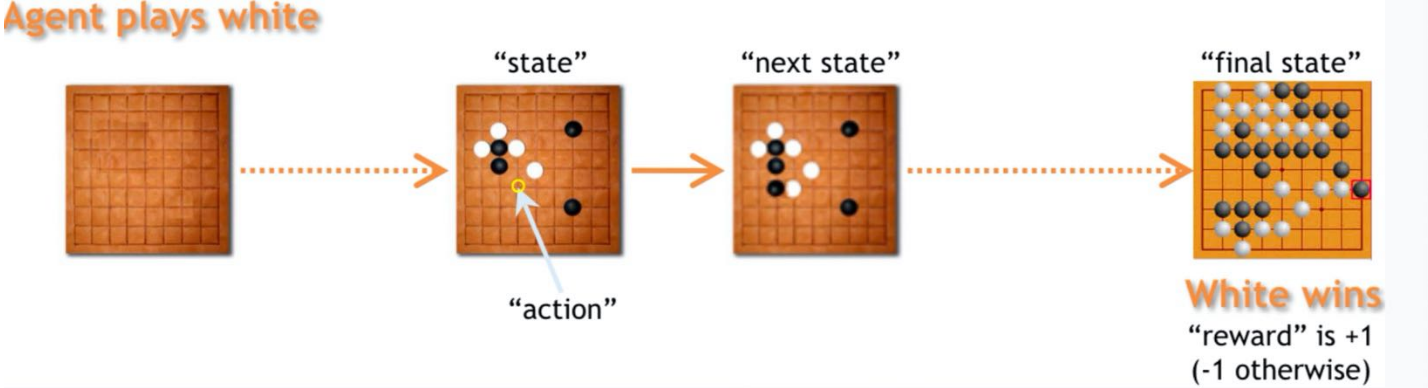
Reinforcement Learning

Hang Su
suhangss@tsinghua.edu.cn
http://www.suhangss.me
State Key Lab of Intelligent Technology & Systems
Tsinghua University

Nov 4th, 2019

Sequential Decision Making

- ◆ Goal: select actions to maximize total future reward
- ◆ Actions may have long term consequences
- ◆ Reward may be delayed
- ◆ It may be better to sacrifice immediate reward to gain more long-term reward



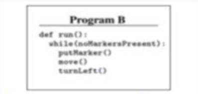

Games (Malmo, Ms. Pac-Man)


Robotics & control


Autonomic computing


News/ads recommendation


Dialogue systems

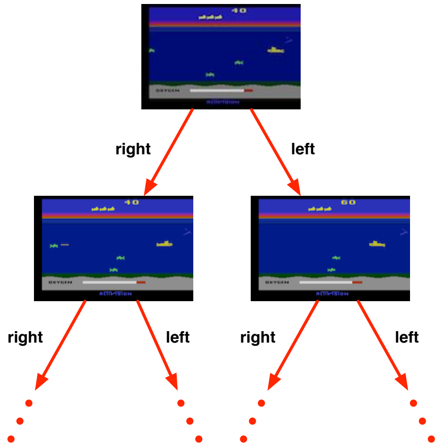

Program synthesis

Learning and Planning

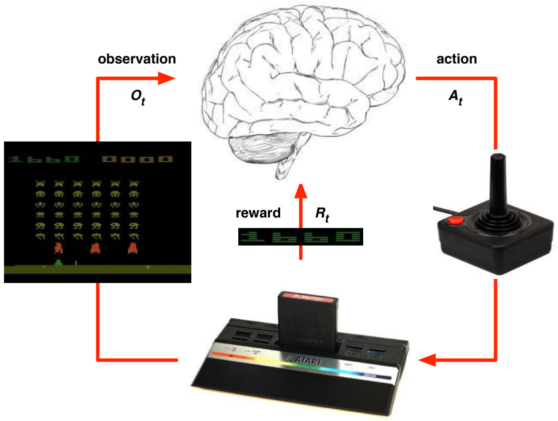
- Two fundamental problems in sequential decision making
- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy via reasoning, search, etc.

Atari Example: Planning

- Rules of the game are known
- Can query emulator
    - perfect model inside agent's brain
- If I take action a from state s:
    - what would the next state be?
    - what would the score be?
- Plan ahead to find optimal policy
    - e.g. tree search



Atari Example: Reinforcement Learning

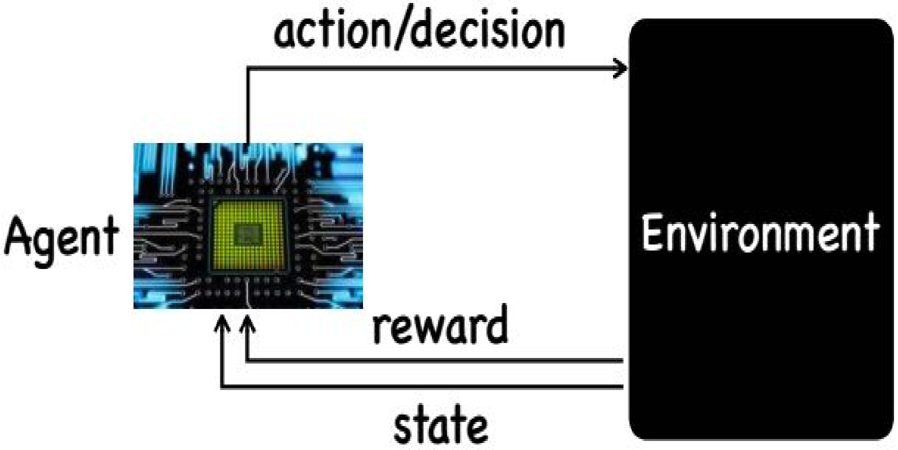


- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Reinforcement learning

- Intelligent animals can learn from interactions to adapt to the environment

Can computers do similarly?



Reinforcement Learning in a nutshell

- RL is a general-purpose framework for decision-making
  - RL is for an agent with the capacity to act
  - Each action influences the agent's future state

    Success is measured by a scalar reward signal
  - Goal: select actions to maximize future reward

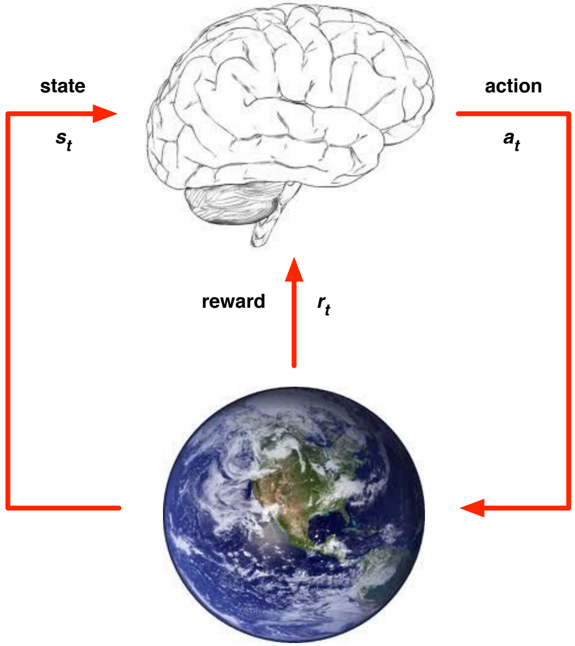
Reinforcement Learning

- The history is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$$

  - Agent chooses actions so as to maximize expected cumulative reward over a time horizon
  - Observations can be vectors or other structures
  - Actions can be multi-dimensional
  - Rewards are scalar but can be arbitrarily information

Agent and Environment



- At each step t the agent:
  - Receives state s_t
  - Receives scalar reward r_t
  - Executes action a_t

- The environment:
  - Receives action a_t
  - Emits state s_t
  - Emits scalar reward r_t

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t)$$

- In a fully observed environment

$$s_t = f(o_t)$$

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - **Policy**: agent’s behavior function
  - **Value function**: how good is each state and/or action
  - **Model**: agent’s representation of the environment

Policy

- A policy is the agent's behavior
- It is a map from state to action, e.g
- Deterministic policy:

$$a = \pi(s)$$

- Stochastic policy:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- **Q-value function** gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

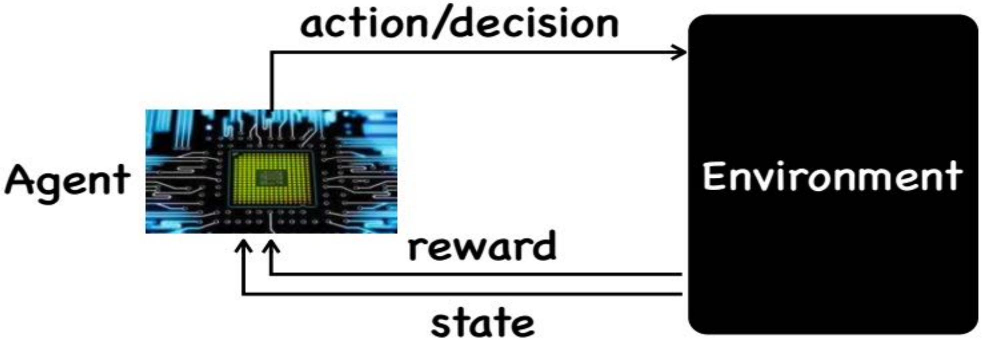
Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Reinforcement Learning



- Agent's inside: Policy: $\pi : S \times A \to \mathbb{R}, \quad \sum_{a \in A} \pi(a|s) = 1$

  Policy (deterministic): $\pi : S \to A$

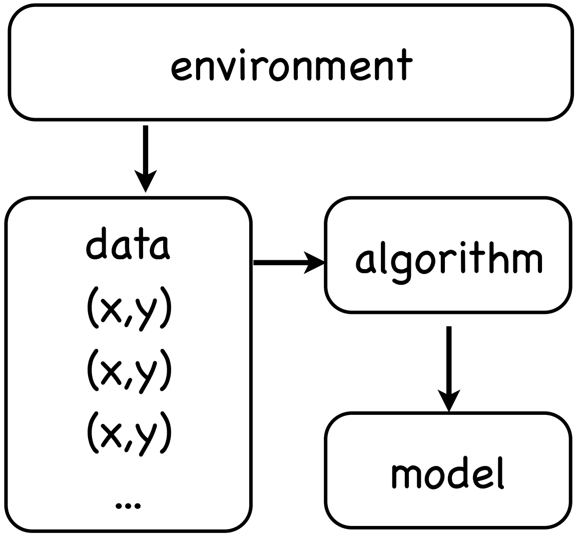
- **Agent's goal**: learn a policy to maximize long-term total reward

T-step: $\sum_{t=1}^{T} r_t$ discounted: $\sum_{t=1}^{\infty} \gamma^t r_t$

Difference between RL and SL?

- Both learn a model …

supervised learning



environment

data
(x,y)
(x,y)
(x,y)
…

algorithm

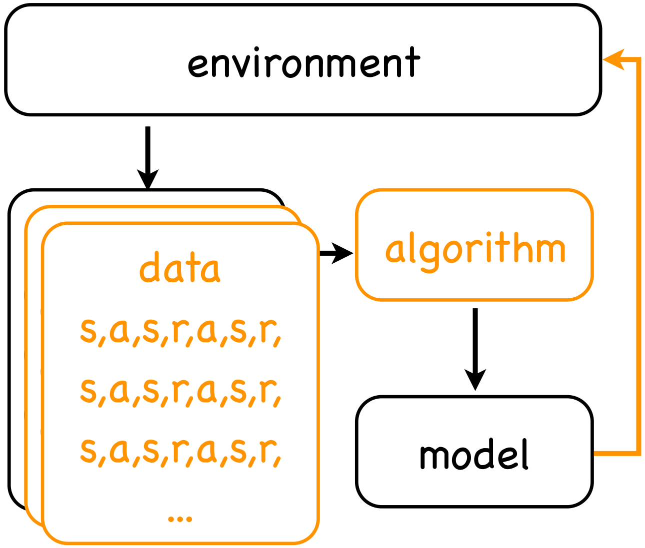
model

open loop

learning from labeled data

passive data

reinforcement learning



environment

data
s,a,s,r,a,s,r,
s,a,s,r,a,s,r,
s,a,s,r,a,s,r,
…

algorithm

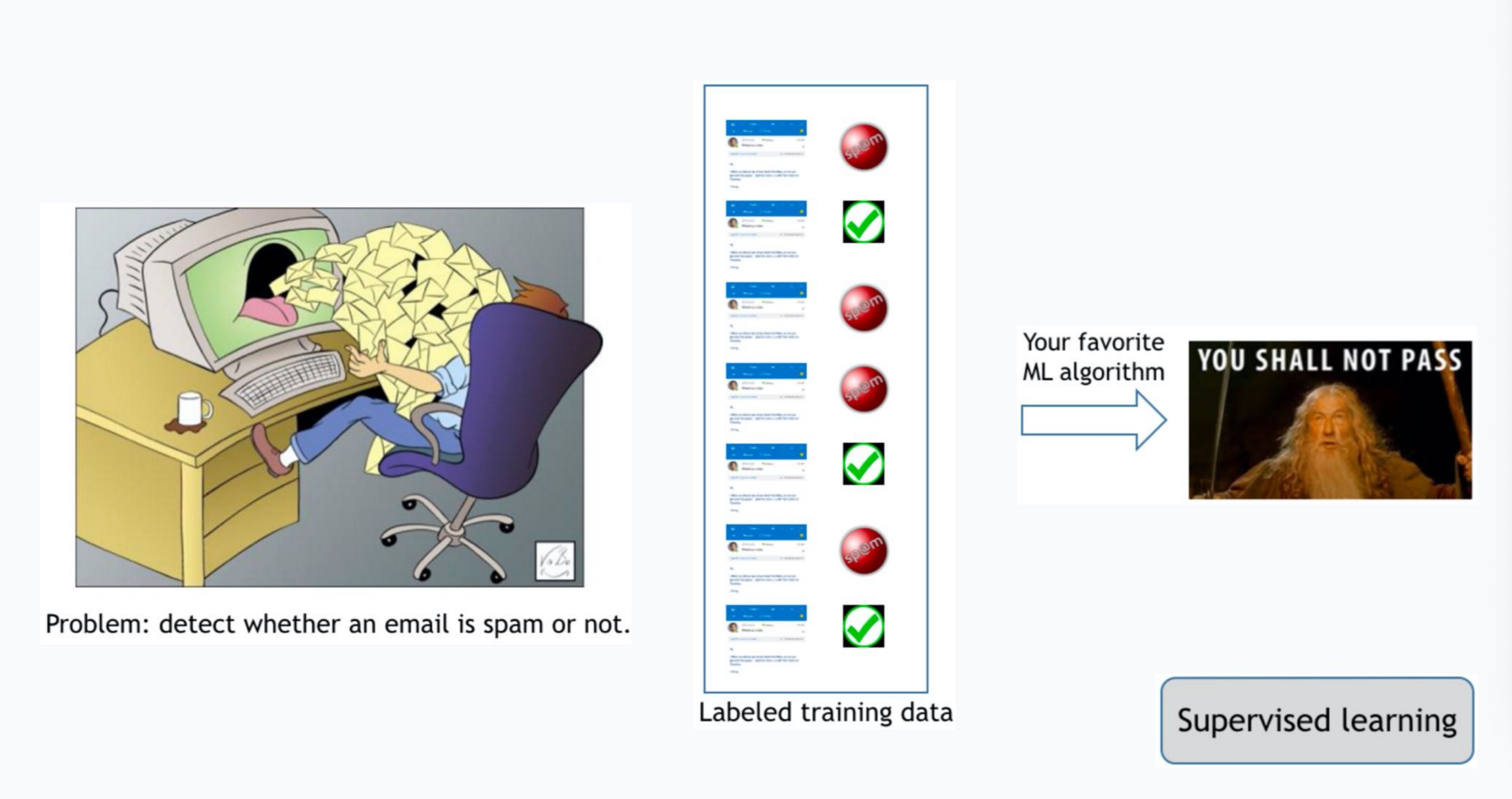
model

closed loop

learning from delayed reward explore environment

Supervised Learning

- Spam detection based on supervised learning



Problem: detect whether an email is spam or not.

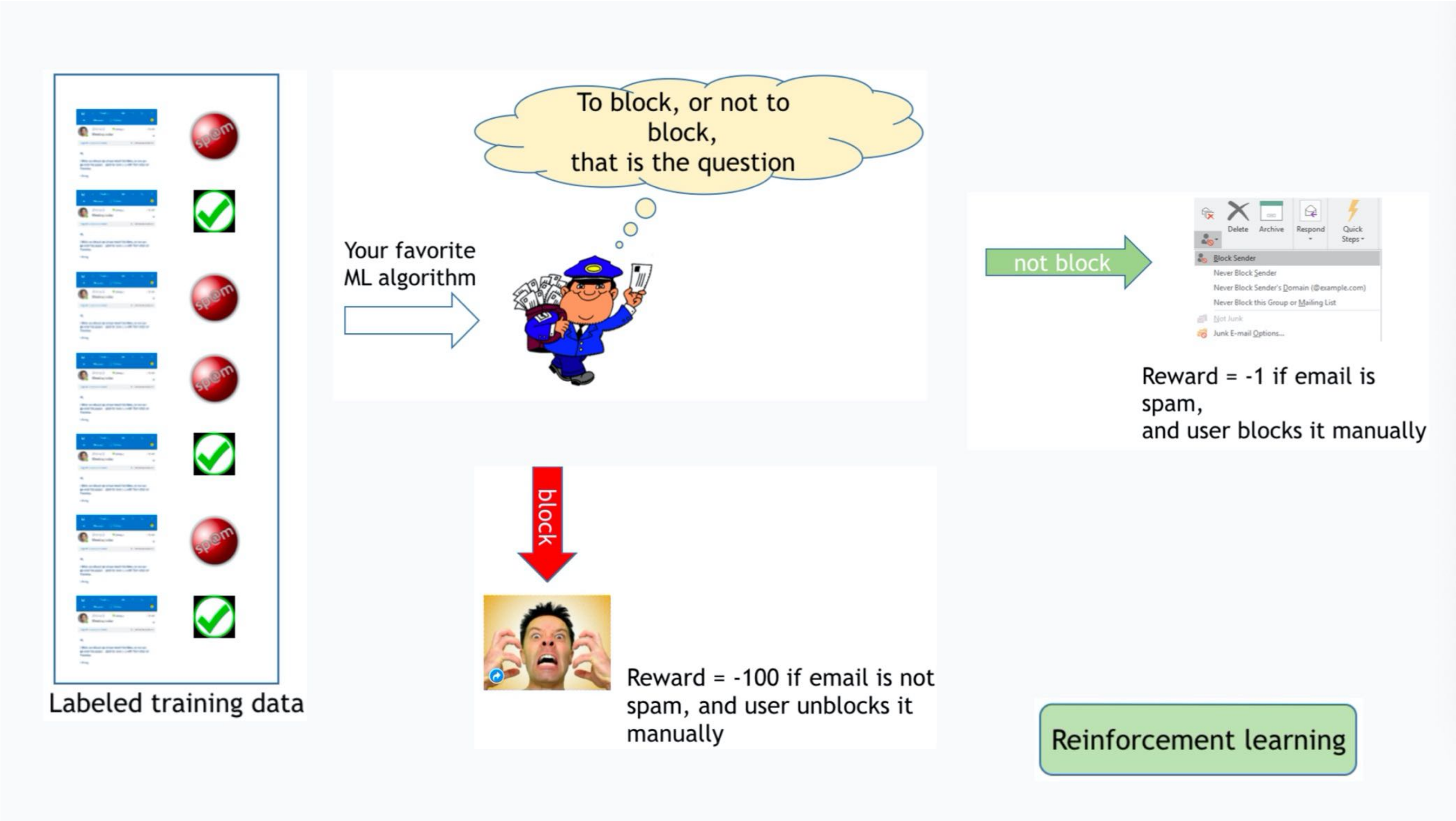
Labeled training data

Your favorite ML algorithm

Supervised learning

Reinforcement Learning

- Spam detection based on reinforcement learning



Labeled training data

Your favorite ML algorithm

To block, or not to block,
that is the question

not block

Reward = -1 if email is spam,
and user blocks it manually

block

Reward = -100 if email is not spam, and user unblocks it manually

Reinforcement learning

Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
    - There is no supervisor
    - Only a reward signal
    - Feedback is delayed, not instantaneous
    - Time really matters (sequential, non i.i.d data)
    - Agent’s actions affect the subsequent data it receives

RL vs SL (Supervised Learning)

- Differences from SL
  - Learn by trial-and-error
    - Need exploration/exploitation trade-off
  - Optimize long-term reward
    - Need temporal credit assignment
- Similarities to SL
  - Representation
  - Generalization
  - Hierarchical problem solving
  - …

Applications: The Atari games

- Deepmind Deep Q-learning on Atari
  - ☐ Mnih et al. Human-level control through deep reinforcement learning. Nature, 518(7540): 529-533, 2015

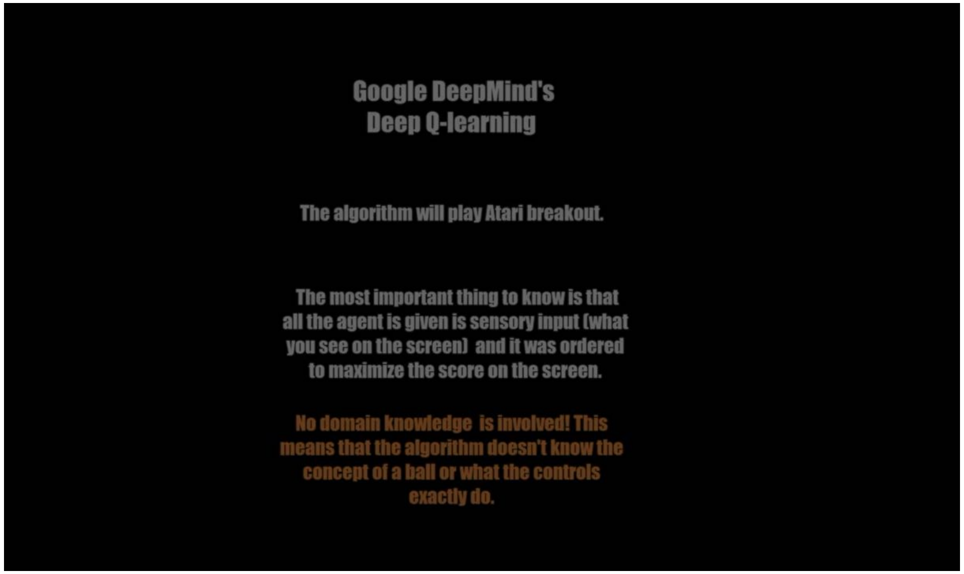
DeepMind

“Human-level control through deep reinforcement learning”

letter

Deep Q-Learning

5



Google DeepMind's
Deep Q-learning

The algorithm will play Atari breakout.

The most important thing to know is that all the agent is given is sensory input (what you see on the screen) and it was ordered to maximize the score on the screen.

No domain knowledge is involved! This means that the algorithm doesn't know the concept of a ball or what the controls exactly do.

Applications: The game of Go

- Deepmind Deep Q-learning on Go
  - ☐ Silver et al. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587): 484–489, 2016

nature

THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE

At last – a computer program that can beat a champion Go player PAGE 484

ALL SYSTEMS GO

CONSERVATION

SONGBIRDS À LA CARTE

Illegal harvest of millions of Mediterranean birds

PAGE 452

RESEARCH ETHICS

SAFEGUARD TRANSPARENCY

Don't let openness backfire on individuals

PAGE 459

POPULAR SCIENCE

WHEN GENES GOT 'SELFISH'

Dawkins's calling card 40 years on

PAGE 462

NATUREASIA.COM

28 January 2016

Vol. 529, No. 7587

Application: Producing flexible behaviors

- NIPS 2017: Learning to Run competition

More applications

- Search
- Recommendation system
- Stock prediction

3021.02 ⬆ +16.75 (+0.56%)

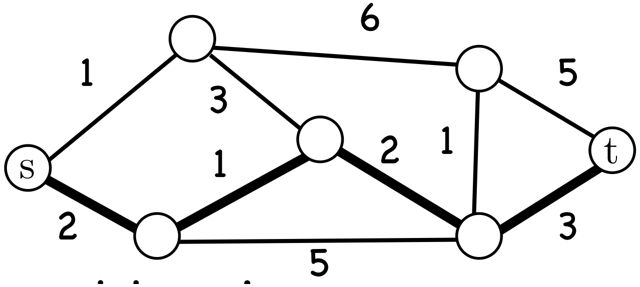
2016/08/09 14:54:25 34秒前更新 （北京时间）

今开	3001.31
昨收	3004.28
最高	3024.86
最低	2998.68
成交量	161.66亿
成交额	1780.85亿

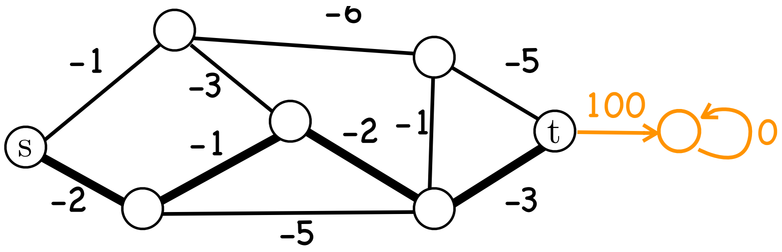
every decision changes the world

Generality of RL

- shortest path problem
  - Dijkstra's algorithm, Bellman–Ford algorithm, etc



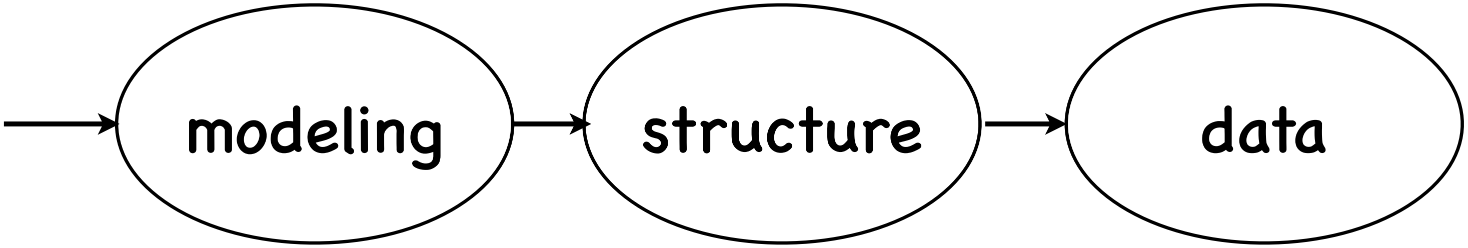
- by reinforcement learning



- every node is a state, an action is an edge out
- reward function = the negative edge weight
- optimal policy leads to the shortest path

More applications

- Also as an differentiable approach for structure learning



[Bahdanau et al., An Actor-Critic Algorithm for Sequence Prediction. ArXiv 1607.07086]
[He et al., Deep Reinforcement Learning with a Natural Language Action Space, ACL'16]
[B. Dhingra et al., End-to-End Reinforcement Learning of Dialogue Agents for Information Access, ArXiv 1609.00777]
[Yu et al., SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient, AAAI'17]

(Partial) History...

- Idea of programming a computer to learn by *trial and error* (Turing, 1954)
- SNARCs (Stochastic Neural-Analog Reinforcement Calculators) (Minsky, 1951)
- Checkers playing program (Samuel, 59)
- Lots of RL in the 60s (e.g., Waltz & Fu 65; Mendel 66; Fu 70)
- MENACE (Matchbox Educable Naughts and Crosses Engine (Mitchie, 63)
- RL based Tic Tac Toe learner (GLEE) (Mitchie 68)
- Classifier Systems (Holland, 75)
- Adaptive Critics (Barto & Sutton, 81)
- Temporal Differences (Sutton, 88)

Outline

- Markov Decision Process
- Value-based methods
- Policy search
- Model-based method
- Deep reinforcement learning

History and State

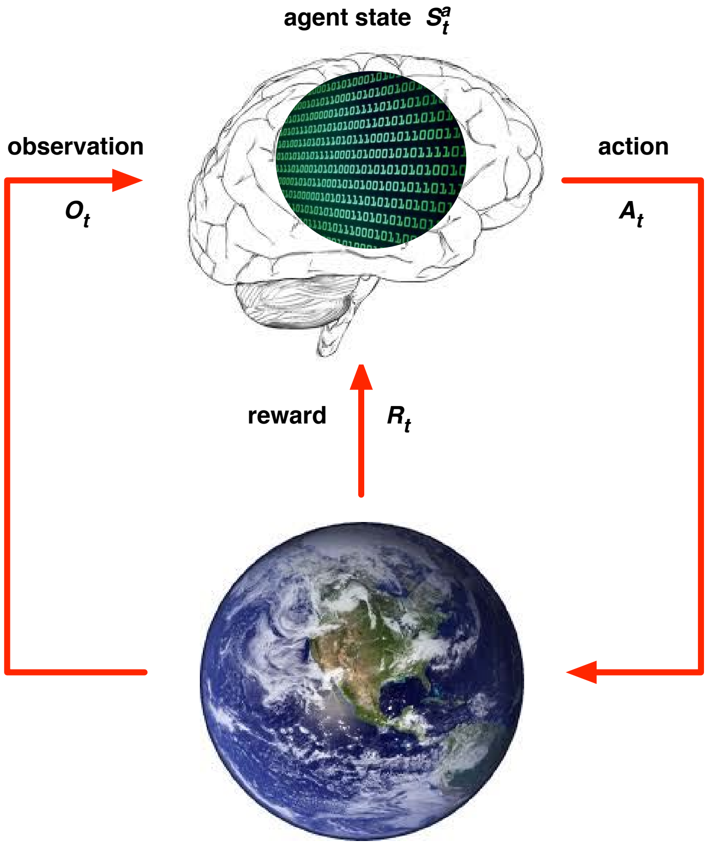
- The history is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- all observable variables up to time t
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

Agent State



- The agent state S_t^a is the agent's internal representation
- whatever information the agent uses to pick the next action
- it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t)$$

Markov state

- An Markov state contains all useful information from the history.

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- "The future is independent of the past given the present"

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- The state is a **sufficient statistic** of the future

Introduction to MDPs

- Markov decision processes formally describe an environment for reinforcement learning
- **Where the environment is fully observable**
  - i.e. The current state completely characterizes the process
- Almost all RL problems can be formalized as MDPs
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

Markov Property

- "The future is independent of the past given the present"

A state S_t is *Markov* if and only if

$$\mathbb{P}\left[S_{t+1} \mid S_t\right] = \mathbb{P}\left[S_{t+1} \mid S_1, ..., S_t\right]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
  - The state is a sufficient statistic of the future

Markov Decision Process

- A Markov reward process is a Markov chain with values
- A Markov decision process (MDP) is a Markov reward process with decisions.

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

RL in MDP

- Observe initial state s_1
- For t = 1,2,3,…
  - Choose action a_t based on s_t and current policy
  - Observe reward r_t and next state s_{t+1}
  - Update policy using new information(s_t,a_t,r_t,s_{t+1})
- Episode length may be finite or infinite
- Agent can have multiple episodes starting from new initial states

Solving the optimal policy in MDP

- Given MDP model, we can compute an optimal policy as
- Value-based RL
  - Estimate the optimal value function $Q^*(s,a)$
  - This is the maximum value achievable under any policy
- Policy-based RL
  - Search directly for the optimal policy π^*
  - This is the policy achieving maximum future reward
- Model-based RL
  - Build a model of the environment
  - Plan (e.g. by look ahead) using model
- What if R and P are unknown?
  - This is what reinforcement learning is about!

Policy Evaluation

- Q: what is the total reward of a policy?
- state value function

$$V^{\pi}(s) = E[\sum_{t=1}^{T} r_t | s]$$

- state-action value function

$$Q^{\pi}(s,a) = E[\sum_{t=1}^{T} r_t | s,a] = \sum_{s'} P(s'|s,a)\big(R(s,a,s') + V^{\pi}(s')\big)$$

- Consequently

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q(s,a)$$

Solving the optimal policy in MDP

- idea:
  - how is the current policy   policy evaluation
  - improve the current policy   policy improvement
- policy iteration:
  - policy evaluation: backward calculation

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V^{\pi}(s')\big)$$

  - policy improvement

$$V(s) \leftarrow \max_{a} Q^{\pi}(s,a)$$

  - value iteration:

$$V_{t+1}(s) = \max_{a} \sum_{s'} P(s'|s,a)\big(R(s,a,s') + \gamma V_t(s)\big)$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximizes over all decisions.

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Value Function Approximation

- So far we have represented value function
  - Every state s has an entry V(s)
  - every state-action pair (s,a) has an entry Q(s,a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with **function approximation**

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

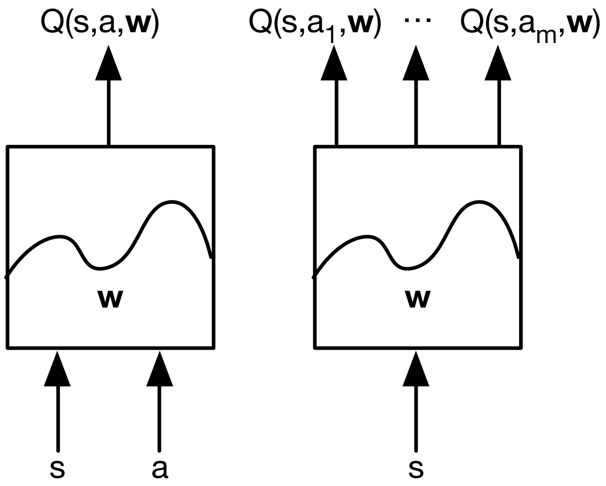
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states

Q-Networks

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$



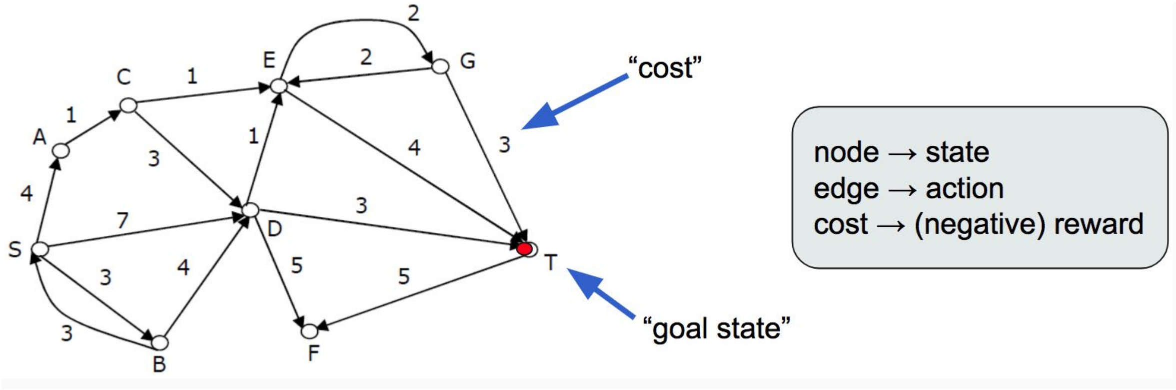
- Minimize mean-squared error between approximate and true action-value

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2\right]$$

- Use stochastic gradient descent to find a local minimum

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

Simple MDP: Shortest Path Problem



"cost"

node → state
edge → action
cost → (negative) reward

"goal state"

$$\forall i: \quad \mathrm{CostToGo}(i) = \min_{j \in \mathrm{Neighbor}(i)} \{\mathrm{cost}(i \to j) + \mathrm{CostToGo}(j)\}$$

- Principle of Optimality (Richard Bellman, 1957)
  - An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellmen Equations for MDPs

Deterministic shortest path

$$\mathsf{CostToGo}(i) = \min_{j \in \mathrm{Neighbor}(i)} \{\mathsf{cost}(i \to j) + \mathsf{CostToGo}(j)\}$$

Markov decision process

$$V^*(s) \quad = \quad \max_{a \in \mathcal{A}} \{R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^*(s')]\}$$

(maximum long-term reward starting from s)

$$Q^*(s, a) \quad = \quad R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q^*(s', a')\right]$$

(maximum long-term reward after choosing a from s)

V* and Q* are called optimal value functions

Policy-Based Reinforcement Learning

- Directly parametrize the policy

$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- Start with arbitrary policy $\pi_0 : S \rightarrow A$
- For k = 0,1,2,…
  - Policy evaluation: solve for Q_k that satisfies

$$\forall(s, a) : Q_k(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) Q_k(s', \pi_k(s'))$$

  - Policy improvement:

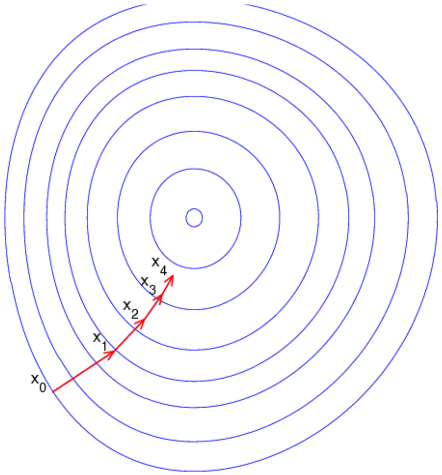
$$\pi_{k+1}(s) \leftarrow \arg\max_a Q_k(s, a)$$

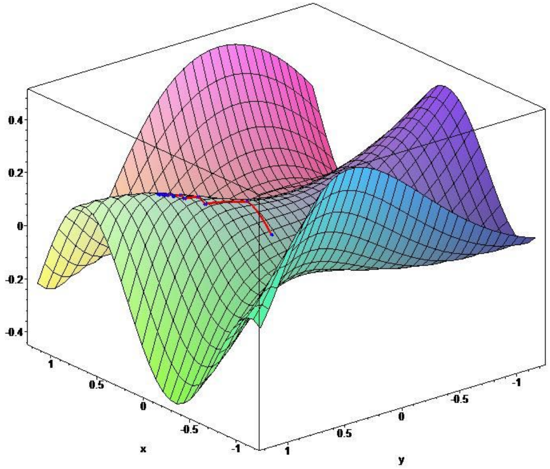
Policy Gradient

- Let J(θ) be any policy objective function

- Policy gradient algorithms search for a local maximum in J(θ) by ascending the gradient of the policy:

$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$





Computing Gradients By Finite Differences

- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension k $\in$ [1, n]
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ε in kth dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective

Score Function

- We now compute the policy gradient analytically
- Assume policy π_θ is differentiable whenever it is non-zero
- Likelihood ratios exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\end{aligned}$$

- The gradient $\nabla_\theta \pi_\theta(s,a)$ can be computed using the score function $\nabla_\theta \log \pi_\theta(s,a)$

Softmax Policy

- Softmax policy:
    - Weight actions using linear combination of features $\phi(s,a)^\top\theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^\top \theta}$$

- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}\left[\phi(s,\cdot)\right]$$

Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \phi(s)^\top\theta$
- Variance may be fixed σ2, or can also parametrized
- Policy is Gaussian $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

Policy Gradient Theorem

- Consider a simple class of one-step MDPs
  - Starting in state s ~ d(s)
  - Terminating after one time-step with reward r
  - Use likelihood ratios to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]$$

- Generalize the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q_\pi(s,a)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\right]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

```
function REINFORCE
    Initialise $\theta$ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
        for $t = 1$ to $T - 1$ do
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return $\theta$
end function
```

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)\right]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)$$

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
- Subtract a baseline function B(s) from the policy gradient

$$B(s) = V^{\pi_\theta}(s)$$

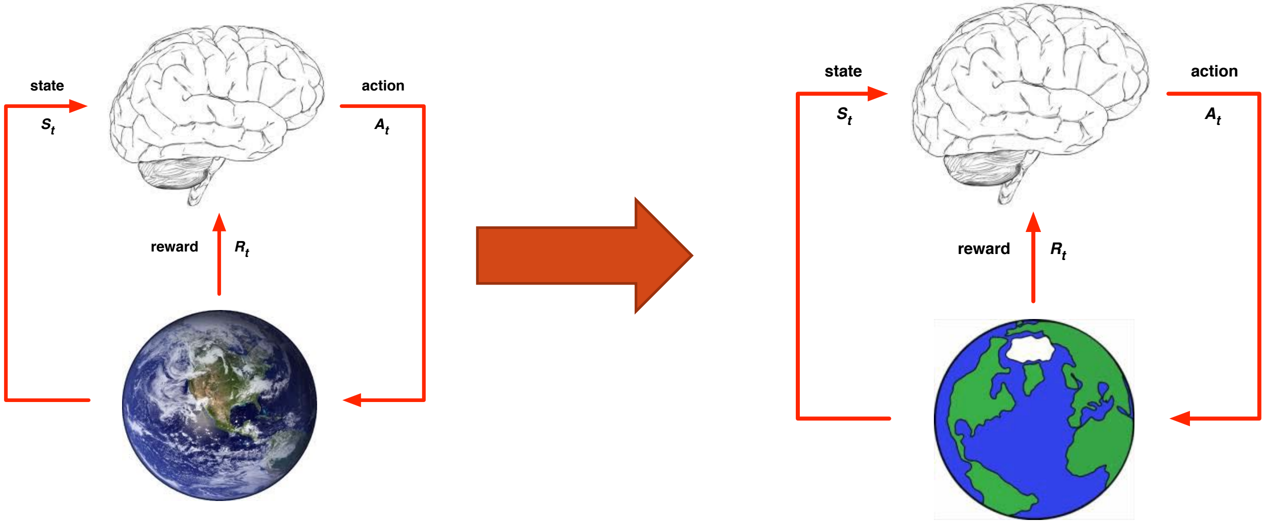
- So we can rewrite the policy gradient using the advantage function

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{\pi_\theta}(s,a)\right]$$

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model



Advantages of Model-Based RL

- Advantages:
  - Can efficiently learn model by supervised learning methods
  - Can reason about model uncertainty
- Disadvantages:
  - First learn a model, then construct a value function

Model Learning

- Goal: estimate model M_η from experience {S1, A1, R2, …, ST }
- This is a supervised learning problem

$$\begin{gathered} S_1, A_1 \to R_2, S_2 \\ S_2, A_2 \to R_3, S_3 \\ \vdots \\ S_{T-1}, A_{T-1} \to R_T, S_T \end{gathered}$$

- Learning s,a → r is a regression problem
- Learning s,a → s′ is a density estimation problem
- Pick loss function, e.g. mean-squared error, KL divergence, …

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- ……

Exploration vs. Exploitation Dilemma

- Online decision-making involves a fundamental choice:
  - Exploitation: Make the best decision given current information
  - Exploration: Gather more information

- The best long-term strategy may involve short-term sacrifices

- Gather enough information to make the best overall decisions

Examples

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation Show the most successful advert
  - Exploration Show a different advert
- Oil Drilling
  - Exploitation Drill at the best known location
  - Exploration Drill at a new location
- Game Playing
  - Exploitation Play the move you believe is best
  - Exploration Play an experimental move

Exploration methods

- exploration only policy: try every action in turn
  - waste many trials
- exploitation only policy: try each action once, follow the best action forever
  - risk of pick a bad action
- balance between exploration and exploitation

Exploration methods

- ε-greedy:
  - follow the best action with probability 1-ε
  - choose action randomly with probability ε
  - ε should decrease along time
- given a policy

$$\pi_\epsilon(s) = \begin{cases} \pi(s), \text{with prob. } 1-\epsilon \\ \text{randomly chosen action, with prob. } \epsilon \end{cases}$$

- ensure probability of visiting every state > 0

Deep Reinforcement Learning

- DL is a general-purpose framework for representation learning
  - Given an objective, and learn representation that is required to achieve objective
  - Directly from raw inputs using minimal domain knowledge
- Deep Reinforcement Learning: AI = RL + DL
- Seek a single agent which can solve any human-level task
  - RL defines the objective
  - DL gives the mechanism
  - RL + DL = general intelligence

Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent

Stochastic Gradient Descent with Experience Replay

- Given experience consisting of ⟨state, value⟩ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Repeat:
  - Sample state, value from experience

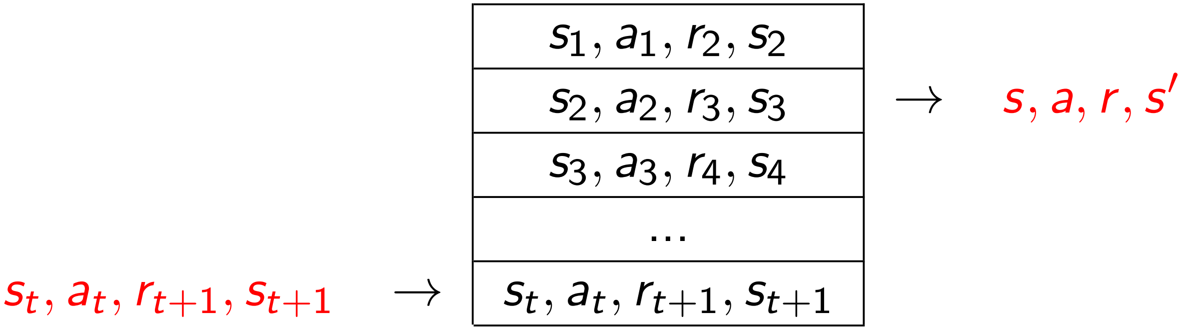
$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

  - Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Deep Q-Networks (DQN): Experience Replay

- To remove correlations, build data-set from agent's own experience



s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

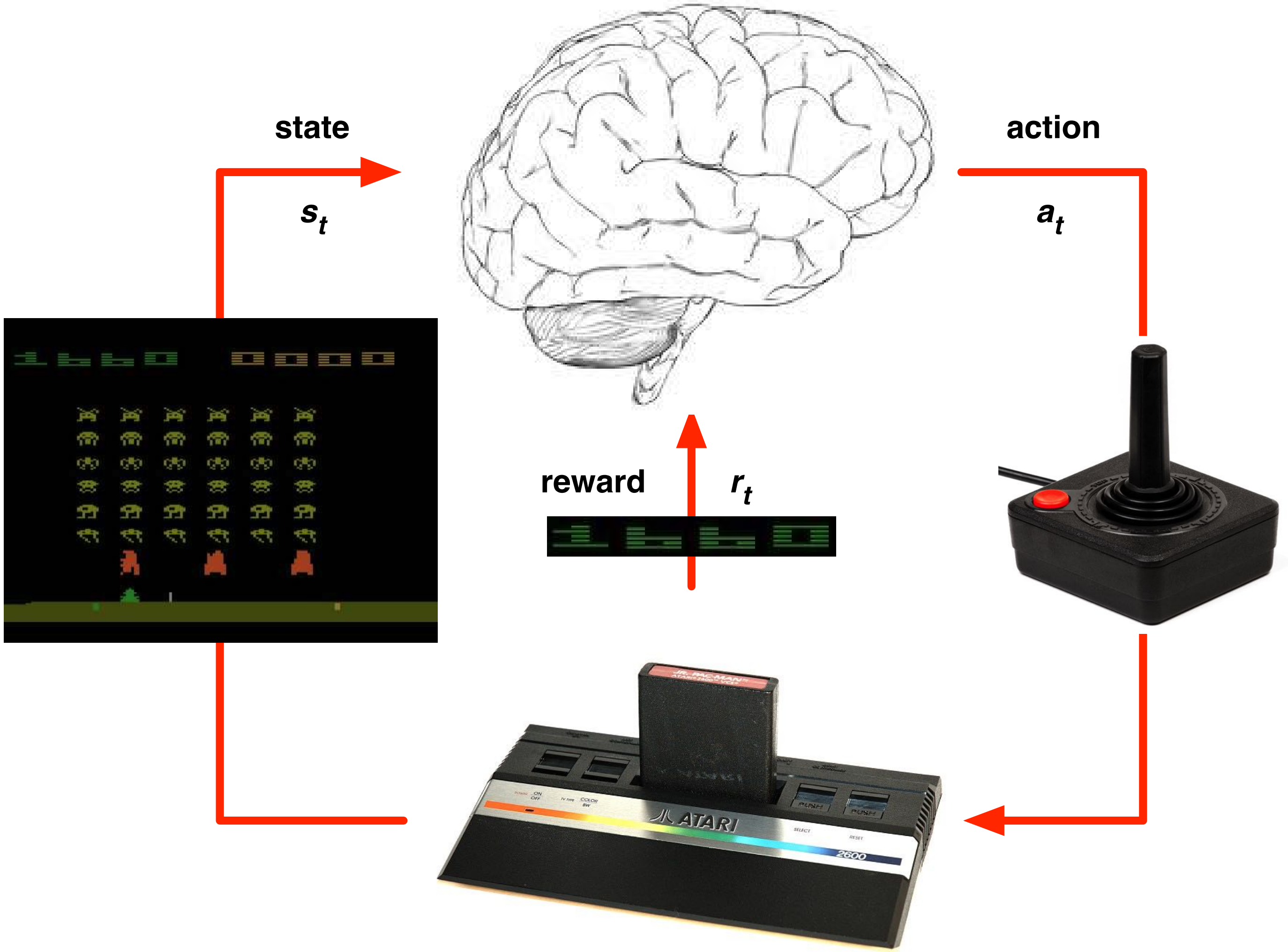
$\rightarrow$ s, a, r, s'

$s_t, a_t, r_{t+1}, s_{t+1}$ $\rightarrow$

- Sample experiences from data-set and apply update

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

Deep Reinforcement Learning in Atari



state

s_t

action

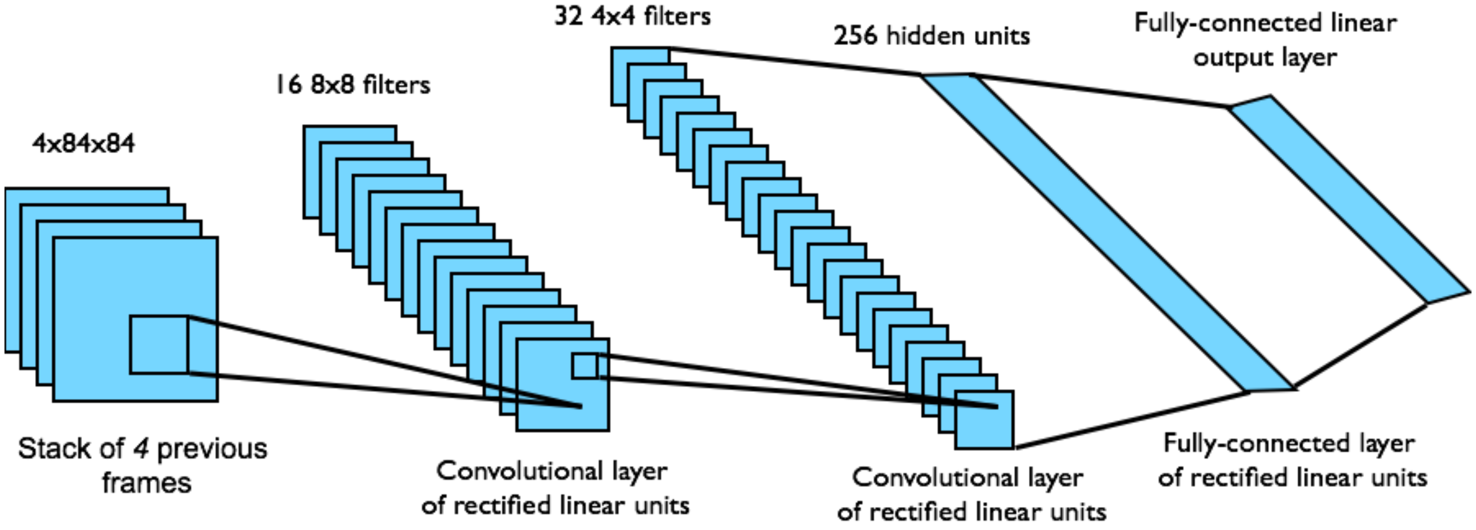
a_t

reward

r_t

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

Deep Policy Networks

- Represent policy by deep network with weights u

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Define objective function as total discounted reward

$$L(\mathbf{u}) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \mid \pi(\cdot, \mathbf{u})\right]$$

- Optimize objective end-to-end by SGD
- Adjust policy parameters u to achieve more reward

Policy Gradients

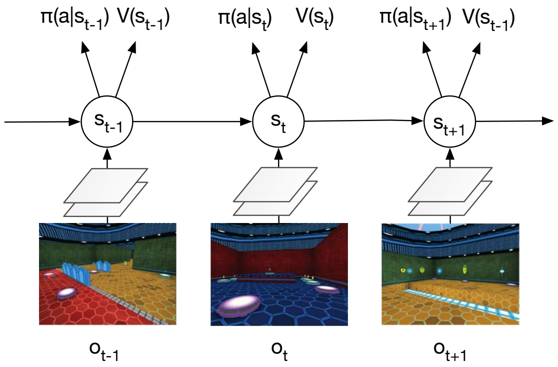
- The gradient of a stochastic policy π(a | s,u) is given by

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)\right]$$

Similar as Policy Gradient Theorem for RL

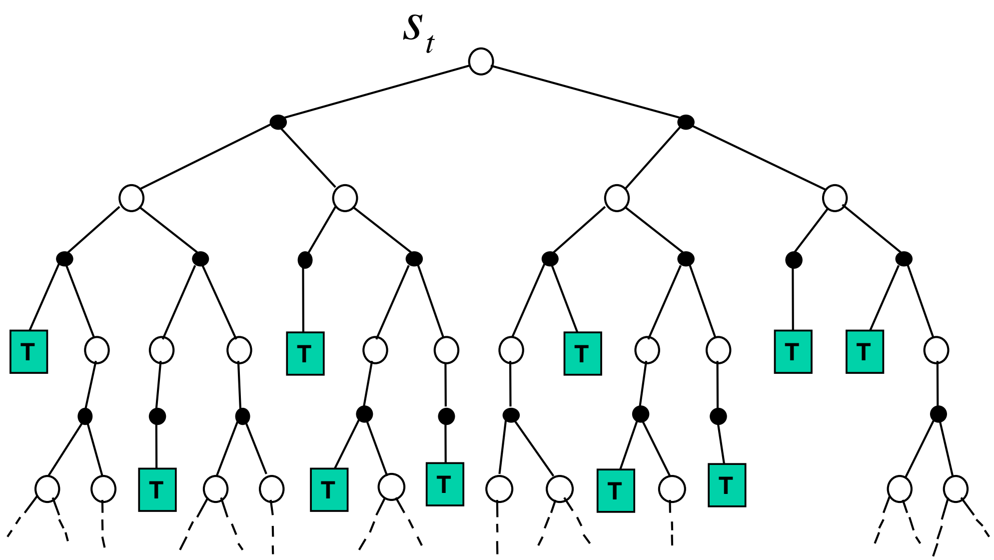
Deep Reinforcement Learning in Labyrinth

- End-to-end learning of softmax policy $\pi(a|st)$ from pixels
- Observations o_t are raw pixels from current frame
- State $s_t = f(o_1, \ldots, o_t)$ is a recurrent neural network(LSTM)
- Outputs both value V(s) and softmax over actions $\pi(a|s)$



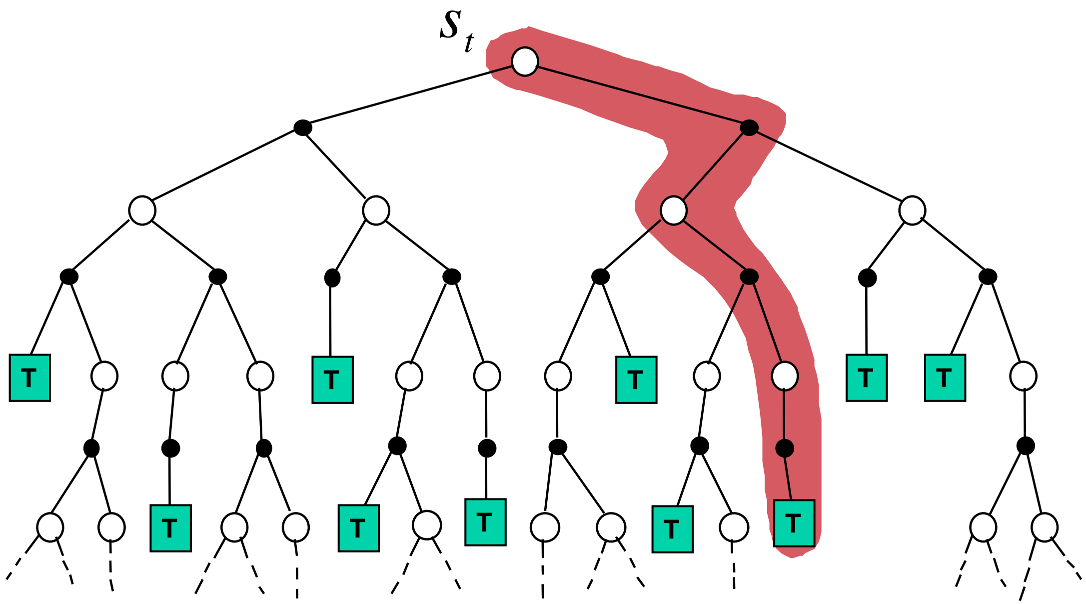
Model-based RL

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead
- No need to solve whole MDP, just sub-MDP starting from now



Simulation-Based Search

- Forward search paradigm using sample-based planning
- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes



Simple Monte-Carlo Search

- Given a model M_v and a simulation policy π
- For each action a $\in$ A
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

  - Evaluate actions by mean return (Monte-Carlo evaluation)

$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a)$$

  - Select current (real) action with maximum value

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s_t, a)$$

Monte-Carlo Tree Search (Evaluation)

- Given a model M_v
- Simulate K episodes from current state s_t using current simulation policy π

$$\{s_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states Q(s, a) by mean return of episodes from s, a

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}(S_u, A_u = s, a) G_u \xrightarrow{P} q_\pi(s, a)$$

- After search is finished, select current (real) action with maximum value in search tree

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q(s_t, a)$$

Monte-Carlo Tree Search (Simulation)

- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximize Q(S,A)
  - Default policy (fixed): pick actions randomly
- Repeat (each simulation)
  - Evaluate states Q(S,A) by Monte-Carlo evaluation
  - Improve tree policy, e.g. by ϵ – greedy(Q)
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, $Q(S,A) \rightarrow q^*(S,A)$

Case Study: the Game of Go

- How good is a position s?
- Reward function (undiscounted):

$$R_t = 0 \text{ for all non-terminal steps } t < T$$

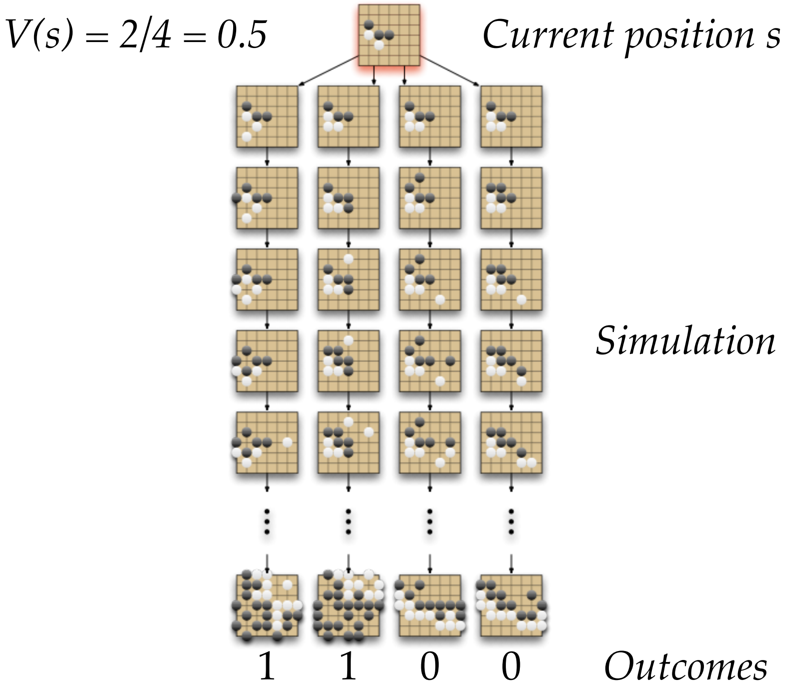
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

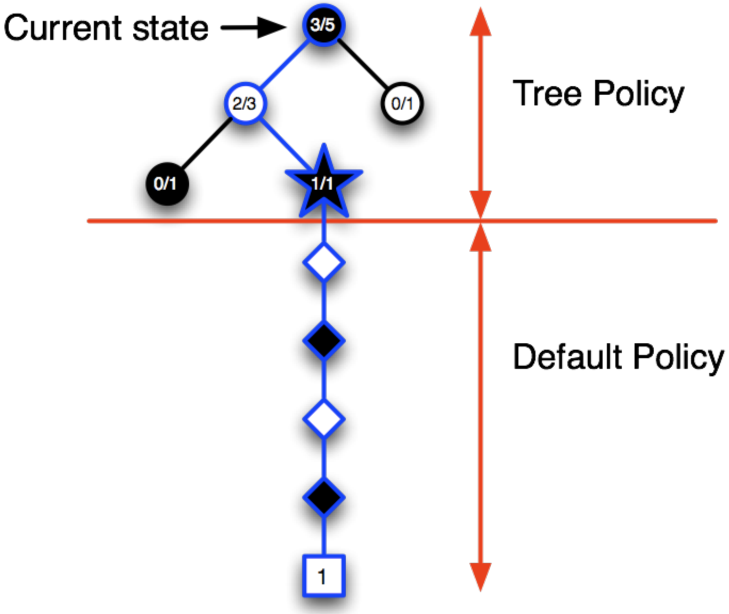
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid S = s \right] = \mathbb{P} \left[\text{Black wins} \mid S = s \right]$$

$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

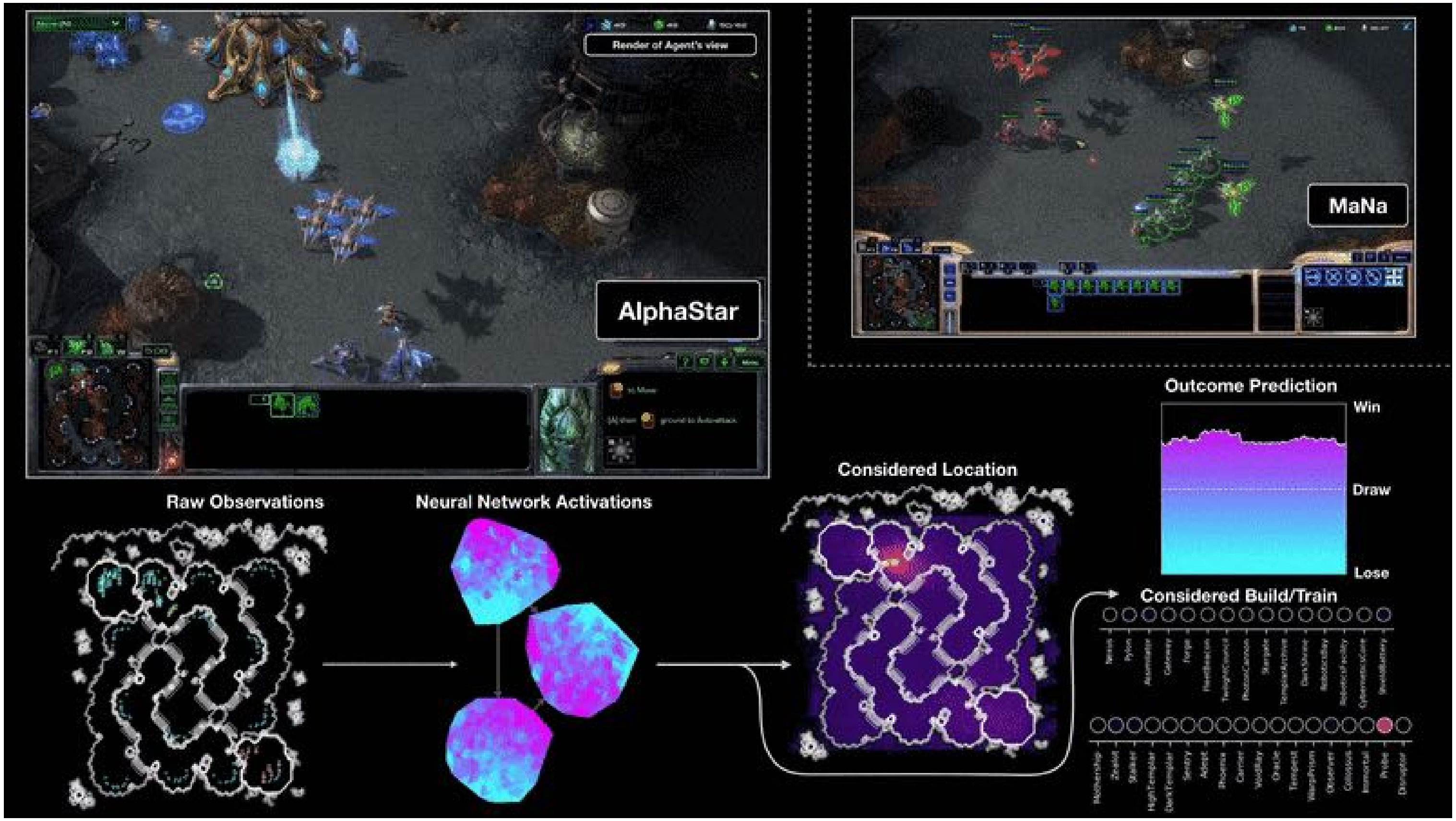
Monte-Carlo Evaluation in Go





AlphaGo paper: www.nature.com/articles/nature16961

AlphaStar



- A visualisation of the AlphaStar agent during game two of the match against MaNa.

AlphaStar – Challenges on StartCraft

- **Game theory**
  - StarCraft is a game where, just like rock-paper-scissors, there is no single best strategy
- **Imperfect information**
  - crucial information is hidden from a StarCraft player and must be actively discovered by “scouting”.
- **Long term planning**
  - Like many real-world problems cause-and-effect is not instantaneous.
- **Real time**
  - StarCraft players must perform actions continually as the game clock progresses
- **Large action space**
  - Hundreds of different units and buildings must be controlled at once, in real-time, resulting in a combinatorial space of possibilities

Summary

- Key concepts:
    - Markov Decision Process
    - Value-based methods
    - Policy gradient
    - Deep reinforcement learning
- What's more
    - POMDP
    - Exploration and Exploition
    - A3C
    - HRL
    - On policy and off policy
    - ……

Questions?