# Deep Reinforcement Learning

Ruohan Zhang

The University of Texas at Austin

April 14, 2019

# Outline

# The Reinforcement Learning Problem

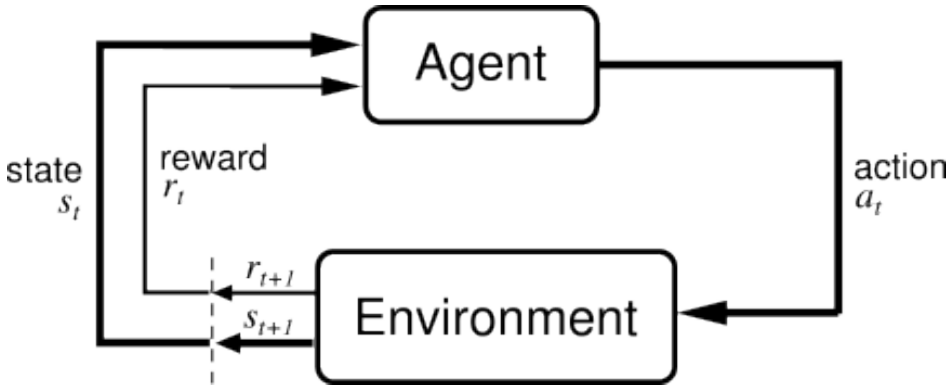



Figure: Agent-Environment Interaction

# Markov Decision Process (MDP)

## Definition

A tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a set of states.
- $\mathcal{A}$ is a set of actions.
- $\mathcal{T}$ is a state transition function. $\mathcal{T}^a_{ss'} = \mathbb{P}[s'|s, a]$.
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[r|s, a]$.
- $\gamma$ is a discount factor, $\gamma \in [0, 1)$.

# Policy and Value Function

## Policy

Agent behavior is fully specified by $\pi(s, a) = \mathbb{P}[a|s]$, one can directly optimize this by trying to maximize expected reward.

# Policy and Value Function

## Policy

Agent behavior is fully specified by $\pi(s, a) = \mathbb{P}[a|s]$, one can directly optimize this by trying to maximize expected reward.

## Action-value function

$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[v_t|s_t = s, a_t = a]$, expected return starting from state $s$, taking action $a$, and then following policy $\pi$.

# Policy and Value Function

## Policy

Agent behavior is fully specified by $\pi(s, a) = \mathbb{P}[a|s]$, one can directly optimize this by trying to maximize expected reward.

## Action-value function

$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[v_t|s_t = s, a_t = a]$, expected return starting from state $s$, taking action $a$, and then following policy $\pi$.

## Optimal action-value function

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

# Bellman Equations

## Action-value function recursive decomposition

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

# Bellman Equations

## Action-value function recursive decomposition

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

## Dynamic programming to solve MDP

Assumption: environment model $\mathcal{T}, \mathcal{R}$ is fully known.

# Model-free Reinforcement Learning: Q-learning

## Q-learning algorithm [4]

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  until $s$ is terminal

# Summary: Approaches to Reinforcement Learning

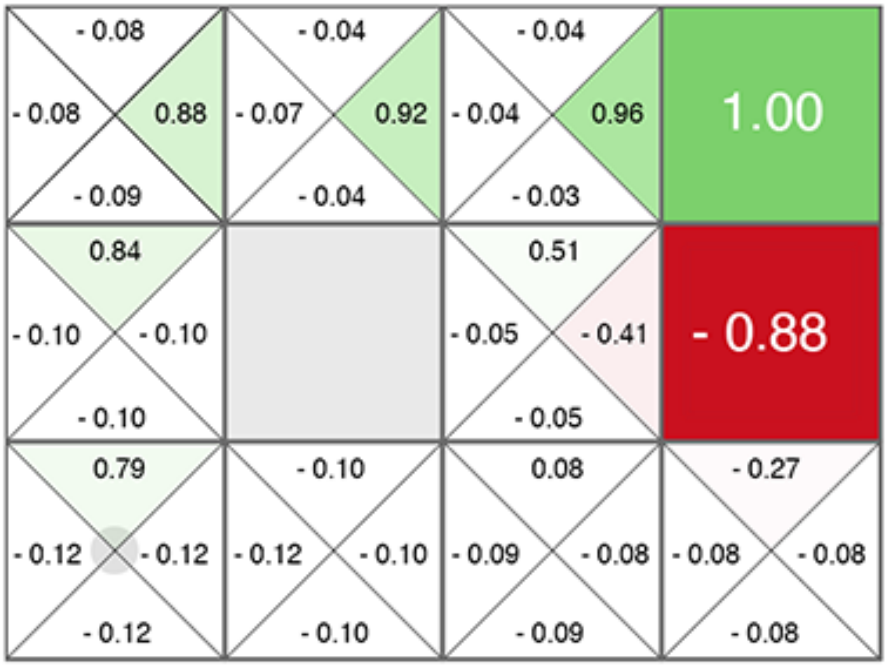
## To find the optimal policy (Source: David Silver)

1. Policy-based RL
   Search directly for the optimal policy $\pi^*$, this is the policy achieving maximum future reward.
2. Value-based RL
   Estimate the optimal value function $Q^*(s, a)$, this is the maximum value achievable under any policy.
3. Model-based RL
   Build a transition model of the environment, plan (e.g. by lookahead) using model.

# Limitations of Basic Reinforcement Learning Algorithms

## Basic reinforcement learning methods

Assumption: discrete states and actions (lookup Q table).



# Limitations of Basic Reinforcement Learning Algorithms

## Basic reinforcement learning methods

Assumption: discrete states and actions (lookup Q table).

## Key issues

- Approximation: continuous, high-dimensional, and noisy state/action space.
- Generalization: predict values of unseen states.

# Linear Function Approximation I

- Representing action-value function as a linear combination of features:

$$Q^{\pi}(s,a) \approx Q_w(s,a) = \sum_{j=1}^{n} \phi_j(s,a) w_j$$

# Linear Function Approximation I

- Representing action-value function as a linear combination of features:

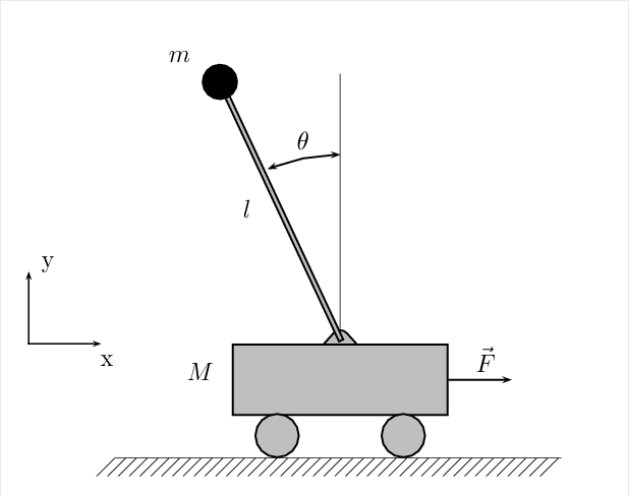
$$Q^{\pi}(s,a) \approx Q_w(s,a) = \sum_{j=1}^{n} \phi_j(s,a) w_j$$

- $\phi_j(s,a)$ are (binary) features:
  - e.g.,$\phi_j(s,a) = 1$ if taking $a$ in $s$ will move the agent closer to a reward
- Linear form prohibits representation of interactions between features.

# Linear Function Approximation II

## Limitation: cart-pole example

Linear form prohibits representation of interactions between features.
Features dependency: angle, angular velocity.



# Nonlinear Function Approximation

## Nonlinear function approximator?

Convergence result:

| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| Gradient Q-learning | ✓ | ✓ | ✗ |

(✓) = chatters around near-optimal value function

Slide courtesy: David Silver

# Selecting Nonlinear Function Approximator

What's special about training data in reinforcement learning? (versus traditional supervised learning, e.g., visual recognition)

- Successive, highly correlated samples (not i.i.d.)
- Non-stationary learning target $Q^{\pi}(s, a)$
- Selected action affect next sample obtained
- Delayed feedback

# One-step Further: Deep Reinforcement Learning

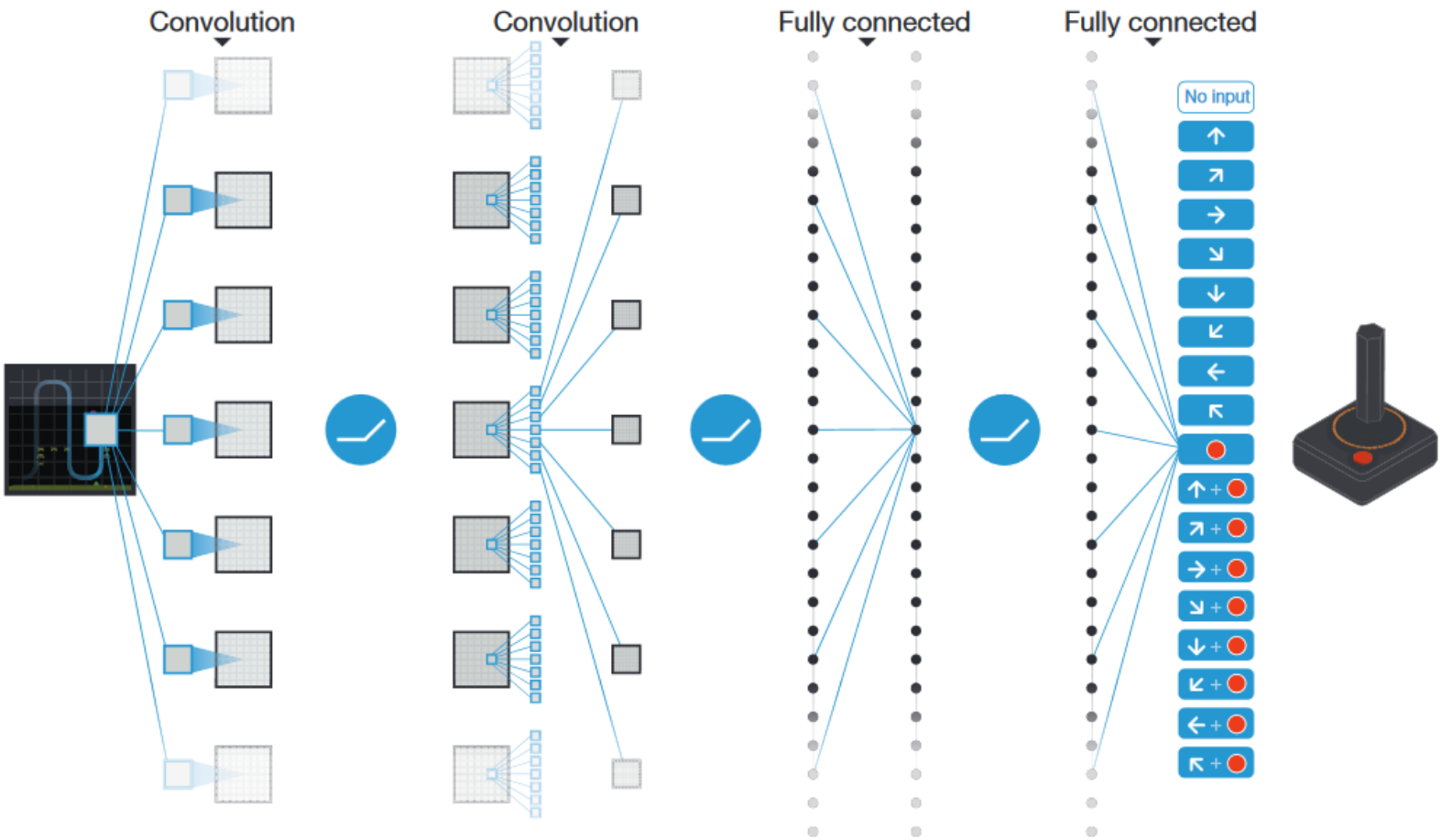
**Human-level control through deep reinforcement learning [2]**

Google DeepMind


# Goals

- The state space is the raw pixel space.
  - In normal RL, the state space is manually defined.
- Build a deep network as function approximator for Q-learning.
  - An end-to-end visuomotor system that can learn features matter for control.

# The Deep Q Network [2]



# Again, Challenges

## The challenge: stability (Source: David Silver)

Naive Q-learning oscillates or diverges with neural nets.

1. Scale of rewards and Q-values is unknown, gradients can be large, unstable when backpropagated.
2. Data is sequential, successive samples are correlated, non-i.i.d.
3. Policy changes rapidly with slight changes to Q-values, may oscillate.

# Addressing the Challenges: Stable Deep RL

## Challenge 1: reward magnitude (Source: David Silver)

Scale of rewards and Q-values is unknown, gradients can be large, unstable when backpropagated.

- Rescaling reward to $[-1, +1]$ to constrain gradient magnitude.
  - What's the drawback of doing this?

# Addressing the Challenges: Stable Deep RL

## Challenge 2: correlated data (Source: David Silver)

Data is sequential, successive samples are correlated, non-i.i.d.

- Experience replay to break data correlations.
  - Take action $a_t$.
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$.
  - Sample random mini-batch of transitions from $\mathcal{D}$.

# Addressing the Challenges: Stable Deep RL

## Challenge 3: unstable learning target (Source: David Silver)

Policy changes rapidly with slight changes to Q-values, may oscillate.

- Fixed target Q-learning
  - Computer Q-learning targets w.r.t. old, fixed parameters $\theta^-$.
  - Periodically update $\theta$.

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s_{t+1}\sim\mathcal{D}}[(r + \gamma \max_{a'} Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s, a; \theta))^2]$$

# Stable Deep RL Results

These techniques make deep Q network trainable!

| | Q-learning | Q-learning + Target Q | Q-learning + Replay | Q-learning + Replay + Target Q |
|---|---|---|---|---|
| Breakout | 3 | 10 | 241 | **317** |
| Enduro | 29 | 142 | 831 | **1006** |
| River Raid | 1453 | 2868 | 4103 | **7447** |
| Seaquest | 276 | 1003 | 823 | **2894** |
| Space Invaders | 302 | 373 | 826 | **1089** |

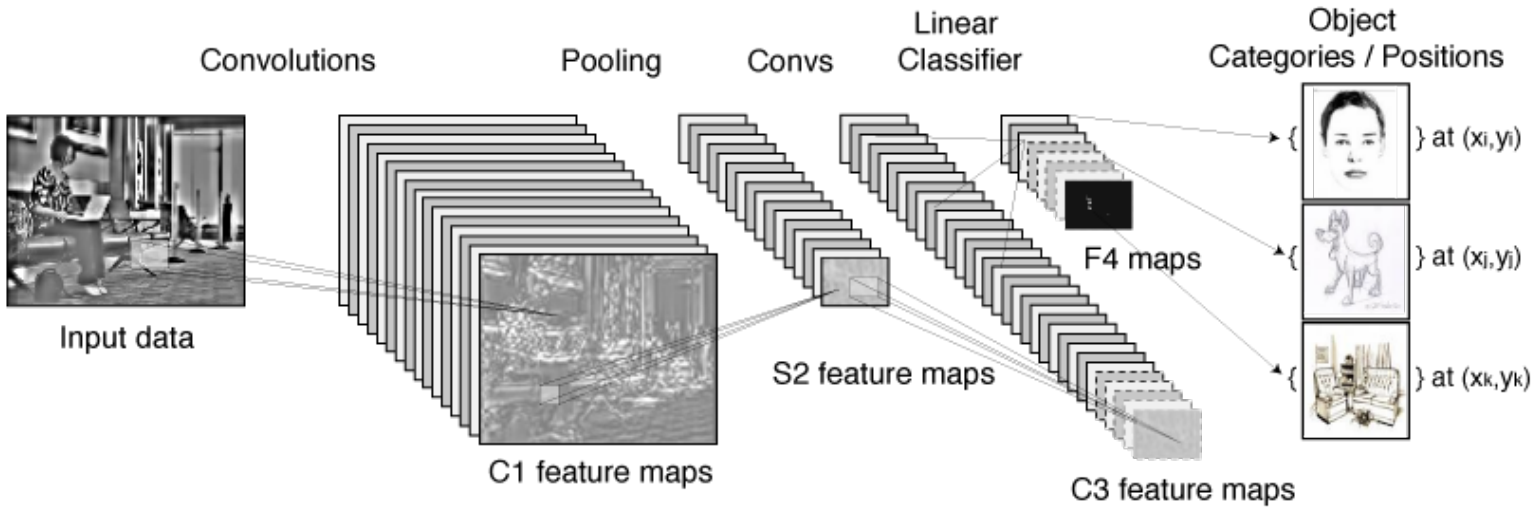
Source: David Silver.

# The Deep Q-Learning Algorithm [1]

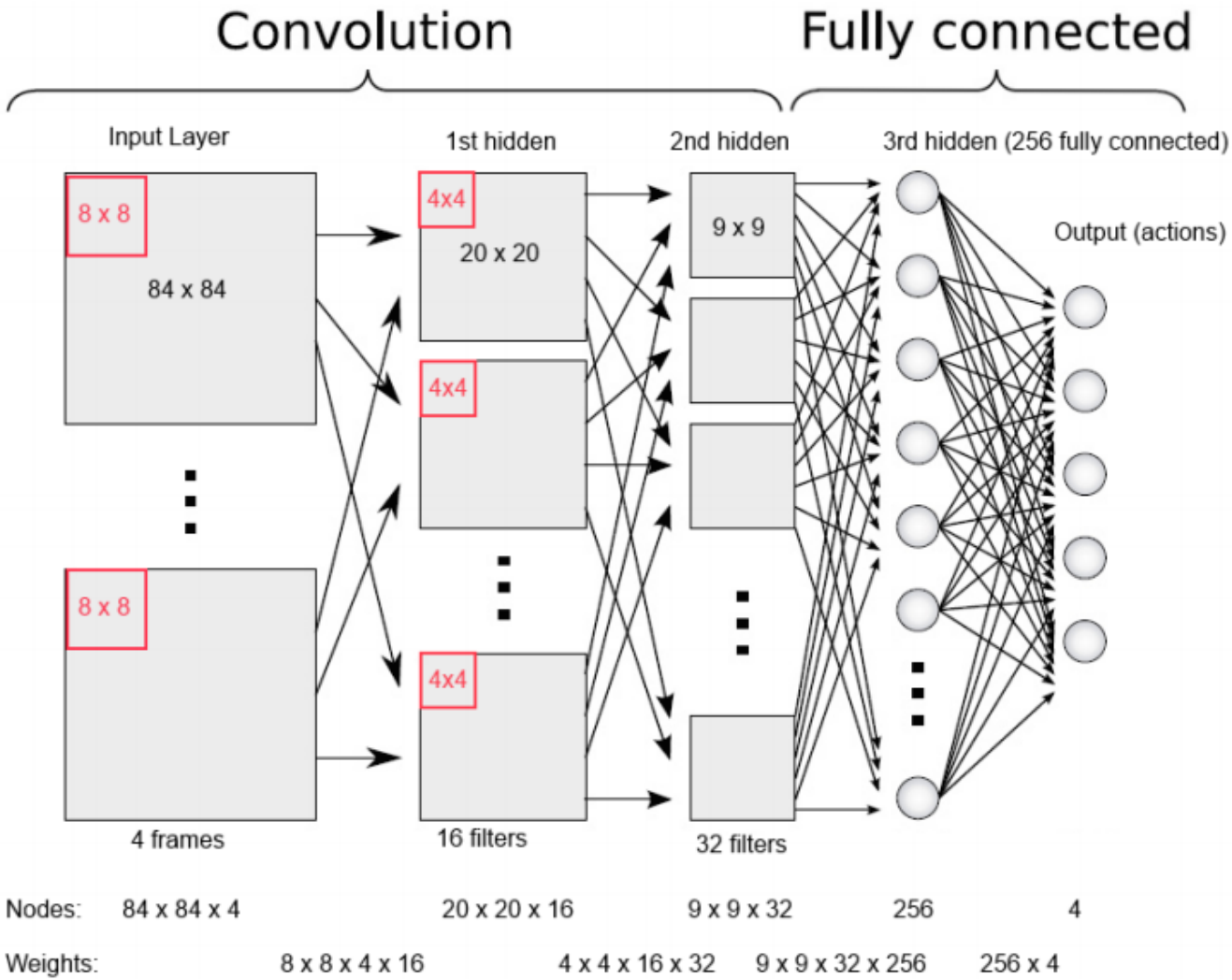
**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  **end for**
**end for**

# Convolutional Neural Network



# Network Architecture: Hyperparameters I



# Network Architecture: Hyperparameters II

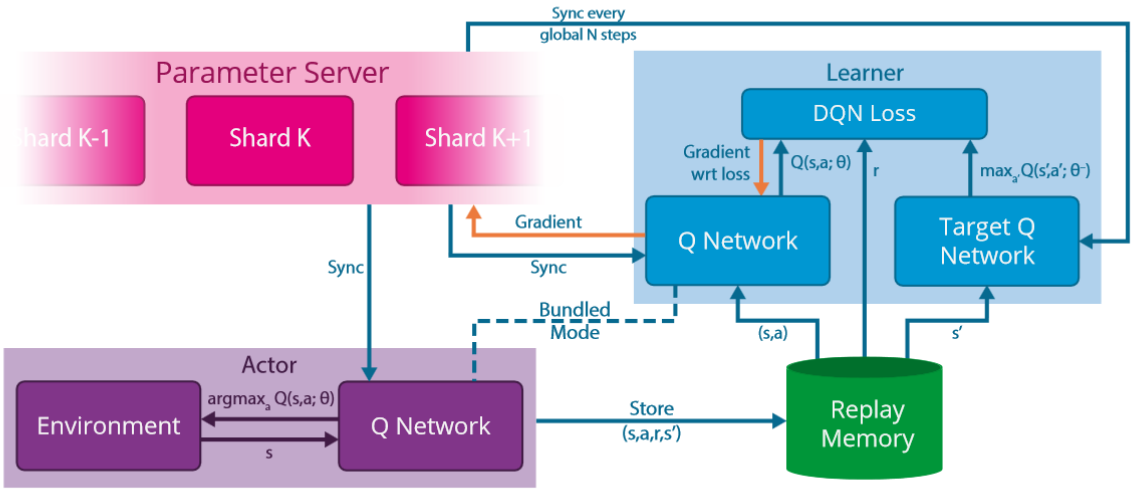
- Input: $84 \times 84 \times 4$ image produced by $\phi$.
- Hidden layer 1 (H1): convolves 16 $8 \times 8$ filters with stride 4 (skip by 4) with input image and applies a rectifier nonlinearity.
  - Rectifier: $f(x) = \max(0, x)$.
- H2: convolves 32 4 x 4 filters with stride 2, followed again by a rectifier nonlinearity.
- H3: fully connected, consists of 256 rectifier units.
- Output: fully connected linear layer with single output for each action.
  - 4 to 18 valid actions, depending on the game.

# Experiments

- 48 Atari games [2]
- Same network architecture, learning algorithm, and parameters.
- Replay memory contains most recent million frames.

# Large Scale Learning

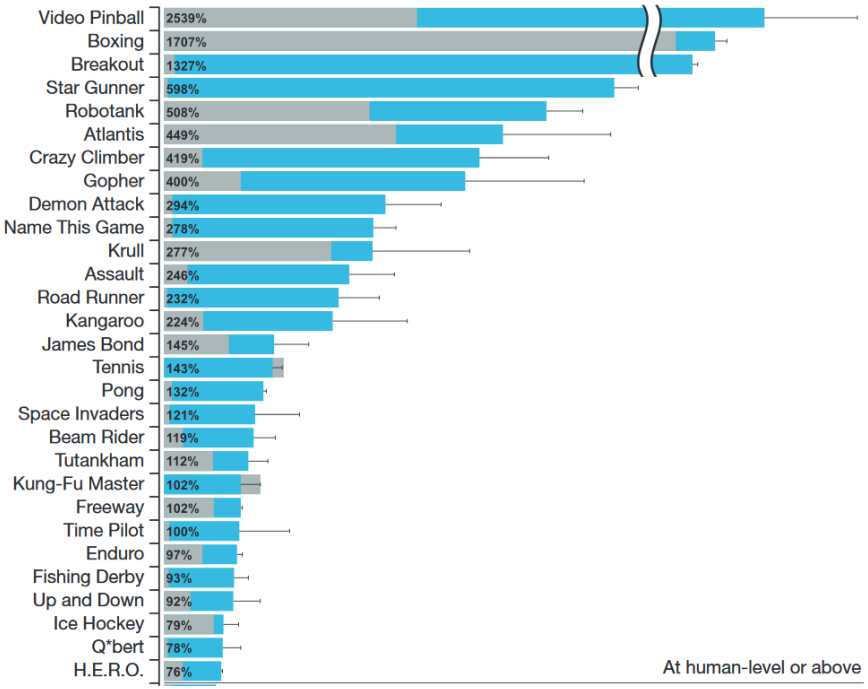
## Gorila (GOogle ReInforcement Learning Architecture)

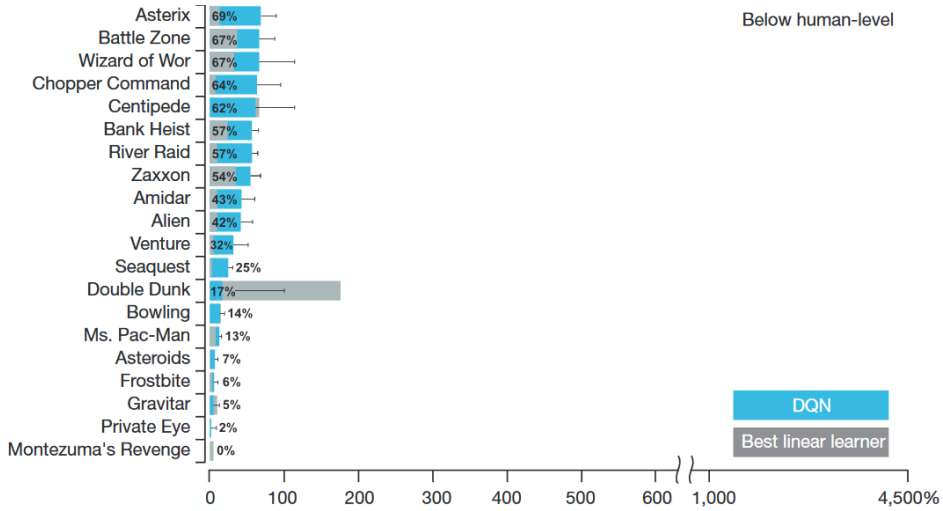



- Parallel acting: generate new interactions
- Distributed replay memory: save interactions
- Parallel learning: compute gradients from replayed interactions
- Distributed neural network: update network from gradients

100 parallel actors and learners result in significant faster training (10x) and doubled performance on majority of the games. Source: David Silver.

# Results [2]


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
At human-level or above

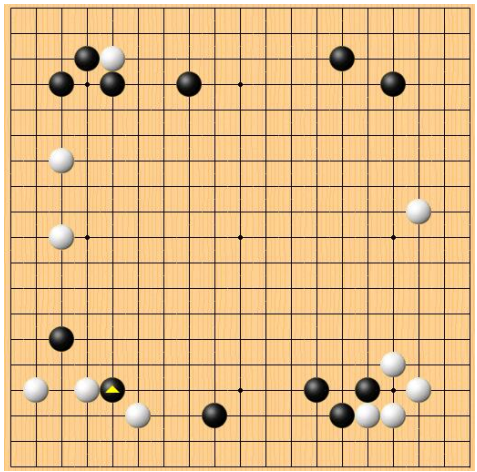

Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

# Another Landmark




Figure: Deep Blue vs. Garry Kasparov; AlphaGo vs Lee Sedol;
*Nature*: Mastering the game of Go with deep neural networks and tree search [3].

# The Game of Go



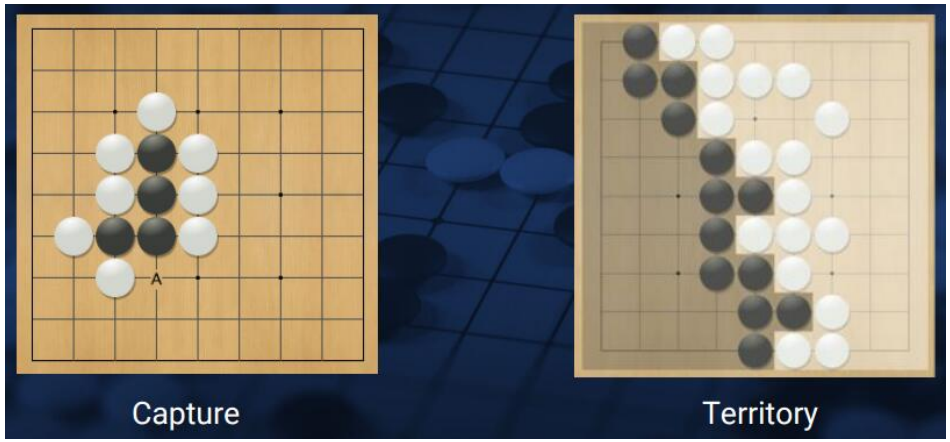


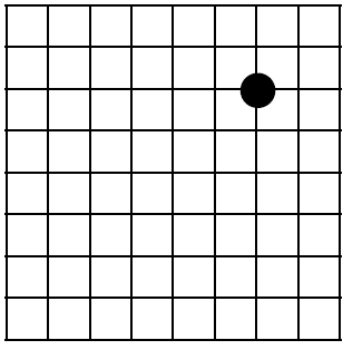

Figure: The game of Go (1046BCE - present). Source: wiki. The standard game board is 19 by 19. Legal game state space size: $2.08x10^{170}$.

# Results


- AlphaGo Master 60:0 world top players in fast games; 3:0 Ke Jie (current Rank1 in human players)
- AlphaGo Zero 100:0 its elder brother AlphaGo Lee; 89:11 AlphaGo Master

# Computer Go AI – Definition

d = 1



***s*** (state)

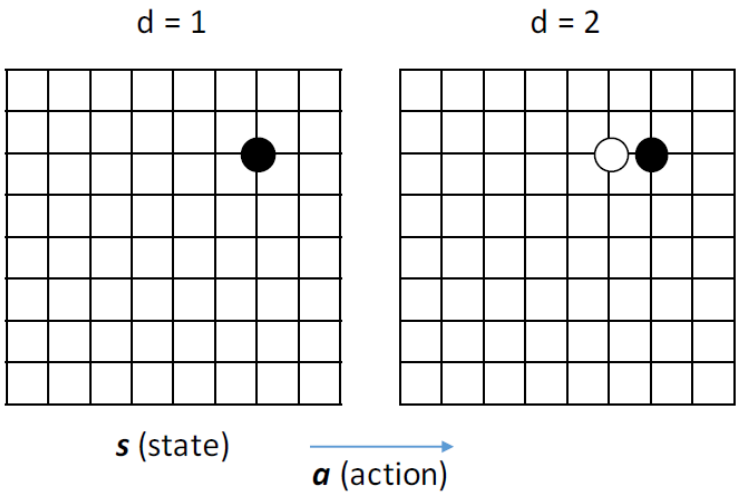
=

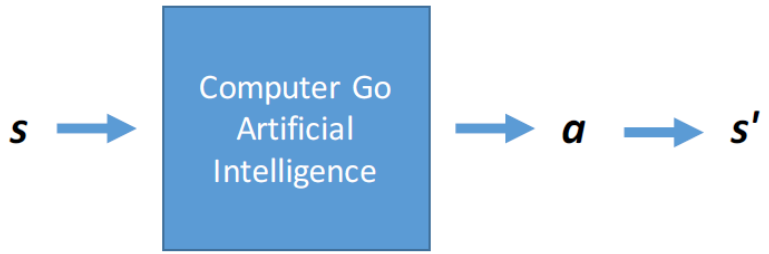
$$\begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&\mathbf{1}&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{pmatrix}$$

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

** The actual model uses other features than board positions as well*

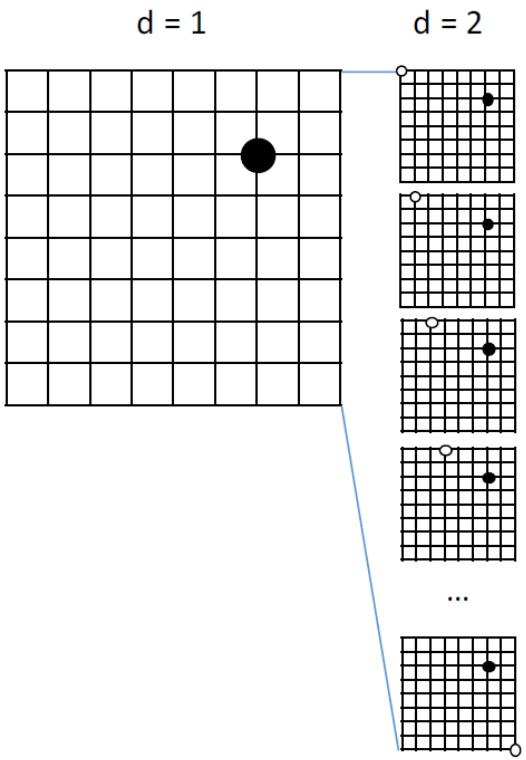
# Computer Go AI – Definition



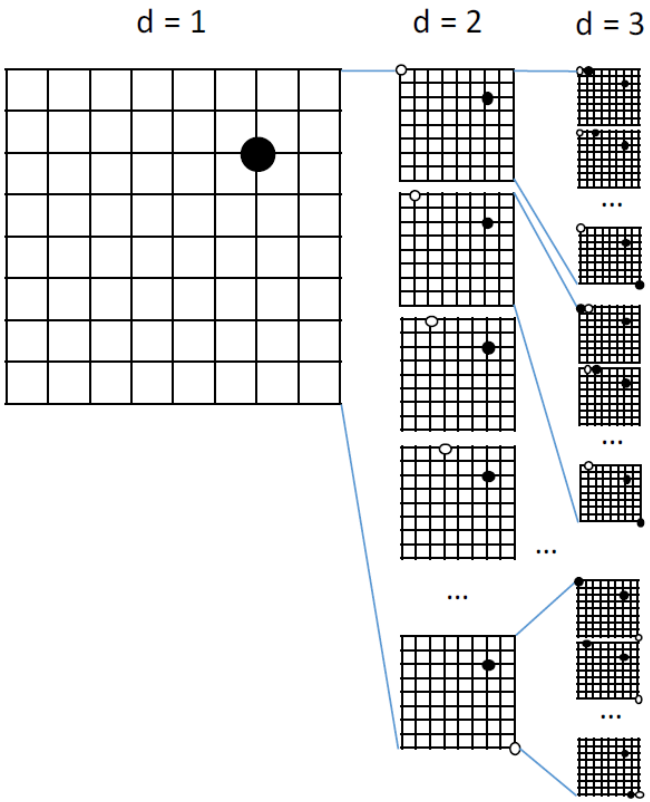

Given $s$, pick the best $a$

# Computer Go AI – An Implementation Idea?

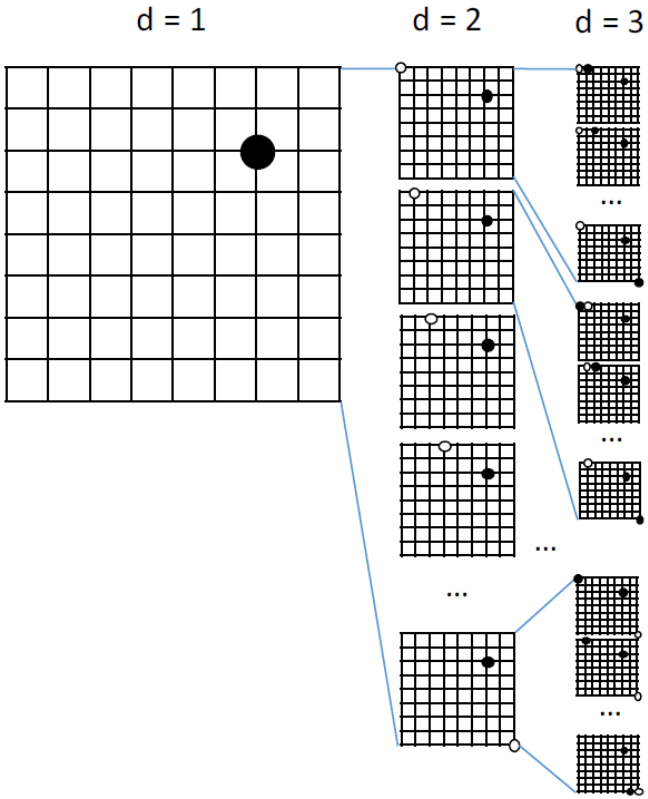



How about simulating all possible board positions?

# Computer Go AI – An Implementation Idea?

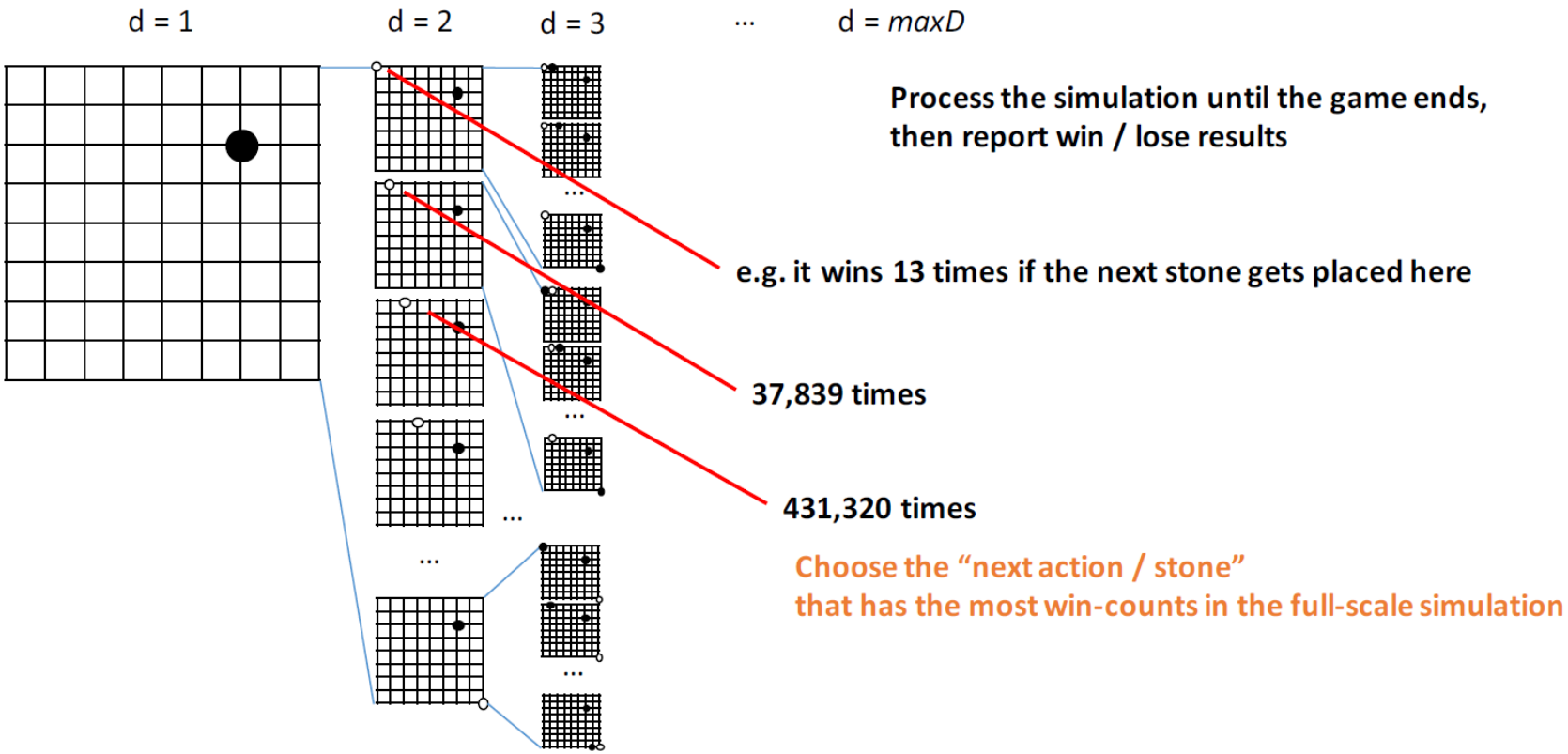


# Computer Go AI – An Implementation Idea?

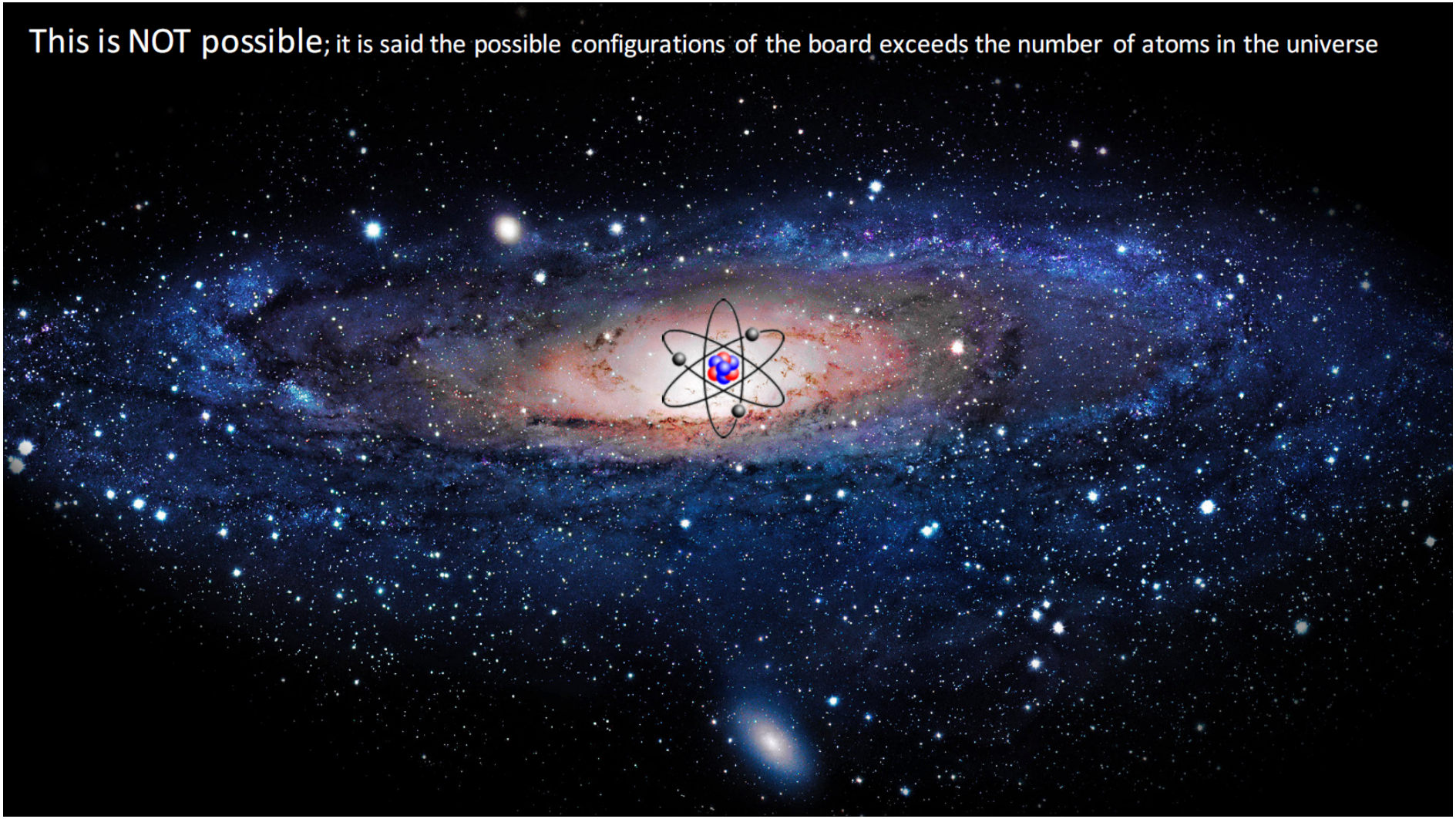



**Process the simulation until the game ends, then report win / lose results**

# Computer Go AI – An Implementation Idea?



# AlphaGo

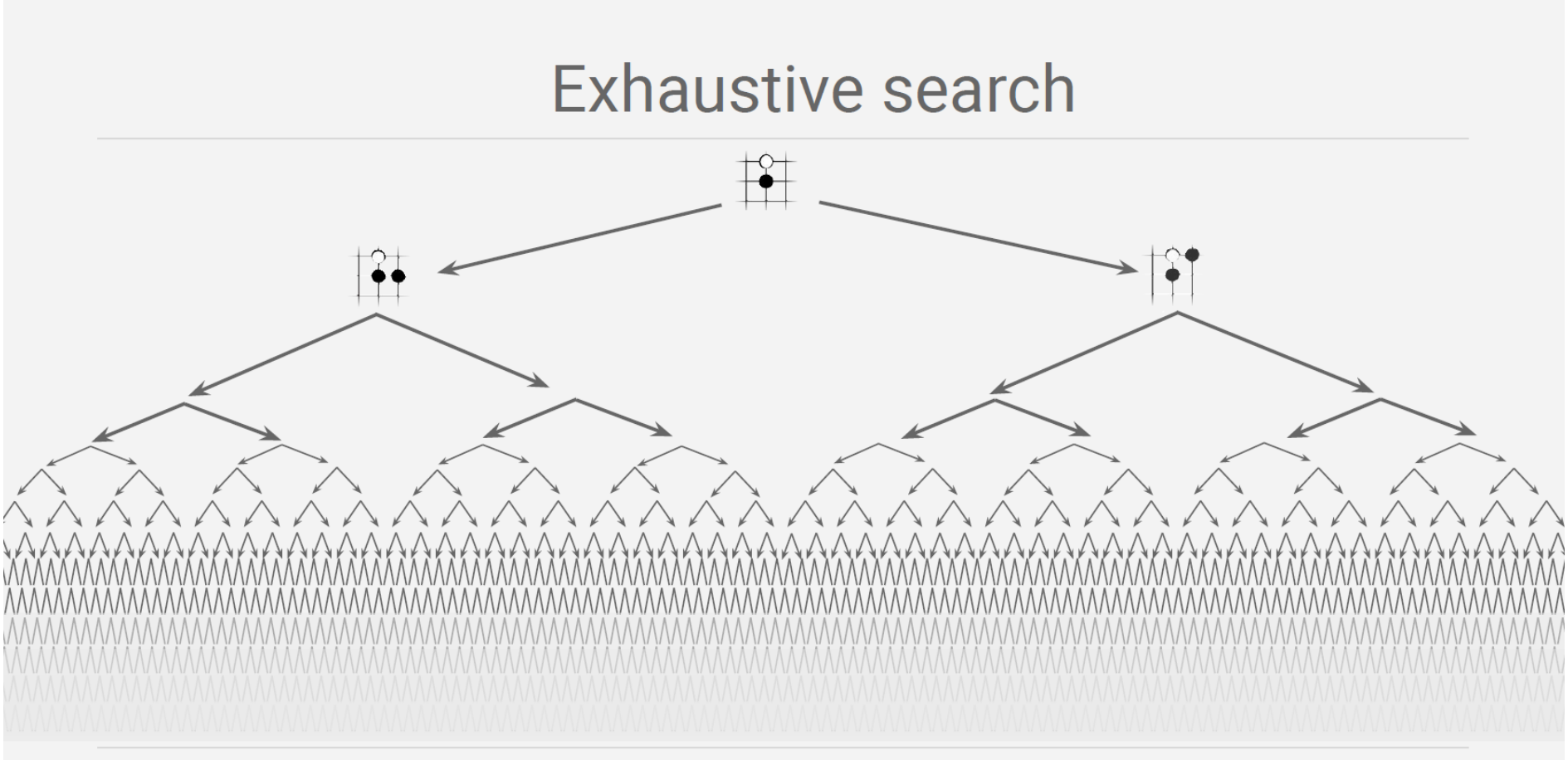

This is NOT possible; it is said the possible configurations of the board exceeds the number of atoms in the universe

# Search Space Reduction

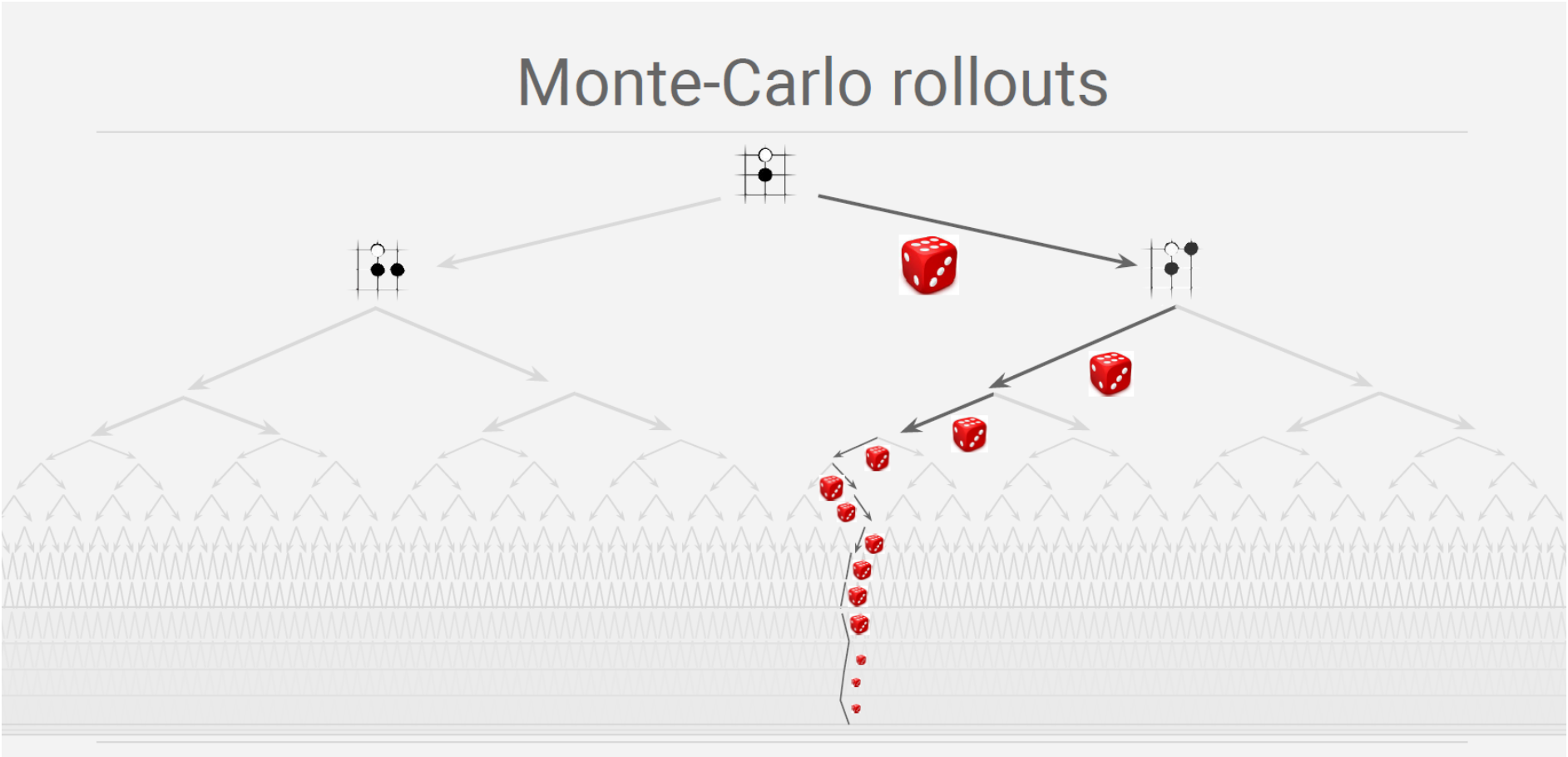
Key: to reduce search space
Three exciting breakthroughs in AI:

- Monte Carlo Tree Search (stochastic tree search)
- Supervised deep learning (deep networks as classifiers)
- Deep reinforcement learning (deep networks as function approximators)

# Monte Carlo Tree Search Source: David Silver (Google DeepMind)

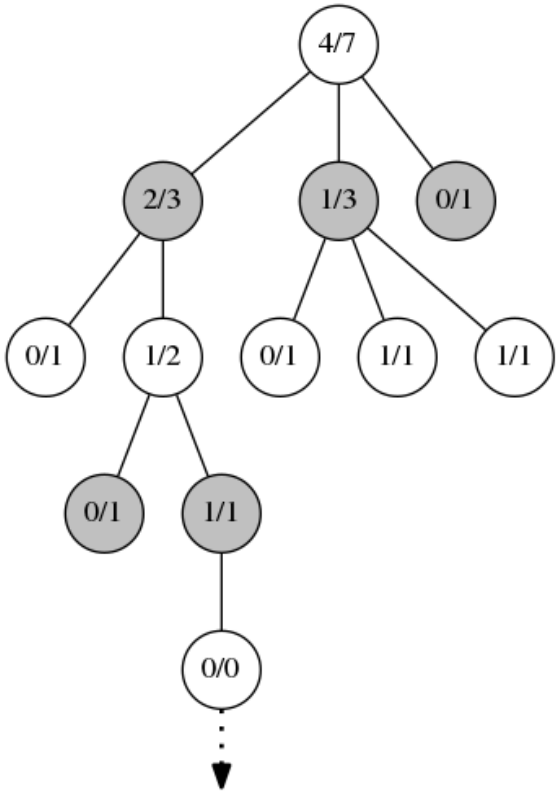


# Monte Carlo Tree Search



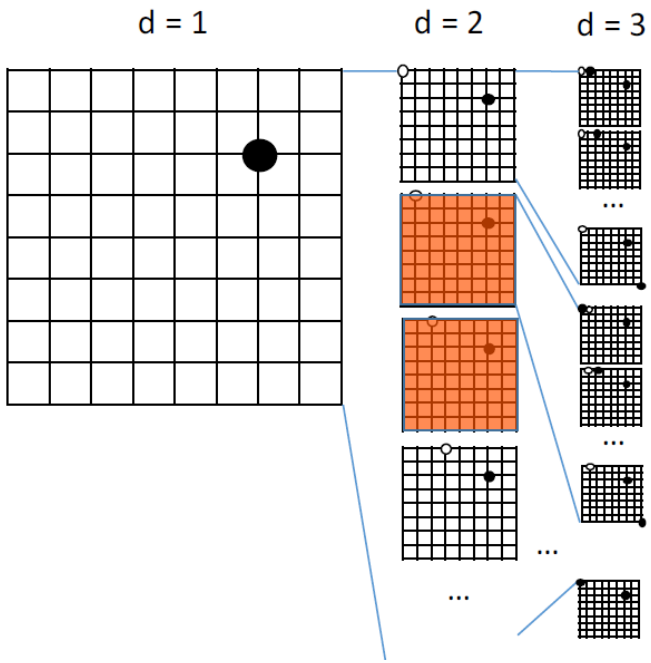
# Monte Carlo Tree Search

Bias the dice using two statistics: reward received and visit count



# Reducing Search Space

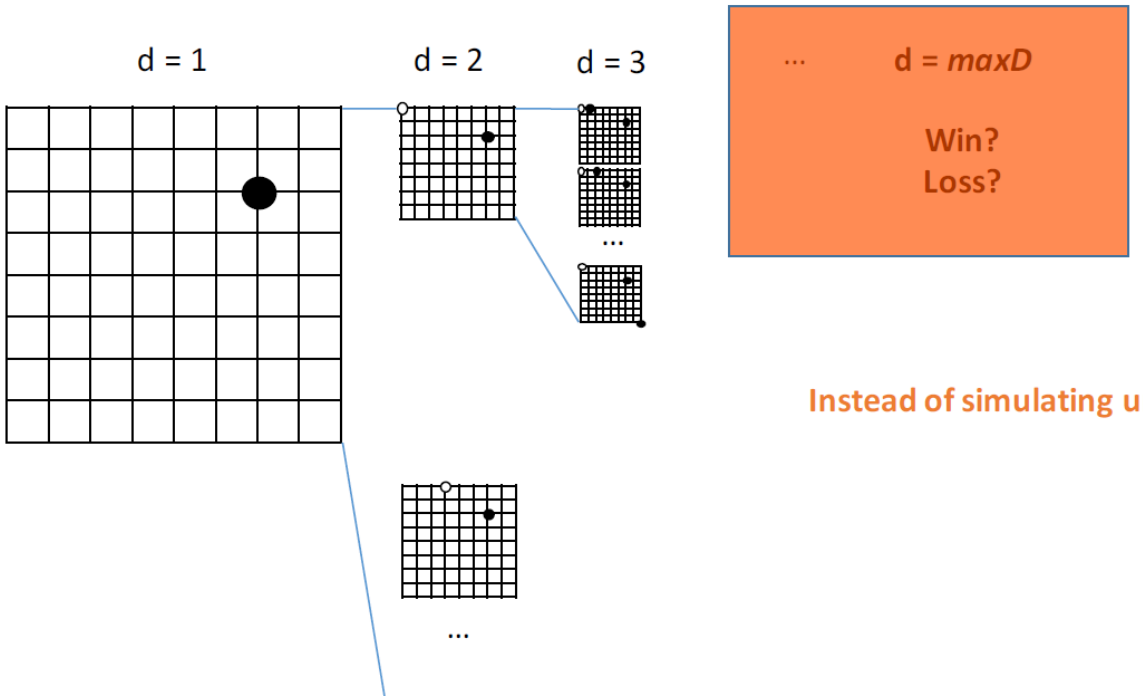
## 1. Reducing "action candidates" (Breadth Reduction)




IF there is a model that can tell you that these moves are not common / probable (e.g. by experts, etc.) ...

# Reducing Search Space

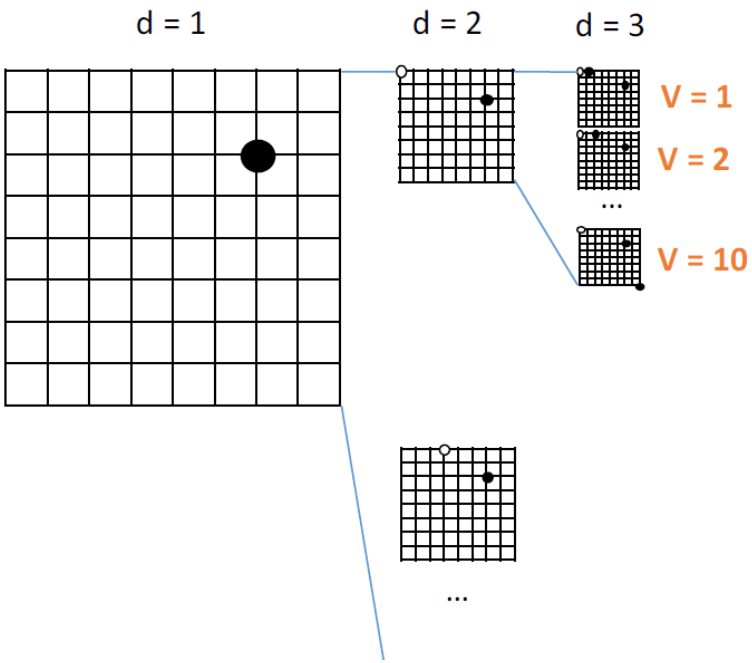
## 2. Position evaluation ahead of time (Depth Reduction)




Instead of simulating until the maximum depth ..

# Reducing Search Space

## 2. Position evaluation ahead of time (Depth Reduction)




**IF there is a function that can measure:**
**V(*s*)**: "board evaluation of state *s*"

## Reducing Search Space

1. **Reducing "action candidates" (Breadth Reduction)**

2. **Position evaluation ahead of time (Depth Reduction)**

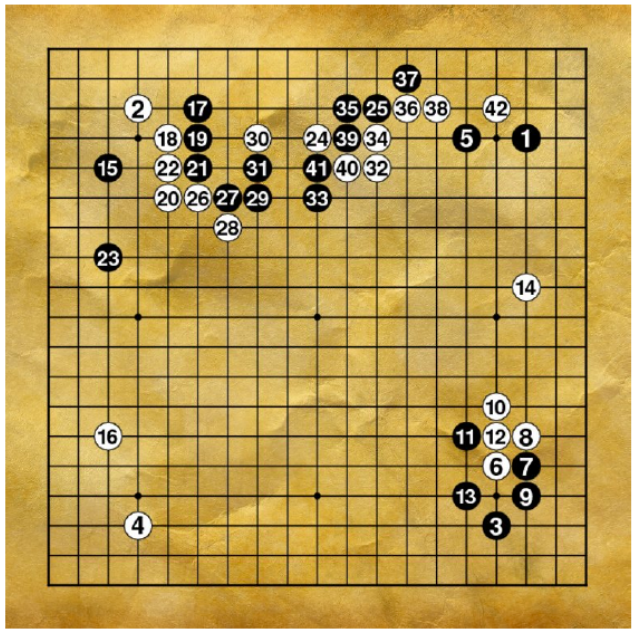
1. Reducing “action candidates”

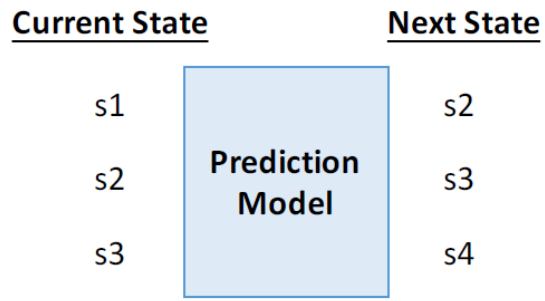
Learning: **P ( next action | current state )**

**= P ( a | s )**

# 1. Reducing "action candidates"

## (1) Imitating expert moves (supervised learning)

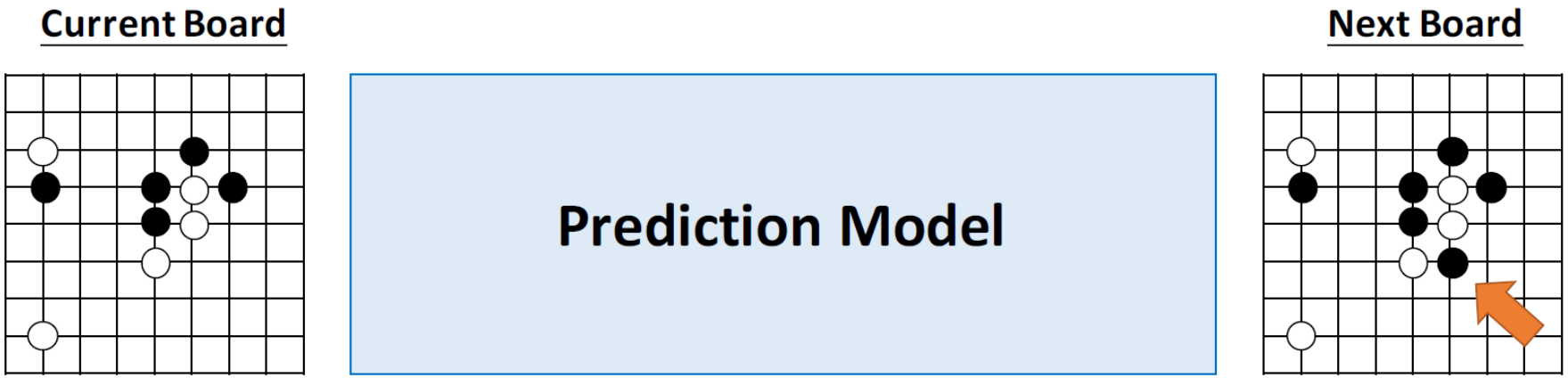





**Data**: Online Go experts (5~9 dan)
160K games, 30M board positions

## 1. Reducing "action candidates"

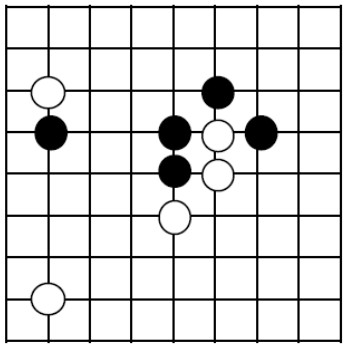
(1) Imitating expert moves (supervised learning)



## 1. Reducing "action candidates"

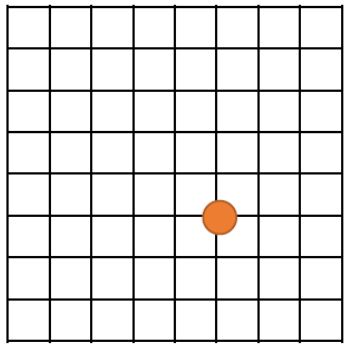
(1) Imitating expert moves (supervised learning)

**Current Board**



**Prediction Model**

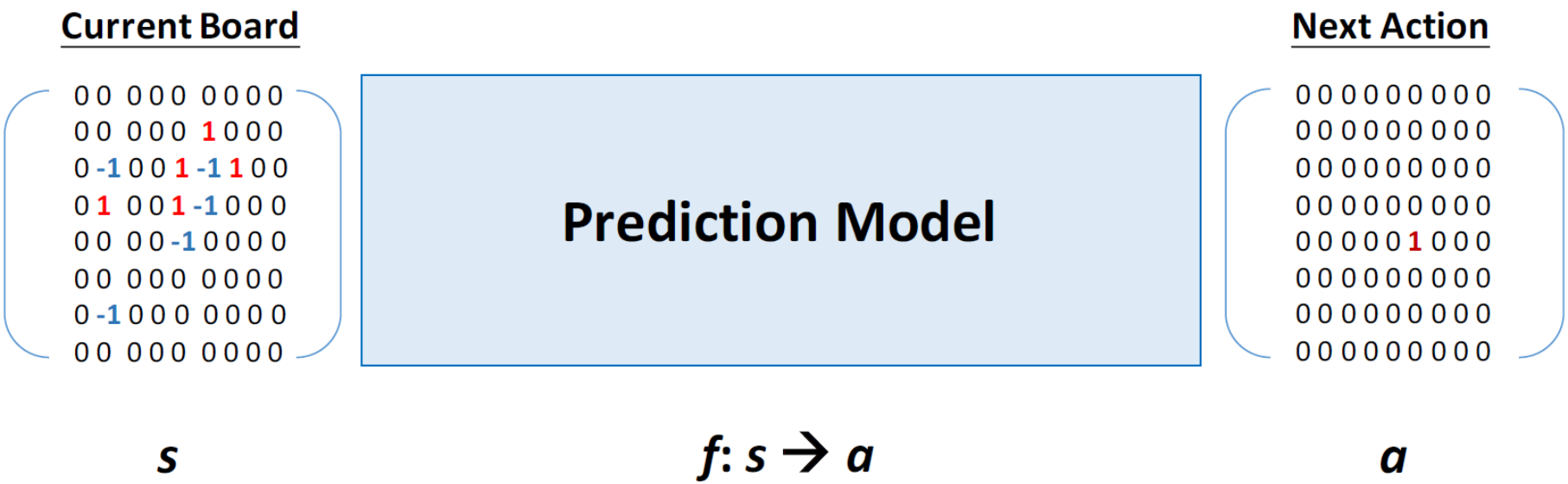
**Next Action**



There are 19 X 19 = 361
possible actions
(with different probabilities)

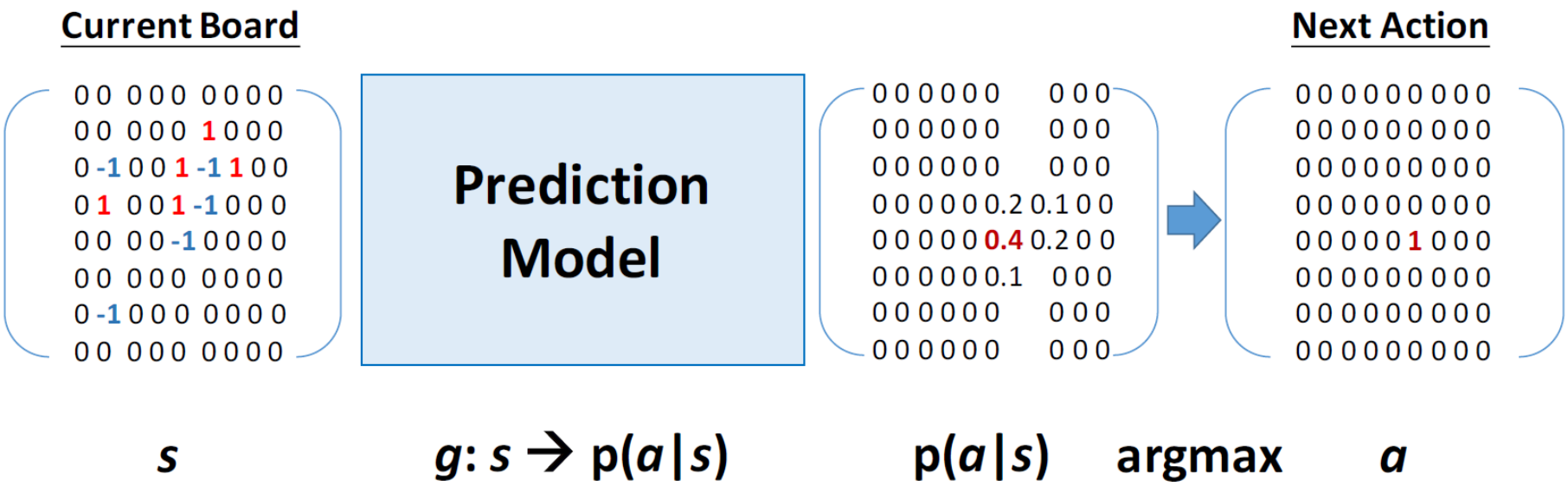
## 1. Reducing "action candidates"

(1) Imitating expert moves (supervised learning)



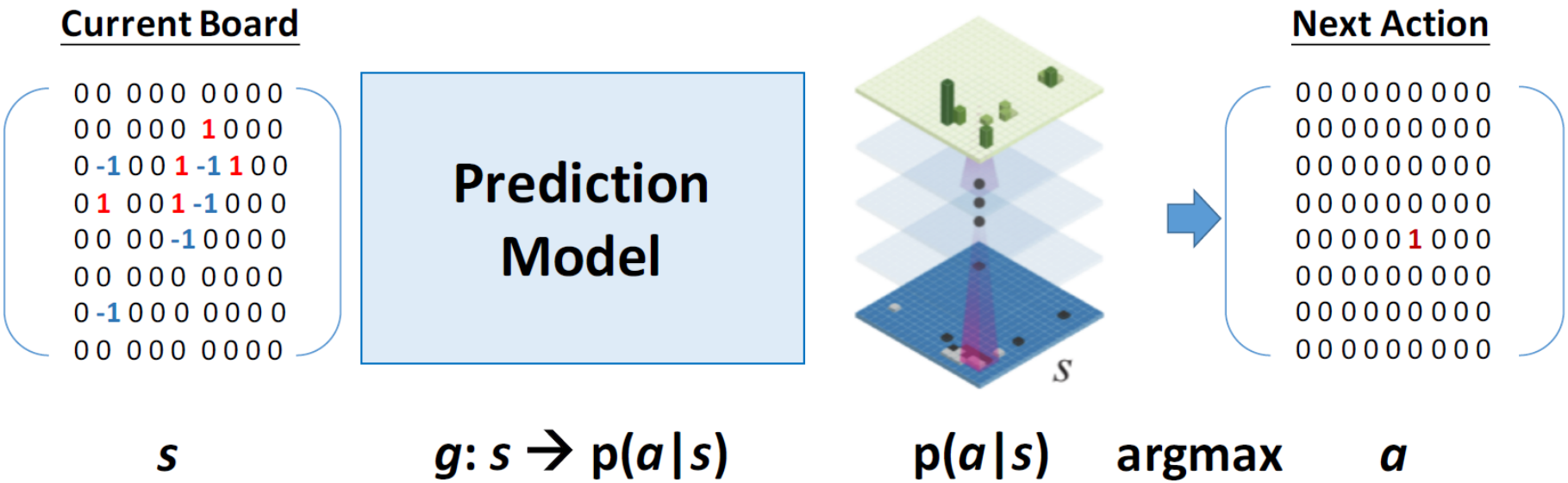
# 1. Reducing "action candidates"

(1) Imitating expert moves (supervised learning)



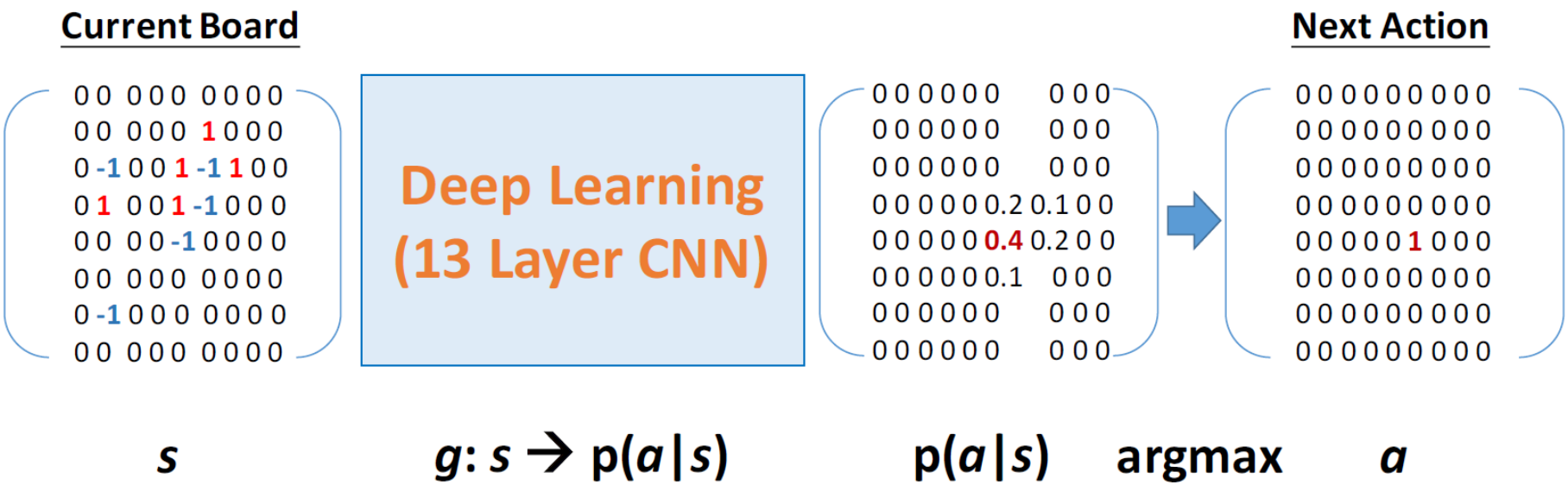
# 1. Reducing "action candidates"

(1) Imitating expert moves (supervised learning)

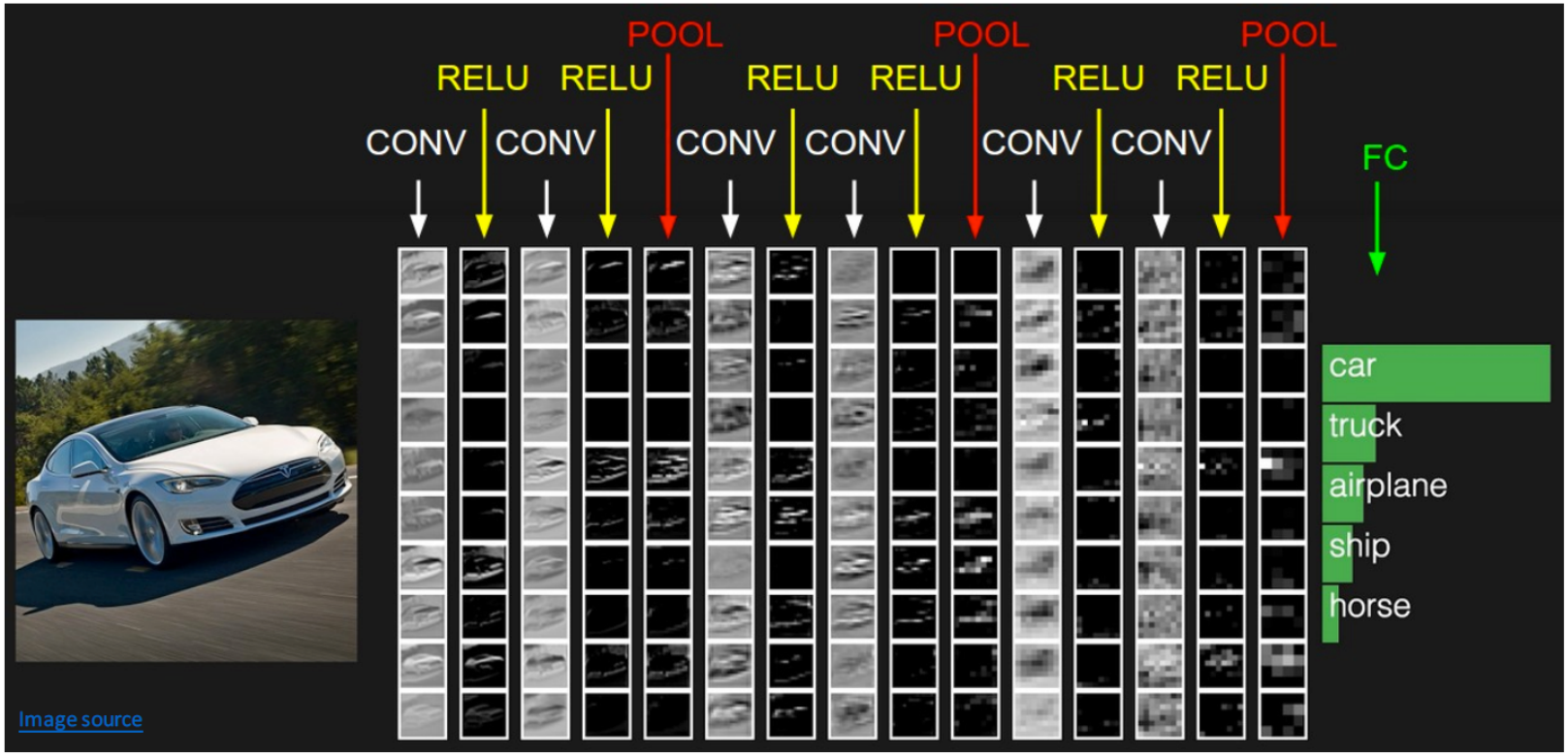


# 1. Reducing "action candidates"

(1) Imitating expert moves (supervised learning)

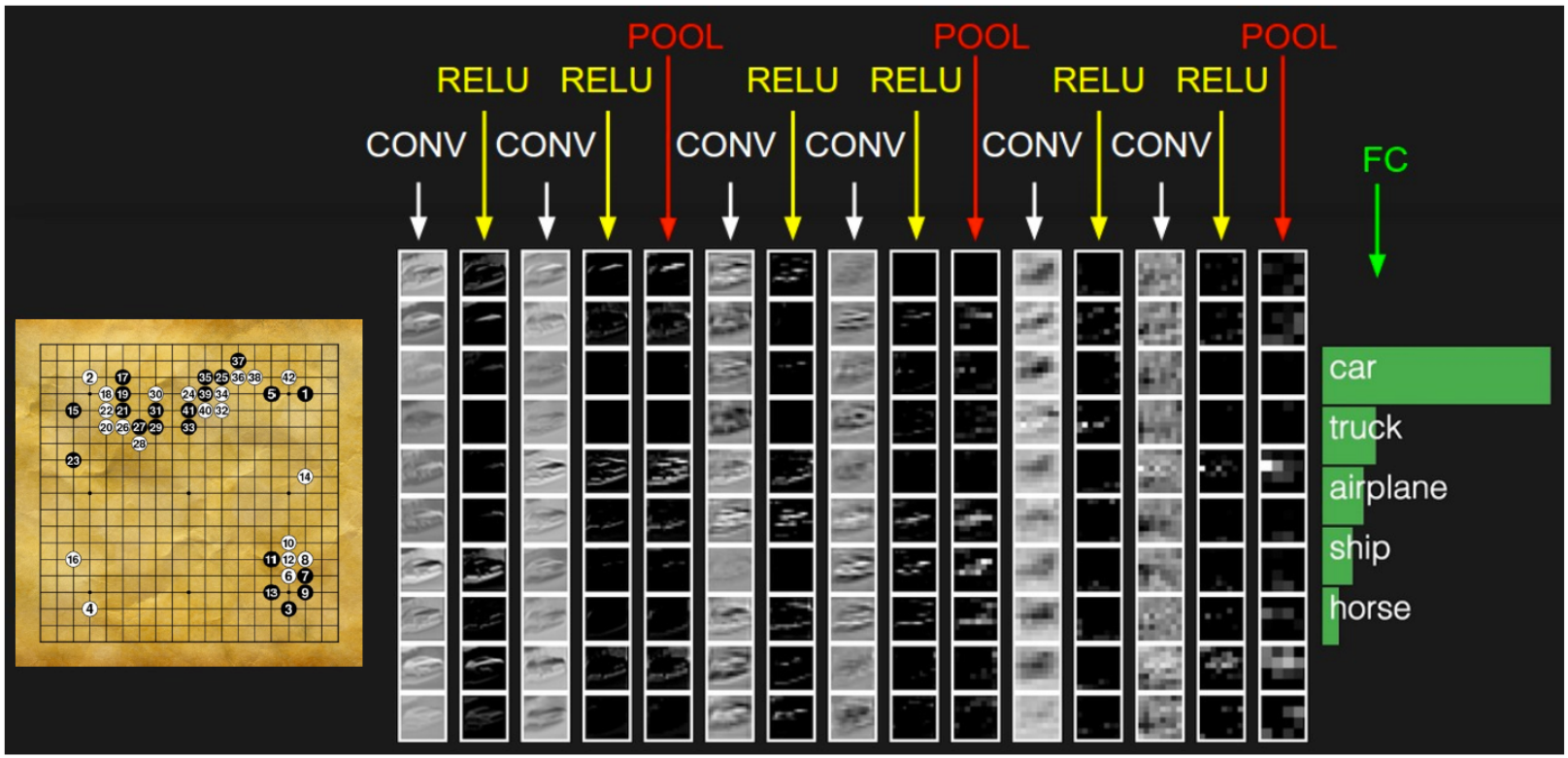


## Convolutional Neural Network (CNN)




CNN is a powerful model for image recognition tasks; it abstracts out the input image through convolution layers

## Convolutional Neural Network (CNN)




And they use this CNN model (similar architecture) to evaluate the board position; *which learns "some" spatial invariance*

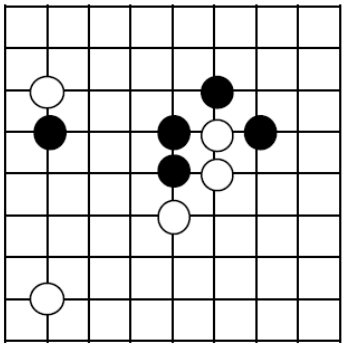
# Question

- Why CNN?

## 1. Reducing "action candidates"

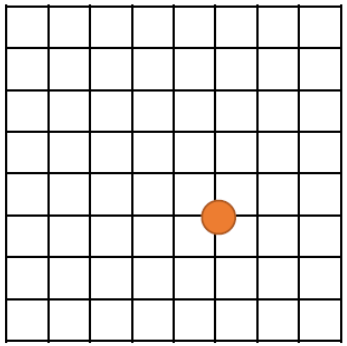
(1) Imitating expert moves (supervised learning)

**Current Board**



**Expert Moves Imitator Model (w/ CNN)**

**Next Action**



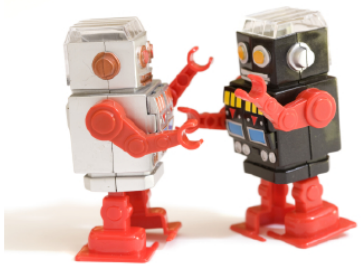
**Training:** $\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$

## 1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

Improving by playing against itself

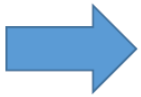
| Expert Moves Imitator Model (w/ CNN) | vs | Expert Moves Imitator Model (w/ CNN) |
|---|---|---|



## 1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

**Expert Moves Imitator Model (w/ CNN)** **vs** **Expert Moves Imitator Model (w/ CNN)**

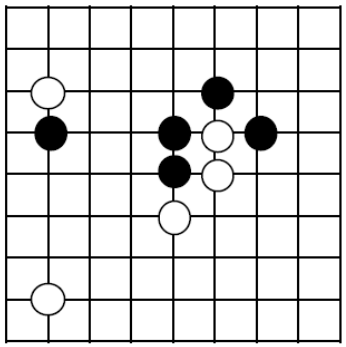


**Return**: board positions, win/lose info

## 1. Reducing “action candidates”

(2) Improving through self-plays (reinforcement learning)

**Board position**



**Expert Moves Imitator Model (w/ CNN)**

**win/loss**

**Loss**

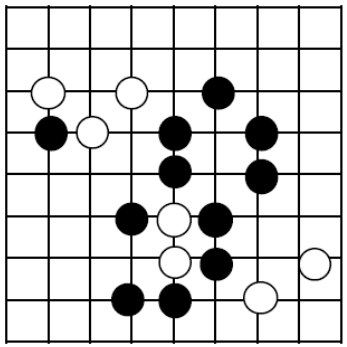
z = -1

**Training:** $\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$ .

## 1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

**Board position**



**Expert Moves Imitator Model (w/ CNN)**

**win/loss**

**Win**

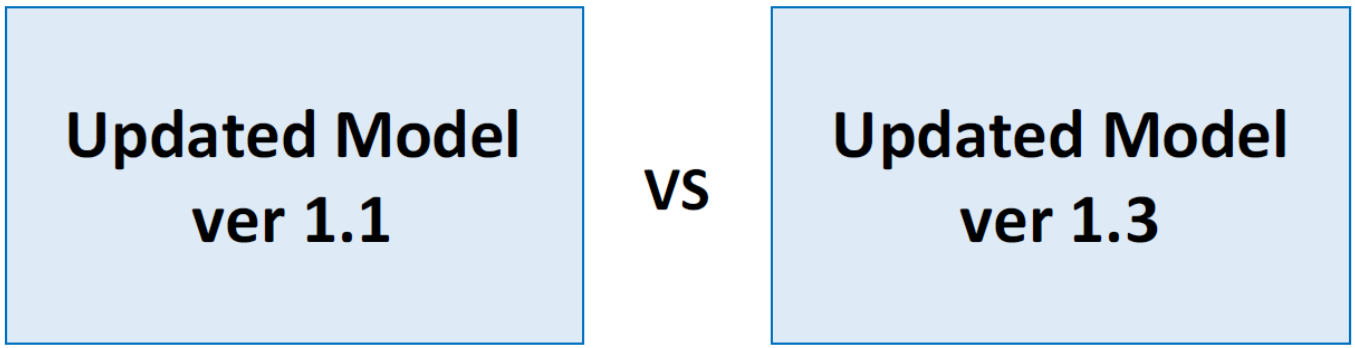
z = +1

**Training:** $\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$ .

# 1. Reducing "action candidates"

## (2) Improving through self-plays (reinforcement learning)

Older models vs. newer models

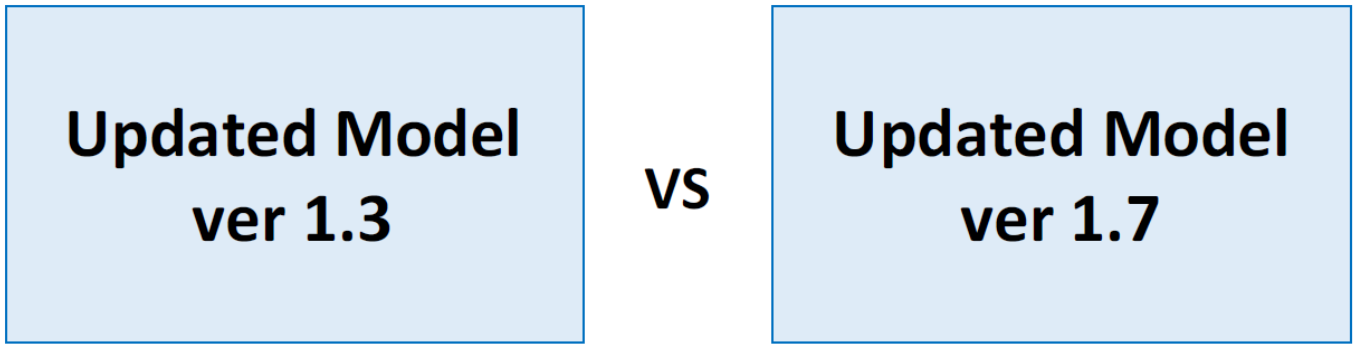



It uses the same topology as the expert moves imitator model, and just uses the updated parameters

**Return**: board positions, win/lose info

## 1. Reducing "action candidates"

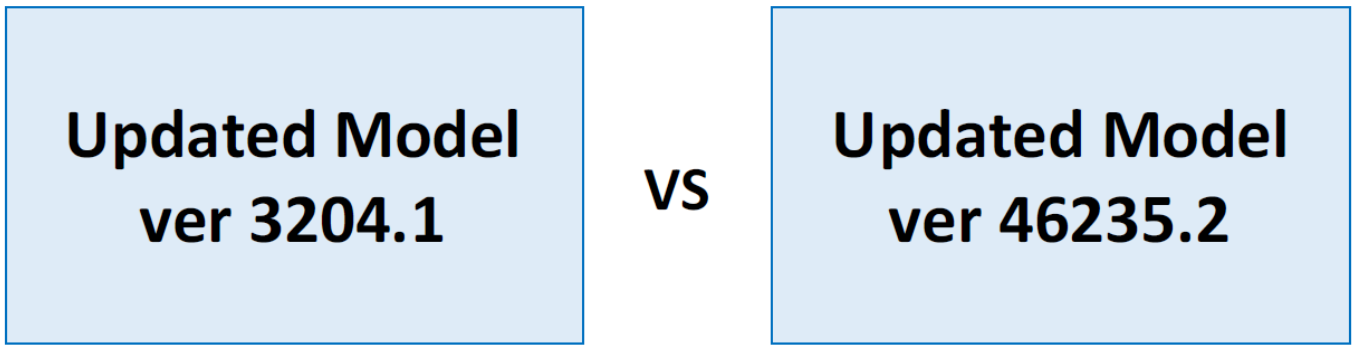
(2) Improving through self-plays (reinforcement learning)




**Return**: board positions, win/lose info

## 1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

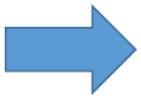



**Return**: board positions, win/lose info

## 1. Reducing "action candidates"

(2) Improving through self-plays (reinforcement learning)

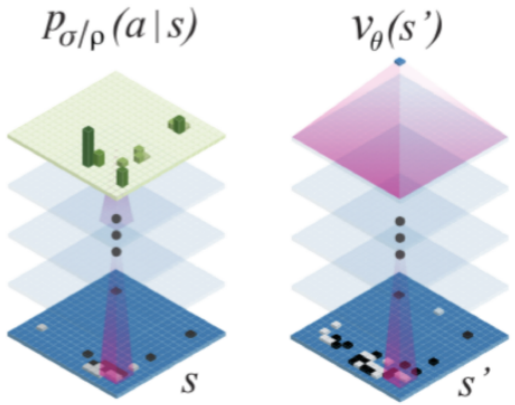
**Expert Moves Imitator Model** **VS** **Updated Model ver 1,000,000**



**The final model wins 80% of the time when playing against the first model**

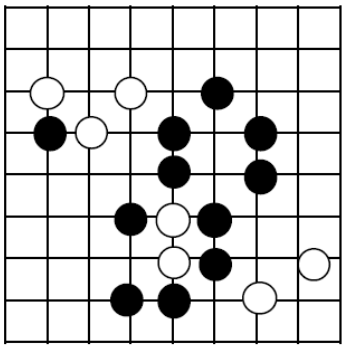
## 2. Board Evaluation

## 2. Board Evaluation




Adds a regression layer to the model
Predicts values between 0~1
Close to 1: a good board position
Close to 0: a bad board position

**Board Position**



**Updated Model ver 1,000,000**

**Value Prediction Model (Regression)**

**Win / Loss**

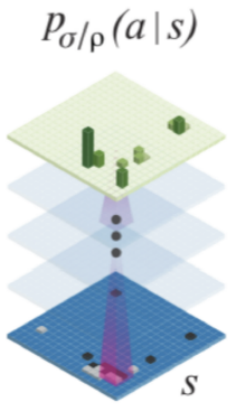
**Win (0~1)**

**Training:** $\Delta\theta \propto \dfrac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$

# Reducing Search Space

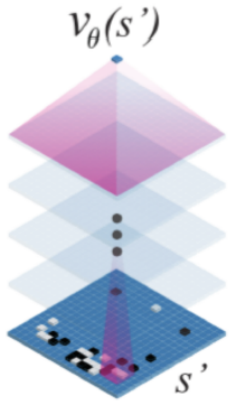
1. **Reducing "action candidates" (Breadth Reduction)**

   Policy Network




2. **Board Evaluation (Depth Reduction)**

   Value Network



# AlphaGo Zero

- Simpler, more elegant version (train with less computational resources; learn faster)
- Get rid of human demonstration and manually defined features
- Use residual network (much deeper)
- Combine value network and policy network into a single network with two heads, aka Siamese Network.

# The Future of Deep RL

What are the key differences between Go and Starcraft?


Figure: Source: Blizzard

# The Future of Deep RL

- Incomplete information: Partially Observable Markov Decision Process (POMDP)
  - Scouting: gathering information
  - Prediction: model learning and decision under uncertainty
- Multitask performance
  - Curriculum learning
  - Modularized model

# The Future of Deep RL

- What's wrong with learning from human demonstration?
  - Credit assignment: good/bad moves?
  - State distribution shift
  - Exactly what do we want machine to learn from human?
    - Behavior: behavior cloning
    - Reward function: inverse reinforcement learning (IRL)
    - Attention: attention guided RL

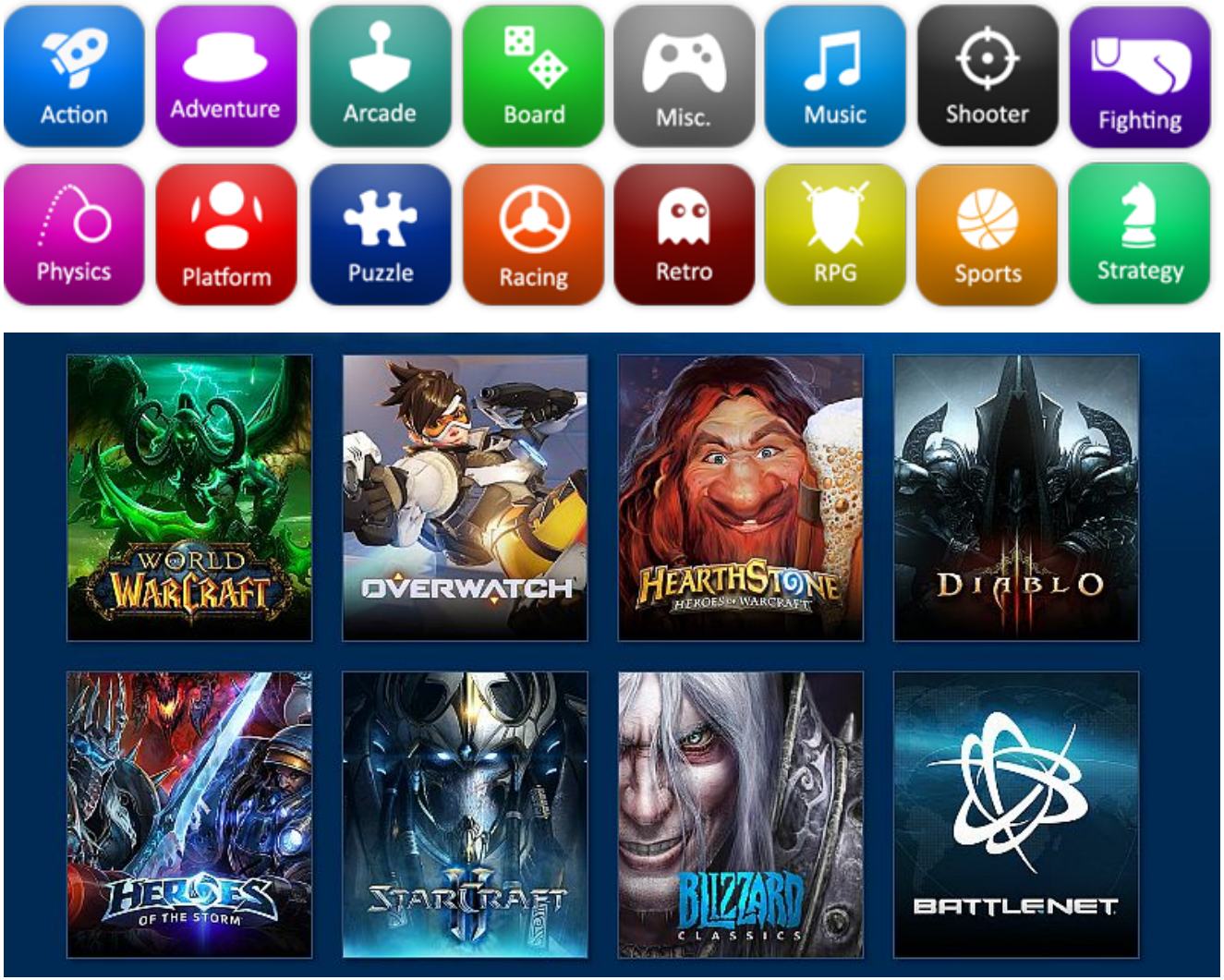
# The Future of Deep RL

- Data efficiency (when you don't have a simulator)
- Simulator to real word transfer
- Robotics: safety in learning
- Financial and health applications: formal guarantees in learned policy
- Multiagent learning: game theory

# Take Home Message

- Function approximation is an important extension of basic RL algorithms.
- Pseudo learning target for RL: $r + \gamma \max_{a'} Q(s', a')$.
- Direct combination of RL + neural net is not stable. But nowadays, stable RL techniques make this combination possible.
- Deep learning + RL can result in an end-to-end visuomotor system.
- RL can be combined with planing method, like MCTS in AlphaGo.
- Many, many exciting unsolved problems.

# Which Game is Most Challenging for the Current Deep RL?



# More References

See Junhyuk Oh's github repository for a comprehensive list of deep reinforcement learning papers here (University of Michigan)
See Dr. David Silver's RL lectures on Youtube.
Professor Peter Stone's RL class at UT-Austin.
Google DeepMind's official website.

# Questions?

Thanks!


Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
*arXiv preprint arXiv:1312.5602*, 2013.


Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al.
Human-level control through deep reinforcement learning.
*Nature*, 518(7540):529–533, 2015.


David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al.
Mastering the game of go with deep neural networks and tree search.
*Nature*, 529(7587):484–489, 2016.


Richard S Sutton and Andrew G Barto.
*Reinforcement learning: An introduction*.
MIT press, 1998.