Reinforcement Learning

Lecture 19
Subir Varma

So Far..

We have learnt how to design Neural Networks that can:

- “See” (ConvNets)
- “Hear” (RNNs, LSTMs)
- “Talk” (Language Models)
- “Draw” (GANs)

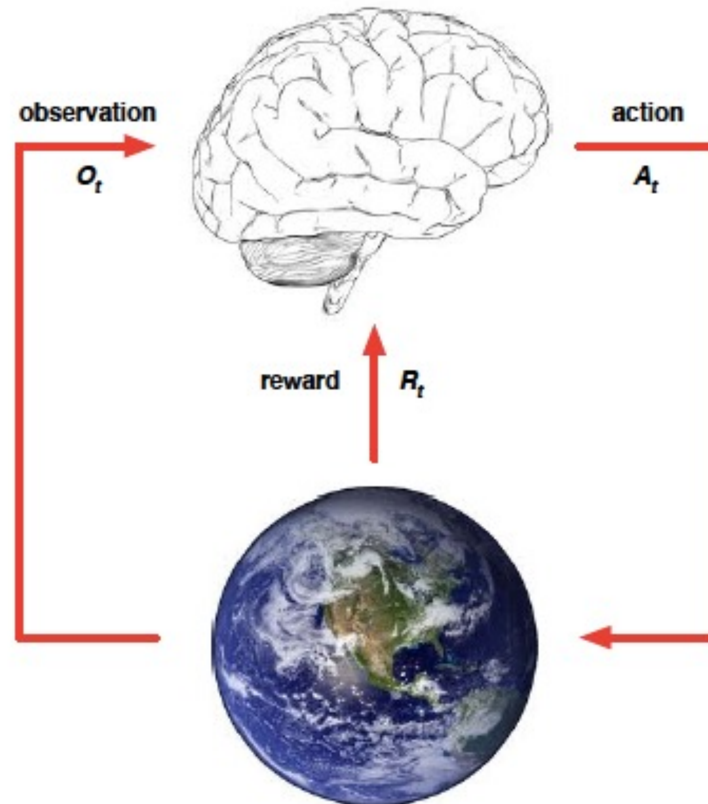
What about Neural Networks that can make decisions?

A Sequence of Decisions to Achieve an Objective

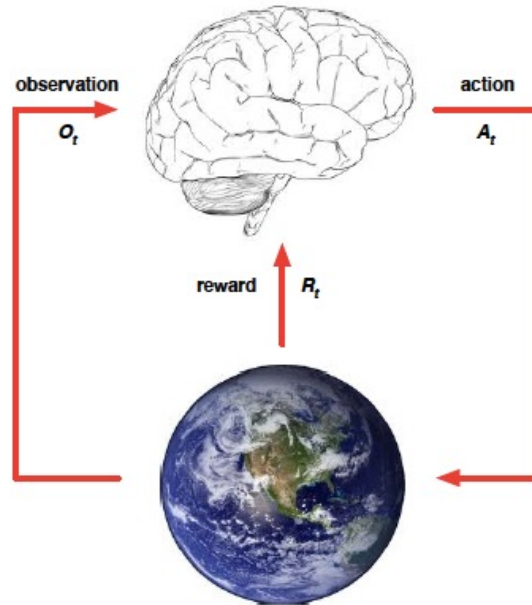
What is Reinforcement Learning?

Science of Making Decisions

By Interacting with the Environment

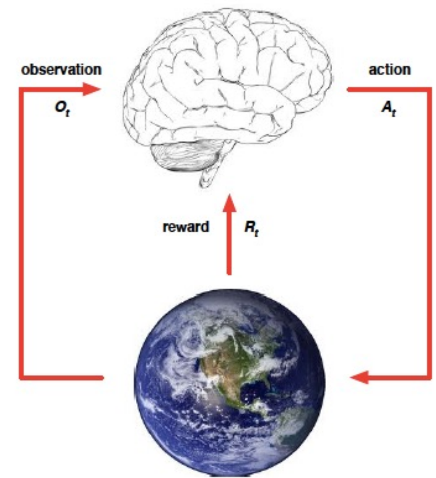


Differences between RL and Supervised Learning



- 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

Examples



Actions: muscle contractions
Observations: sight, smell
Rewards: food

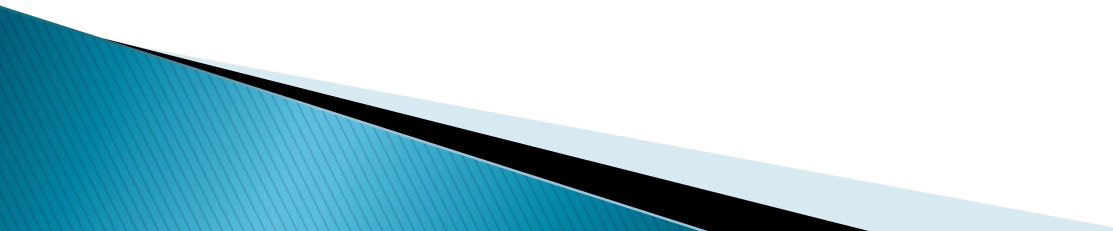
Actions: motor current or torque
Observations: camera images
Rewards: task success measure (e.g., running speed)

Actions: what to purchase
Observations: inventory levels
Rewards: profit

Examples of Reinforcement Learning

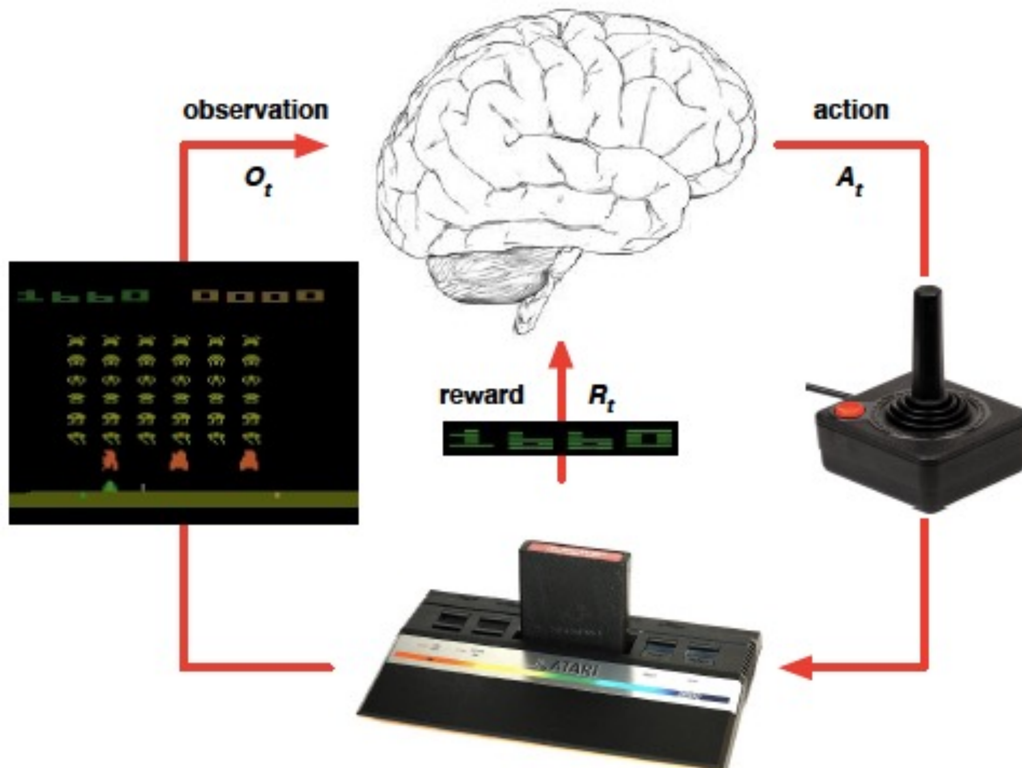
- Fly stunt manoeuvres in a helicopter
- Defeat the world champion at Backgammon
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

Components of an RL System



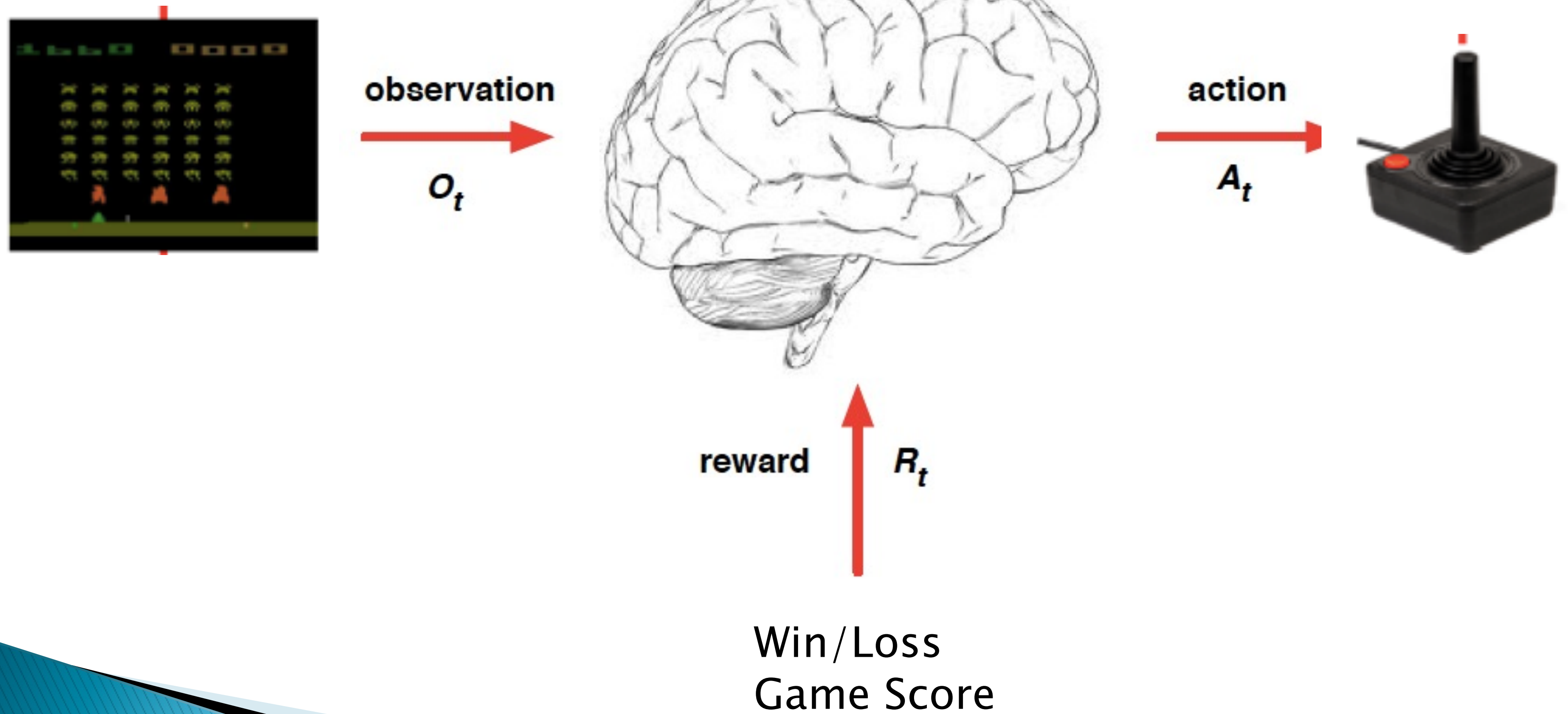
Atari Example: Reinforcement Learning

Agent: Player

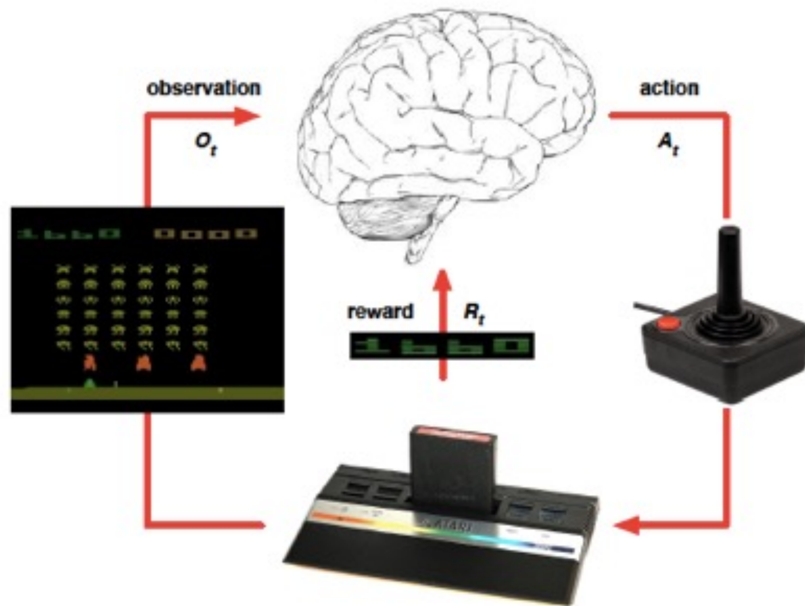


Environment: Game Software

Agent



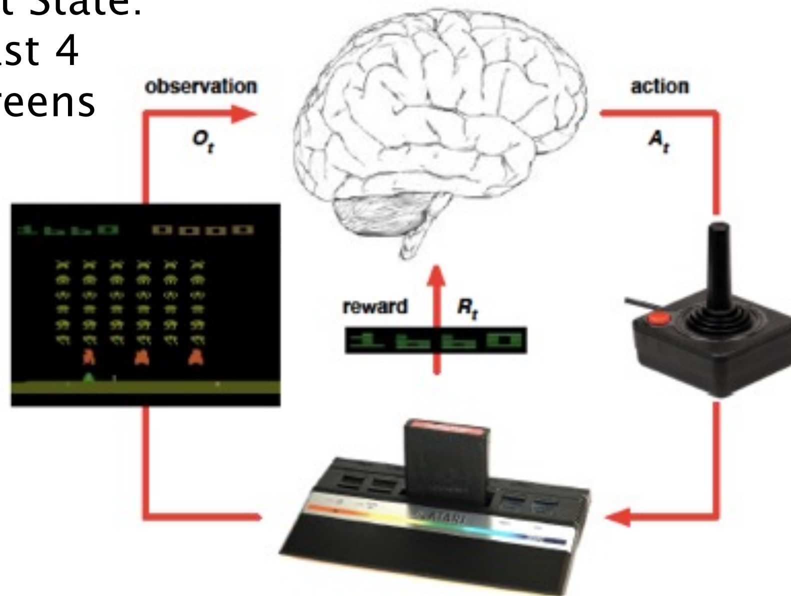
Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Agent State

Agent State:
Last 4
Screens



- The agent state S^a_t is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action

  - i.e. it is the information used by reinforcement learning algorithms
  - It can be any function of history:

$$S^a_t = f(H_t)$$

Rewards

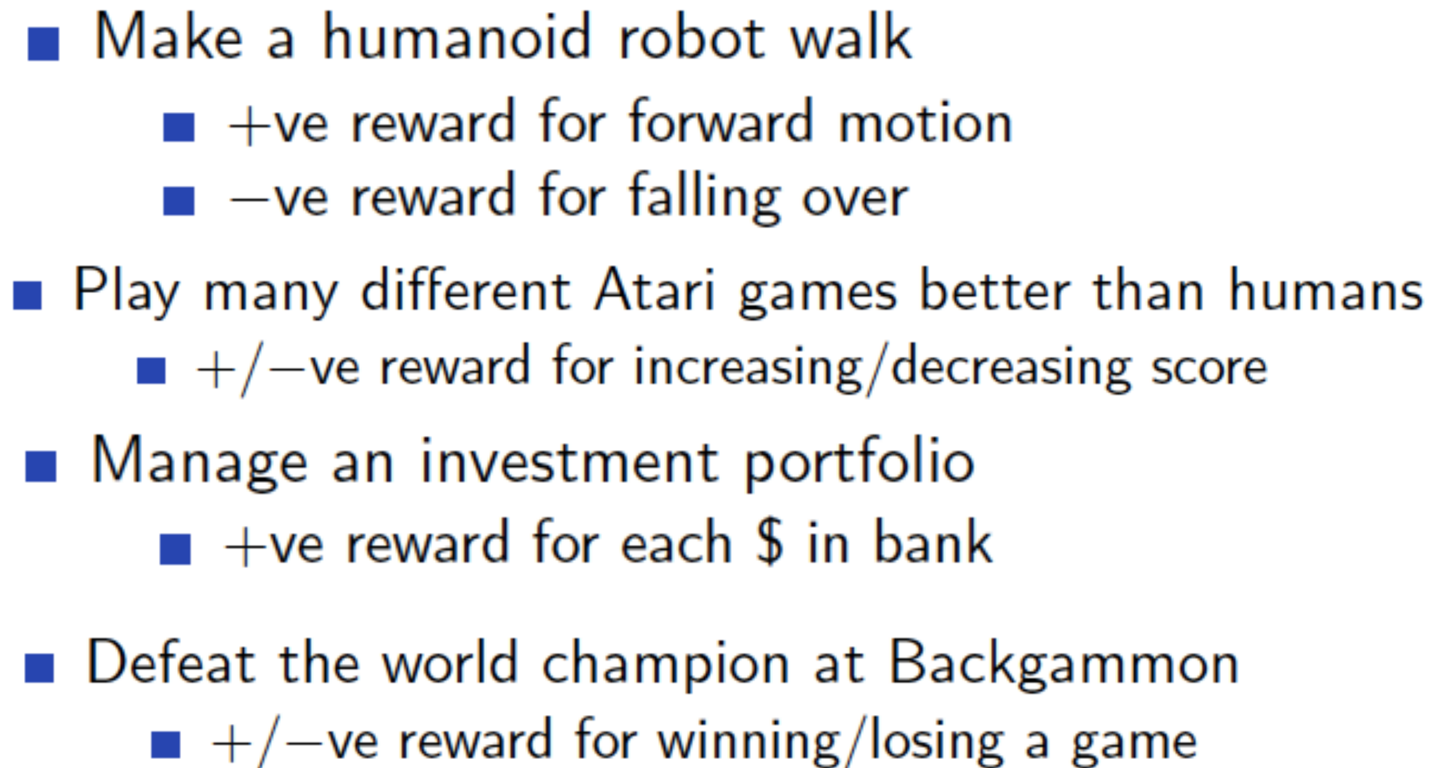
- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward

Reinforcement learning is based on the reward hypothesis

Definition (Reward Hypothesis)

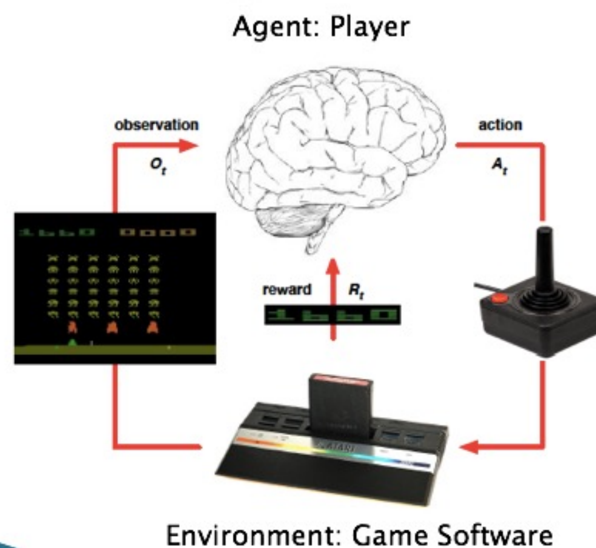
All goals can be described by the maximisation of expected cumulative reward

Examples of Rewards

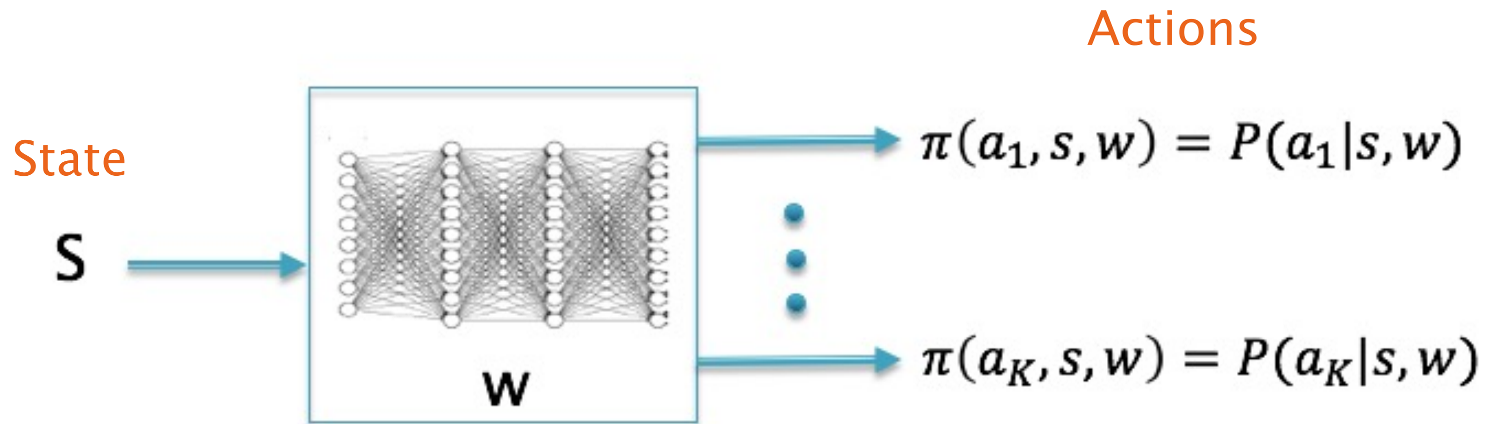


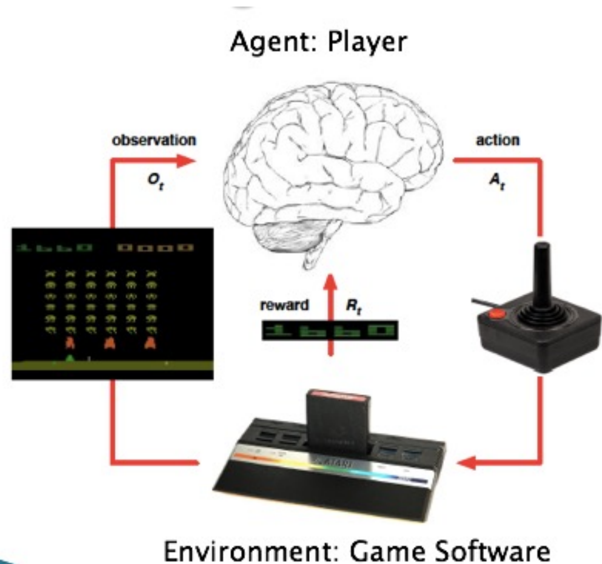
Policy

- A policy is the agent's behaviour
- It is a map from state to action,
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$



Can the Policy be Generated by a Neural Network?



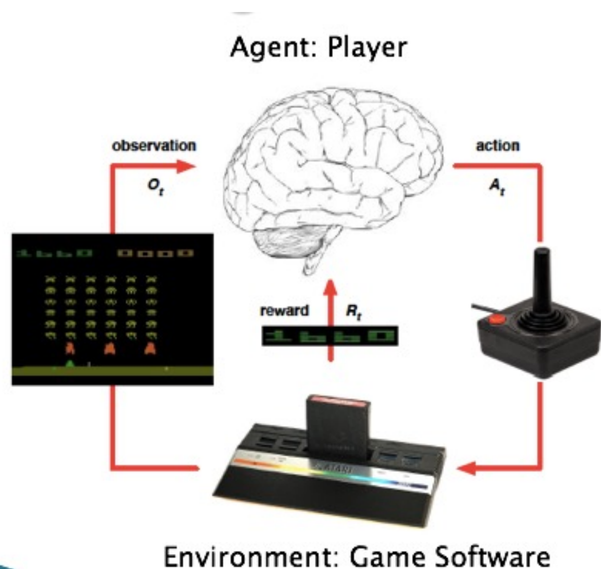


Deep Reinforcement Learning

Model

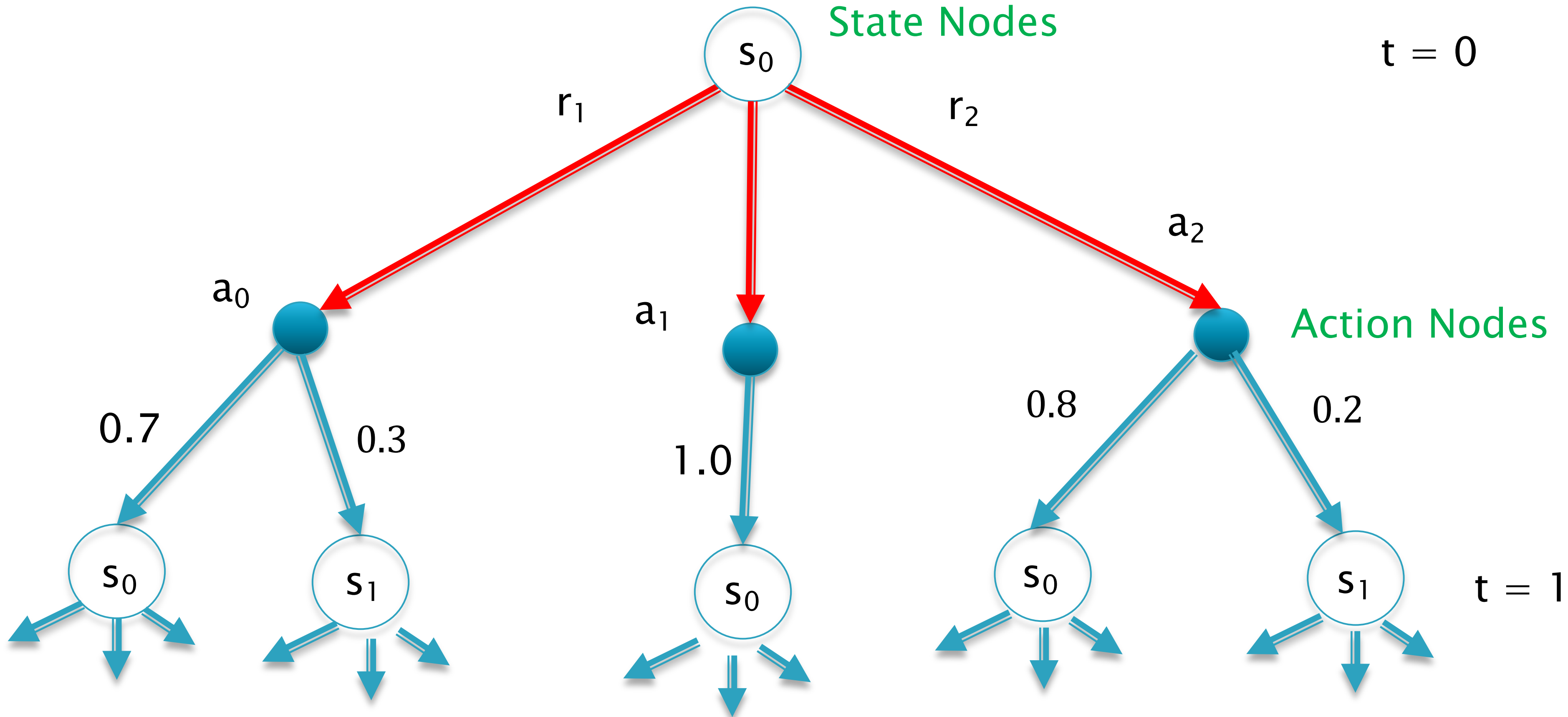
- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

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



In Reinforcement Learning a Model is represented by a Markov Decision Process

MDP Tree Representation



Value Functions and Action-Value Functions

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

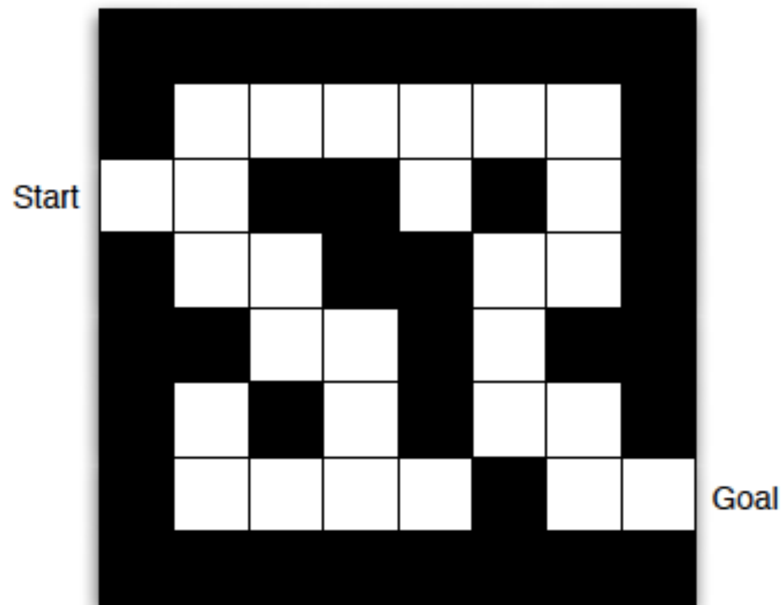
$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s \right]$$

$$q(s,a) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$$

A Value Function specifies what is good in the long run

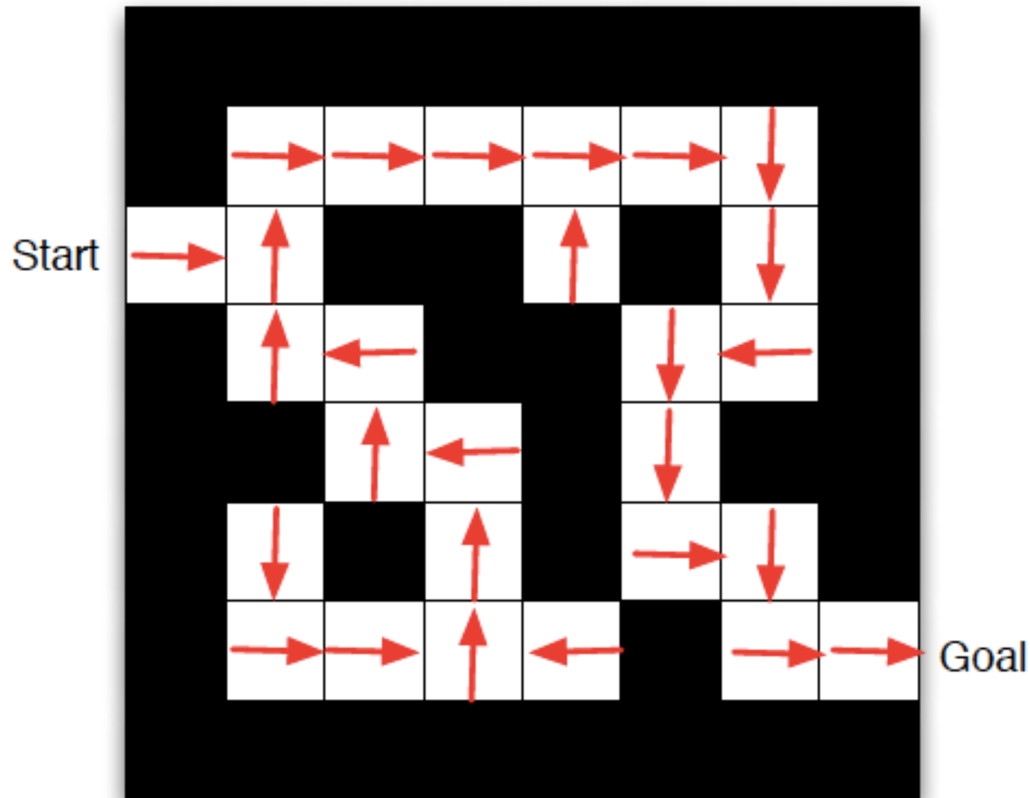
It is better to make decisions on the basis of Action Value Functions rather than Immediate Rewards

Maze Example



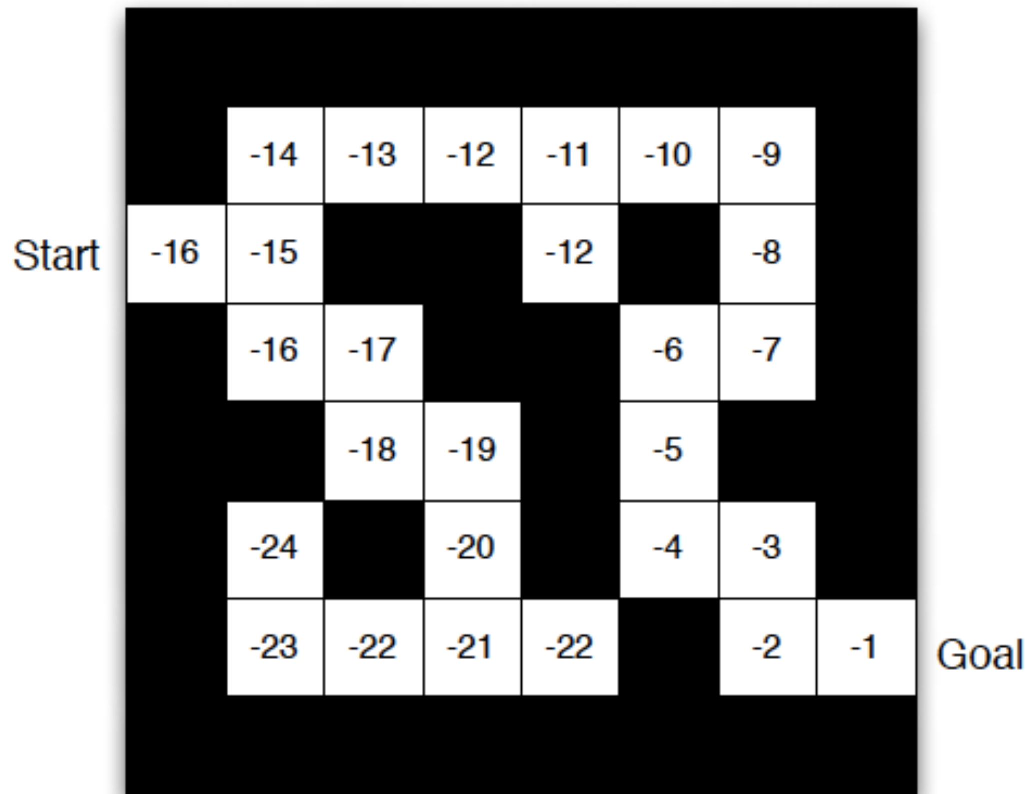
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



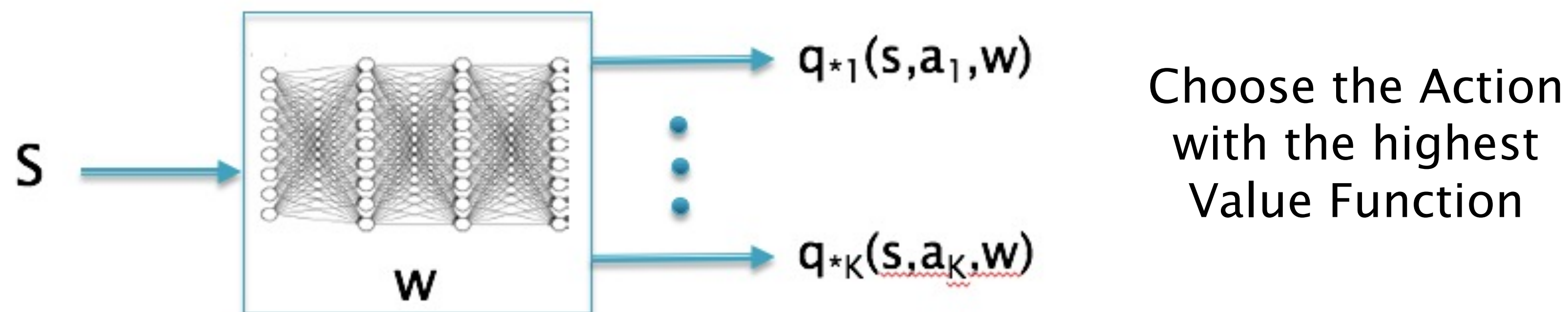
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state s

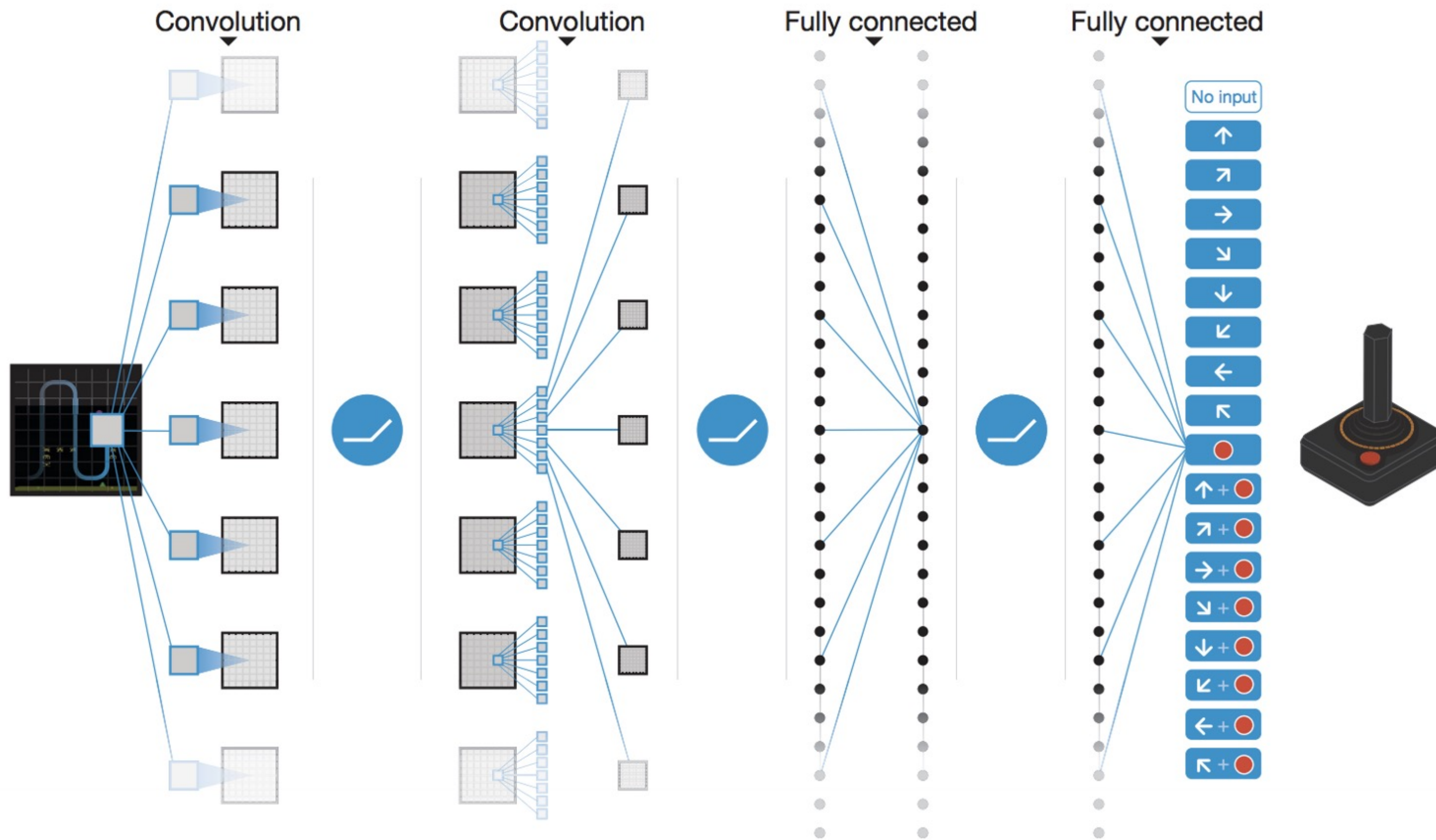
Generating Action Value Functions using Neural Networks



$$q(s,a) = E_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^{2} R_{t+3} + \cdots | S_t = s, A_t = a]$$

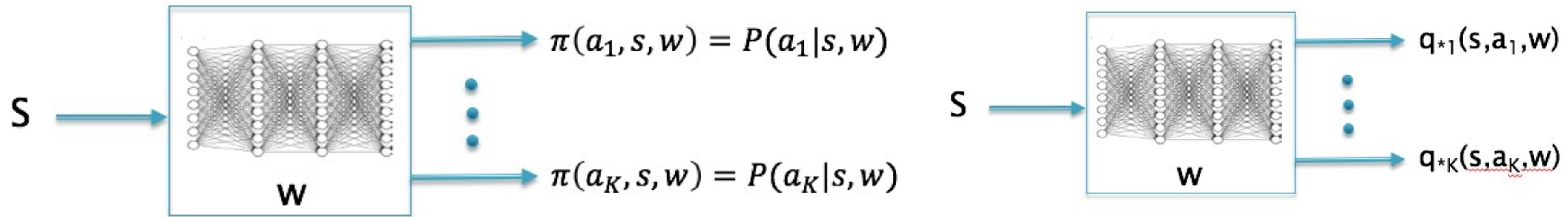
Deep Q Networks

Deep Reinforcement Learning



Deep Models allows RL algorithms to solve Complex Decision Making Problems End-to-End

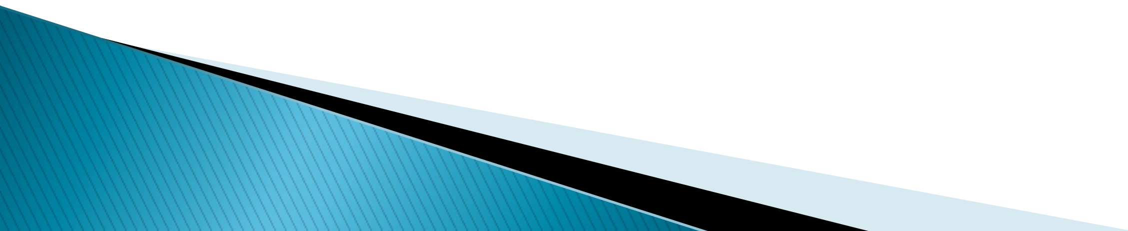
Central Problems of Deep RL



Train a Neural Network to implement the Policy Function $\pi(s)$

Train a Neural Network to implement the Action Value Function q(s,a)

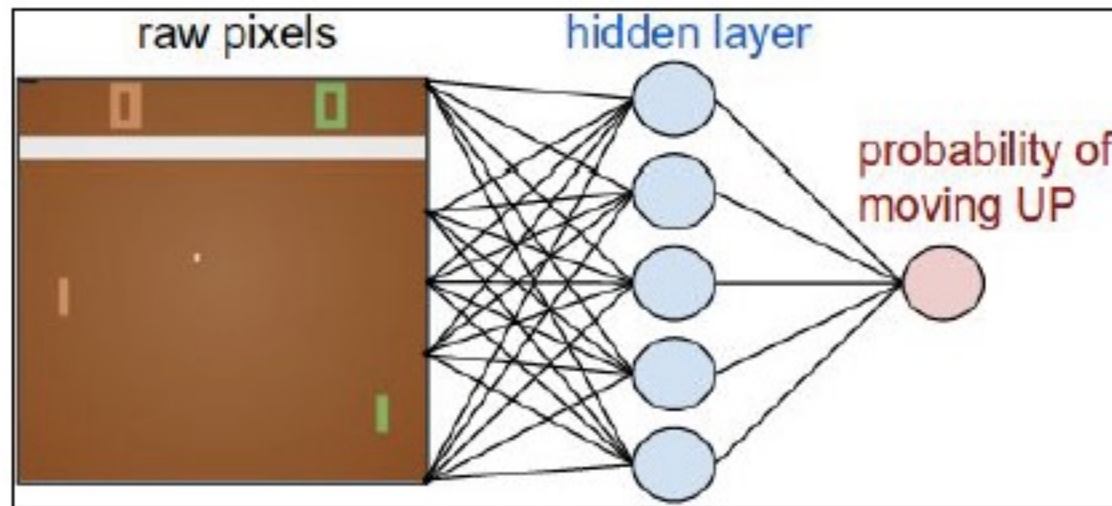
Playing Pong using Policy Gradients

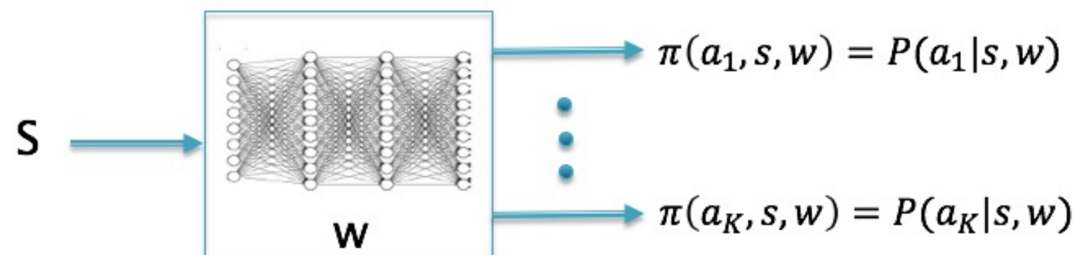


Policy Network for Pong

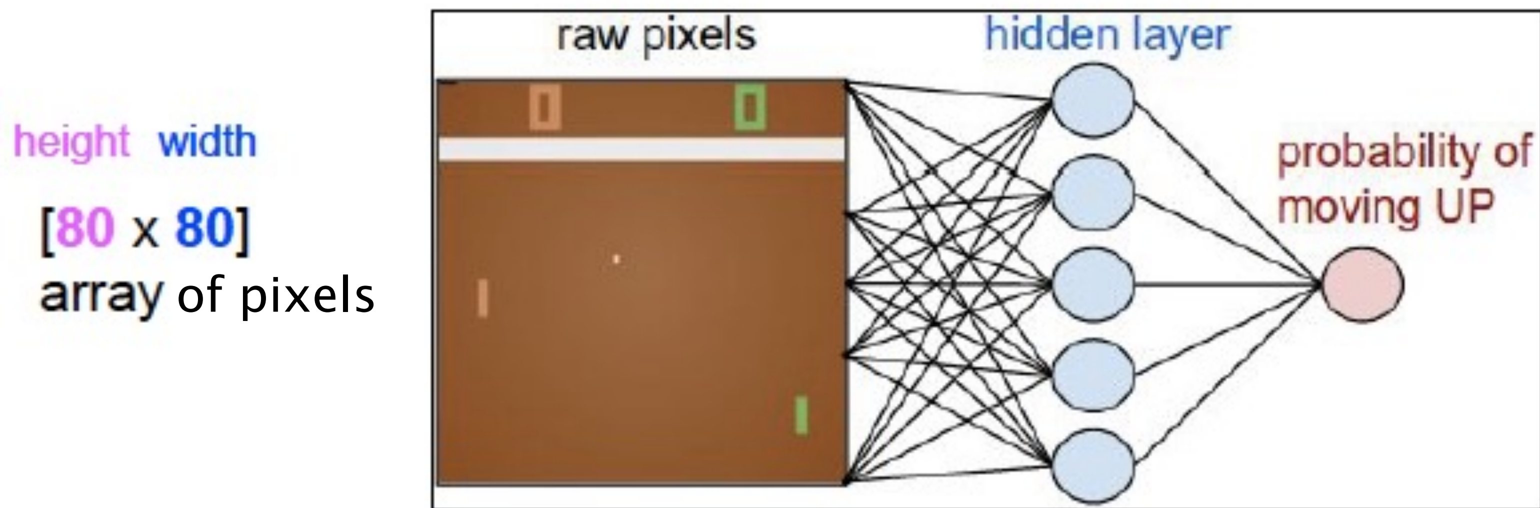
height width

[80 x 80]
array of
pixels





Policy Network for Pong



E.g. 200 nodes in the hidden network, so:

[(80*80)*200 + 200] + [200*1 + 1] = **~1.3M parameters**

Layer 1 Layer 2

Training a Pong Player using Labels – Supervised Learning

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

Training a Pong Player using Labels – Supervised Lerning

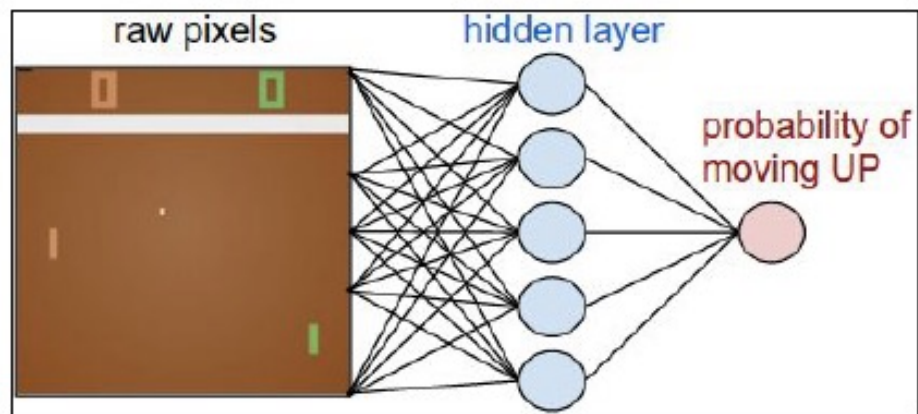
Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

maximize:

$$\sum_i \log p(y_i | x_i)$$

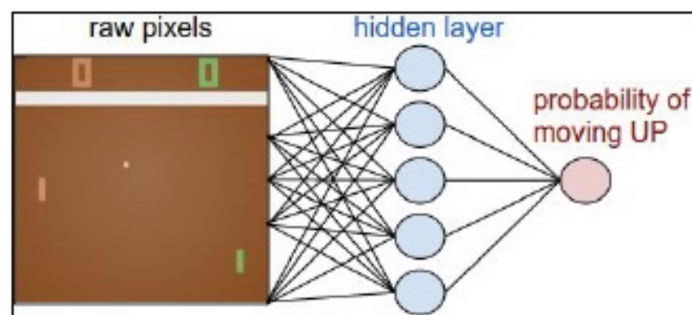
$$\mathcal{L} = -[t \log y + (1-t)\log(1-y)]$$



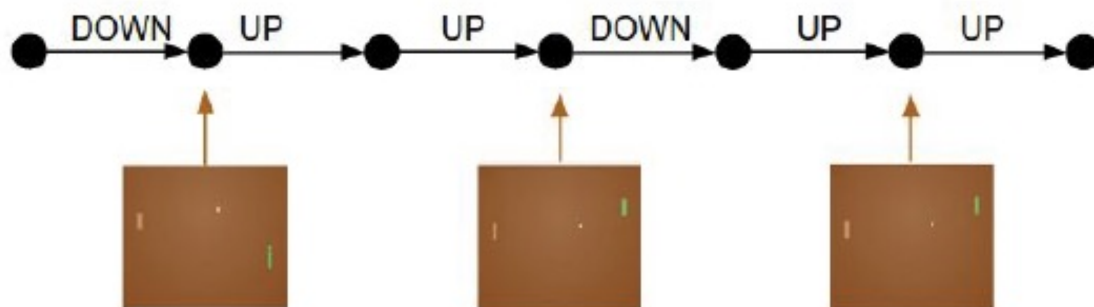
Except, we don't have labels...

Training a Pong Player using Sample Episodes – Reinforcement Learning

Let's just act according to our current policy...



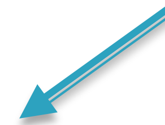
Rollout the policy and collect an episode



WIN

Training a Pong Player using Sample Episodes – Reinforcement Learning

Training Data



Collect many rollouts...

4 rollouts:

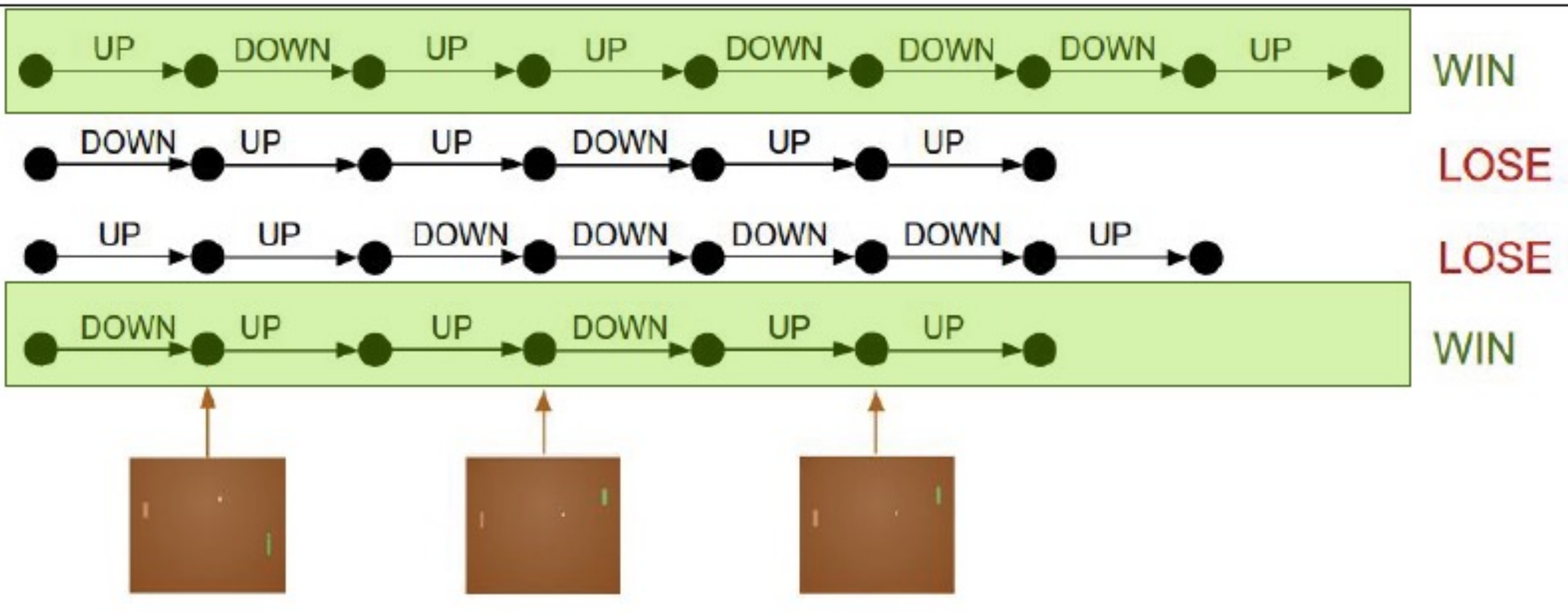
UP → DOWN → UP → UP → DOWN → DOWN → DOWN → UP — WIN

DOWN → UP → UP → DOWN → UP → UP — LOSE

UP → UP → DOWN → DOWN → DOWN → DOWN → UP — LOSE

DOWN → UP → UP → DOWN → UP → UP — WIN

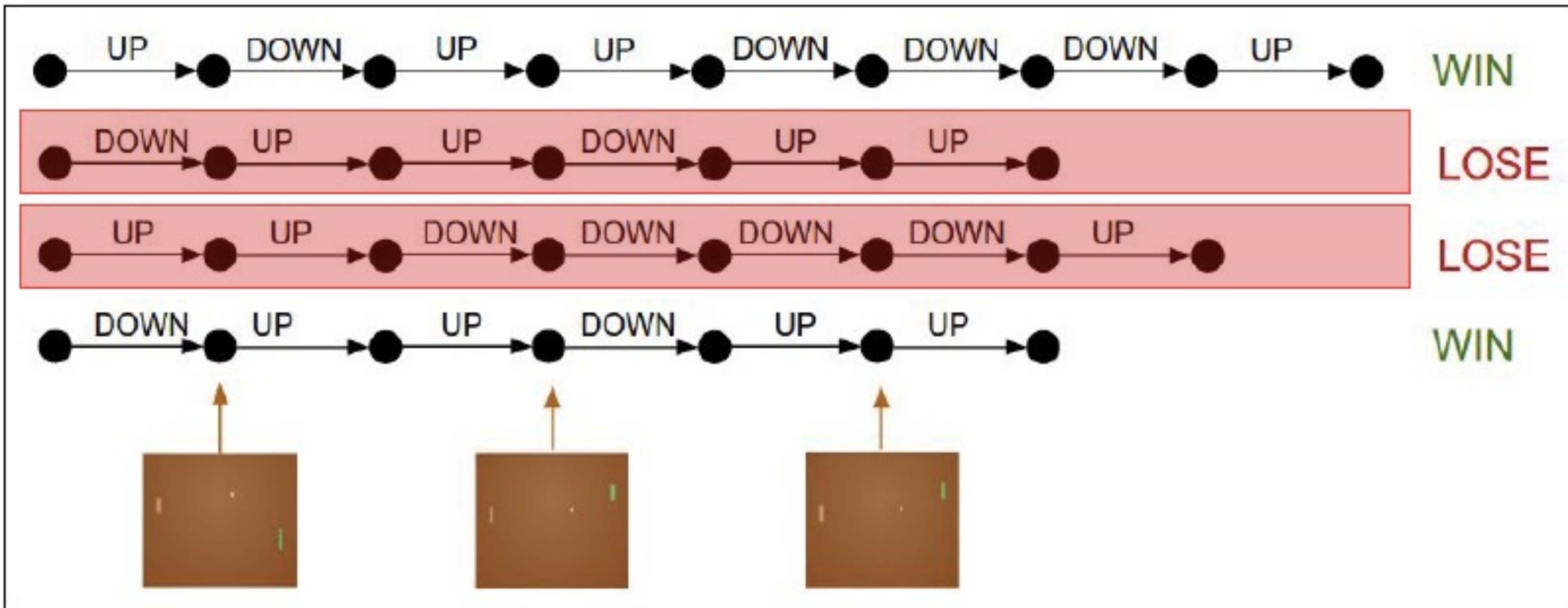
Training Data: WIN Episodes



Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

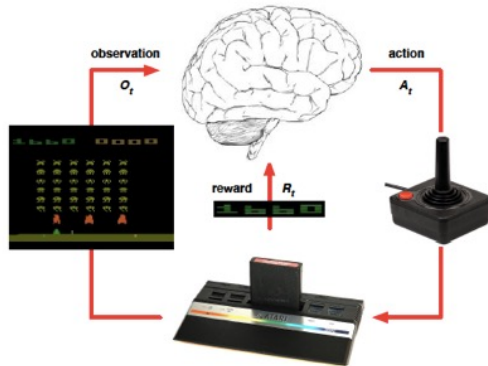
Training Data: LOSE Episodes

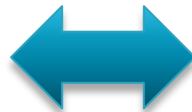


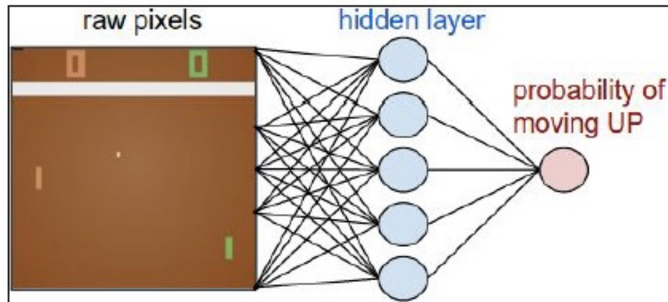
Pretend every action we took here was the wrong label.

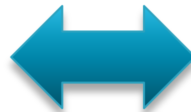
maximize: $(-1) * \log p(y_i \mid x_i)$

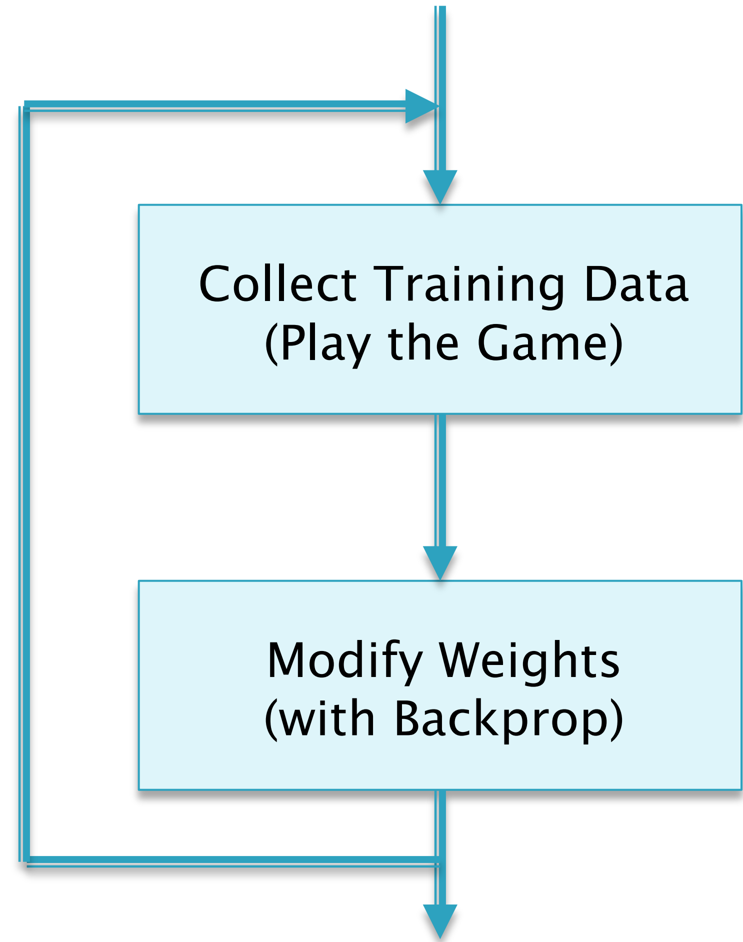
Training Loop



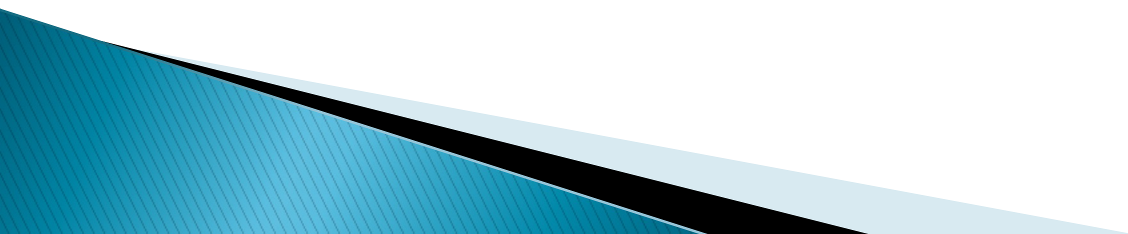




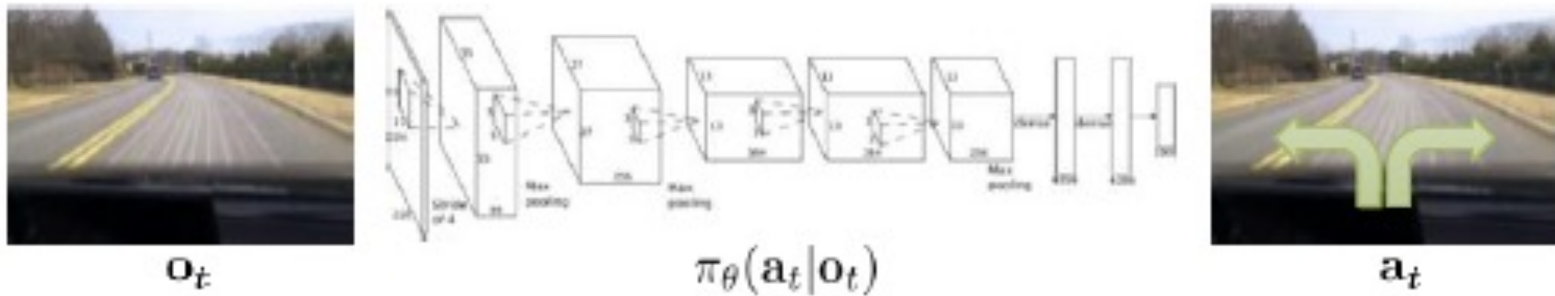




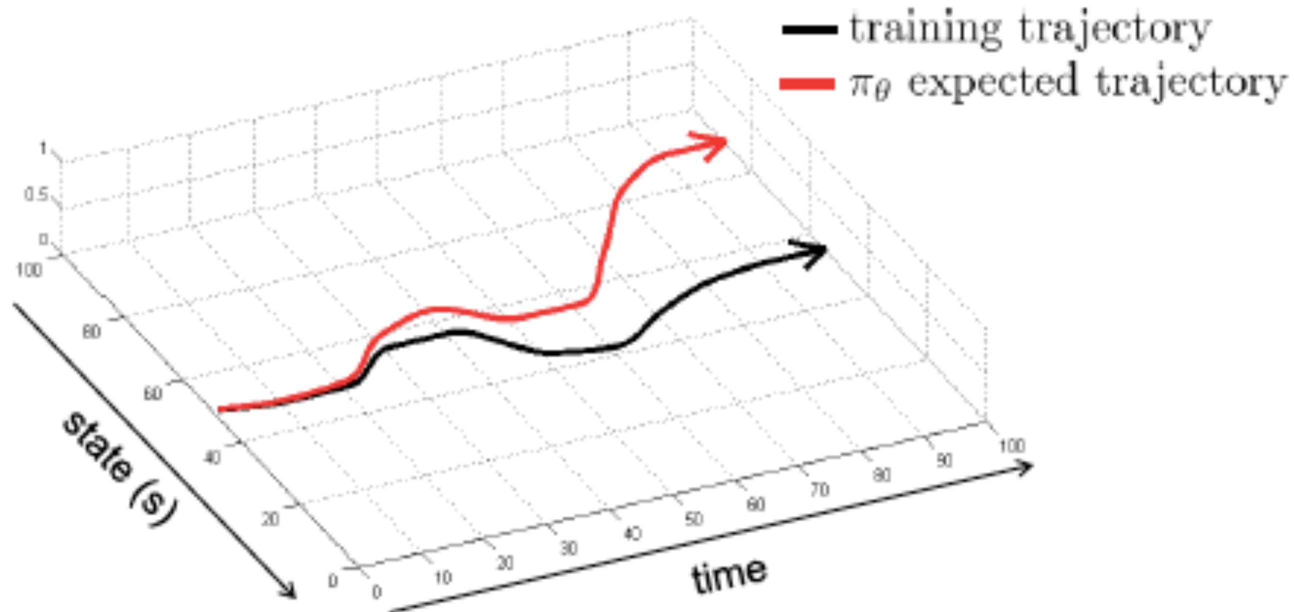
Self Driving Cars and Drones Using Imitation Learning



Training a Self Driving Car Using Imitation Learning



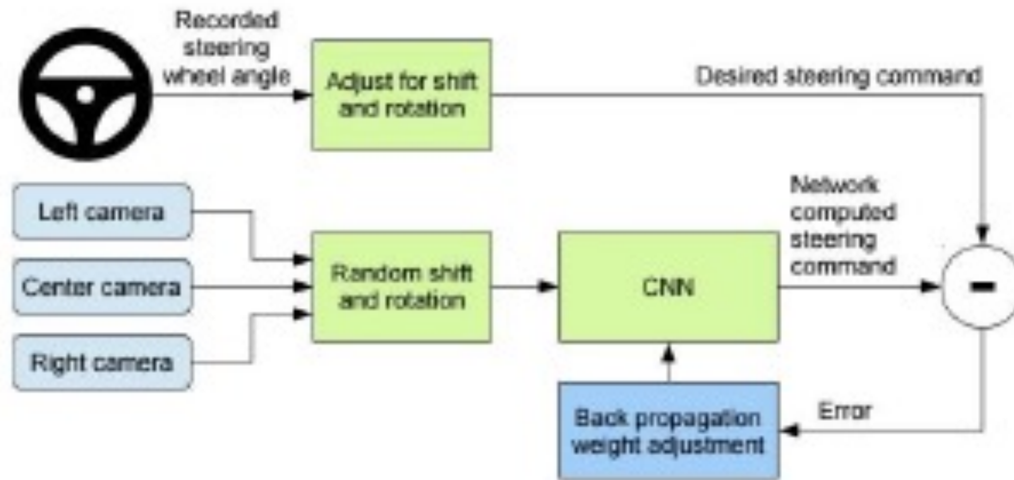
Does it Work?

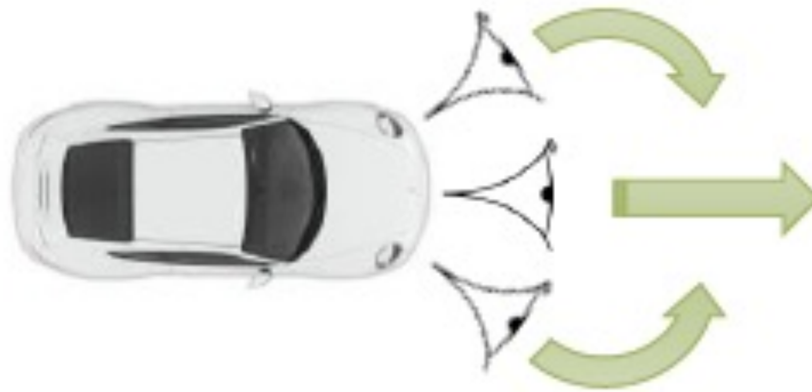


No!
Two Solutions:
- Multiple Cameras (NVIDIA)
- The Dagger Algorithm

NVIDIA Self Driving Car

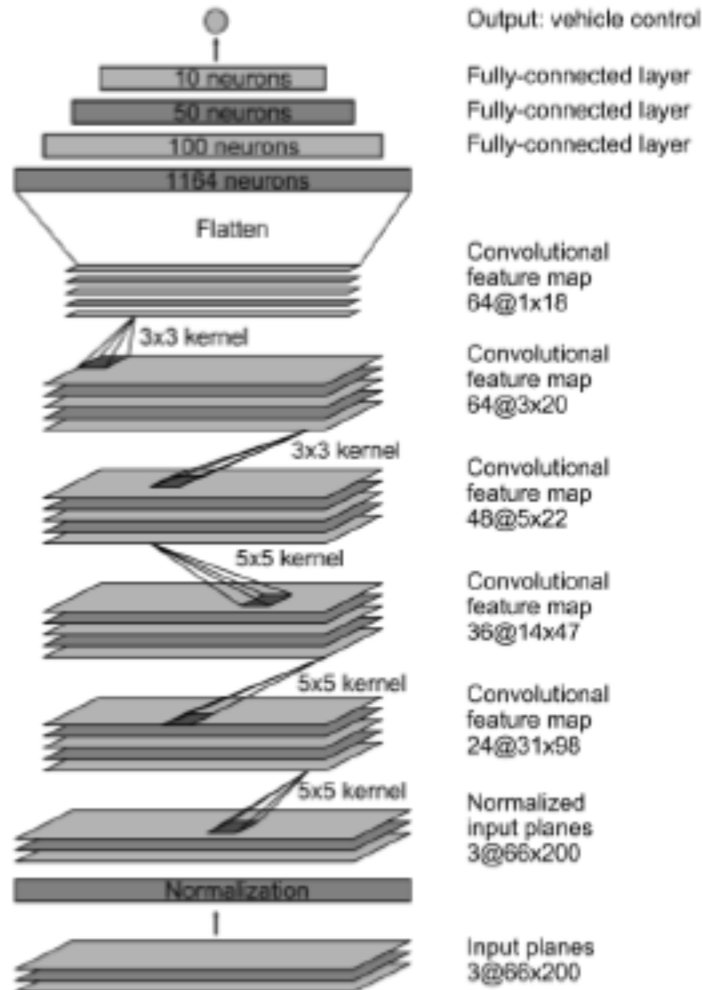
The NVIDIA Self Driving System





Bojarski et al. '16, NVIDIA

ConvNet used in the NVIDIA System



- 9 layers
  - 1 normalization layer
  - 5 convolutional layers
  - 3 fully connected layers
- 27 million connections
- 250 thousand parameters

The Dagger Algorithm

can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

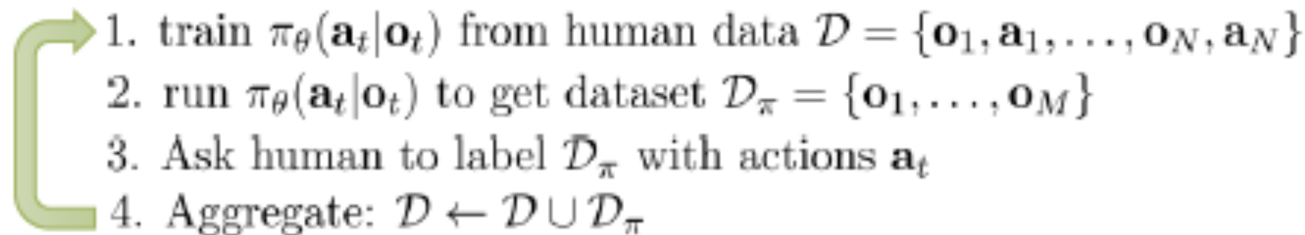
idea: instead of being clever about $p_{\pi_\theta}(\mathbf{o}_t)$, be clever about $p_{\text{data}}(\mathbf{o}_t)$!

DAgger: **D**ataset **Agg**regation

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\text{data}}(\mathbf{o}_t)$

how? just run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

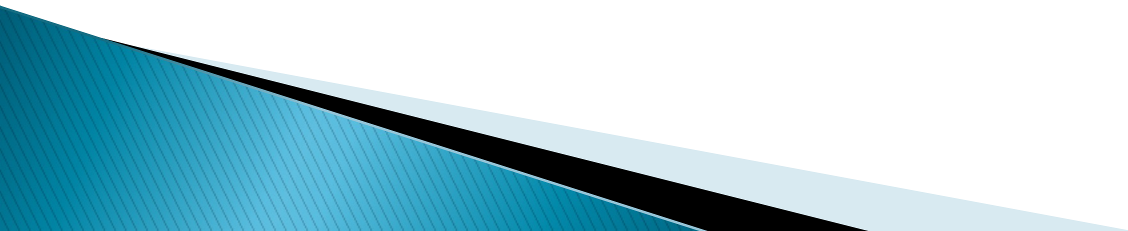
but need labels $\mathbf{a}_t$!



1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Drone Navigation Using Dagger

Playing GO with Deep Reinforcement Learning



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 450

POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card 40 years on
PAGE 462

NATUREASIA.COM
28 January 2016
Vol. 529, No. 7587

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1*], Aja Huang[1*], Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant

policies[13–15] or value functions[16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification[17], face recognition[18], and playing Atari games[19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image[20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

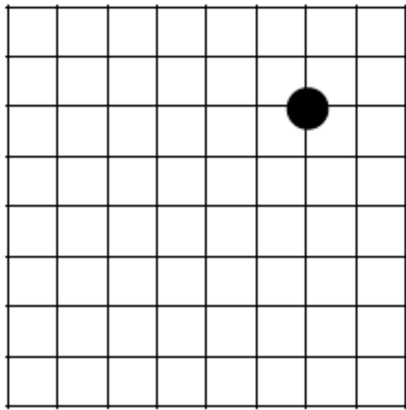
We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work[13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Game of Go

- The ancient oriental game of Go is 2500 years old
- Considered to be the hardest classic board game
- Considered a grand challenge task for AI *(John McCarthy)*
- Traditional game-tree search has failed in Go

The Game of GO

d = 1



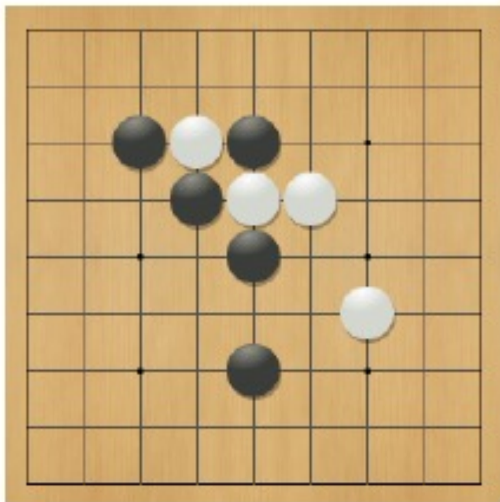
s (state)

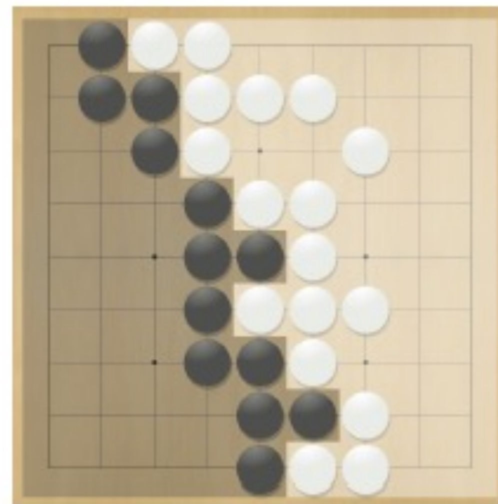
$$
= \begin{pmatrix}
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0
\end{pmatrix}
$$

(e.g. we can represent the board into a matrix-like form)

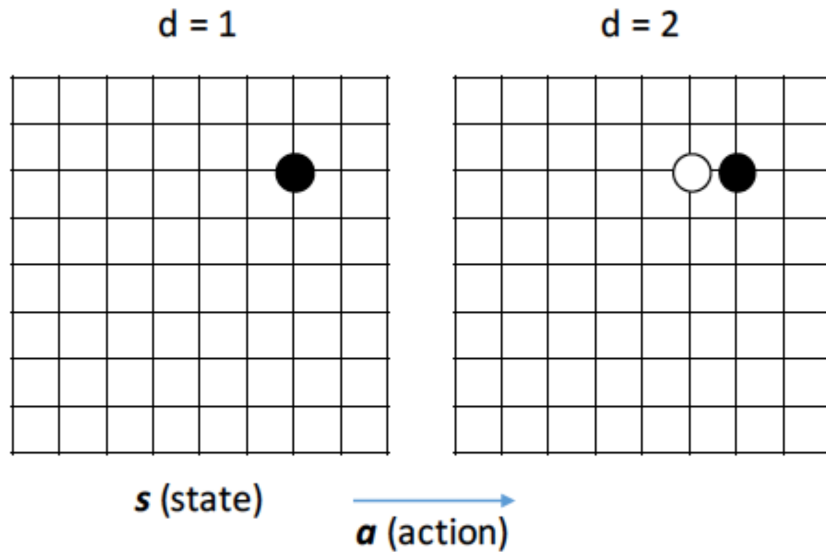
Rules of Go

- Usually played on 19x19, also 13x13 or 9x9 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game

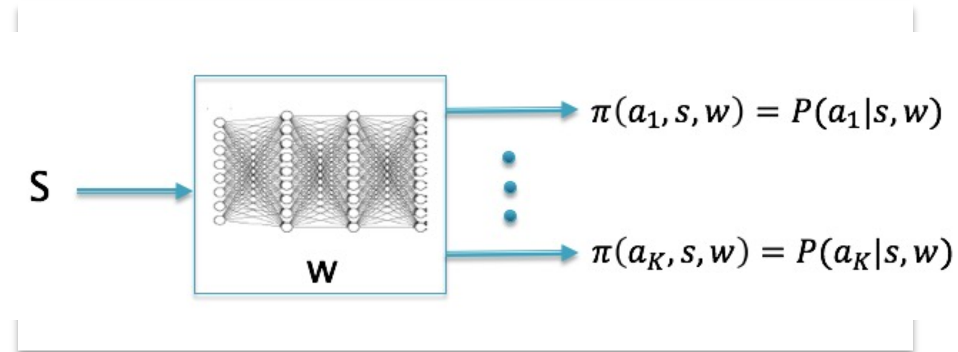




Computer Aided GO: AlphaGO



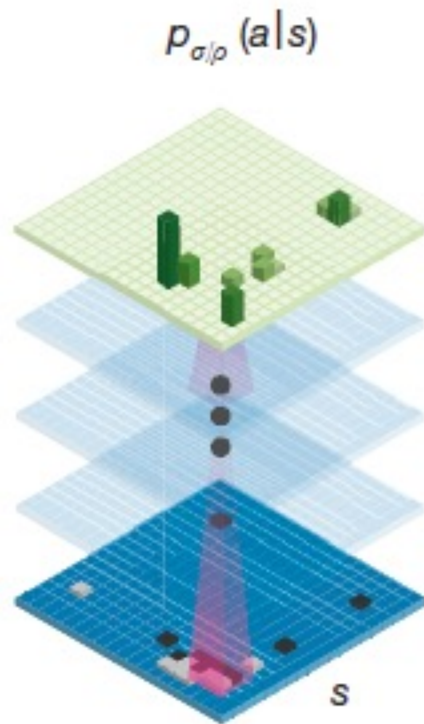
Given ***s***, pick the best ***a***



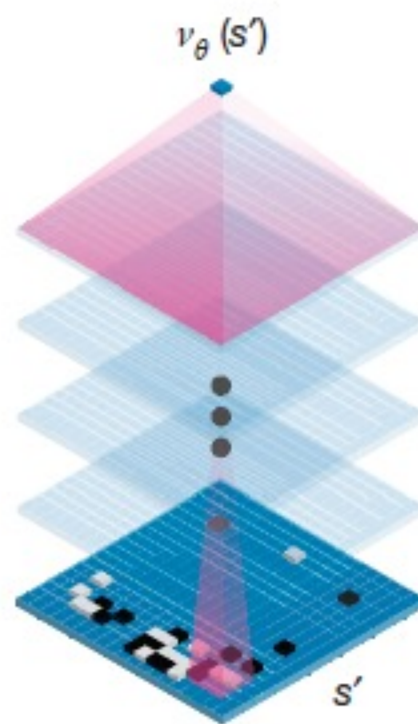
Algorithms Used:

1. Policy Gradients
2. Monte Carlo Tree Search

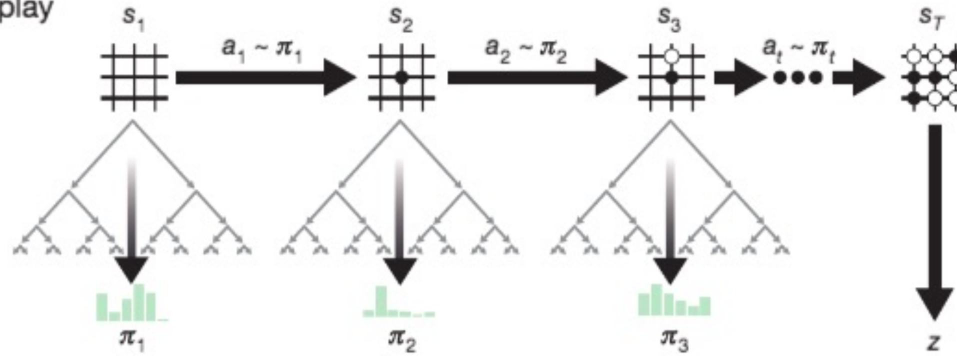
Policy network

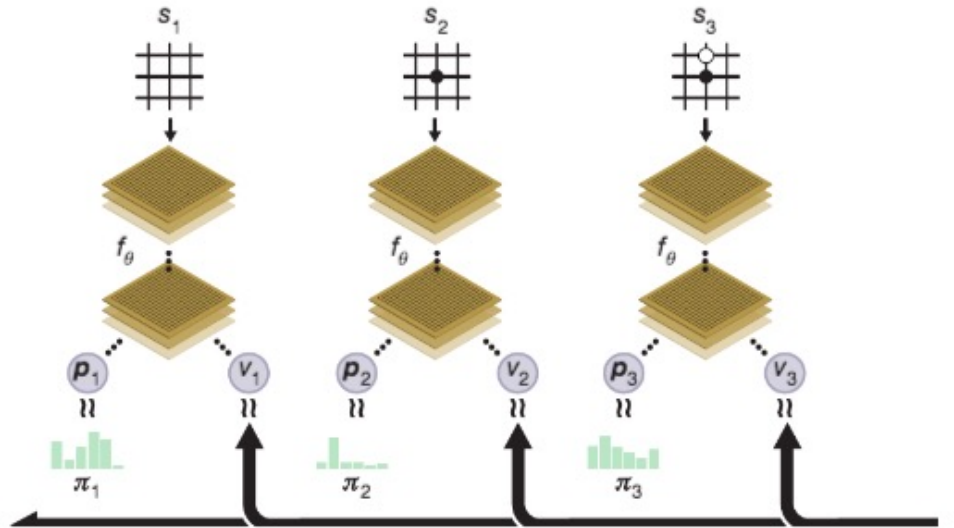


Value network



AlphaGo Zero





$$(\boldsymbol{p}, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

Further Reading

- “Reinforcement Learning, 2nd Edition” by Barto and Sutton