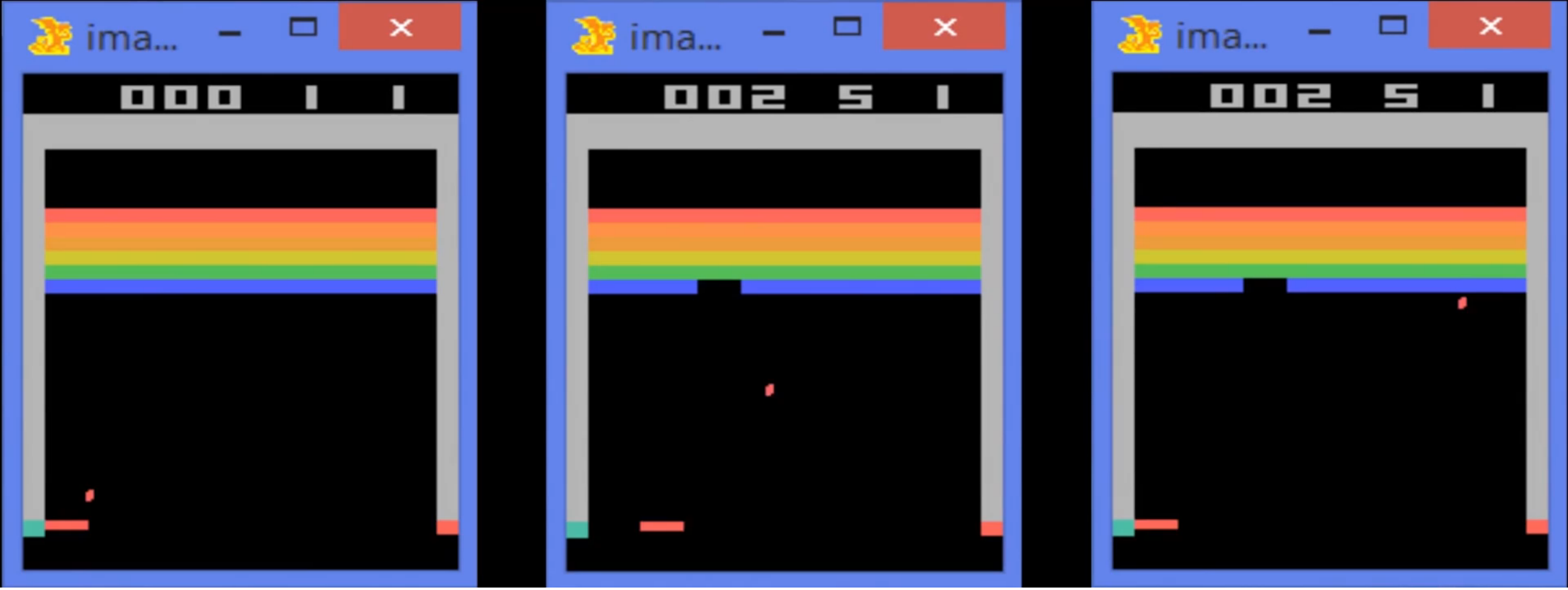
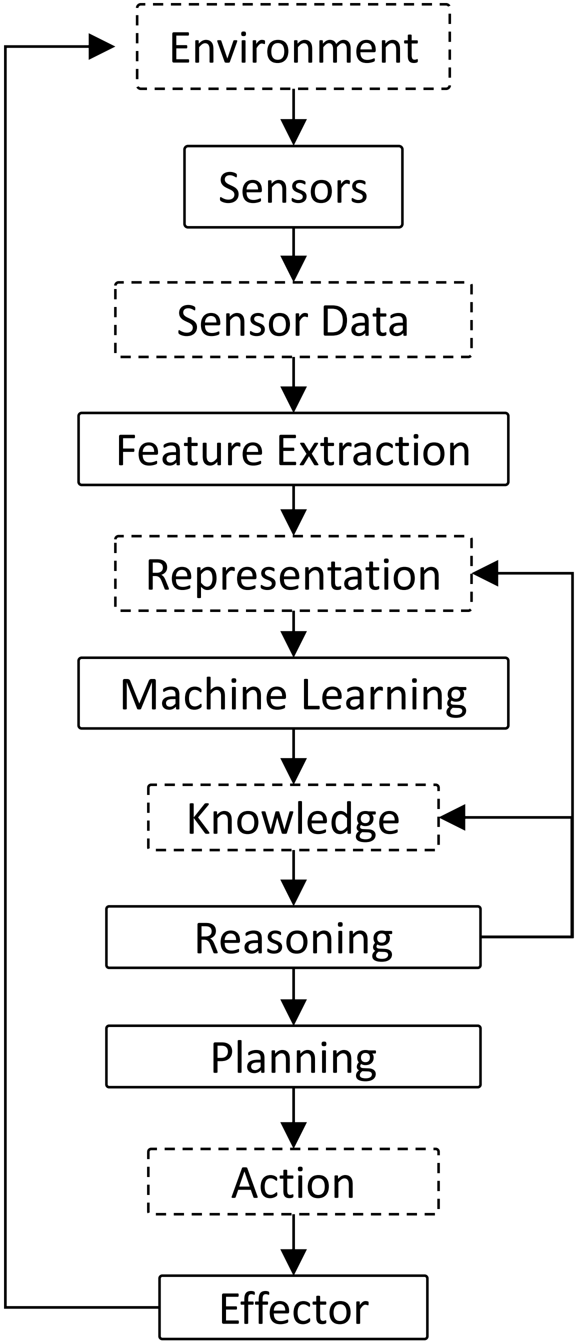
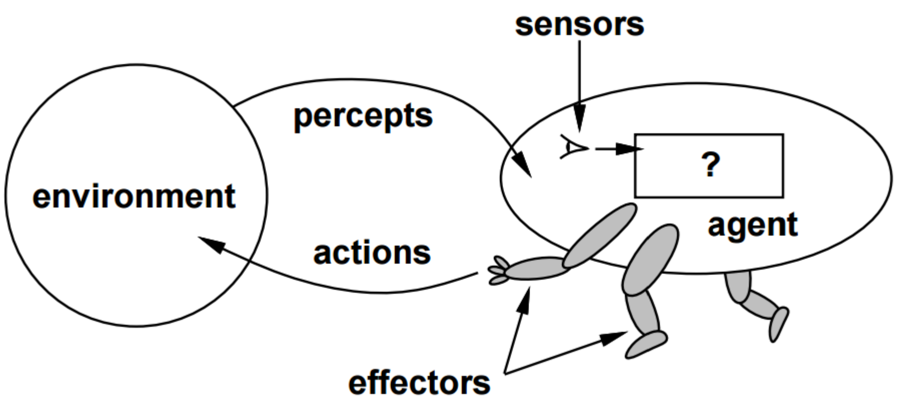


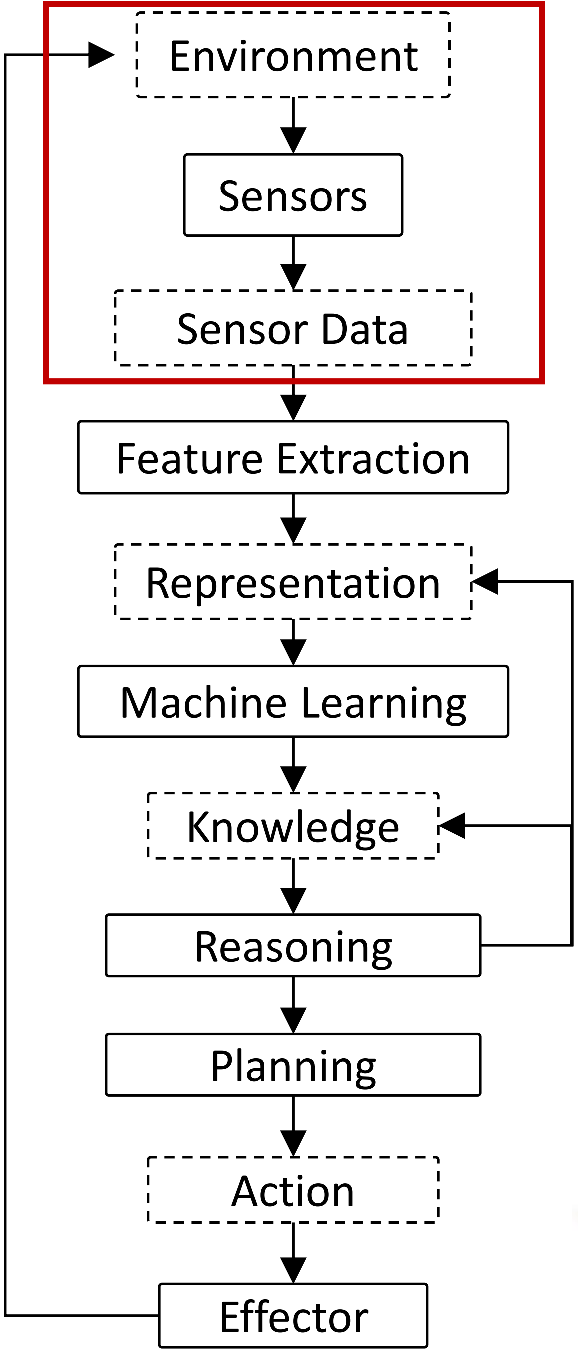
Deep Reinforcement Learning

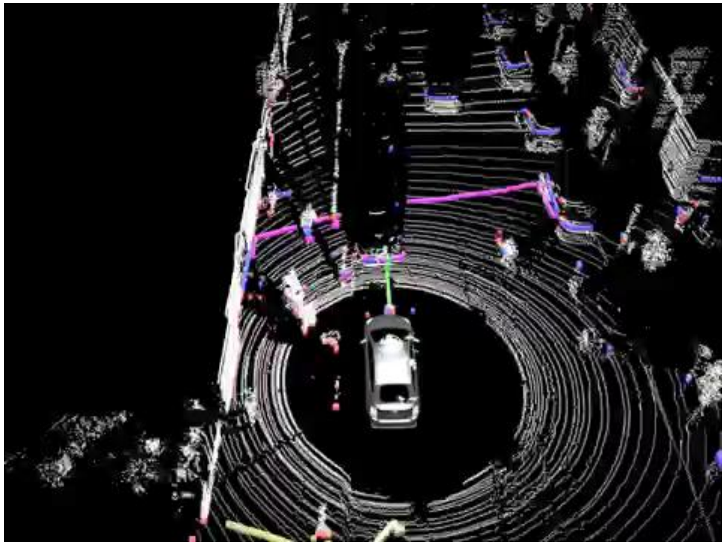
Lex Fridman

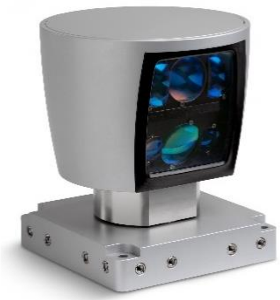


Open Question:
What can be learned from data?









Lidar

Camera
(Visible, Infrared)

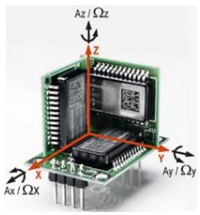
Radar

GPS

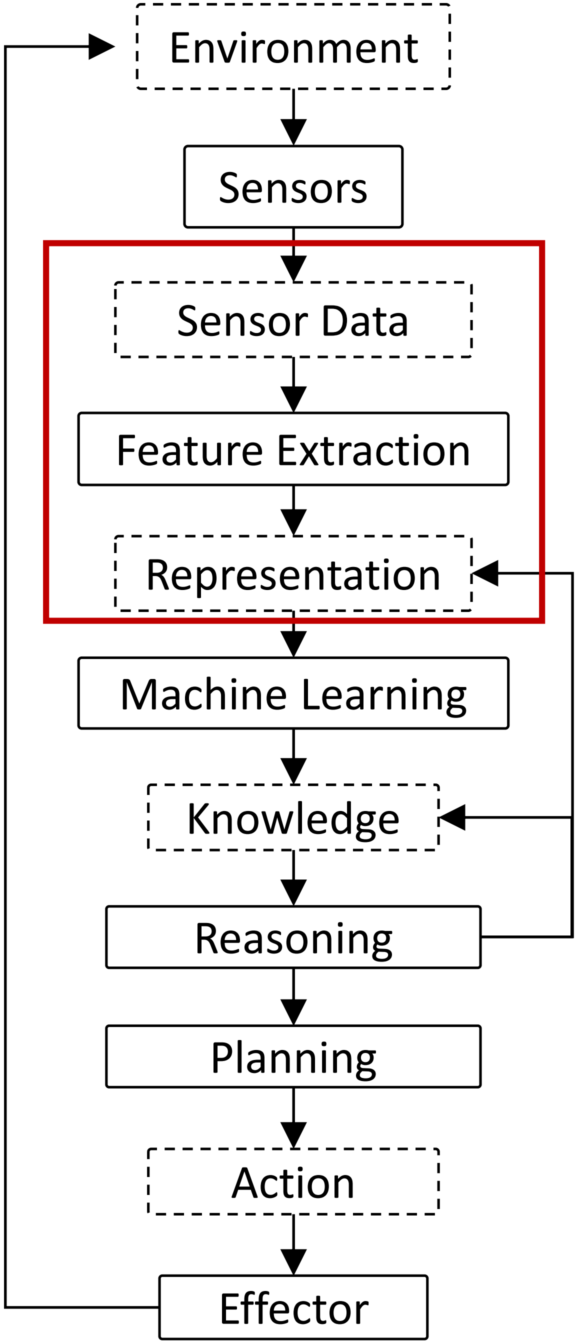
Stereo Camera

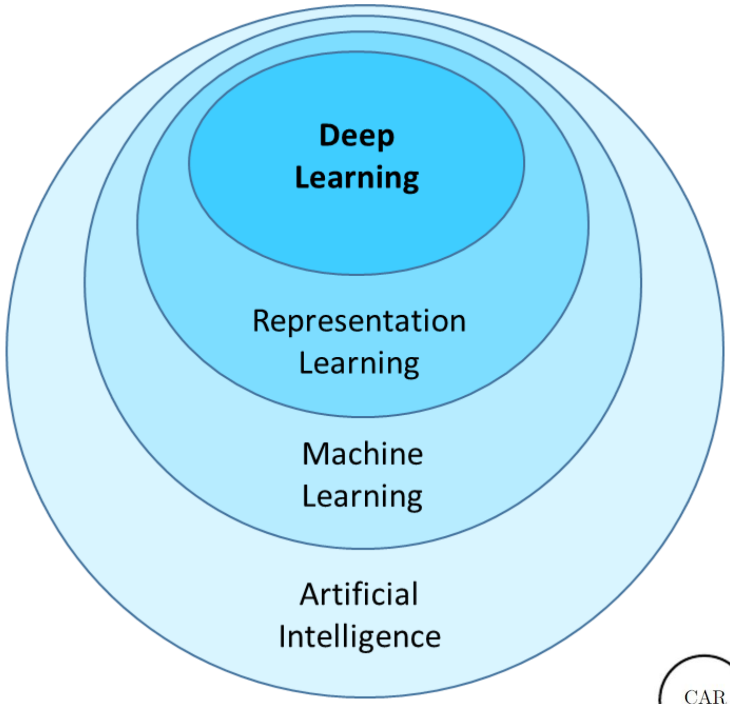
Microphone

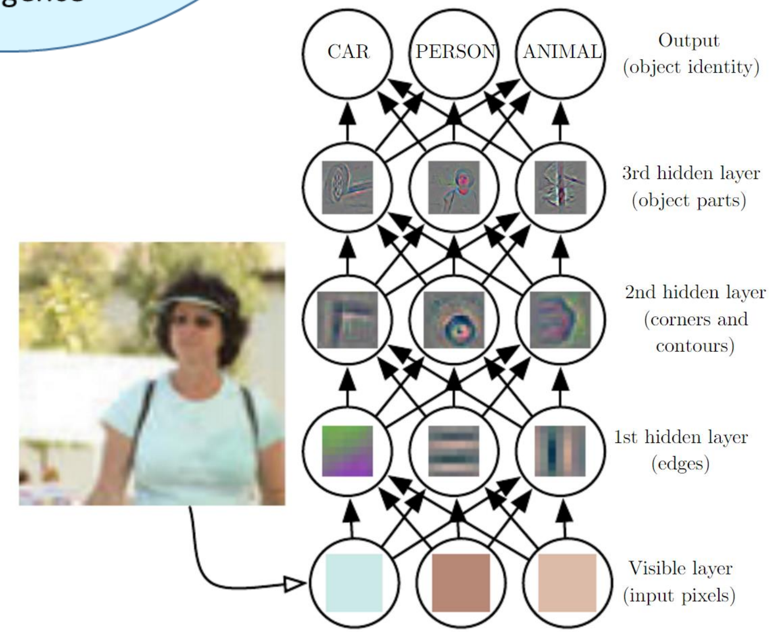
Networking
(Wired, Wireless)

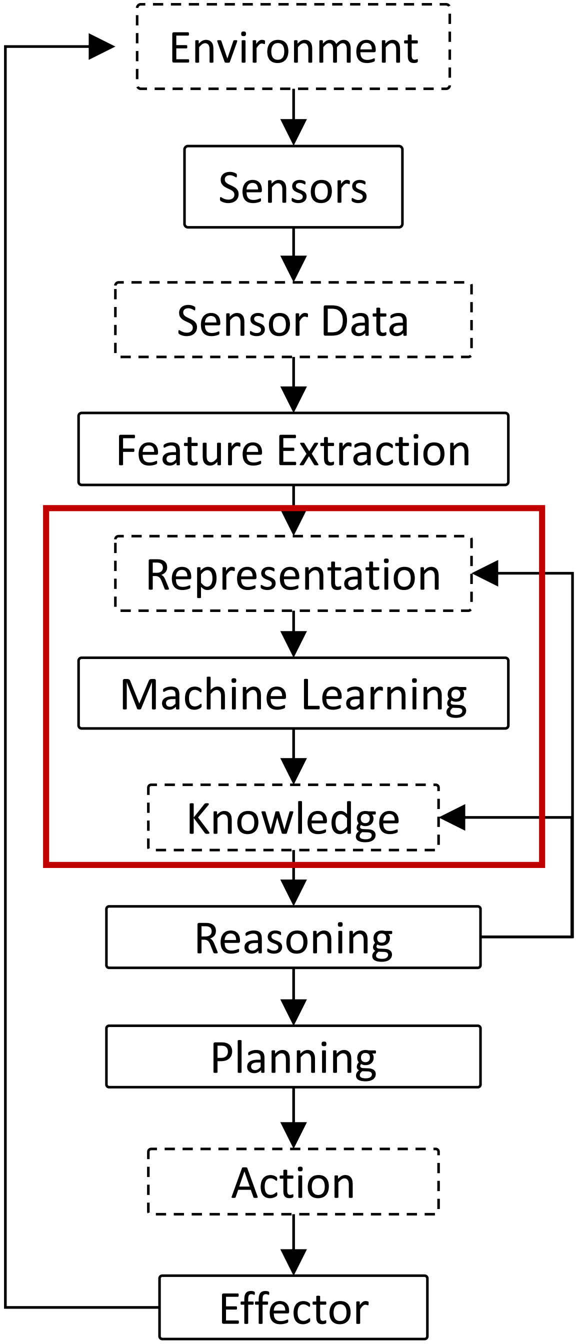


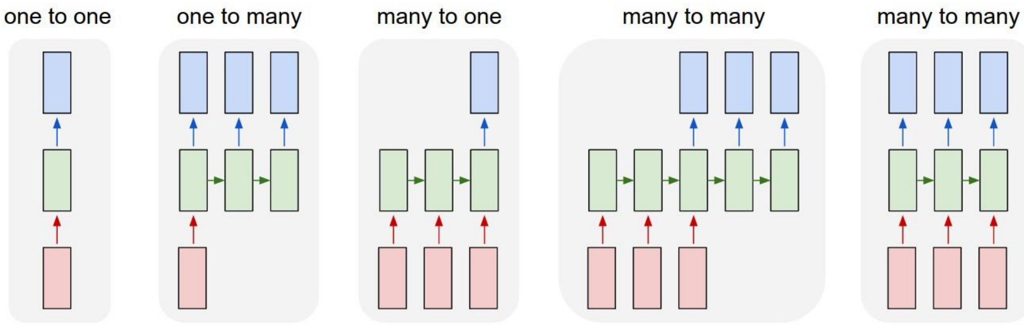
IMU


Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector


Deep Learning
Representation Learning
Machine Learning
Artificial Intelligence


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)


Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector


one to one
one to many
many to one
many to many
many to many

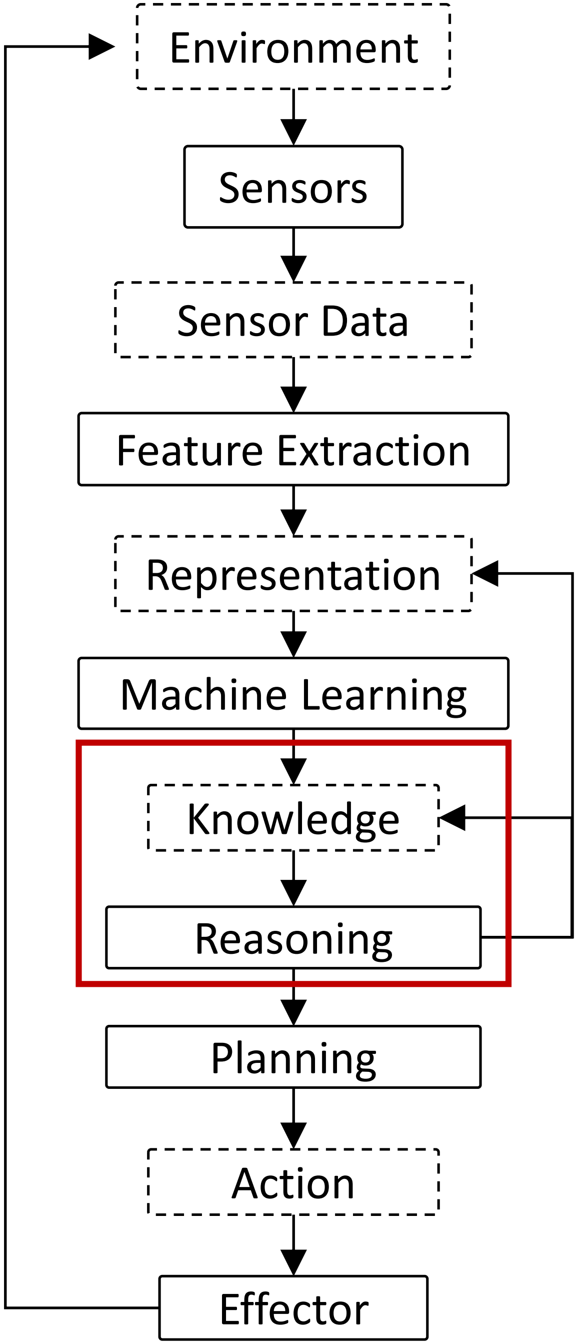
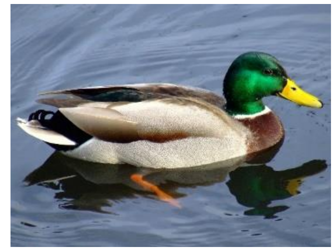
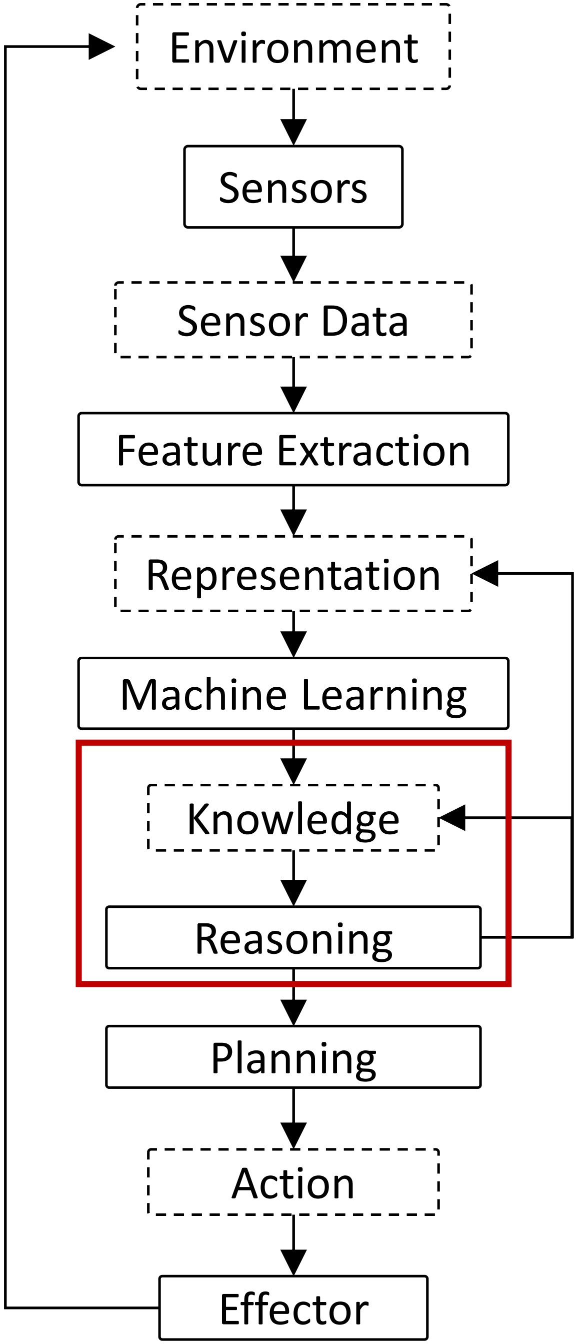


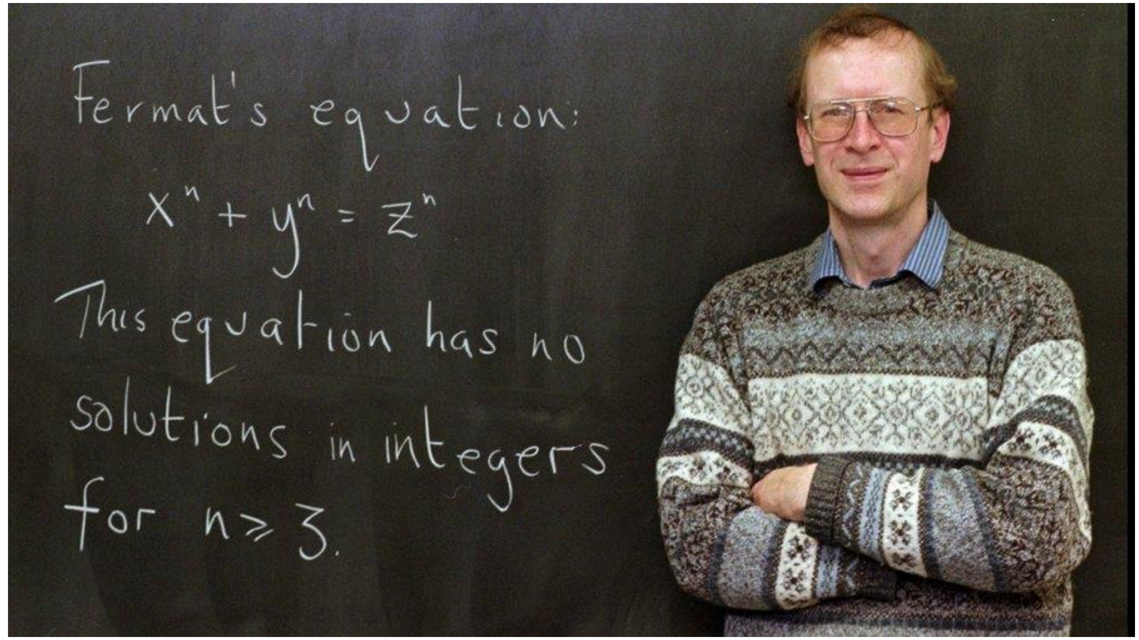
Image Recognition:
If it looks like a duck



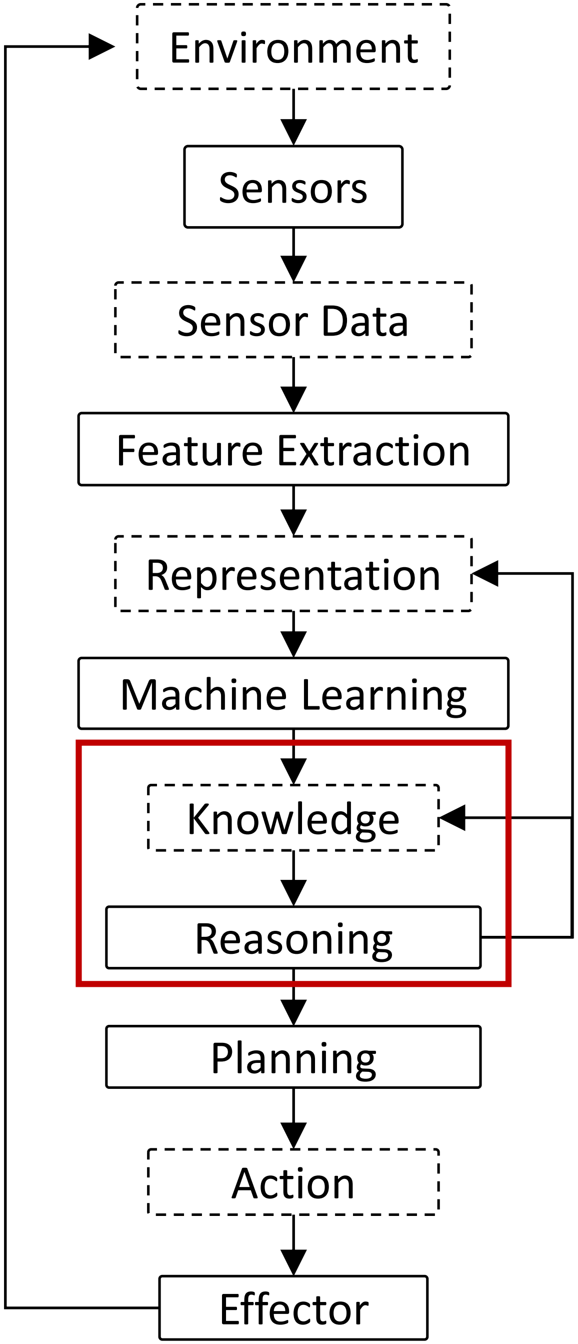
Audio Recognition:
Quacks like a duck

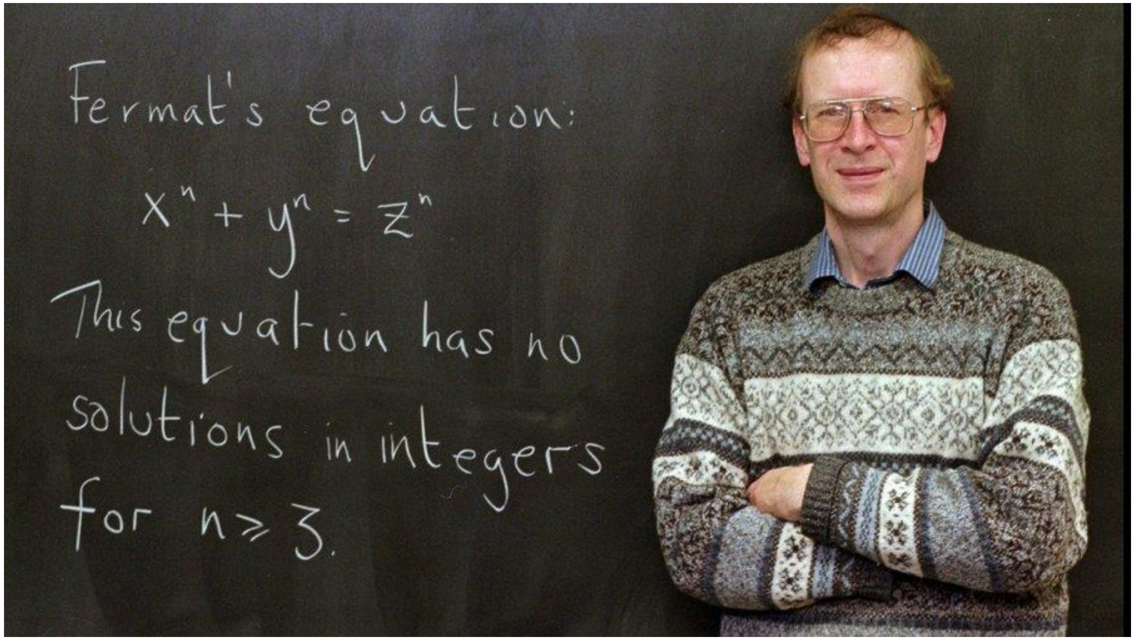
Activity Recognition:
Swims like a duck

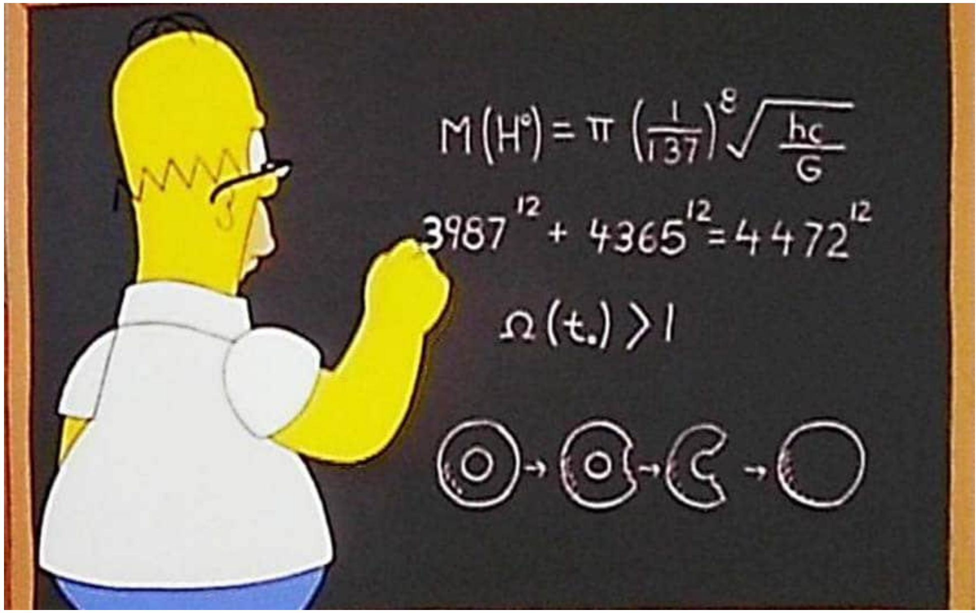


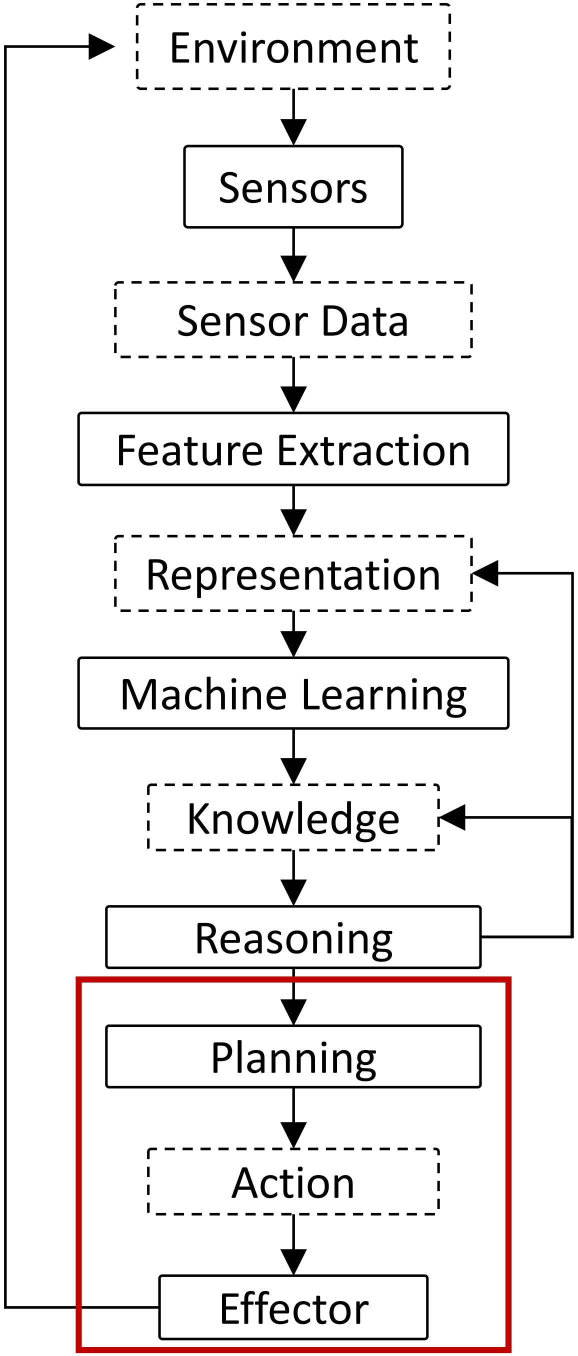


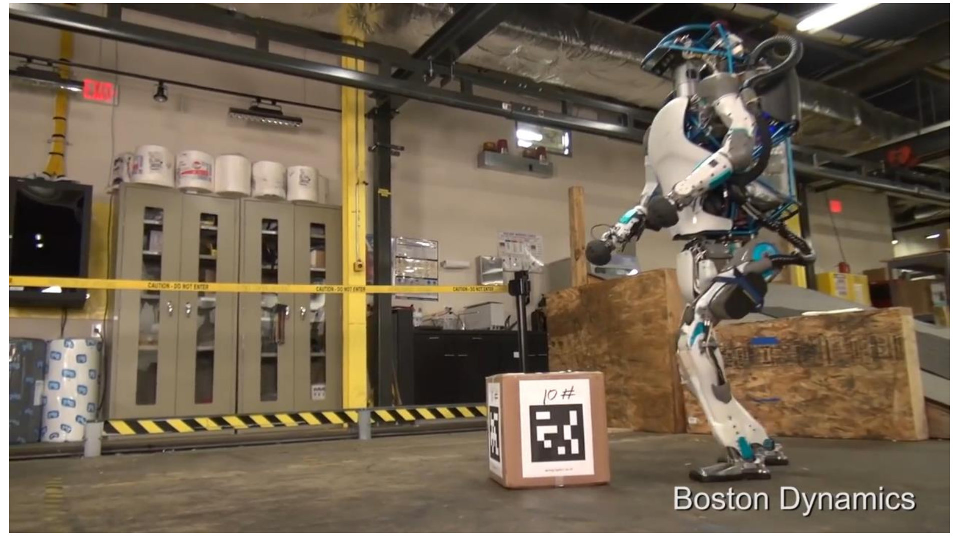
Final **breakthrough**, 358 years after its conjecture:
“It was so indescribably beautiful; it was so simple and so elegant. I couldn’t understand how I’d missed it and I just stared at it in disbelief for twenty minutes. Then during the day I walked around the department, and I’d keep coming back to my desk looking to see if it was still there. It was still there. I couldn’t contain myself, I was so excited. It was the most important moment of my working life. Nothing I ever do again will mean as much."

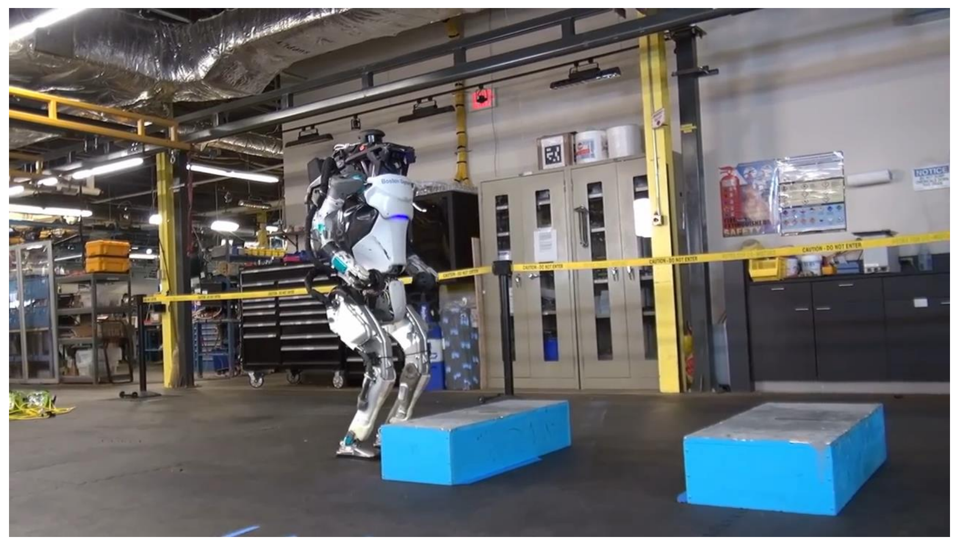

Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector


Fermat's equation:
$x^n + y^n = z^n$
This equation has no solutions in integers for $n \geq 3$.

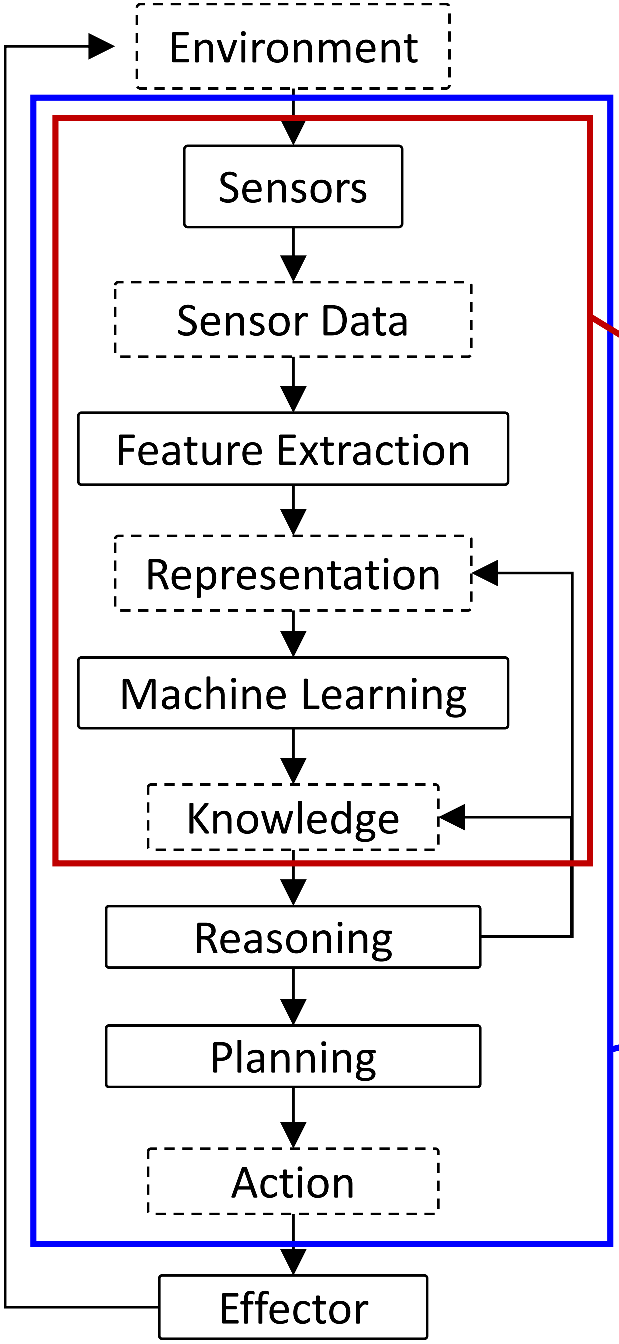

$M(H^\circ) = \pi \left(\frac{1}{137}\right)^8 \sqrt{\frac{hc}{G}}$
$3987^{12} + 4365^{12} = 4472^{12}$
$\Omega(t_0) > 1$







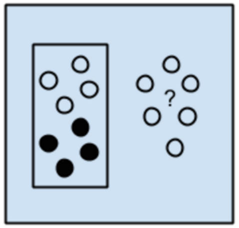
References: [133]



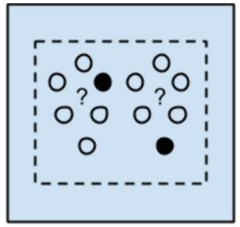
The promise of
Deep Learning

The promise of
Deep Reinforcement Learning

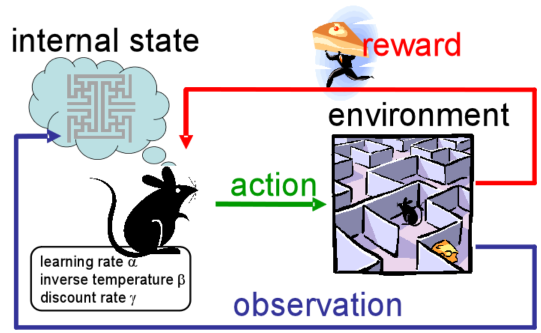
Types of Deep Learning



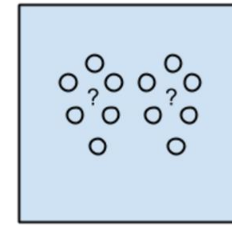
Supervised Learning



Semi-Supervised Learning



Reinforcement Learning



Unsupervised Learning

Philosophical Motivation for Reinforcement Learning

Takeaway from Supervised Learning:

Neural networks are great at memorization and not (yet) great at reasoning.

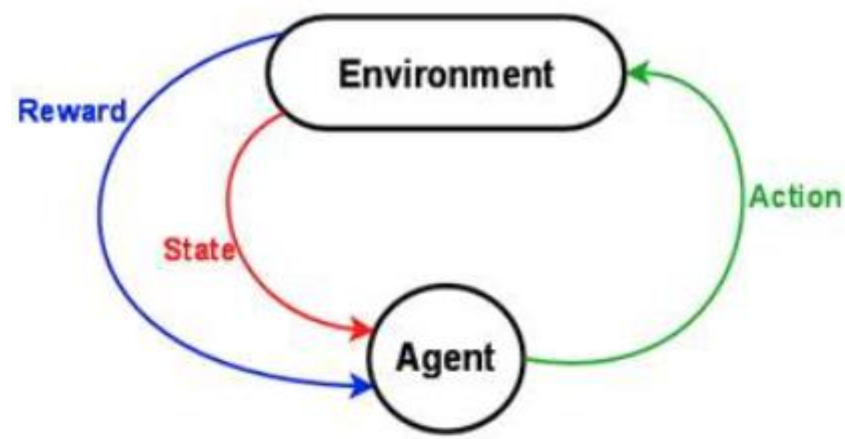
Hope for Reinforcement Learning:

Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force “reasoning”.

Agent and Environment

- At each step the agent:
  - Executes action
  - Receives observation (new state)
  - Receives reward

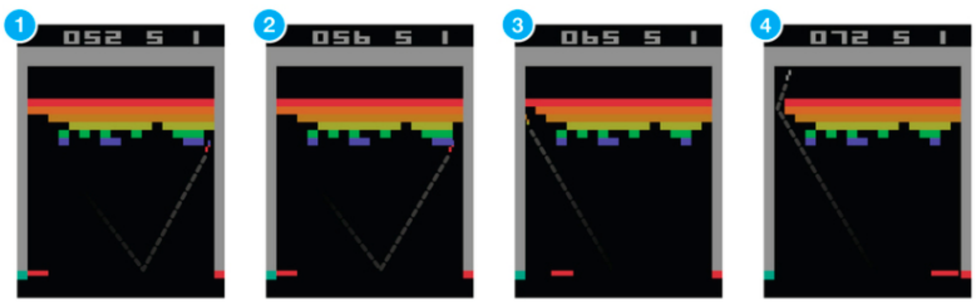
- The environment:
  - Receives action
  - Emits observation (new state)
  - Emits reward



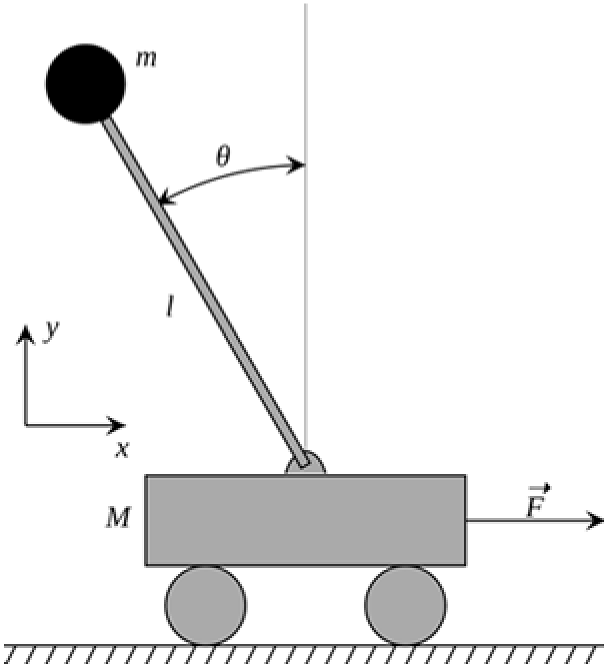
Examples of Reinforcement Learning

Reinforcement learning is a general-purpose framework for decision-making:

- An agent operates in an environment: **Atari Breakout**
- An agent has the capacity to **act**
- Each action influences the agent's **future state**
- Success is measured by a **reward** signal
- **Goal** is to select actions to maximize future reward



Examples of Reinforcement Learning



Cart-Pole Balancing

- **Goal** — Balance the pole on top of a moving cart
- **State** — Pole angle, angular speed. Cart position, horizontal velocity.
- **Actions** — horizontal force to the cart
- **Reward** — 1 at each time step if the pole is upright

Examples of Reinforcement Learning

Doom

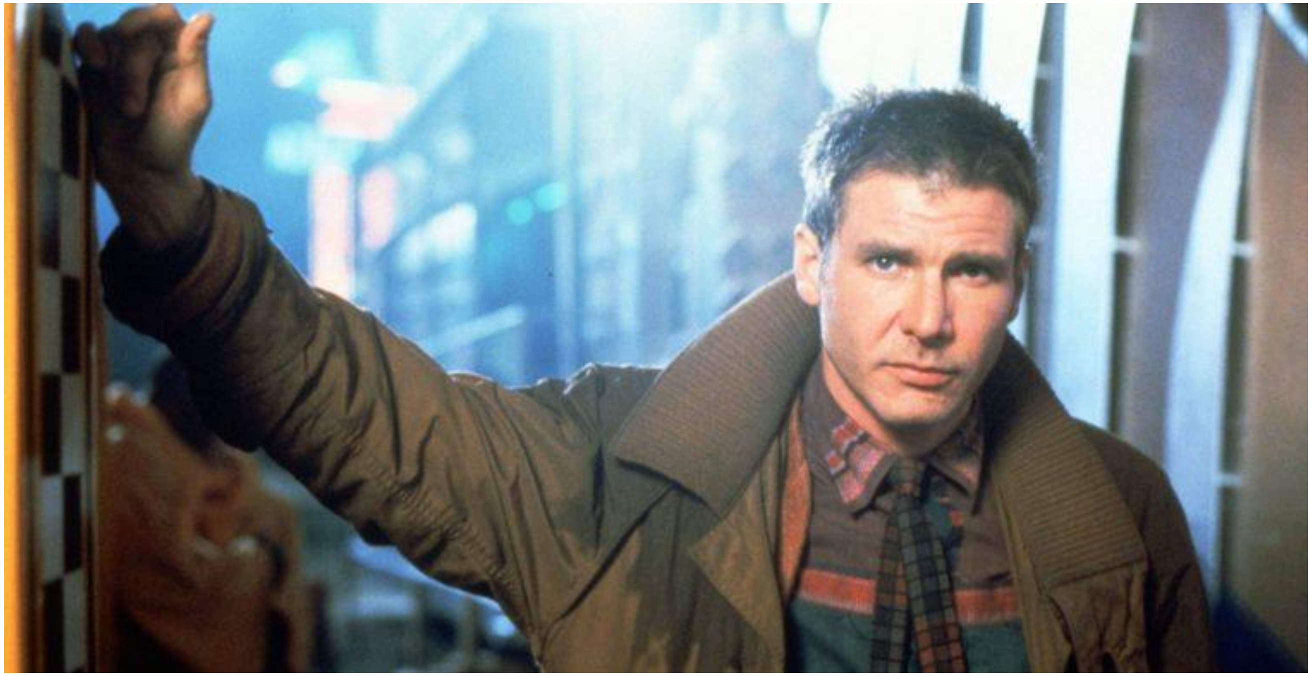
- **Goal** — Eliminate all opponents
- **State** — Raw game pixels of the game
- **Actions** — Up, Down, Left, Right etc
- **Reward** — Positive when eliminating an opponent, negative when the agent is eliminated

Examples of Reinforcement Learning

Bin Packing

- **Goal** - Pick a device from a box and put it into a container
- **State** - Raw pixels of the real world
- **Actions** - Possible actions of the robot
- **Reward** - Positive when placing a device successfully, negative otherwise

Examples of Reinforcement Learning



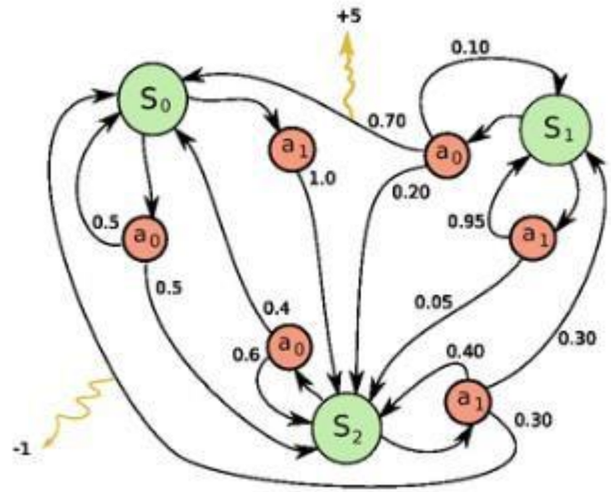
Human Life

- **Goal** - Survival? Happiness?
- **State** - Sight. Hearing. Taste. Smell. Touch.
- **Actions** - Think. Move.
- **Reward** – Homeostasis?

Key Takeaways for Real-World Impact

- Deep Learning:
  - **Fun part:** Good algorithms that learn from data.
  - **Hard part:** Huge amounts of representative data.
- Deep Reinforcement Learning:
  - **Fun part:** Good algorithms that learn from data.
  - **Hard part:** Defining a useful state space, action space, and reward.
  - **Hardest part:** Getting meaningful data for the above formalization.

Markov Decision Process



$$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

state

action

reward

Terminal state

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

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

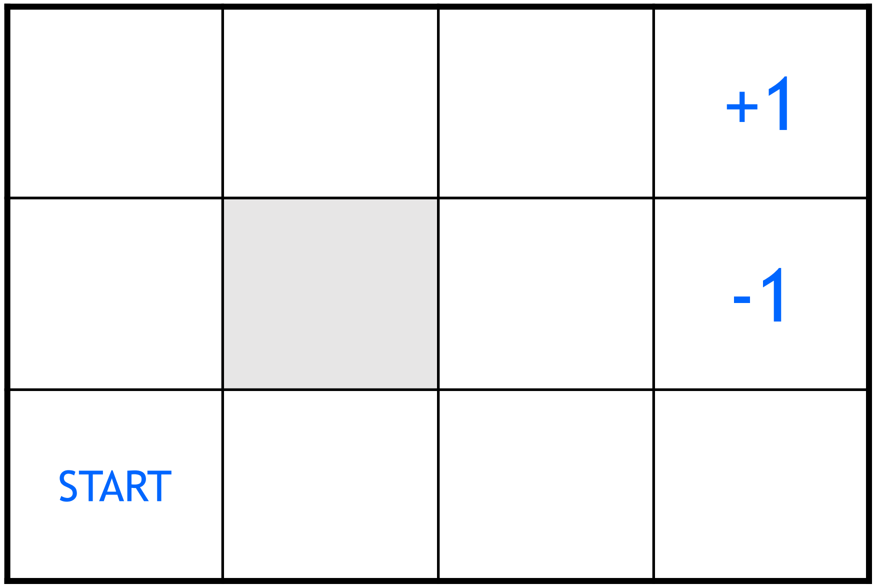
state

action

reward

Terminal state

Robot in a Room

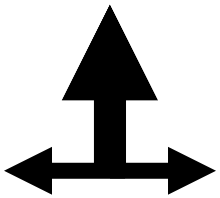


actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

UP

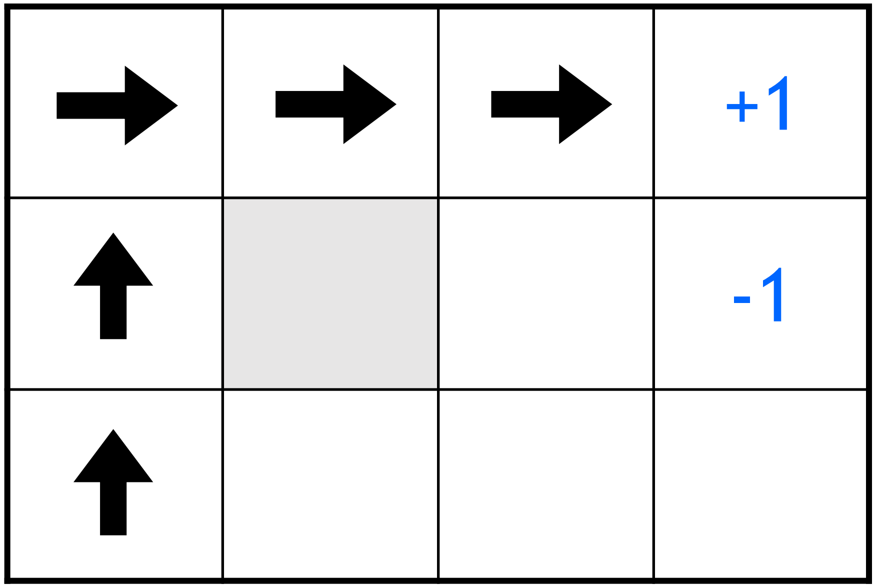
80%	move UP
10%	move LEFT
10%	move RIGHT



- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what’s the strategy to achieve max reward?
- what if the actions were deterministic?

Is this a solution?



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

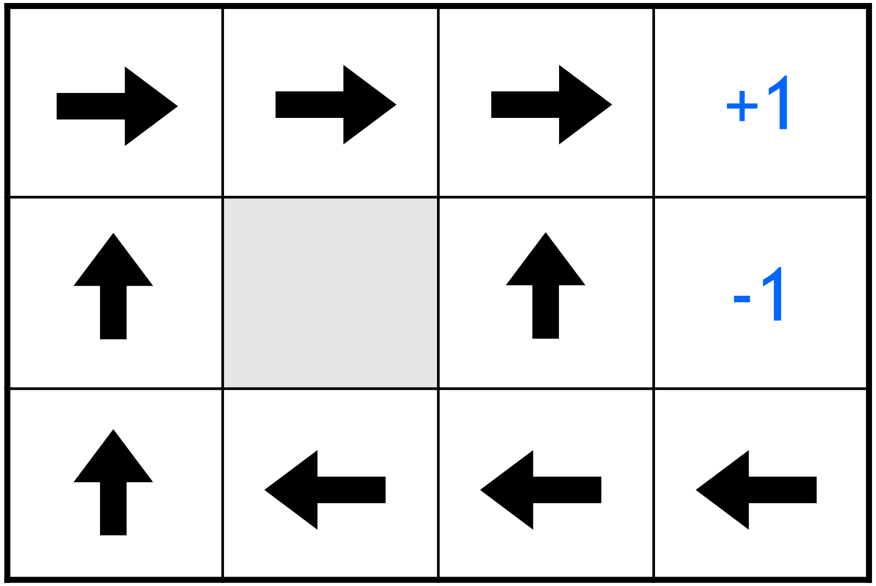
UP

80% move UP
10% move LEFT
10% move RIGHT

- only if actions deterministic
  - not in this case (actions are stochastic)

- solution/policy
  - mapping from each state to an action

Optimal policy



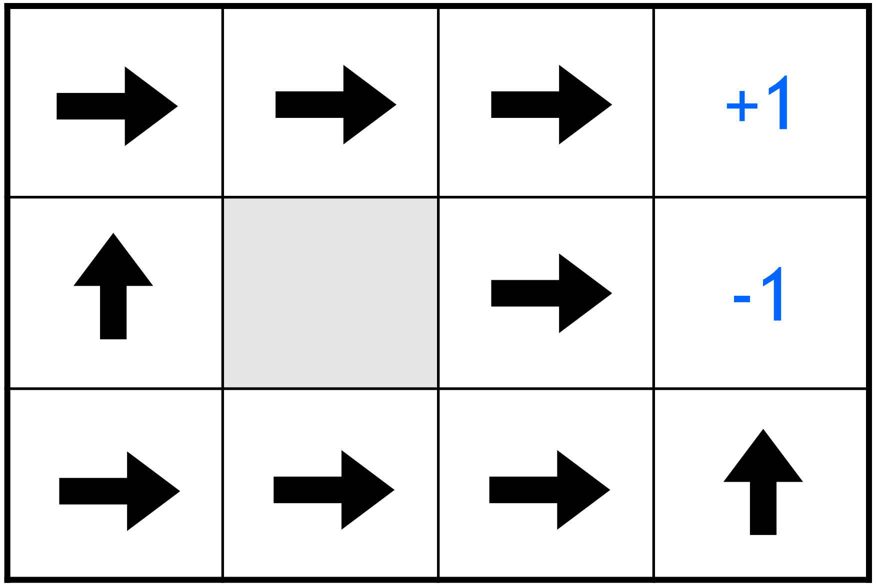
actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

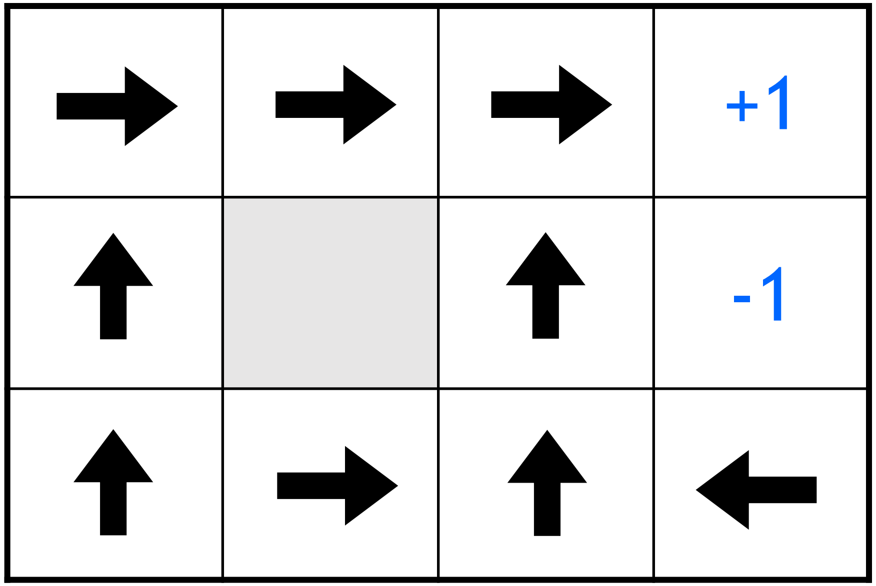
UP

80%	move UP
10%	move LEFT
10%	move RIGHT

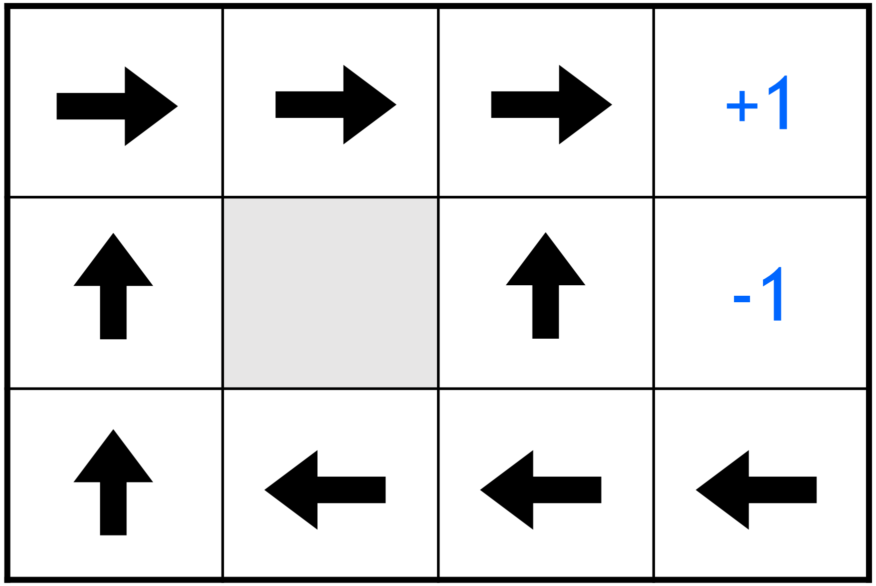
Reward for each step -2



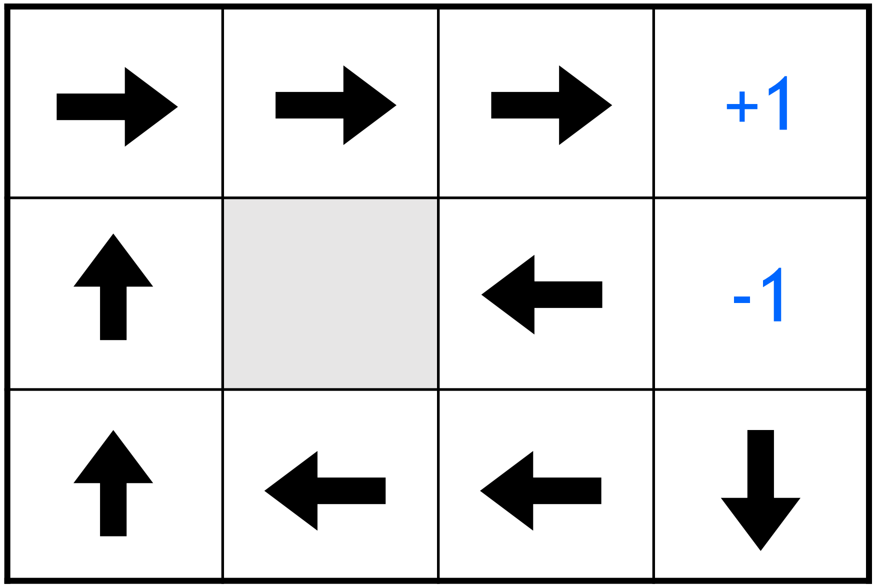
Reward for each step: -0.1



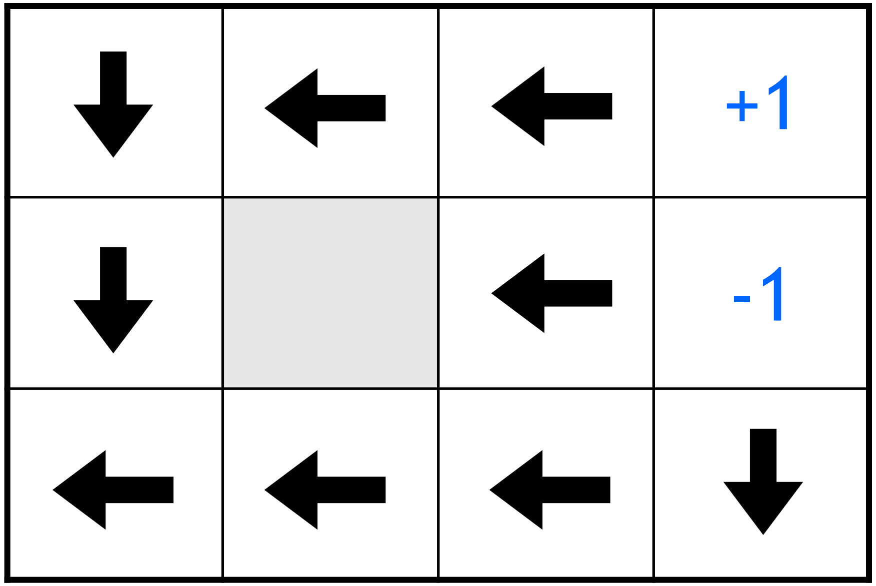
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01



Value Function

- Future reward $R = r_1 + r_2 + r_3 + \cdots + r_n$

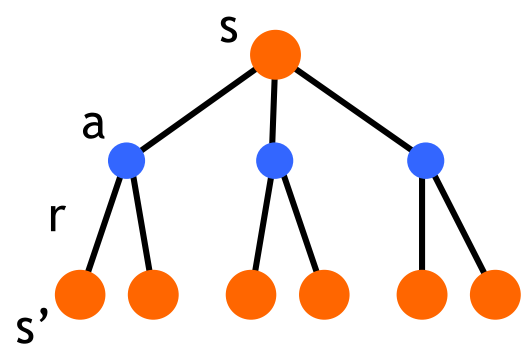
$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

- Discounted future reward (environment is stochastic)

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \\ &= r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \cdots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

- A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward

Q-Learning



- State-action value function: $Q^{\pi}(s,a)$
  - Expected return when starting in *s*, performing *a,* and following π
- Q-Learning: Use **any policy** to estimate Q that maximizes future reward:
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

Learning Rate

Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New State

Old State

Reward

Exploration vs Exploitation

- Deterministic/greedy policy won’t explore all actions
  - Don’t know anything about the environment at the beginning
  - Need to try all actions to find the optimal one
- ε-greedy policy
  - With probability 1-ε perform the optimal/greedy action, otherwise random action
  - Slowly move it towards greedy policy: ε -> 0

Q-Learning: Value Iteration

Learning Rate

Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New State

Old State

Reward

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
    select and carry out an action a
    observe reward r and new state s'
    Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
    s = s'
until terminated
```

References: [84]

Q-Learning: Representation Matters

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $\mathbf{84 \times 84}$ (resized)
    - Consecutive **4** images
    - Grayscale with **256** gray levels

$\mathbf{256^{84\times84\times4}}$ rows in the Q-table!

Philosophical Motivation for **Deep** Reinforcement Learning

Takeaway from Supervised Learning:

Neural networks are great at memorization and not (yet) great at reasoning.

Hope for Reinforcement Learning:

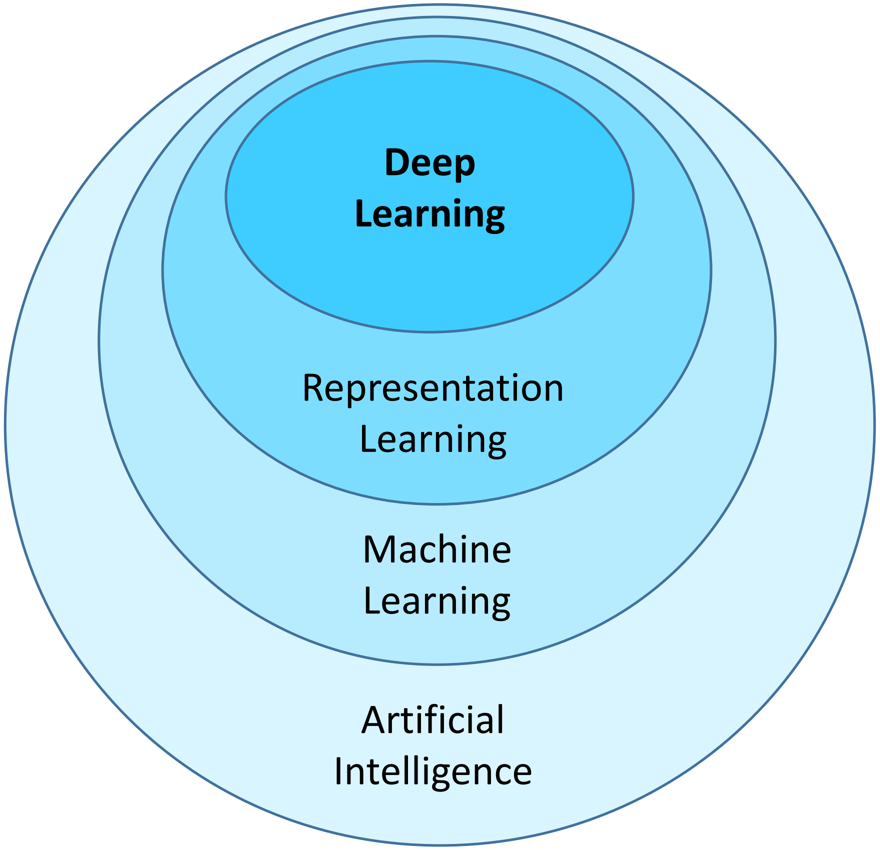
Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force "reasoning".

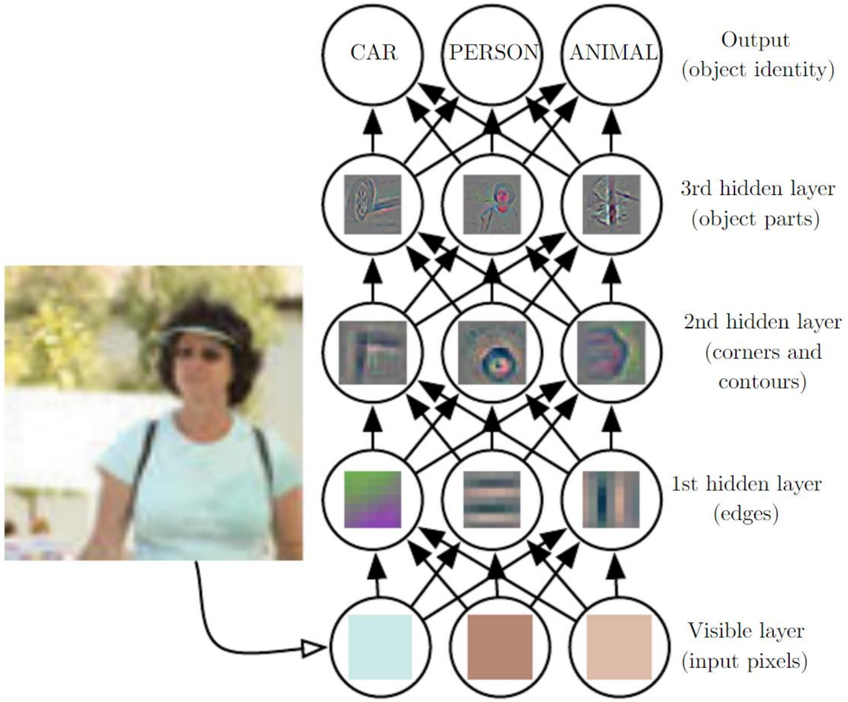
Hope for Deep Learning + Reinforcement Learning:

General purpose artificial intelligence through efficient generalizable learning of the optimal thing to do given a formalized set of actions and states (possibly huge).

Deep Learning is **Representation Learning**

(aka Feature Learning)



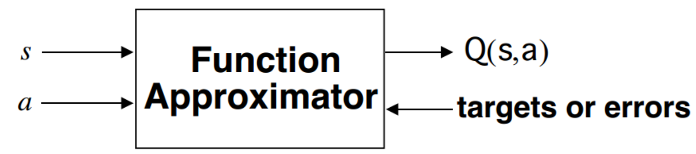


Intelligence: Ability to accomplish complex goals.

Understanding: Ability to turn complex information to into simple, useful information.

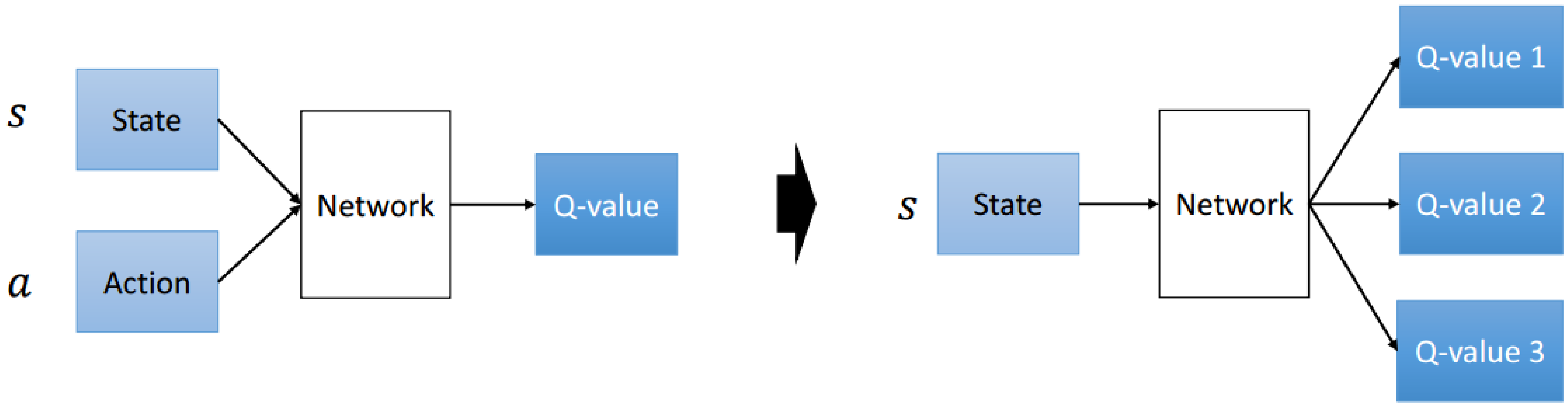
DQN: Deep Q-Learning

Use a function (with parameters) to approximate the Q-function

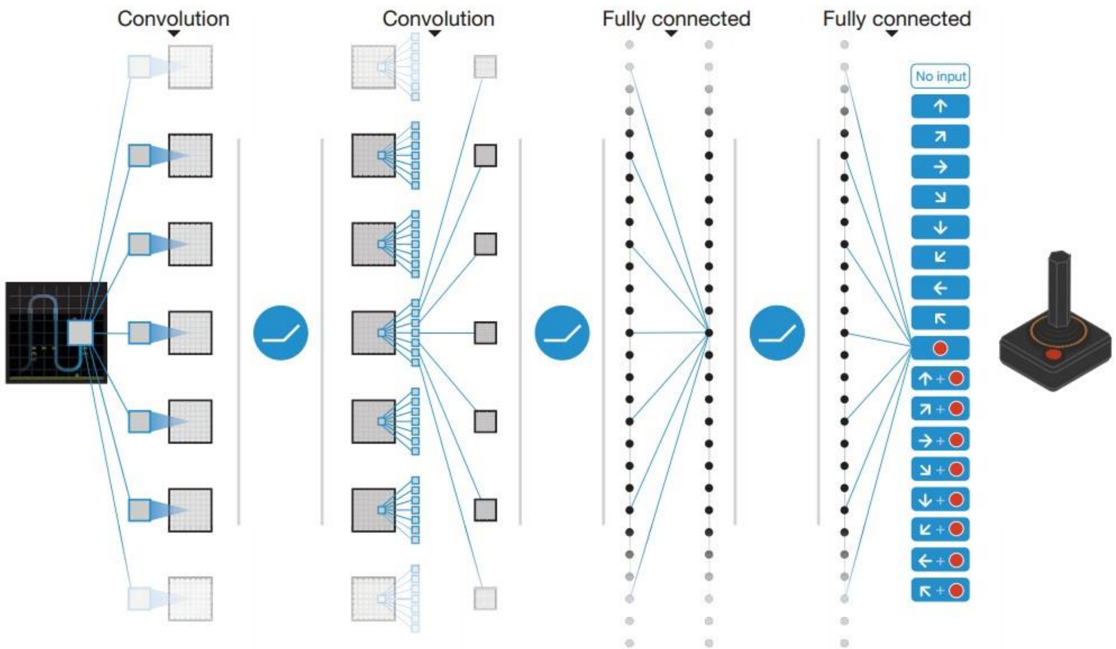


- Linear
- Non-linear: **Q-Network**

$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$



Deep Q-Network (DQN): Atari



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Mnih et al. "Playing atari with deep reinforcement learning." 2013.

DQN and Double DQN (DDQN)

- Loss function (squared error):

$$L = \mathbb{E}[(\underbrace{r + \gamma max_{a'} Q(s', a')}_{\textbf{target}} - \underbrace{Q(s, a)}_{\textbf{prediction}})^2]$$

- DQN: same network for both Q
- DDQN: separate network for each Q
  - Helps reduce bias introduced by the inaccuracies of Q network at the beginning of training

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

- Reward Clipping
  - To standardize rewards across games by setting all positive rewards to +1 and all negative to -1.
- Skipping Frames
  - Skip every 4 frames to take action

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

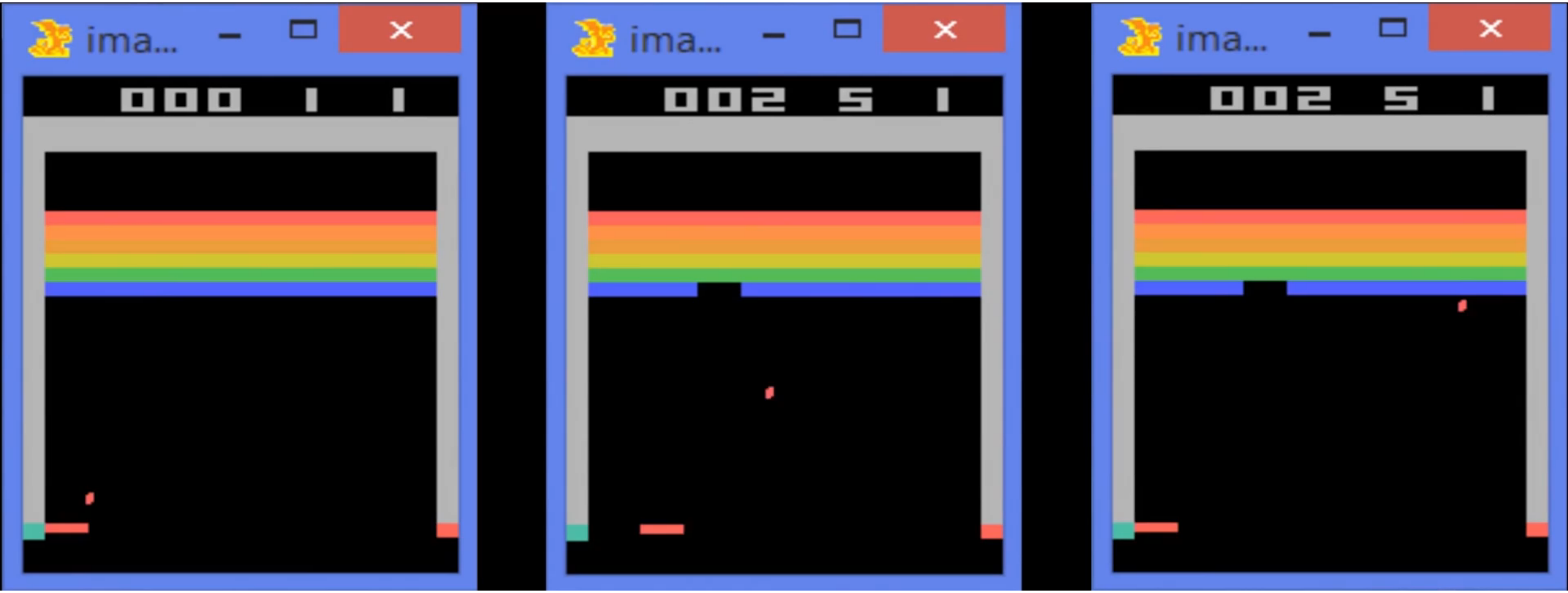
Deep Q-Learning Algorithm

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
     select an action a
          with probability ε select a random action
          otherwise select a = argmax_a' Q(s,a')
     carry out action a
     observe reward r and new state s'
     store experience <s, a, r, s'> in replay memory D

     sample random transitions <ss, aa, rr, ss'> from replay memory D
     calculate target for each minibatch transition
          if ss' is terminal state then tt = rr
          otherwise tt = rr + γmax_a' Q(ss', aa')
     train the Q network using (tt - Q(ss, aa))^2 as loss

     s = s'
until terminated
```

Atari Breakout

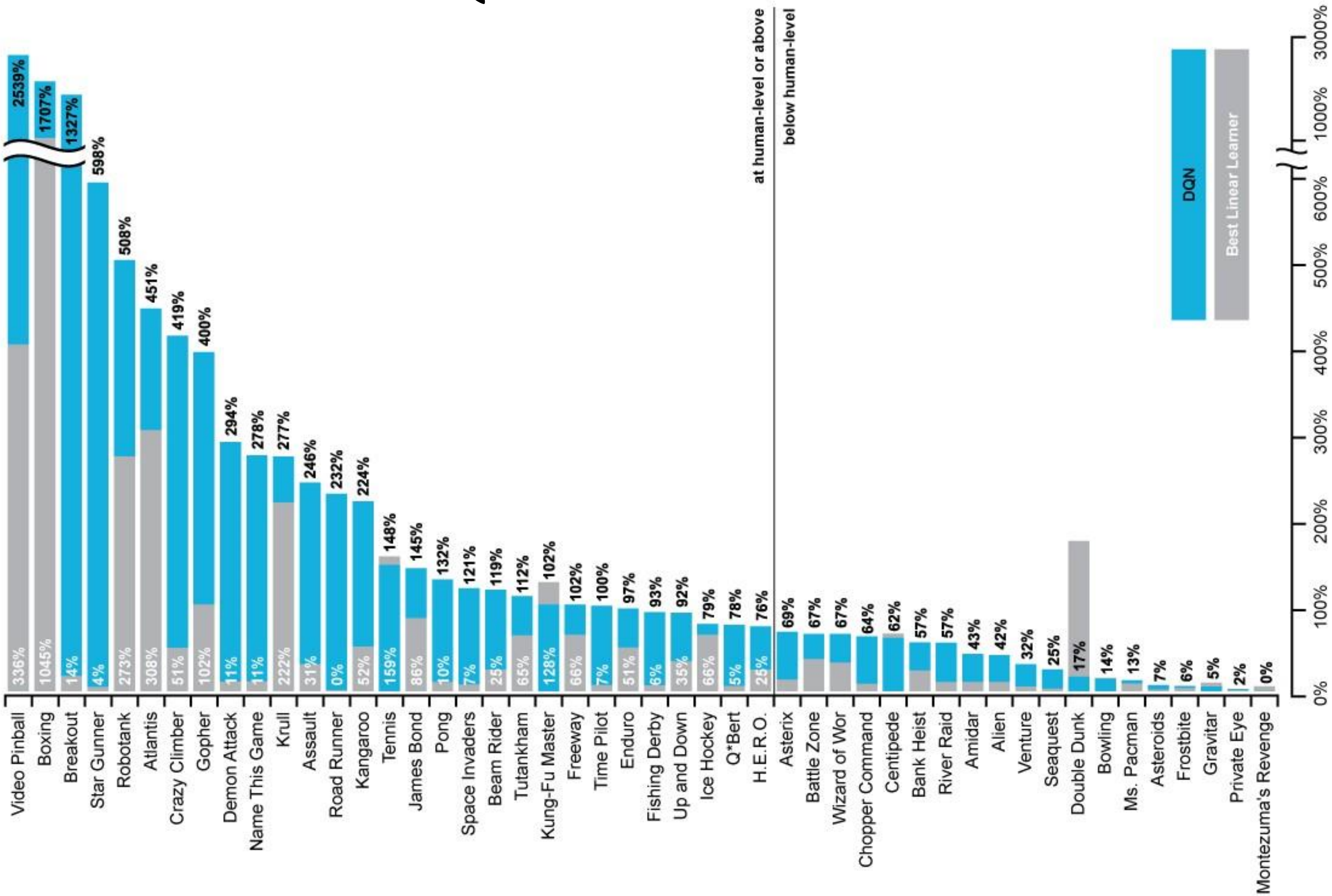


After **10 Minutes** of Training

After **120 Minutes** of Training

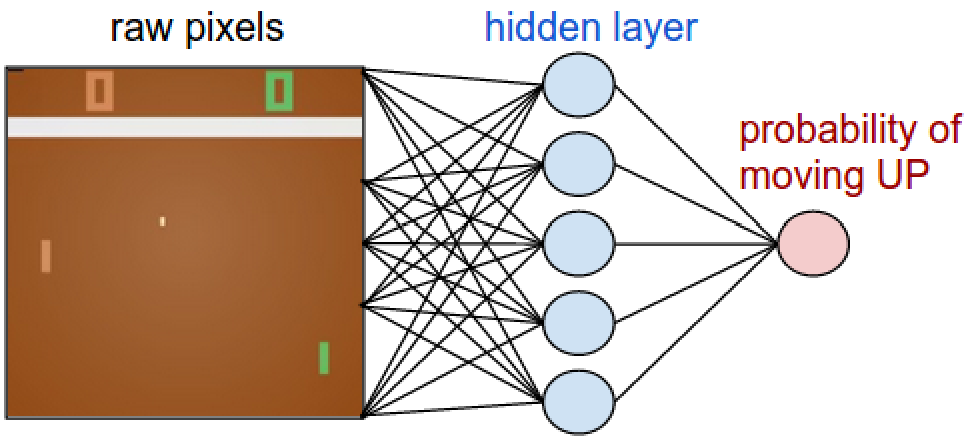
After **240 Minutes** of Training

DQN Results in Atari



Policy Gradients (PG)

- **DQN (off-policy):** Approximate Q and infer optimal policy
- **PG (on-policy):** Directly optimize policy space

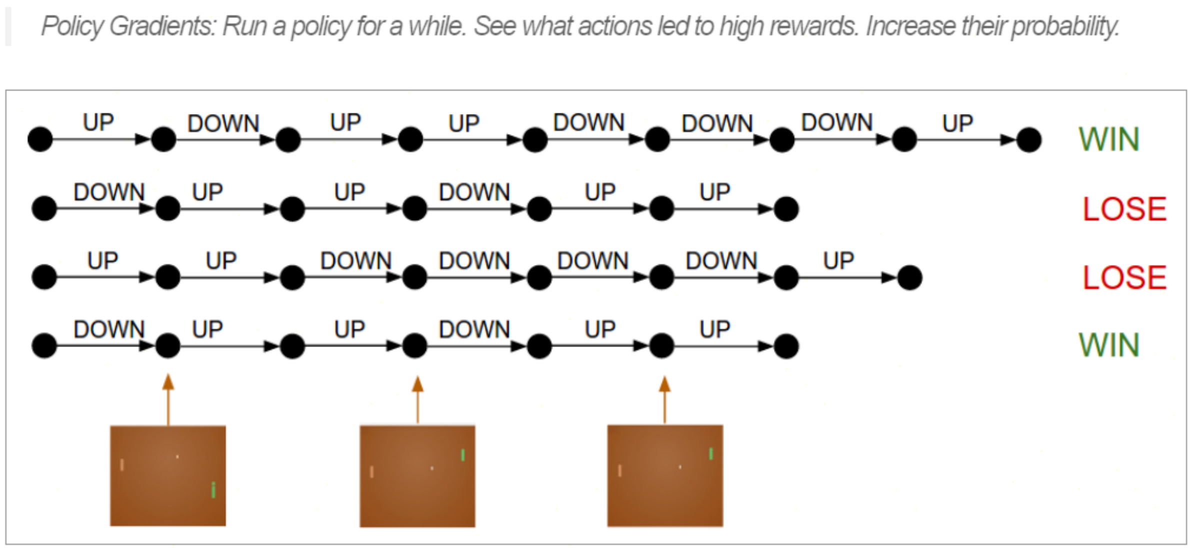


Policy Network

Good illustrative explanation:
http://karpathy.github.io/2016/05/31/rl/

"Deep Reinforcement Learning: Pong from Pixels"

Policy Gradients – Training



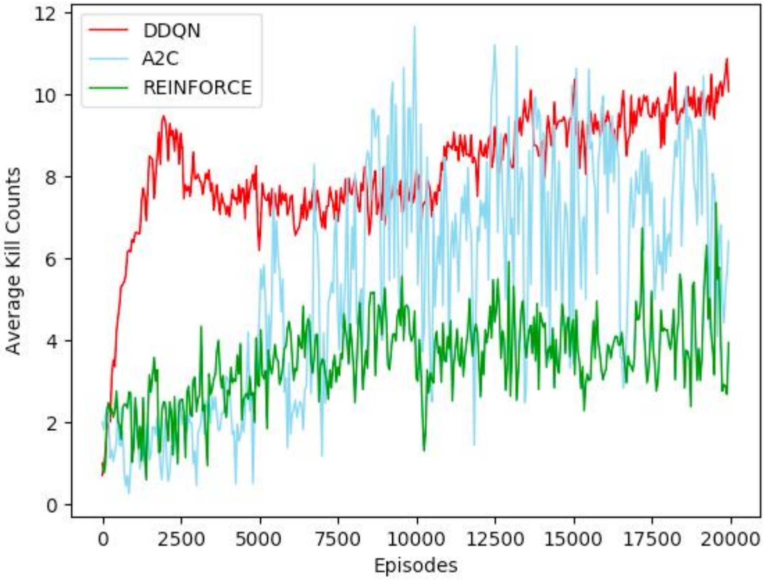
- **REINFORCE (aka Actor-Critic):** Policy gradient that increases probability of good actions and decreases probability of bad action:

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(a) R_t]$$

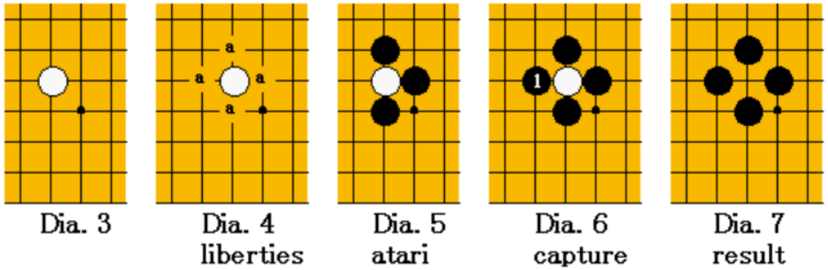
- Policy network is the "actor"
- R_t is the "critic"

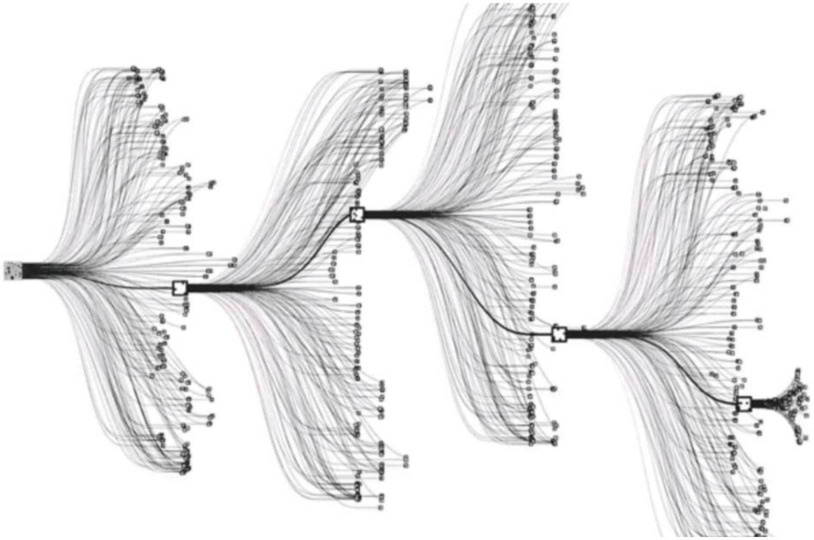
Policy Gradients (PG)

- Pros vs DQN:
  - Able to deal with more complex Q function
  - Faster convergence
  - Since Policy Gradients model probabilities of actions, it is capable of learning stochastic policies, while DQN can't.
- Cons:
  - Needs more data



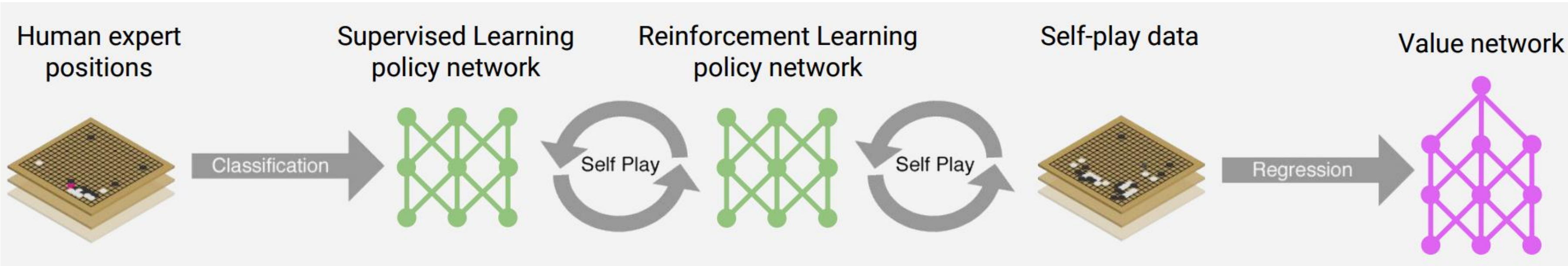
Game of Go

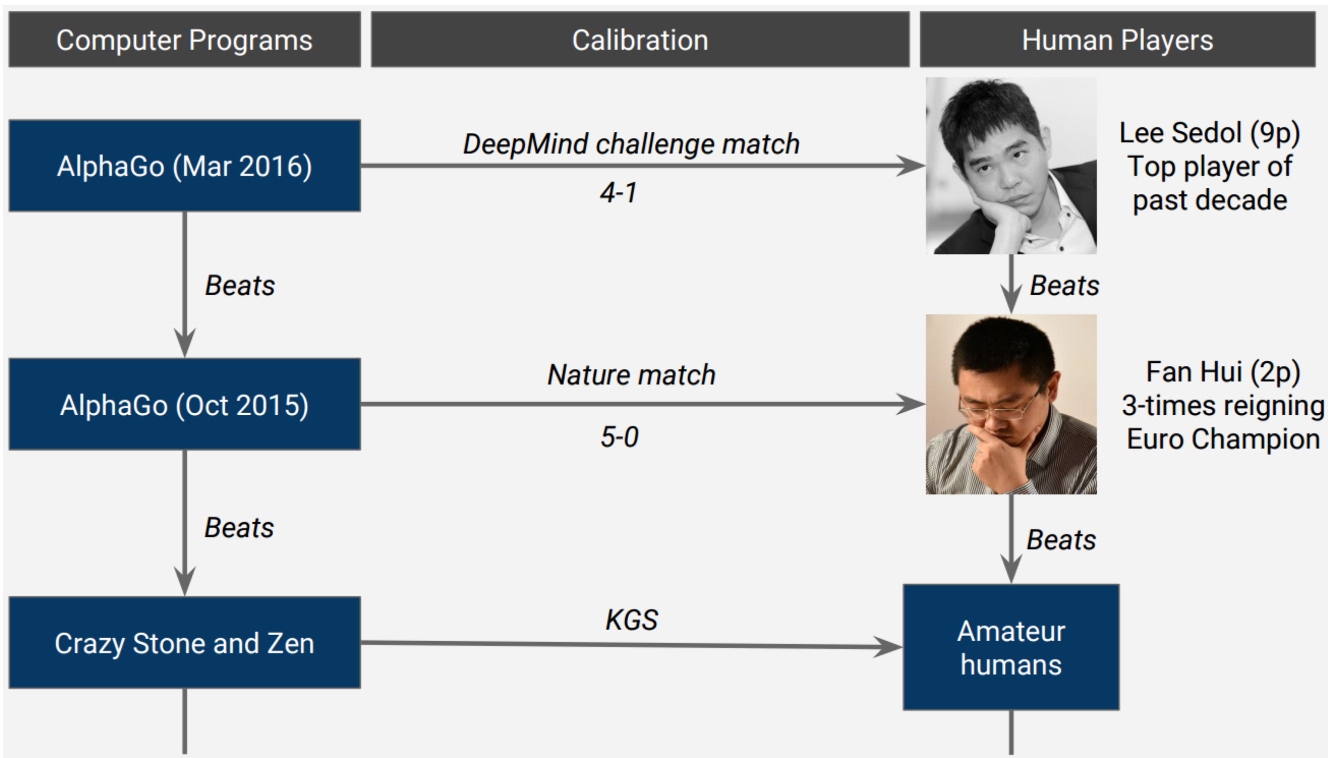




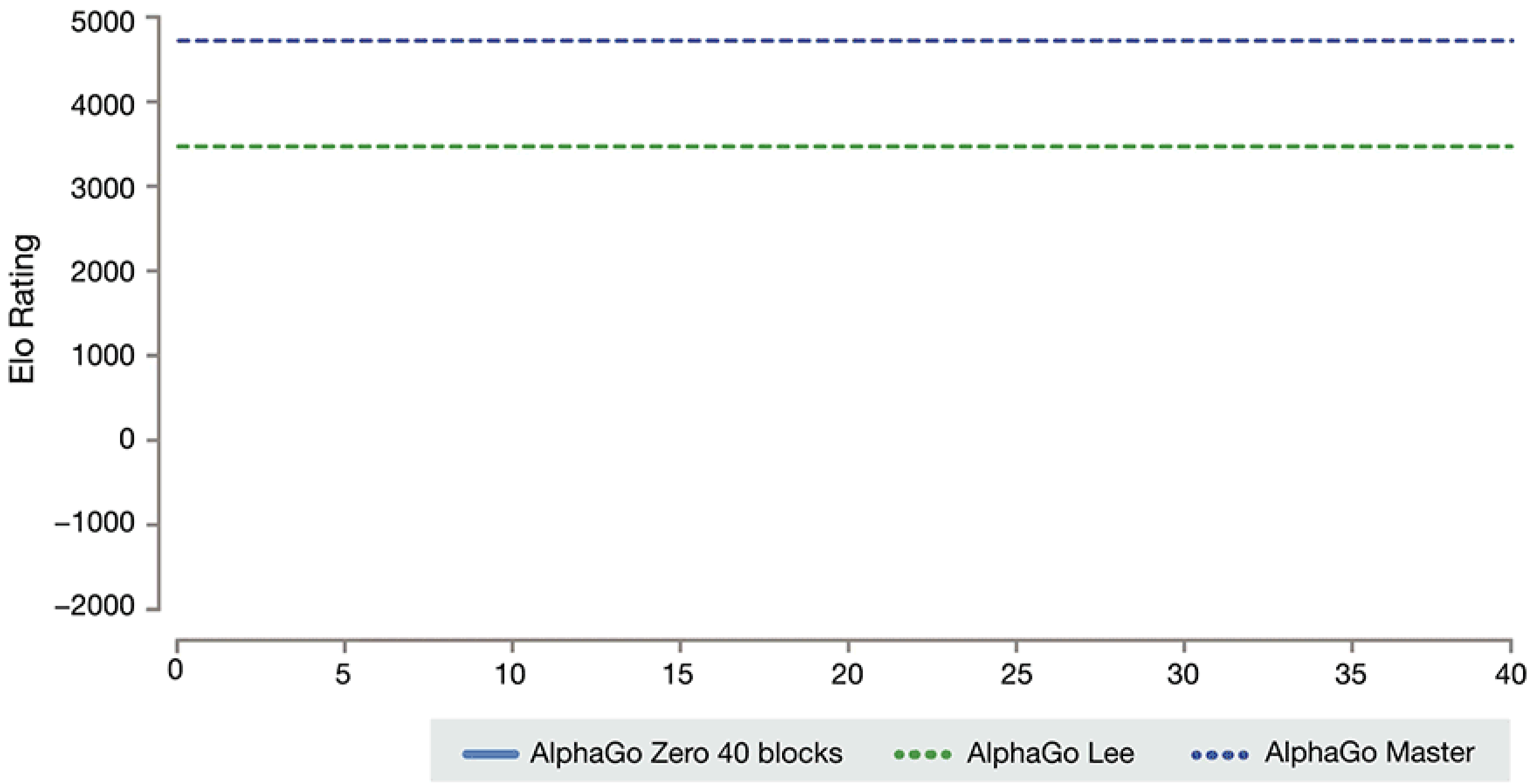
Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}	1.196%	$2.08168199382\times10^{170}$

AlphaGo (2016) Beat Top Human at **Go**



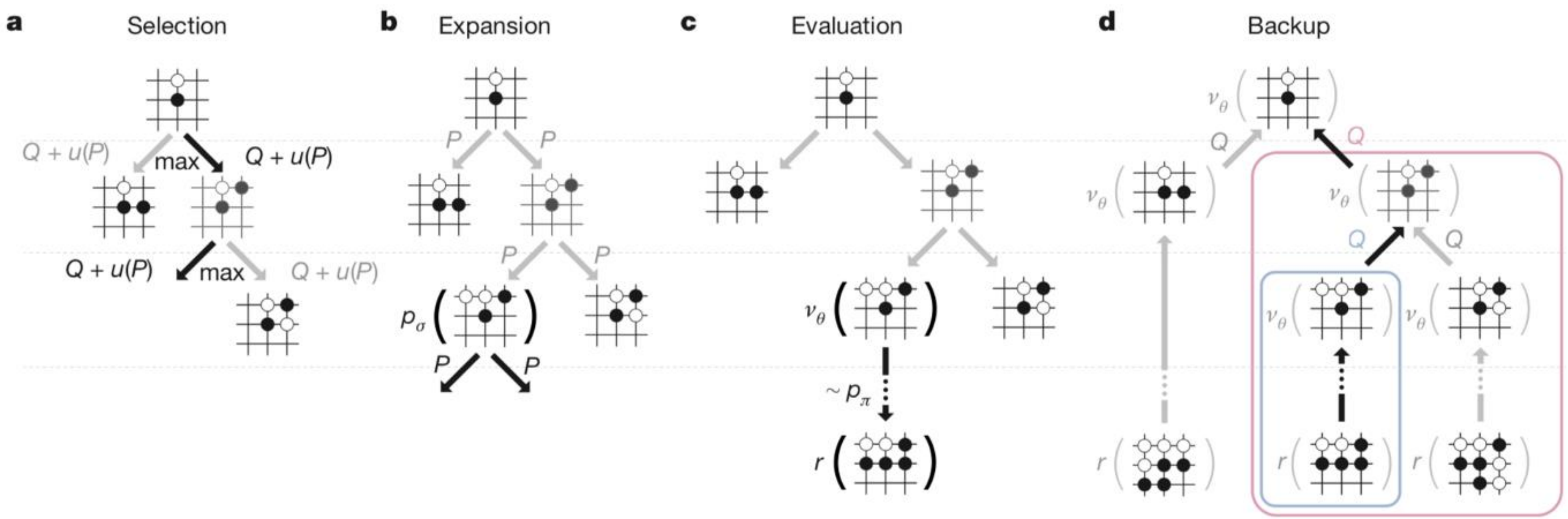


AlphaGo Zero (2017): Beats AlphaGo



AlphaGo Zero Approach

- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)

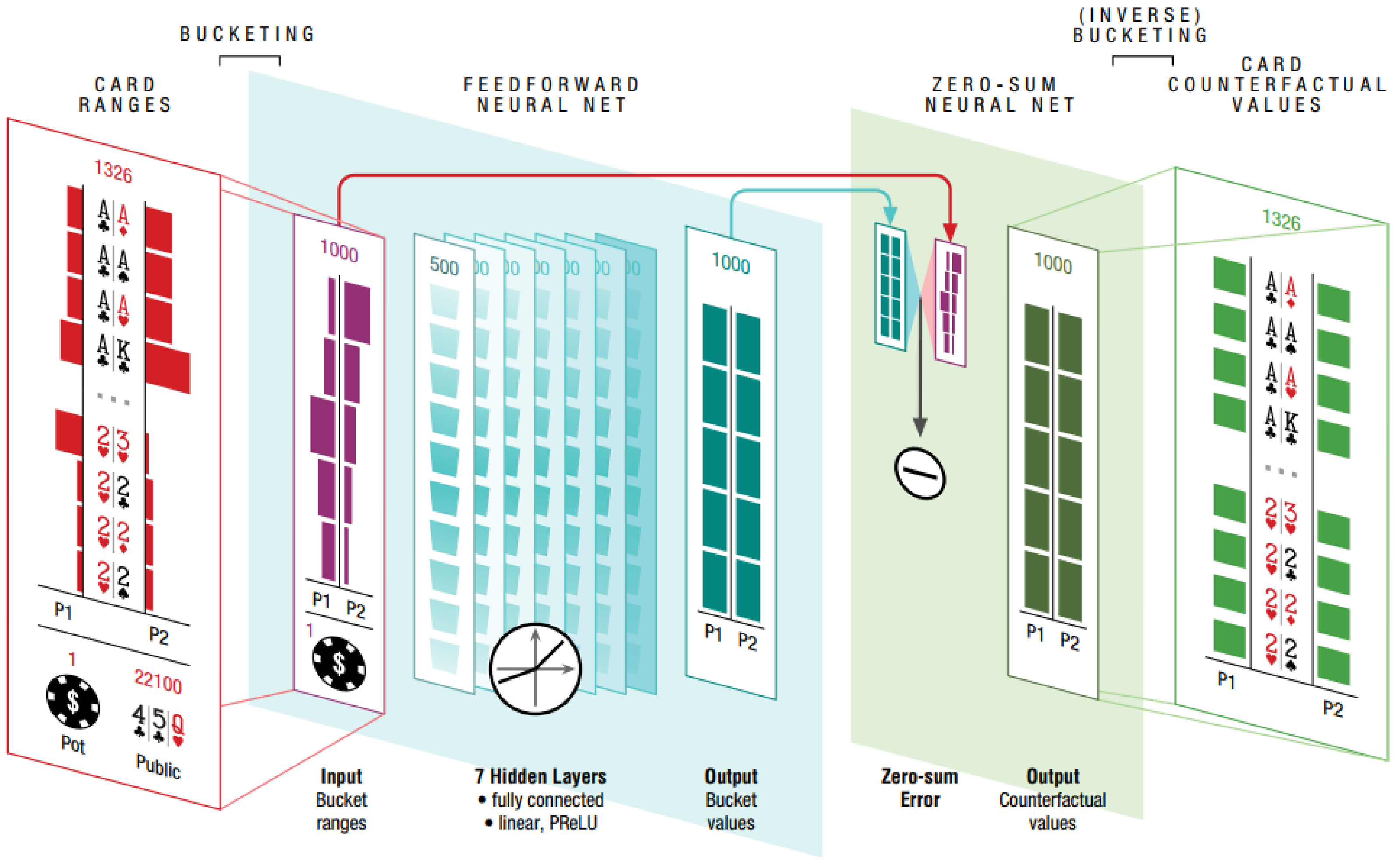


AlphaGo Zero Approach

- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)
- "Tricks"
  - Use MCTS intelligent look-ahead (instead of human games) to improve value estimates of play options
  - Multi-task learning: "two-headed" network that outputs (1) move probability and (2) probability of winning.
  - Updated architecture: use residual networks

DeepStack first to beat professional poker players (2017)

(in heads-up poker)



To date, for most successful robots operating in the real world:

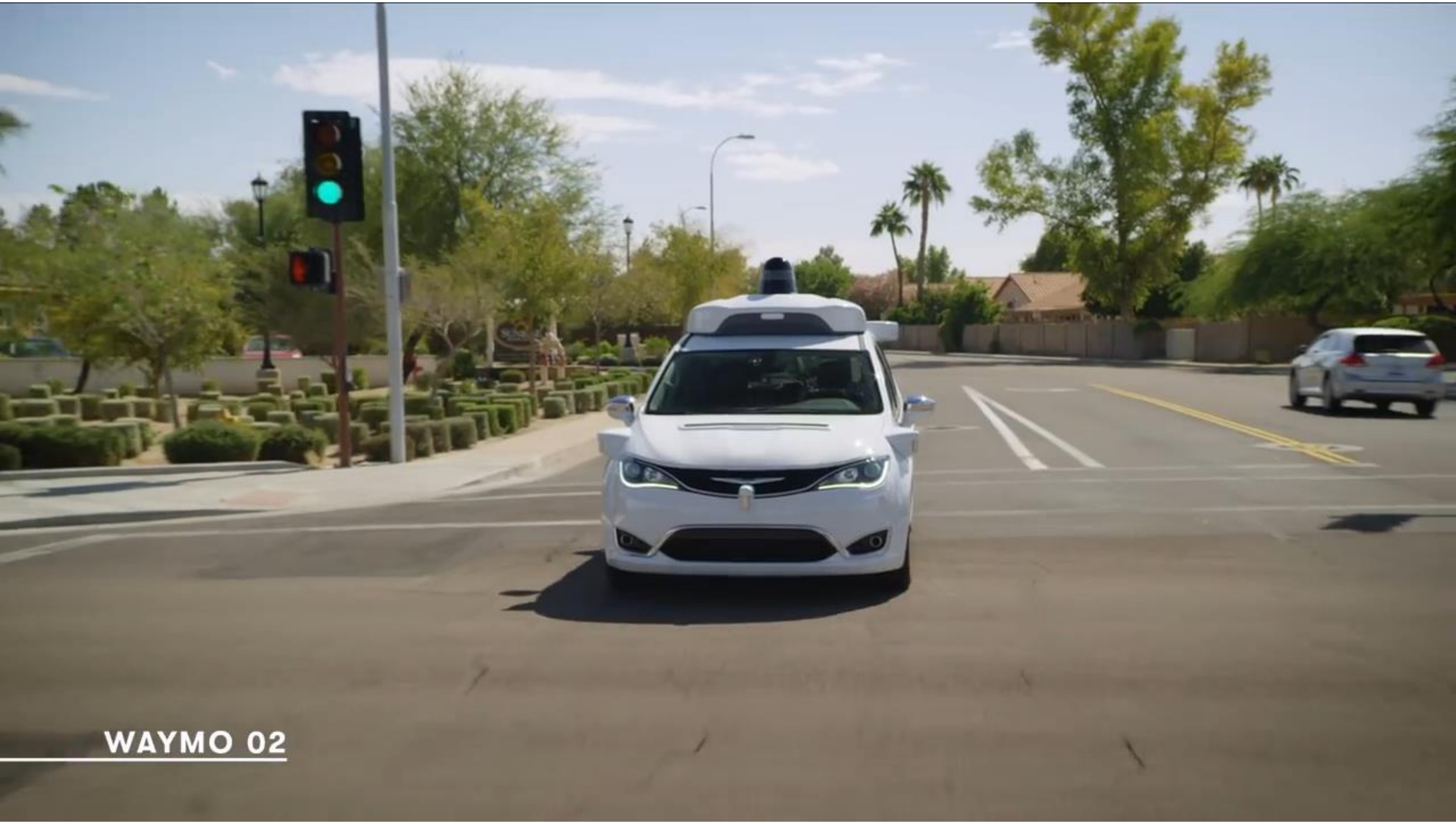
Deep RL is not involved

(to the best of our knowledge)

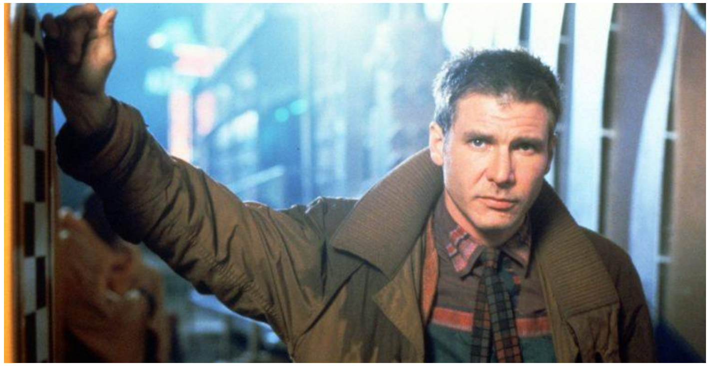
To date, for most successful robots operating in the real world:

Deep RL is not involved

(to the best of our knowledge)



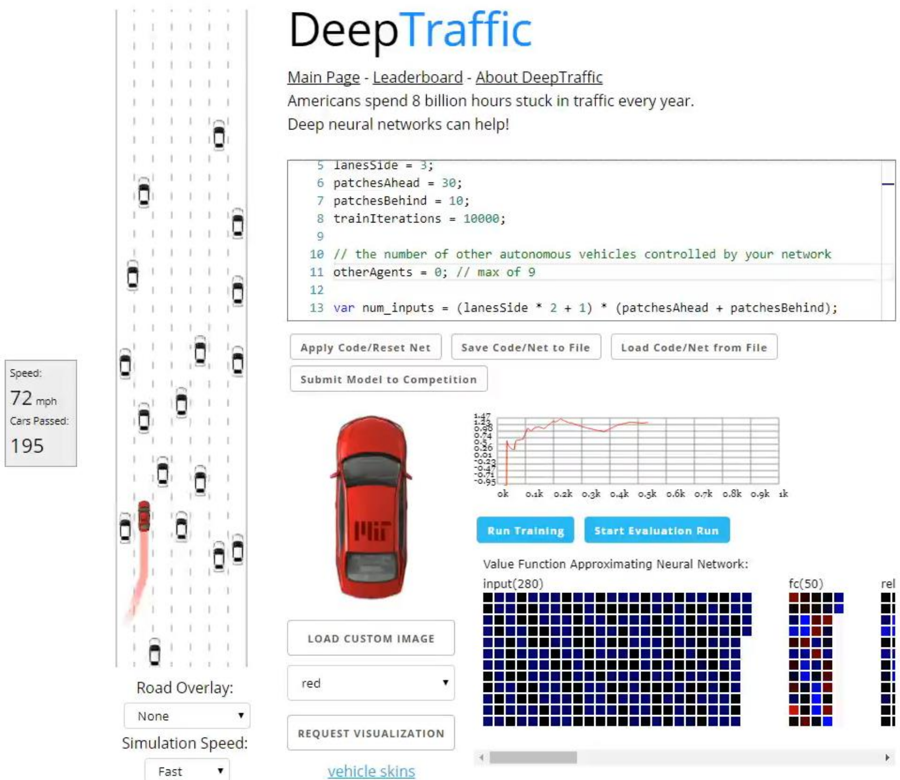
Unexpected Local Pockets of High Reward



AI Safety

Risk *(and thus Human Life)* Part of the Loss Function

DeepTraffic: Deep Reinforcement Learning Competition



https://selfdrivingcars.mit.edu/**deeptraffic**