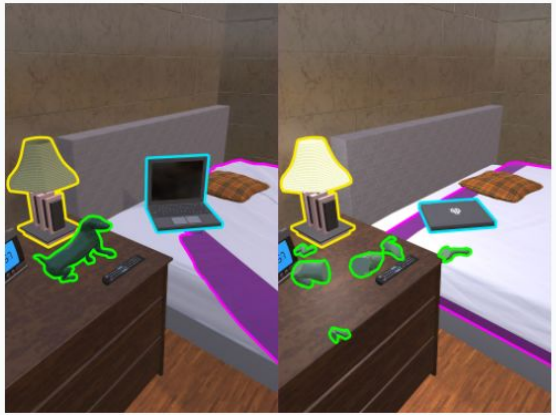


Overhead view of photo-realistic synthetic environments

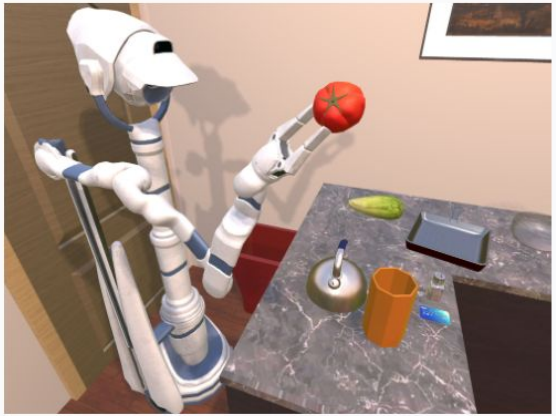
What an agent sees

Deitke et al. “ProcTHOR: Large-Scale Embodied AI Using Procedural Generation” NeurIPS 2022

From object recognition to object manipulation: Embodied AI



Objects in 3D spaces



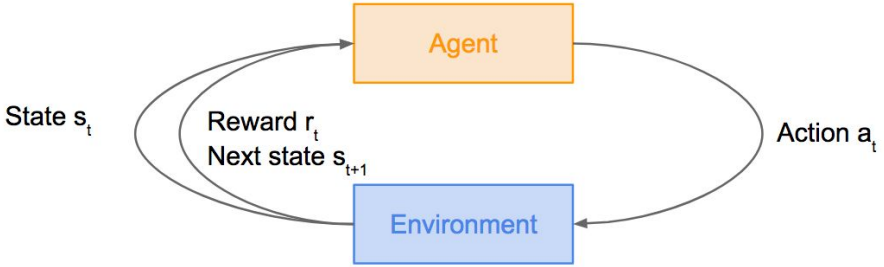
Manipulating objects

Multi-agent collaboration

Deitke et al. “ProcTHOR: Large-Scale Embodied AI Using Procedural Generation” NeurIPS 2022

Today: Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals



Goal: Learn how to take actions in order to maximize reward

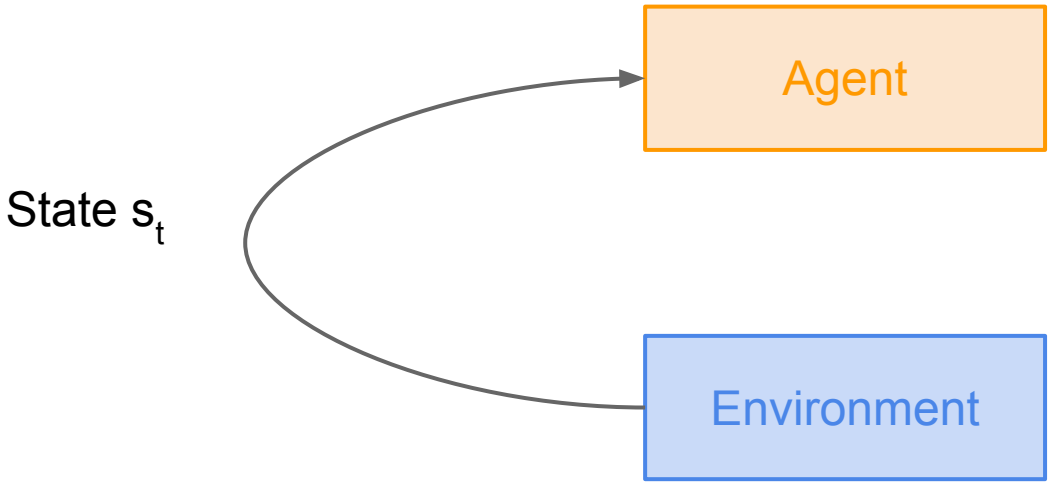
Atari games figure copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Overview

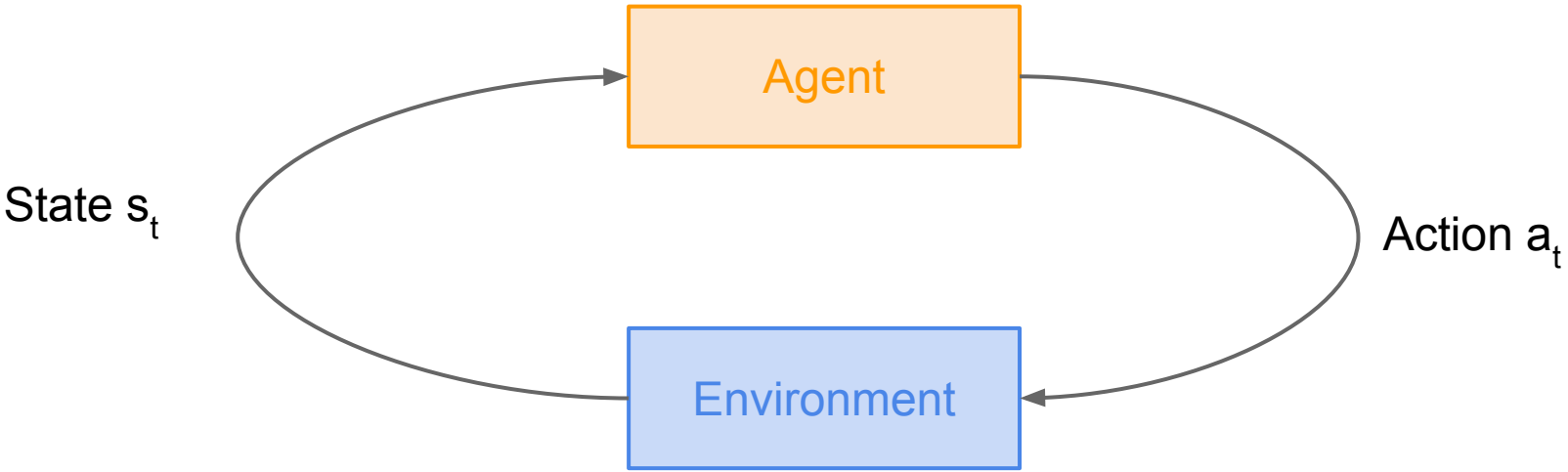
- What is Reinforcement Learning?
- Markov Decision Processes
- Behavior Cloning (imitation learning)
- Policy Gradient

Reinforcement Learning

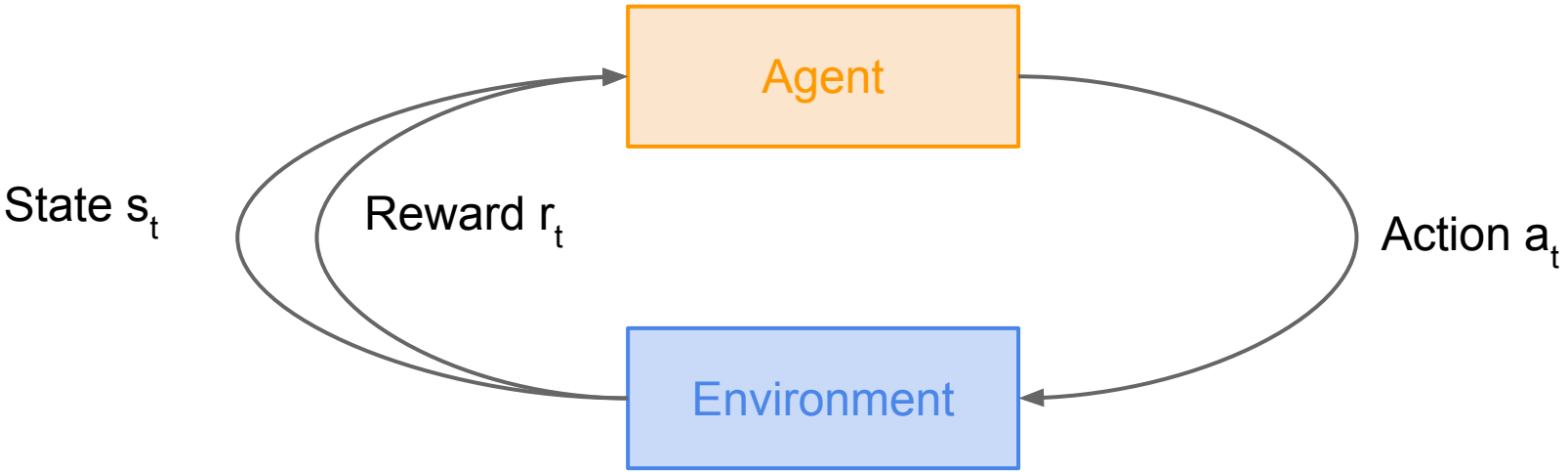
Reinforcement Learning



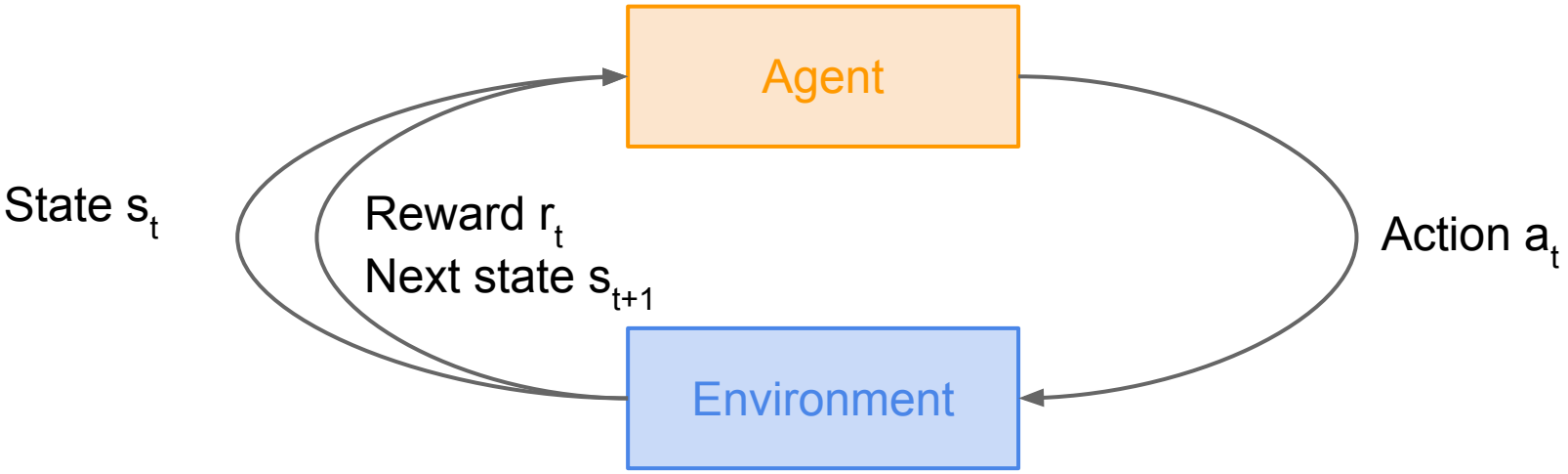
Reinforcement Learning



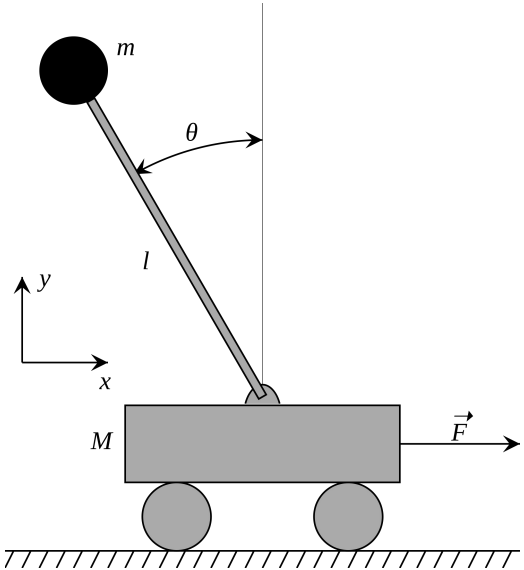
Reinforcement Learning



Reinforcement Learning



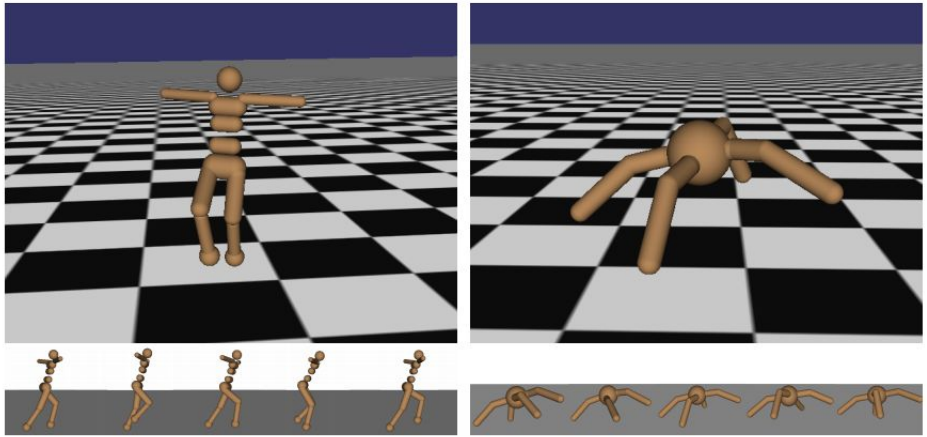
Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Robot Locomotion



Objective: Make the robot move forward

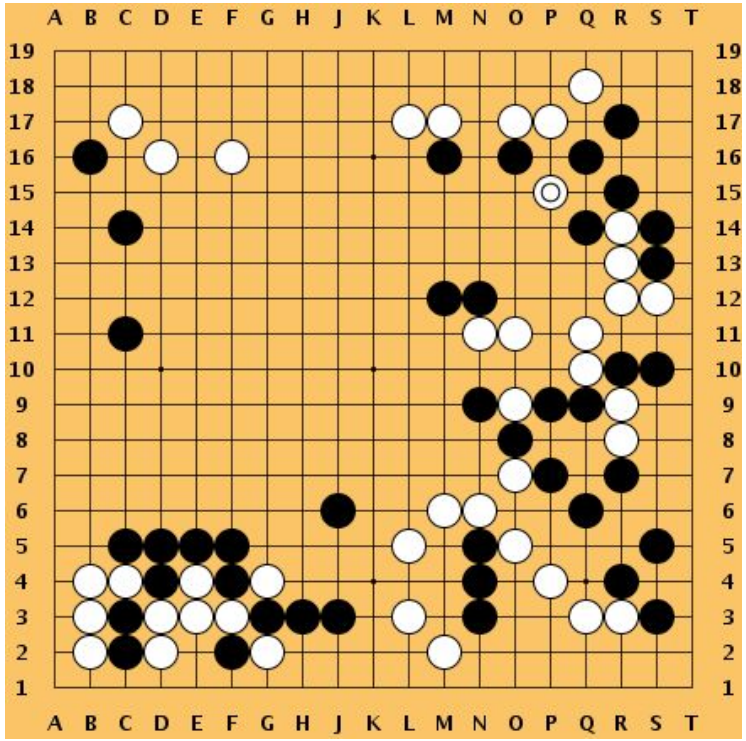
State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

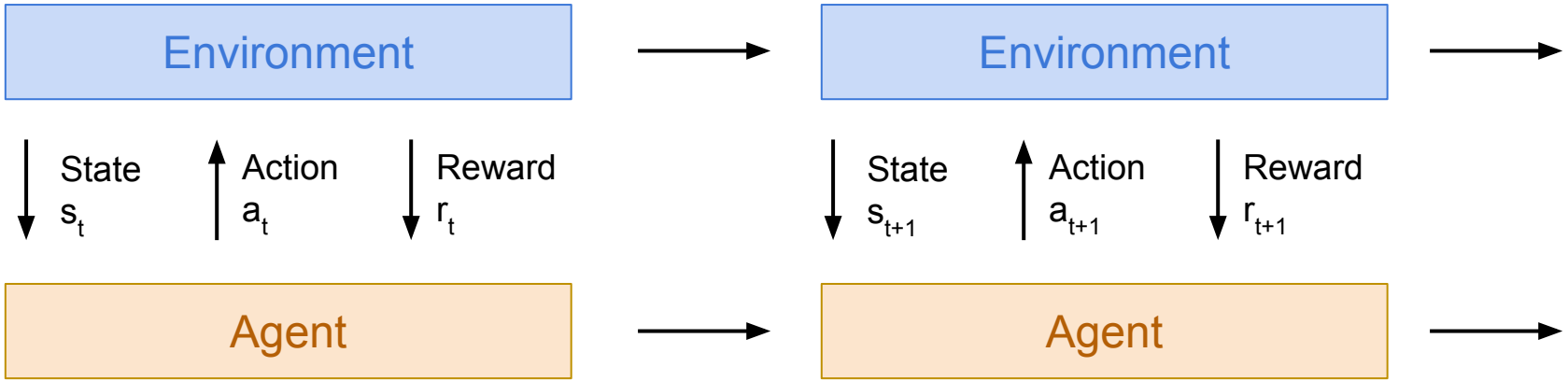
Go



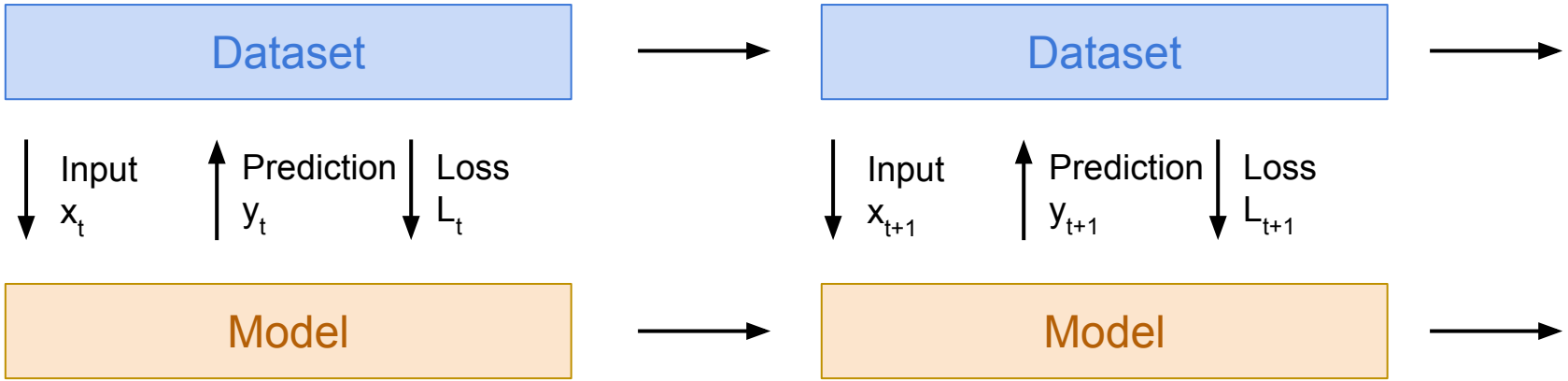
Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Reinforcement Learning vs Supervised Learning

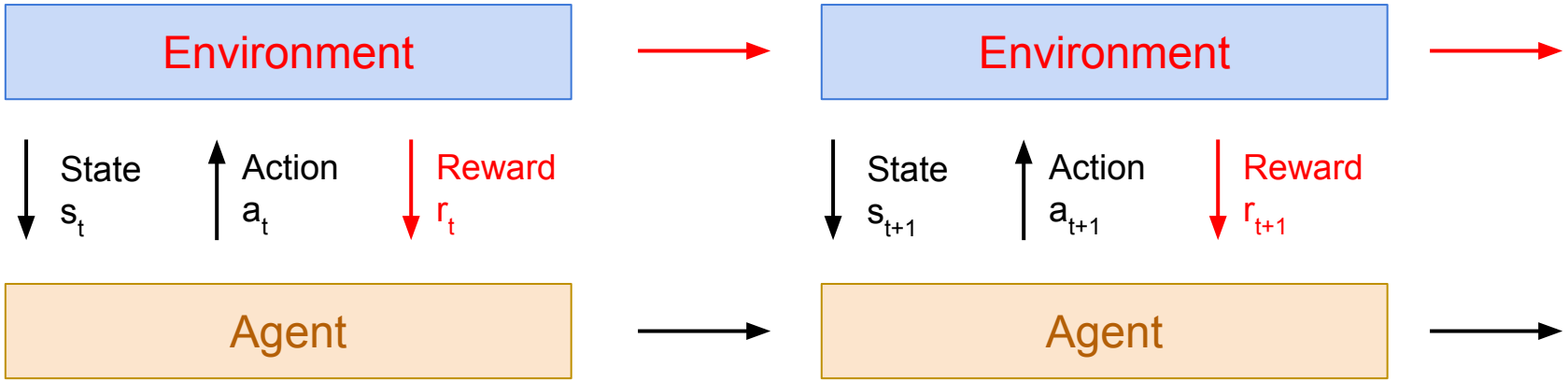


Reinforcement Learning vs Supervised Learning



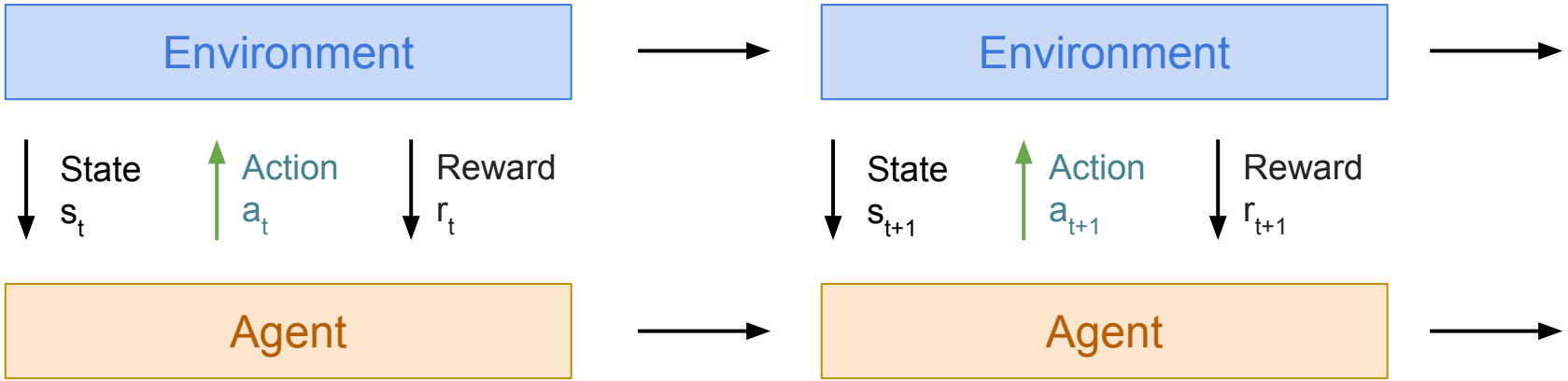
How is supervised learning different from reinforcement learning?

Reinforcement Learning vs Supervised Learning



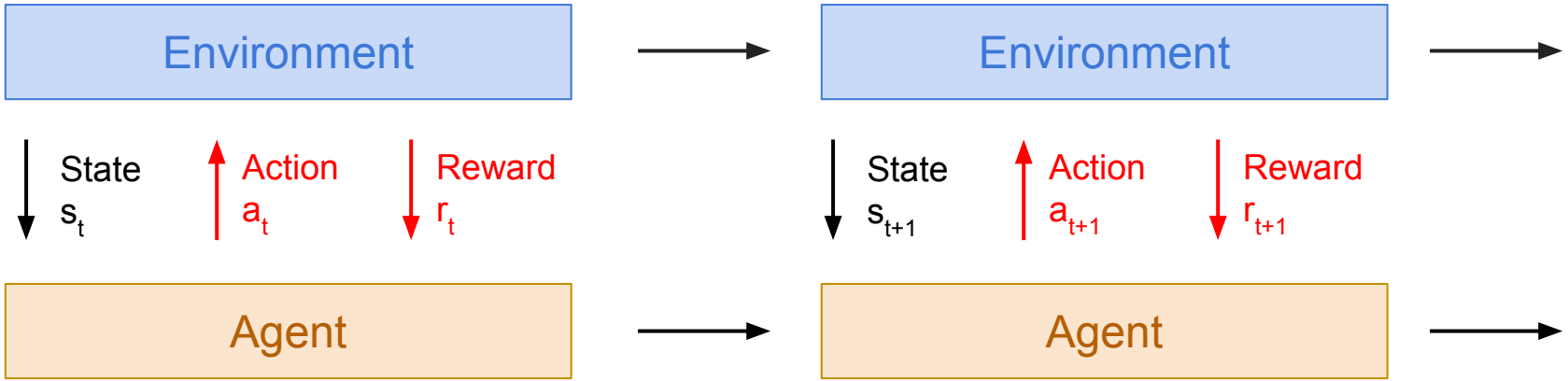
Stochasticity: Rewards and state transitions may be random

Reinforcement Learning vs Supervised Learning



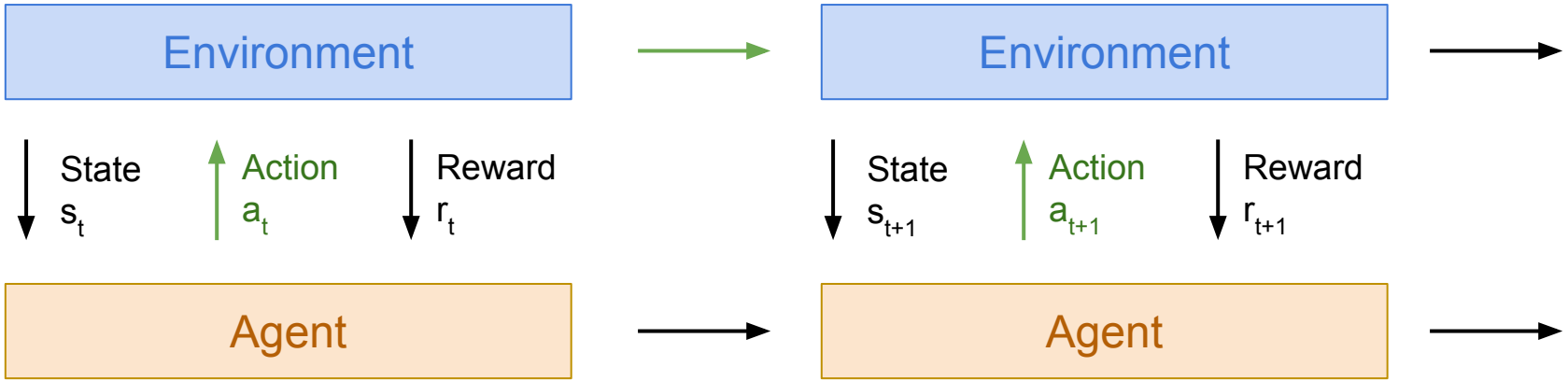
Credit assignment: Reward r_t may not directly depend on action a_t

Reinforcement Learning vs Supervised Learning



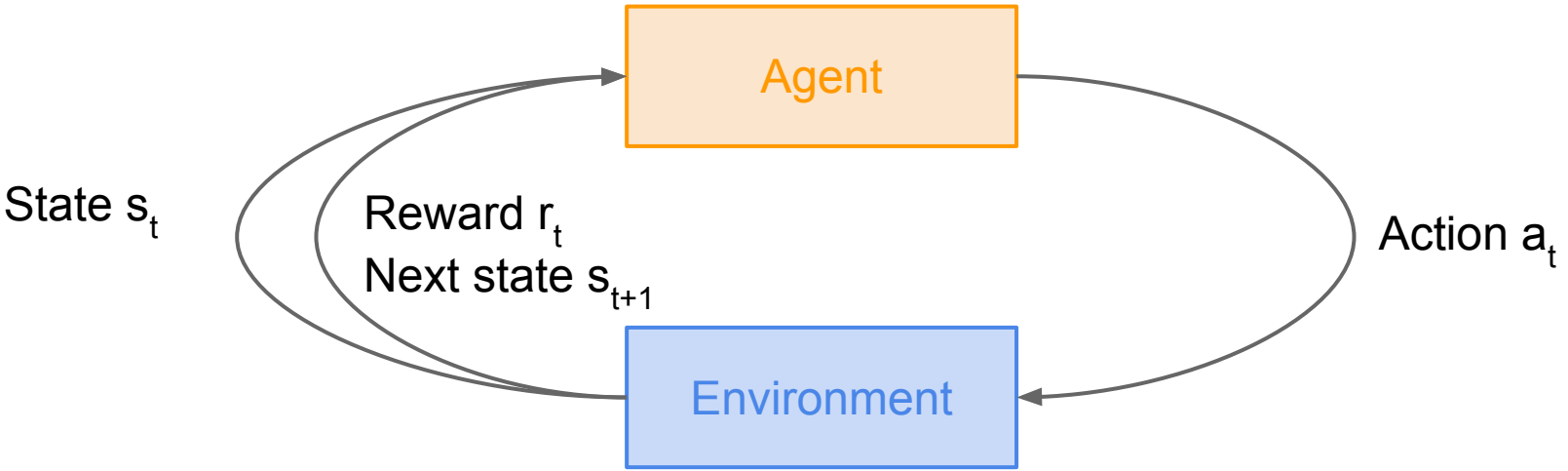
Nondifferentiable: Can't backprop through world; can't compute dr_t/da_t

Reinforcement Learning vs Supervised Learning



Nonstationary: What the agent experiences depends on how it acts

How can we mathematically formalize the RL problem?



Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

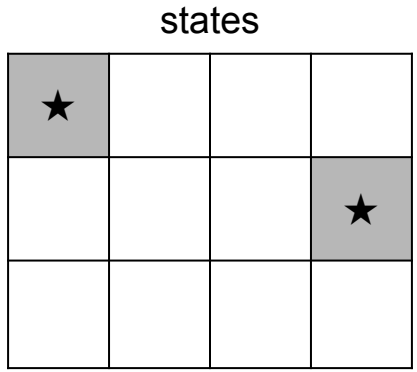
- A policy **π** is a function from S to A that specifies what action to take in each state
- **Objective**: find policy **π*** that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

A simple MDP: Grid World

actions = {

1. right
2. left
3. up
4. down

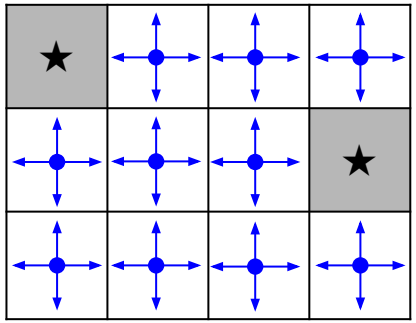
}



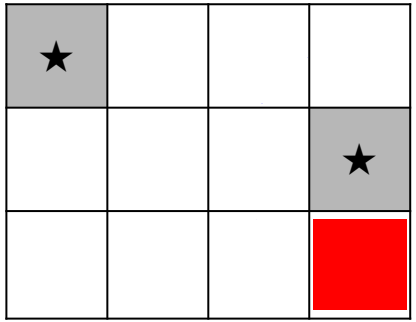
Set a negative “reward” for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

A simple MDP: Grid World

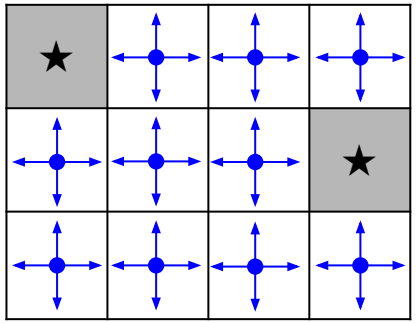


Random Policy

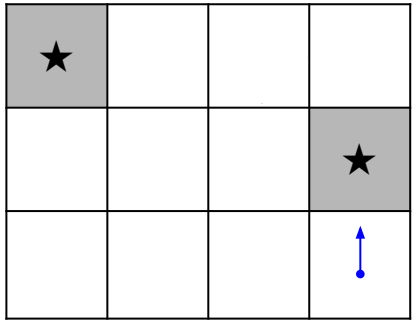


Optimal Policy

A simple MDP: Grid World

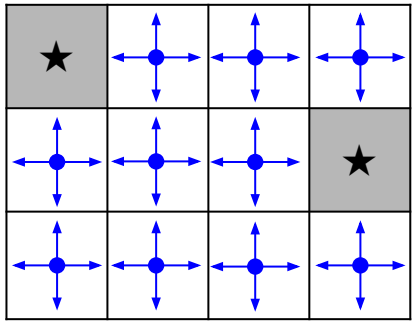


Random Policy

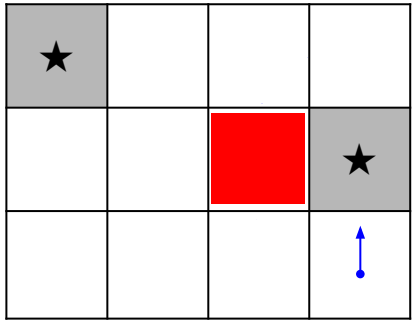


Optimal Policy

A simple MDP: Grid World

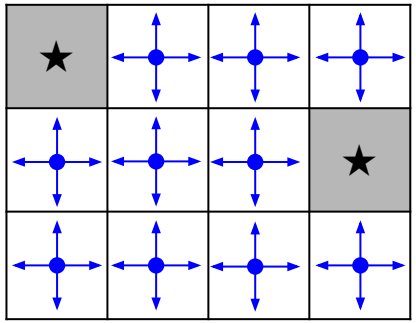


Random Policy

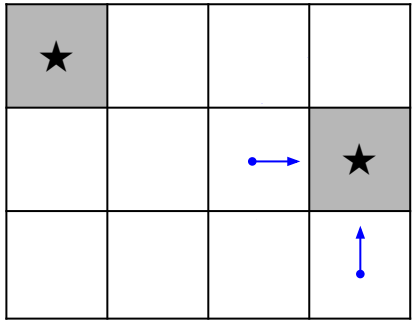


Optimal Policy

A simple MDP: Grid World

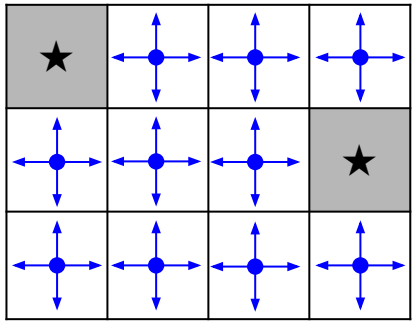


Random Policy

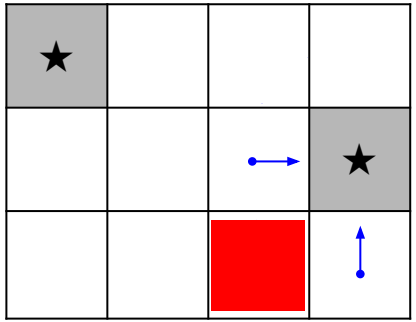


Optimal Policy

A simple MDP: Grid World

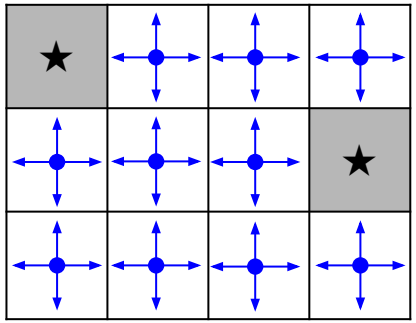


Random Policy

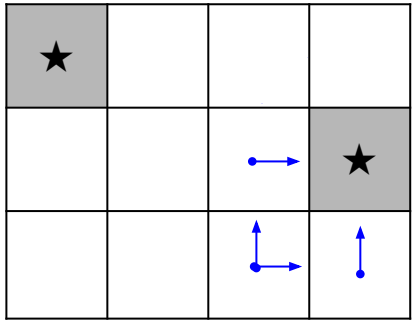


Optimal Policy

A simple MDP: Grid World

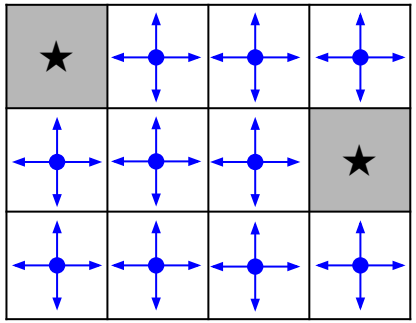


Random Policy

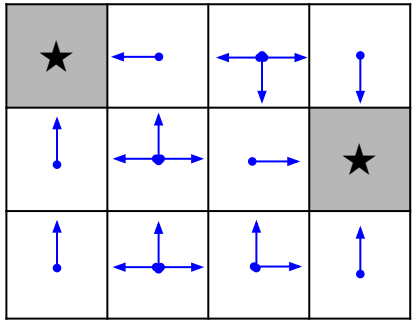


Optimal Policy

A simple MDP: Grid World



Random Policy



Optimal Policy

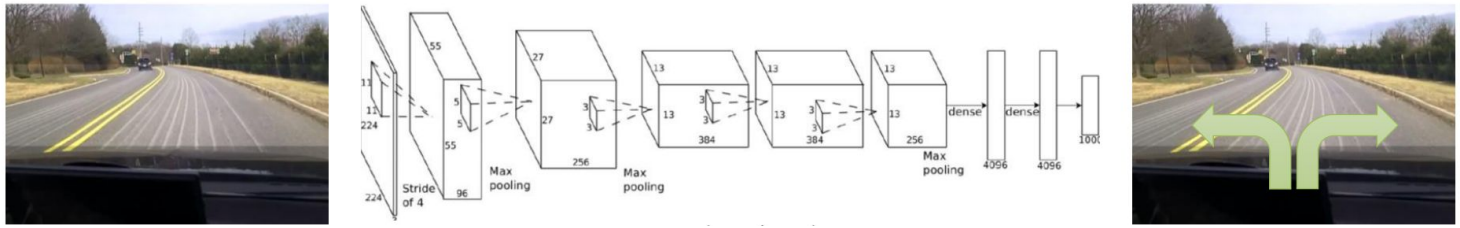
The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?

Idea 1: Let's ignore multiple time steps

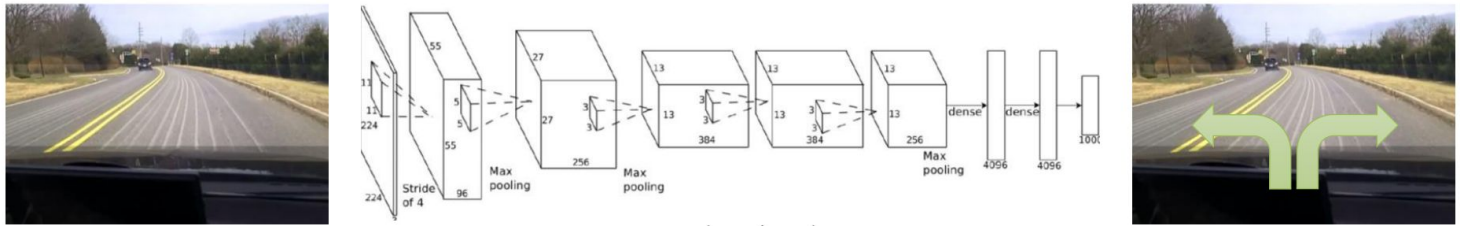
Treat the problem like a supervised learning problem at any time step t:



Driving policy: Map input observations to steering wheel angles

Idea 1: Let's ignore multiple time steps

Treat the problem like a supervised learning problem at any time step t:

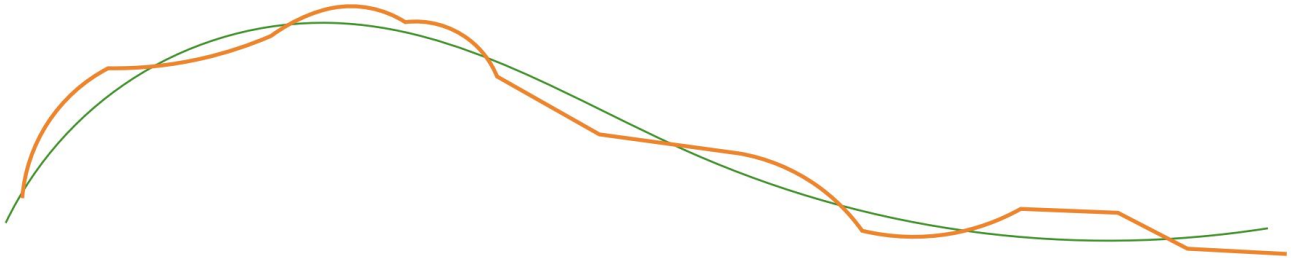


Driving policy: Map input observations to steering wheel angles

This is called Behavior Cloning

Challenges with Behavior cloning: Compounding errors

During training, the agent wakes up at time step t on a state drawn from the data distribution of the expert trajectories, and executes an action.

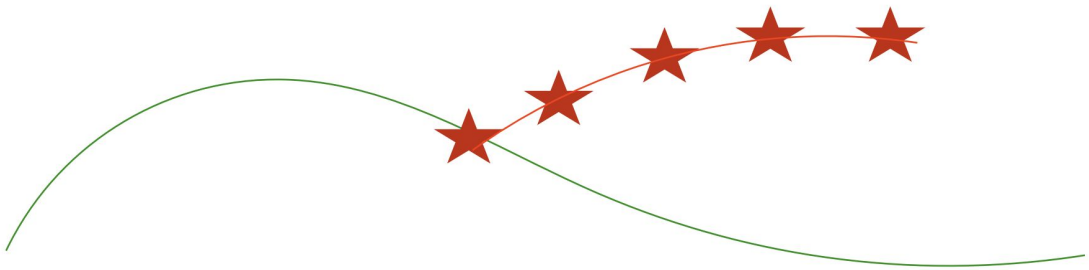


Problem: the agent never sees behavior that differs from experts

Expert trajectory
Agent's trajectory

Challenges with Behavior cloning: Compounding errors

During training, the agent wakes up at time step t on a state drawn from the data distribution of the expert trajectories, and executes an action.

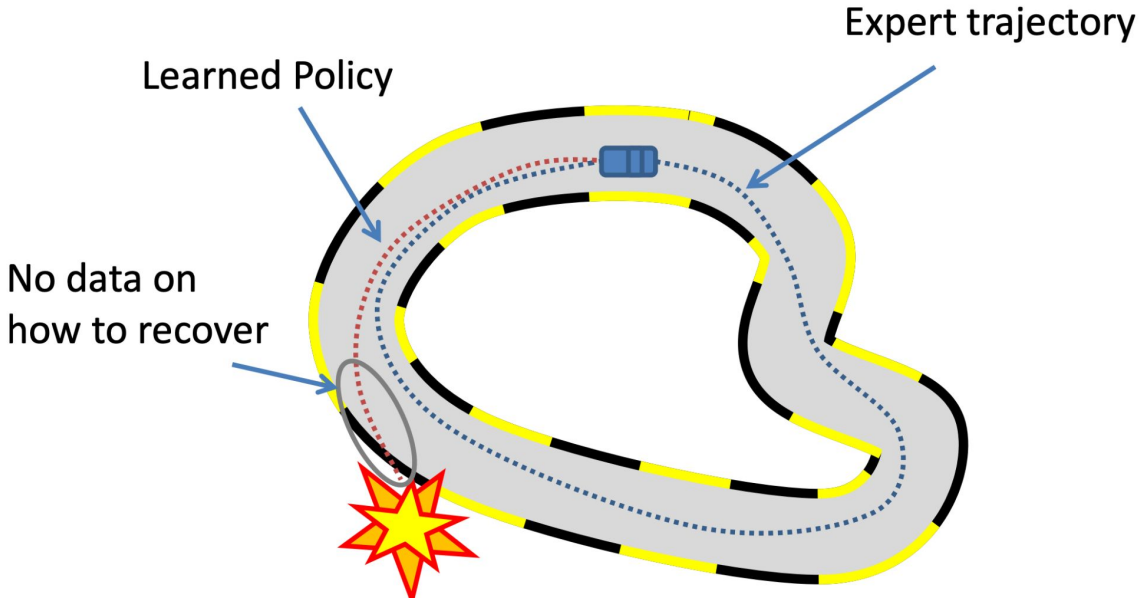


During testing, the agent will diverge from experts and behave unpredictably

Expert trajectory
Agent's trajectory

Challenges with Behavior cloning: Compounding errors

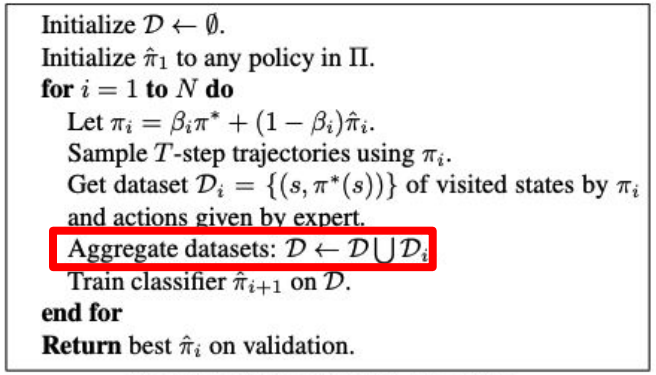
There is no way to recover from bad behavior.



Solution: data augmentation

Two ways of data augmentation:

- **Option 1:** Dagger (Dataset aggregation): Aggregate the initial dataset with additional data sampled from the current policy
  - (see Ross et al. cited below for details)
- **Option 2:** Use synthetic environments to programmatically generate millions of expert trajectories
  - (see Kiana et al. SPOC below for details)



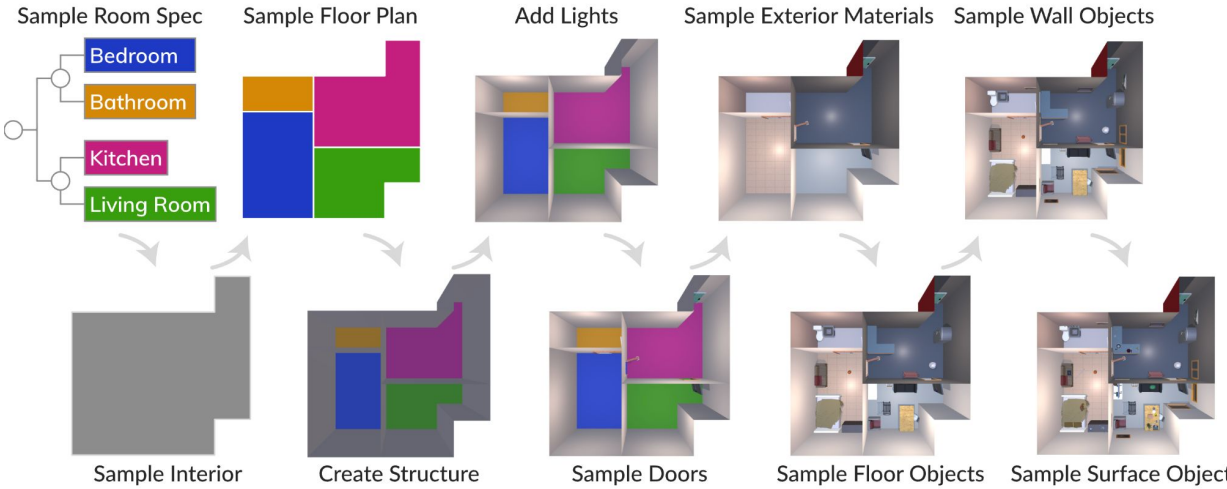
Algorithm 3.1: DAGGER Algorithm.

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning." *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011.

Ehsani, Kiana, et al. "Imitating Shortest Paths in Simulation Enables Effective Navigation and Manipulation in the Real World." arXiv preprint arXiv:2312.02976 (2023). [link to project]

SPOC: Shortest Path Oracle Clone

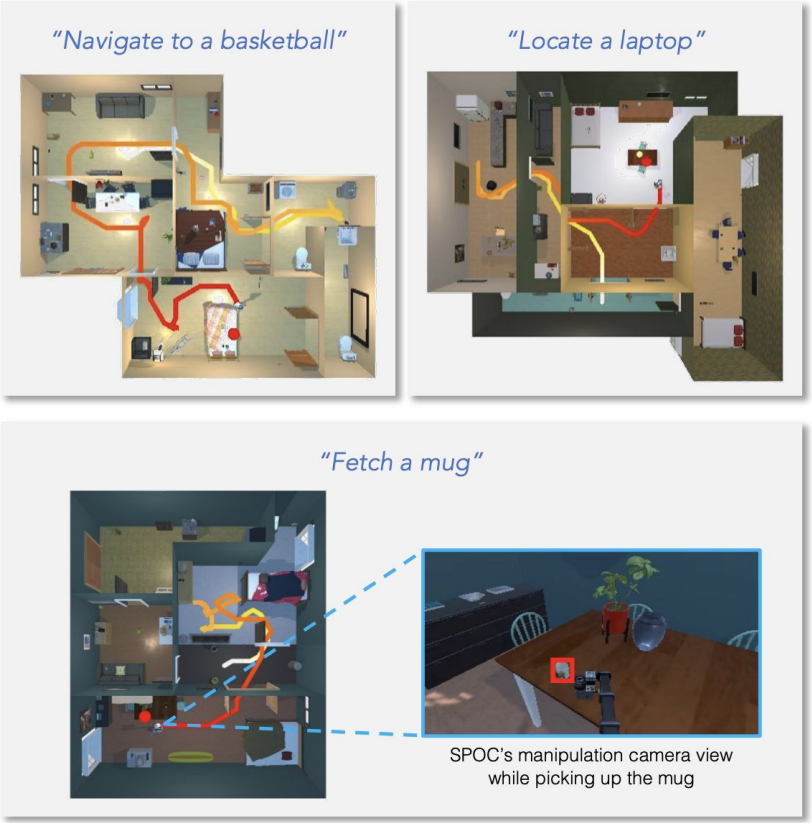
Step 1: programmatically generate millions of houses, populated with light sources, furniture, objects, etc.



Deitke, Matt, et al. "🏘 ProcTHOR: Large-Scale Embodied AI Using Procedural Generation." Advances in Neural Information Processing Systems 35 (2022): 5982-5994.

SPOC: Shortest Path Oracle Clone

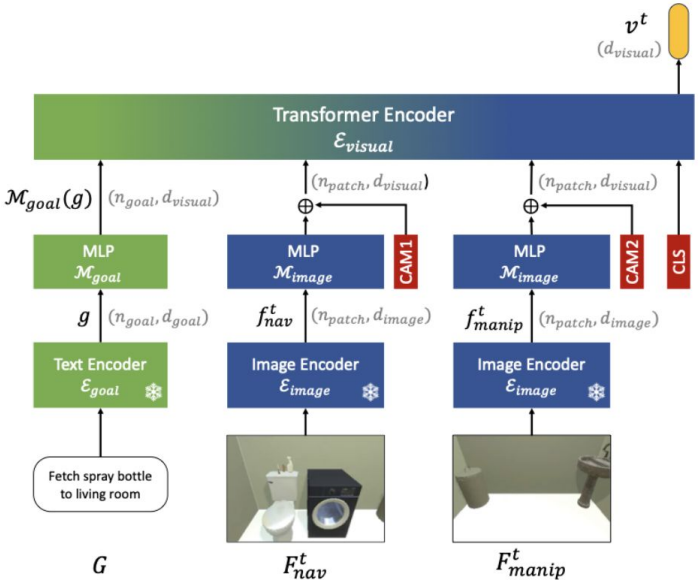
Step 2: Programmatically generate thousands of expert trajectories across all the houses

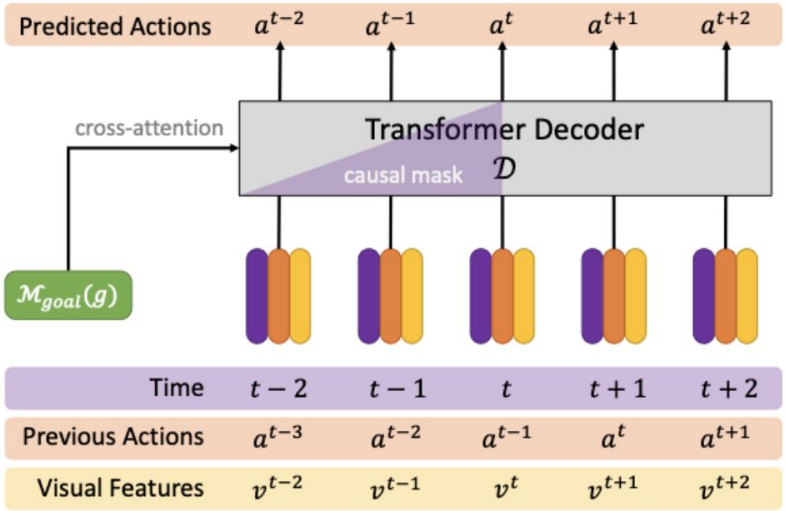


Ehsani, Kiana, et al. "Imitating Shortest Paths in Simulation Enables Effective Navigation and Manipulation in the Real World." arXiv preprint arXiv:2312.02976 (2023). [link to project]

SPOC: Shortest Path Oracle Clone

Step 3: Train a behavior cloning model using the data





Ehsani, Kiana, et al. "Imitating Shortest Paths in Simulation Enables Effective Navigation and Manipulation in the Real World." arXiv preprint arXiv:2312.02976 (2023). [link to project]

SPOC: Shortest Path Oracle Clone

Step 4: Test on new tasks in the real world

Ehsani, Kiana, et al. "Imitating Shortest Paths in Simulation Enables Effective Navigation and Manipulation in the Real World." arXiv preprint arXiv:2312.02976 (2023). [link to project]

Challenges with Behavior cloning: Unable to planning

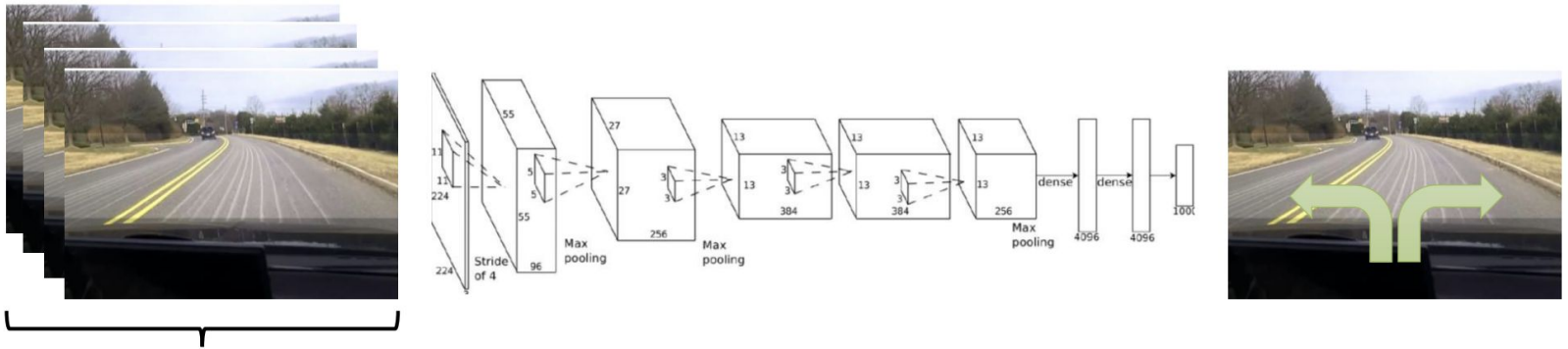
$$\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$$

behavior depends only on current observation

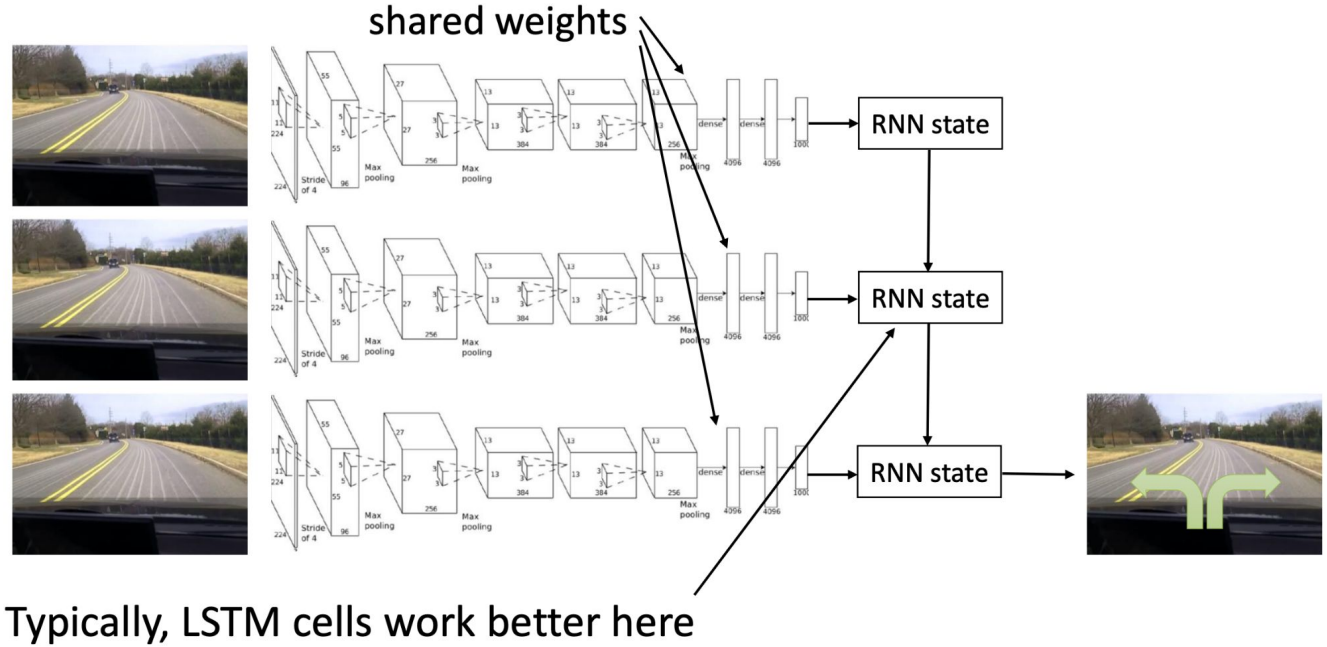
$$\pi_\theta(\mathbf{u}_t|\mathbf{o}_1, \ldots, \mathbf{o}_t)$$

behavior depends on all past observations

Solution: Use an RNN



Challenges with Behavior cloning: Unable to planning



Next: Policy gradient

A more principled approach.

Behavior cloning is maximizing expert behavior.
But what are the experts doing?

They are trying to maximize rewards.
Can we derive a policy to maximize rewards?

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?

Goal: Find the policy that maximizes the **expected sum of rewards:**

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi\right]$$

$$s_0 \sim p(s_0)$$
$$a_t \sim \pi(a \mid s_t)$$
$$s_{t+1} \sim P(s \mid s_t, a_t)$$

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value as this **objective function**:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value as this **objective function**:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Q. How can we do this?

Policy Gradients

Formally, let's define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Q. How can we do this?

Ideally, we would use gradient ascent on policy parameters!

REINFORCE algorithm

Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

REINFORCE algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $$\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

REINFORCE algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

Intractable! Gradient of an expectation is problematic when p depends on θ

REINFORCE algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$

REINFORCE algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_{\tau} \left(r(\tau) \nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta) \mathrm{d}\tau$$

$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau) \nabla_\theta \log p(\tau;\theta)\right]$$

Can estimate with Monte Carlo sampling

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

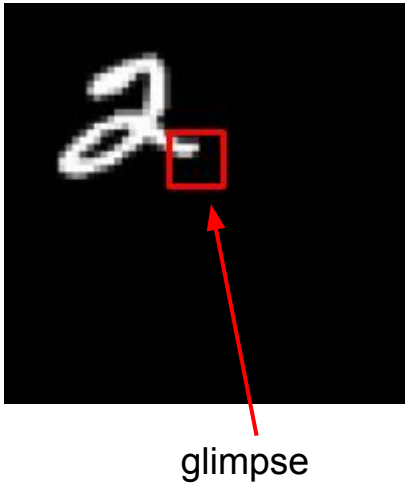
REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



[Mnih et al. 2014]

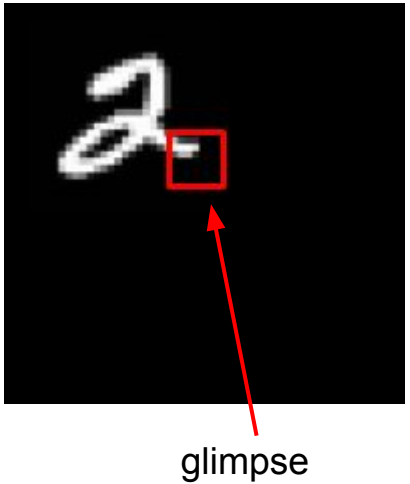
REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

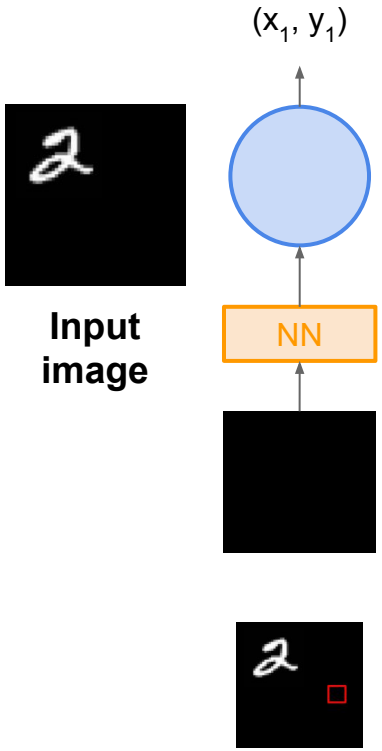
State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

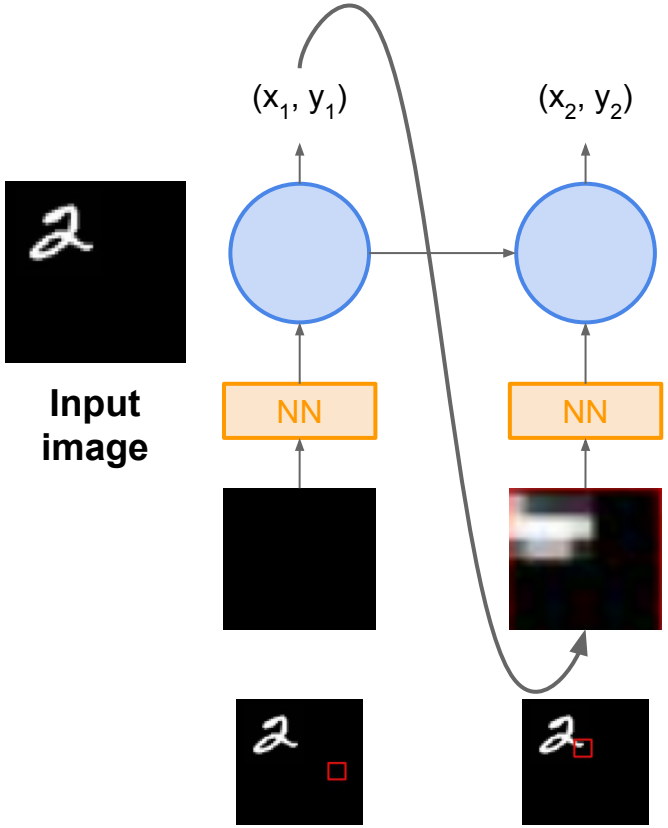
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



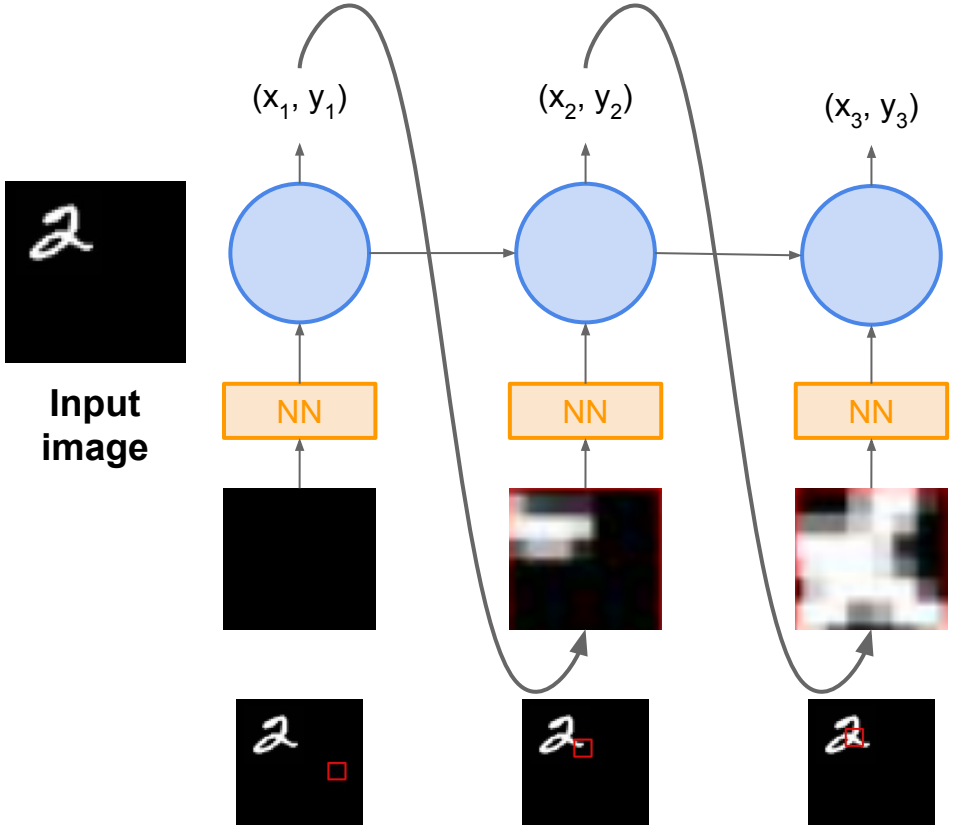
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



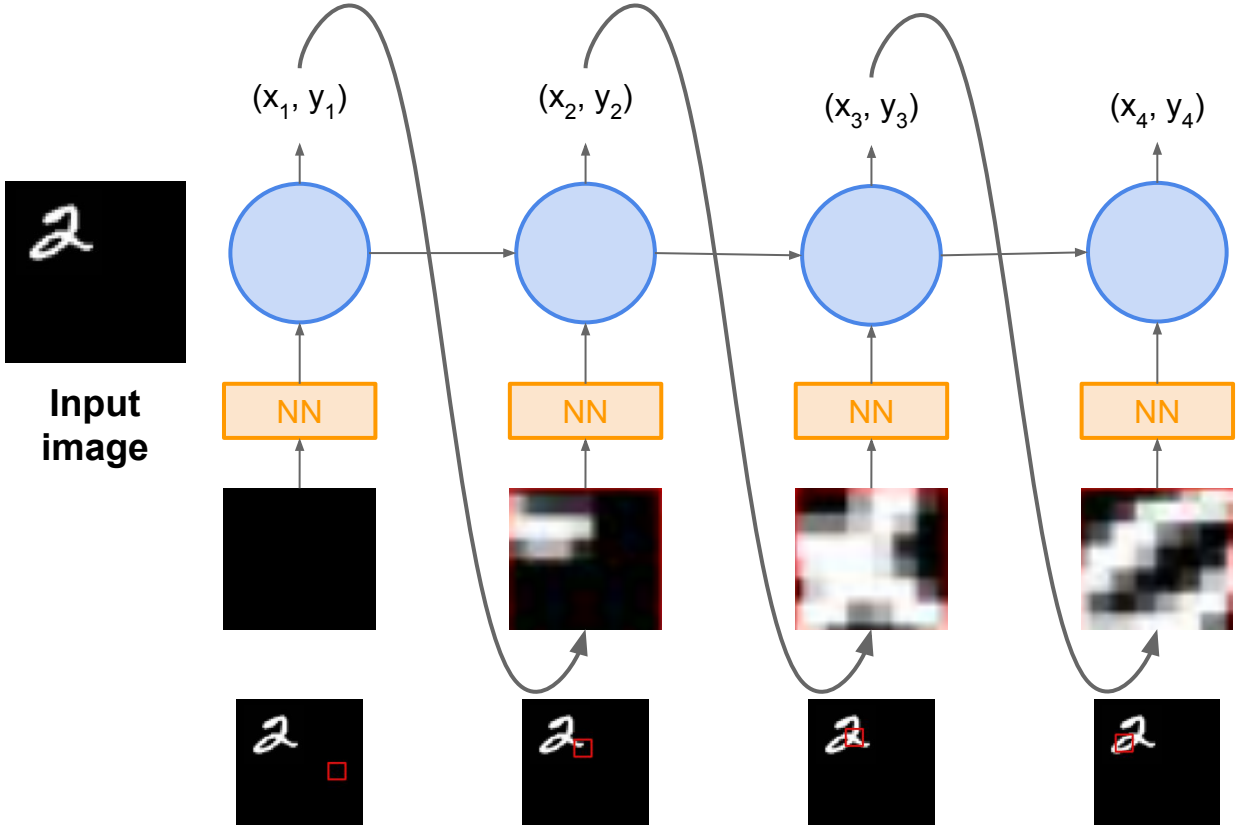
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



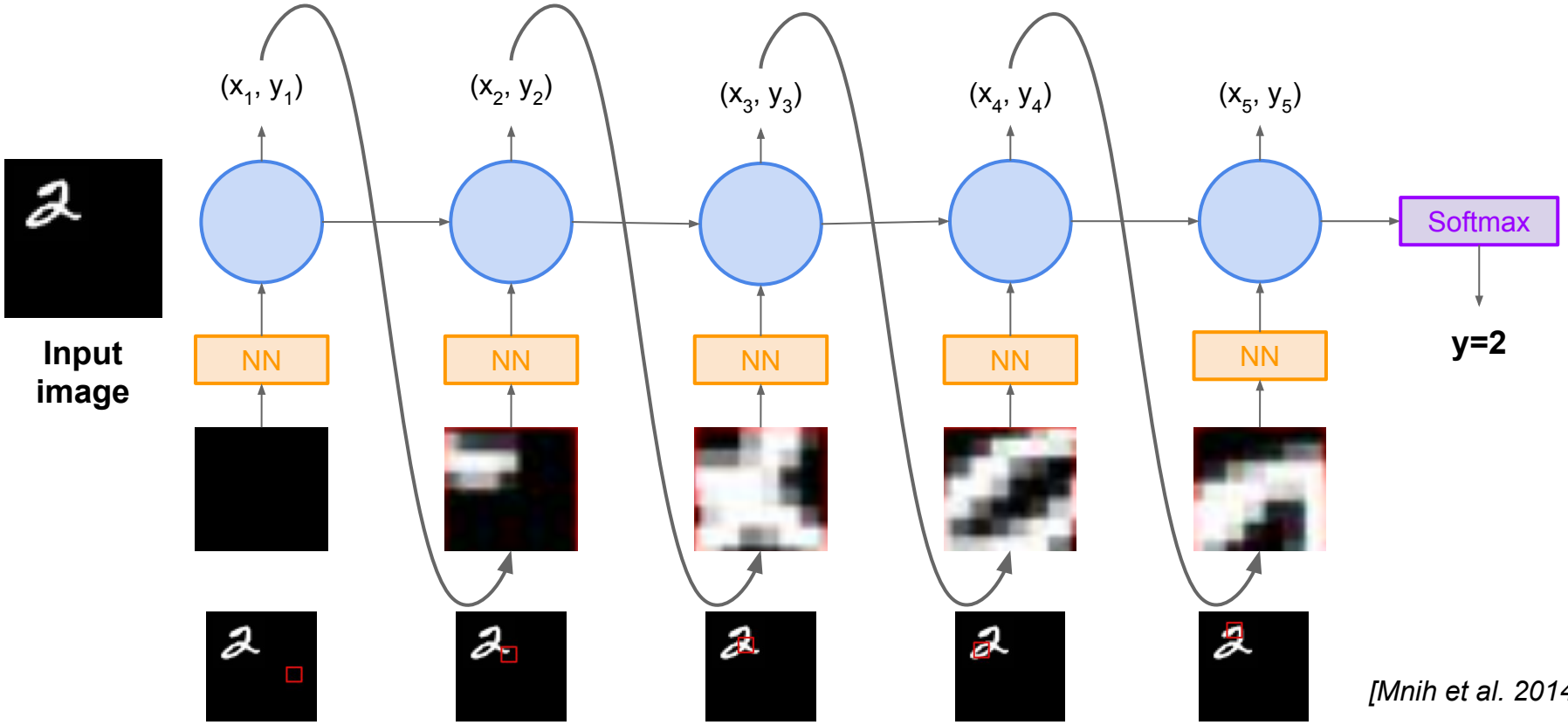
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



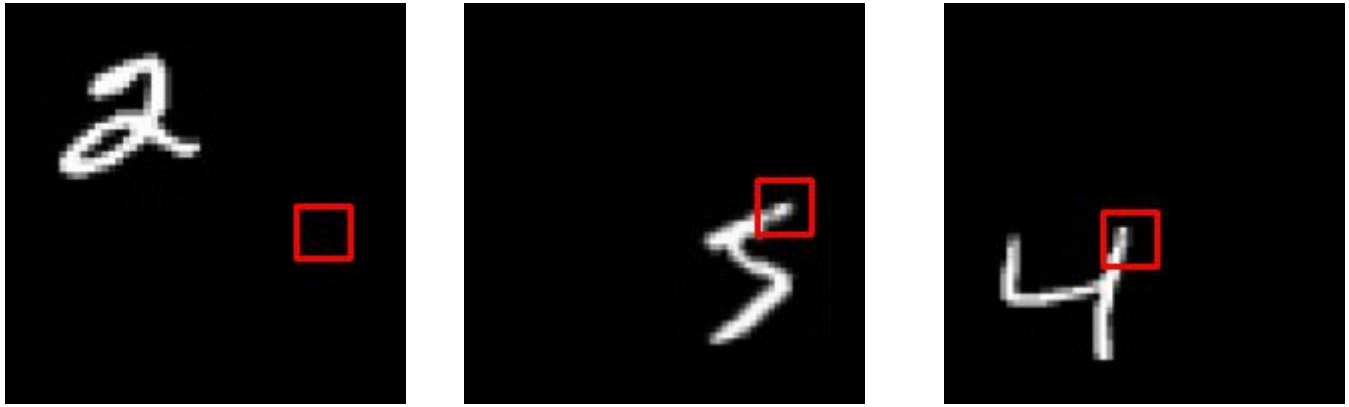
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



[Mnih et al. 2014]

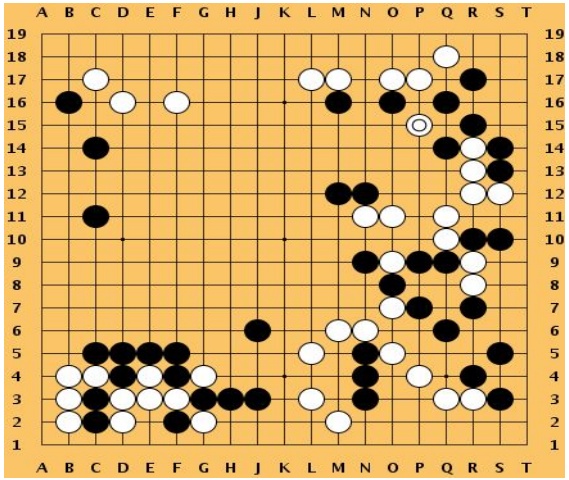
REINFORCE in action: Recurrent Attention Model (RAM)



Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih et al. 2014]

More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with behavior cloning from professional go games,
- Continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- (out of scope:) Also learn value network (critic)
- (out of scope:) Finally, combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016]

Summary

- **Policy gradients**: very general but suffer from high variance so requires a lot of samples. **Challenge**: sample-efficiency
- **Behavior cloning**: does not always work but when it works, usually more sample-efficient. **Challenge**: compounding errors

Next time:
Generative Models

What all we covered: Syllabus

Deep learning Fundamentals	Practical training skills	Applications
Data-driven approaches	Pytorch 1.4 / Tensorflow 2.0	Image captioning
Linear classification & kNN	Activation functions	Interpreting machine learning
Loss functions	Batch normalization	Generative AI
Optimization	Transfer learning	Fairness & ethics
Backpropagation	Data augmentation	Data-centric AI
Multi-layer perceptrons	Momentum / RMSProp / Adam	Deep reinforcement learning
Neural Networks	Architecture design	Self-supervised learning
Convolutions		Diffusion
RNNs / LSTMs		LLMs
Transformers		