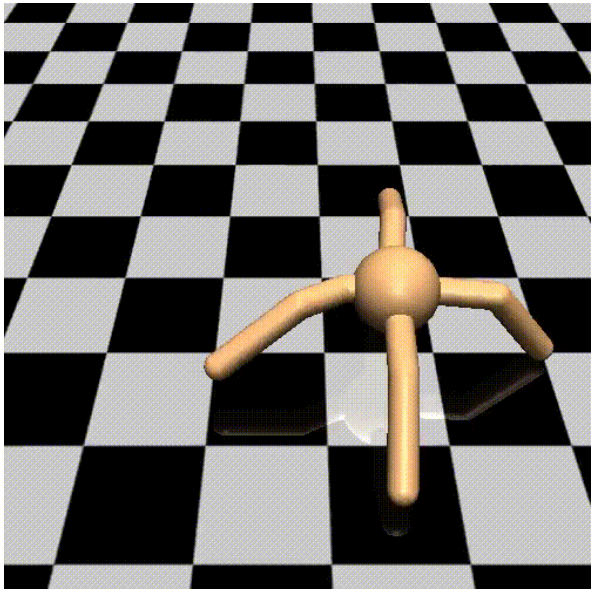
Reinforcement learning and Policy Gradient

Aibek Myrzatay

Deep Reinforcement Learning (Deep RL)

Deep Learning

Deep RL



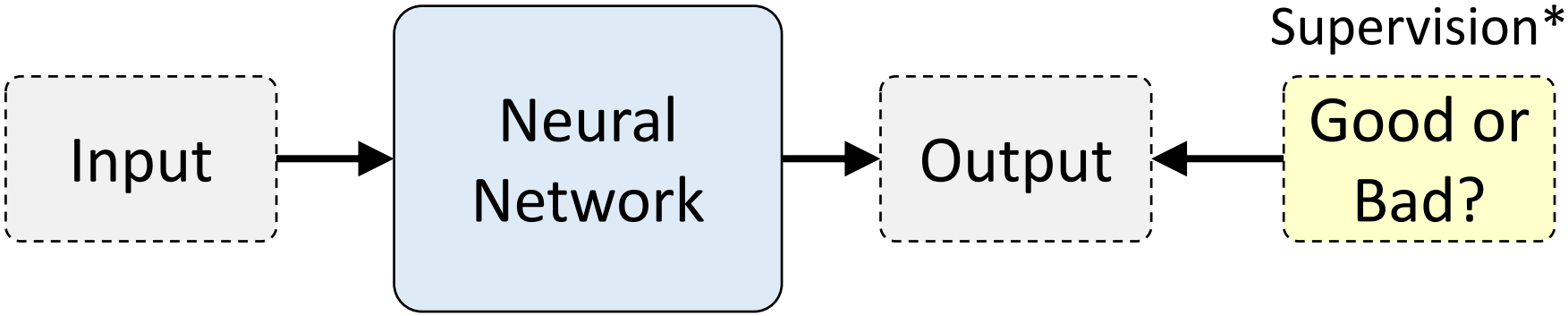
- **What is it?** Framework for learning to solve sequential decision making problems.
- **How?** Trial and error in a world that provides occasional rewards
- **Deep?** Deep RL = RL + Neural Networks

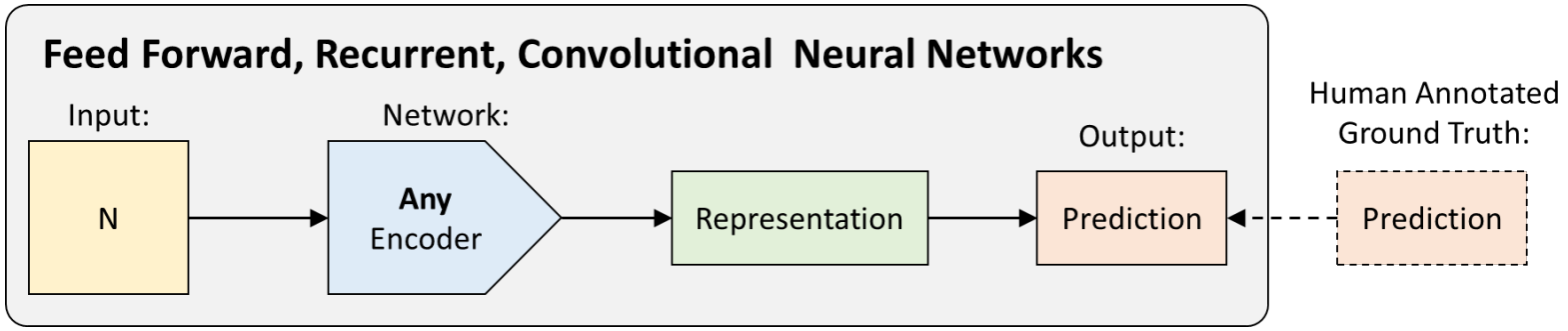
Types of Learning

- Supervised Learning
- Semi-Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

It's all "supervised" by a loss function!

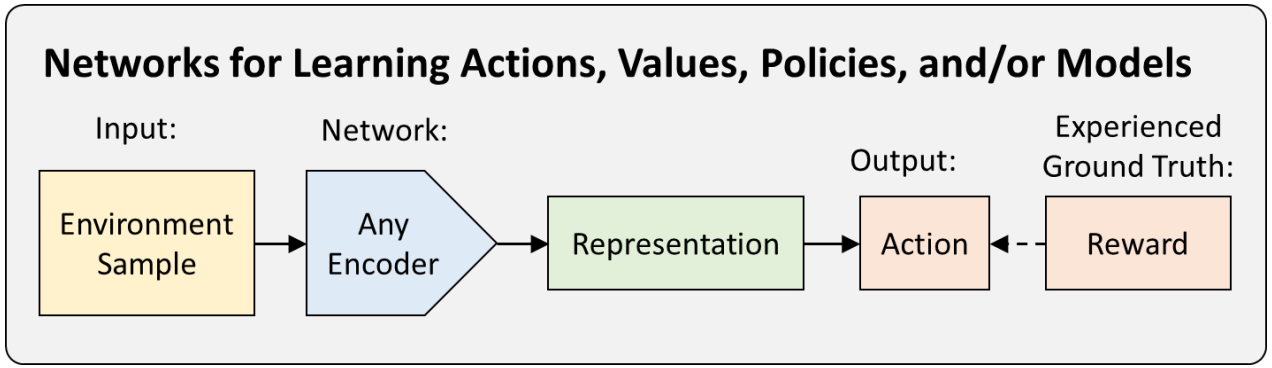
**Someone has to say what's good and what's bad (see Socrates, Epictetus, Kant, Nietzsche, etc.)*





Supervised learning is "teach by **example**":
Here's some examples, now learn patterns in these example.

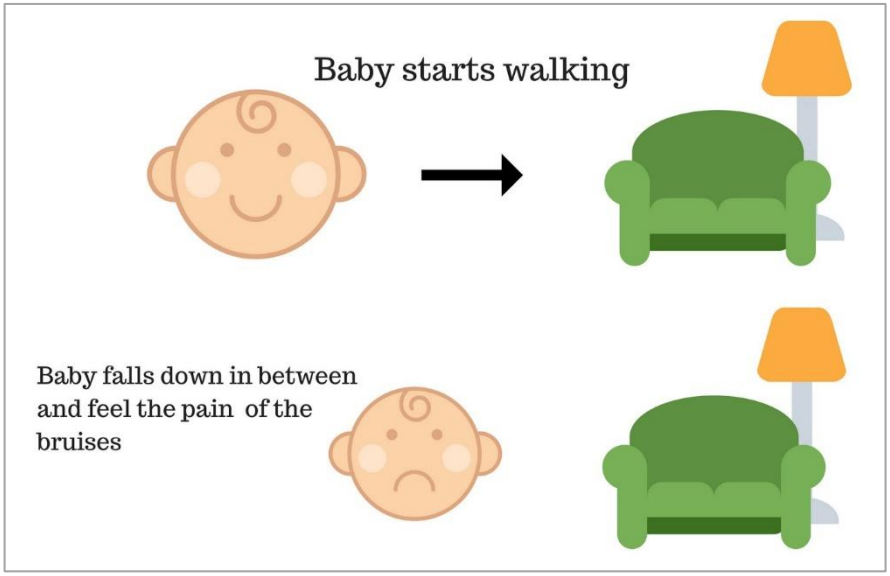
Reinforcement learning is "teach by **experience**":
Here's a world, now learn patterns by exploring it.



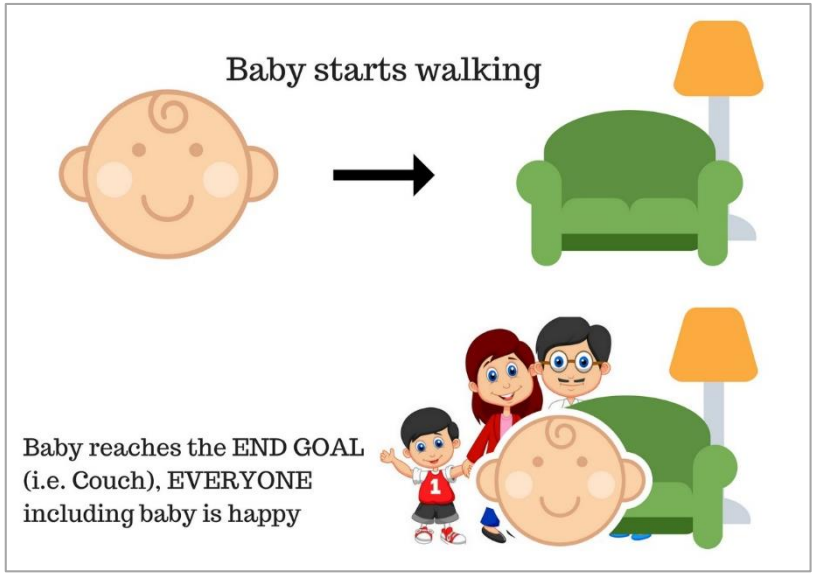
Machine Learning: Supervised vs Reinforcement

Supervised learning is "teach by **example**":
Here's some examples, now learn patterns in these example.

Reinforcement learning is "teach by **experience**":
Here's a world, now learn patterns by exploring it.



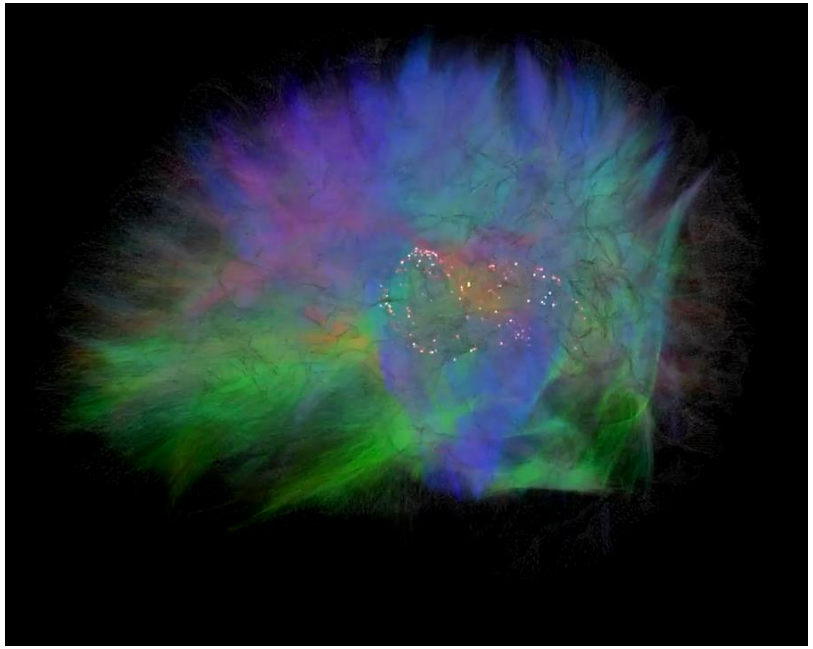
Failure

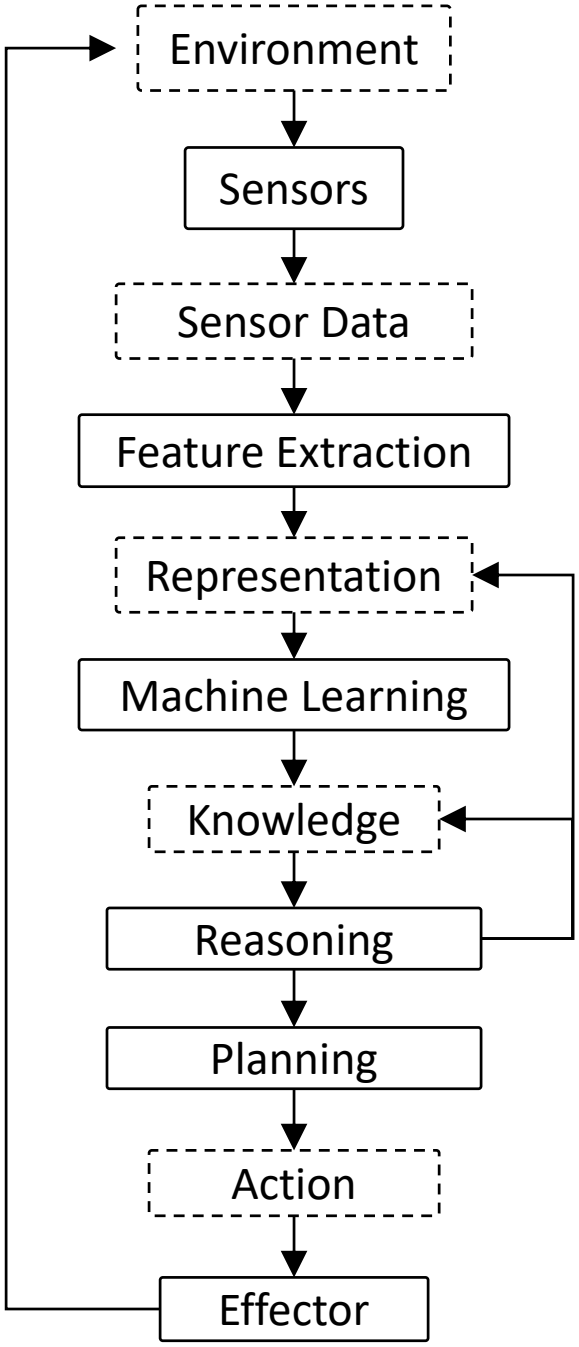


Success

Reinforcement Learning in Humans

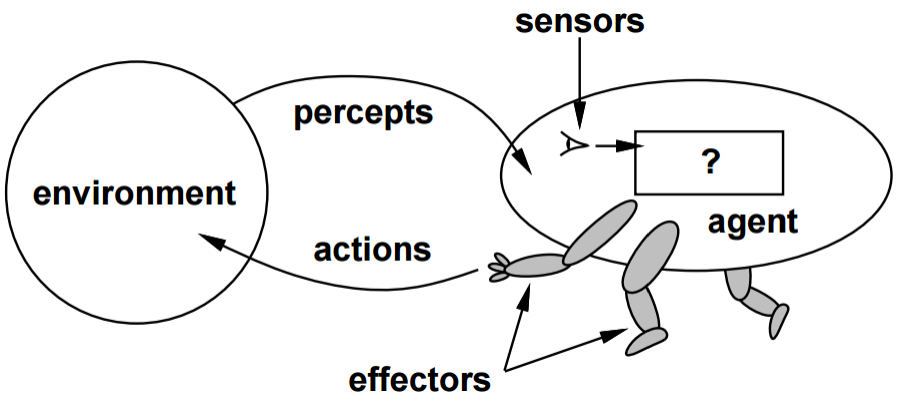
- Human appear to learn to walk through “very few examples” of trial and error. **How** is an open question…
- Possible answers:
  - **Hardware:** 230 million years of bipedal movement data.
  - **Imitation Learning:** Observation of other humans walking.
  - **Algorithms:** Better than backpropagation and stochastic gradient descent

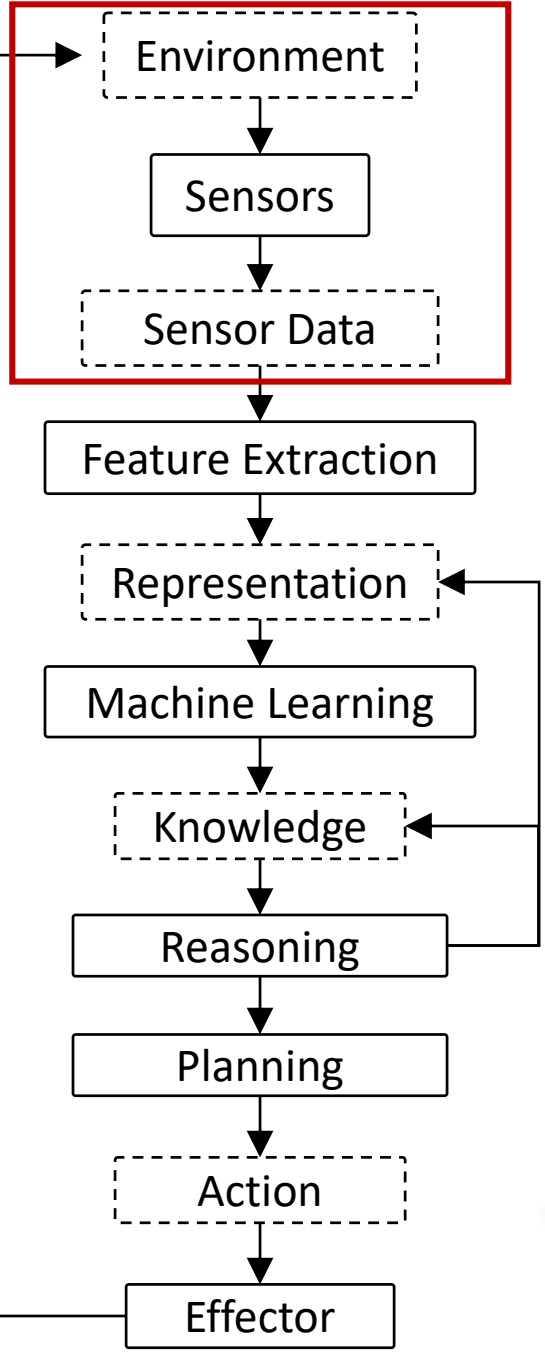


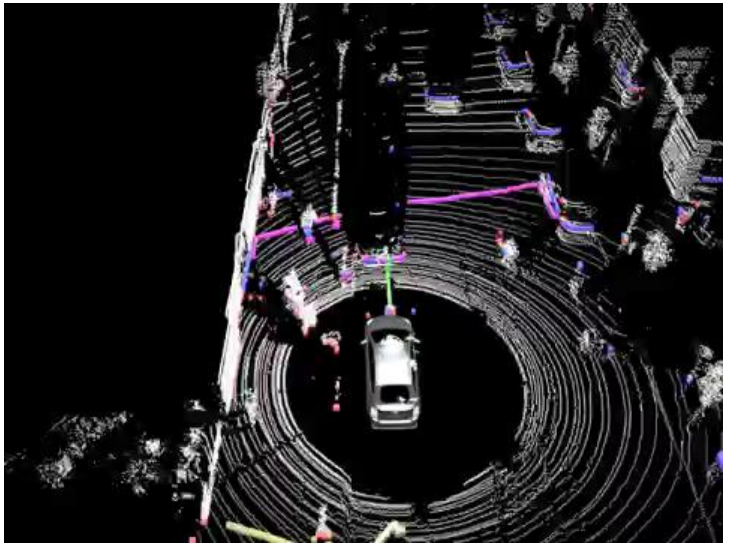


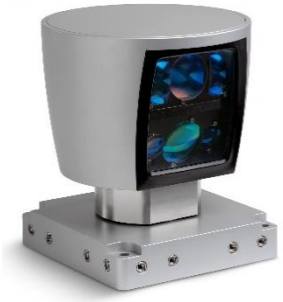
Open Question:

What can be learned from data?









Lidar

Camera
(Visible, Infrared)

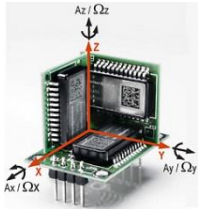
Radar

GPS

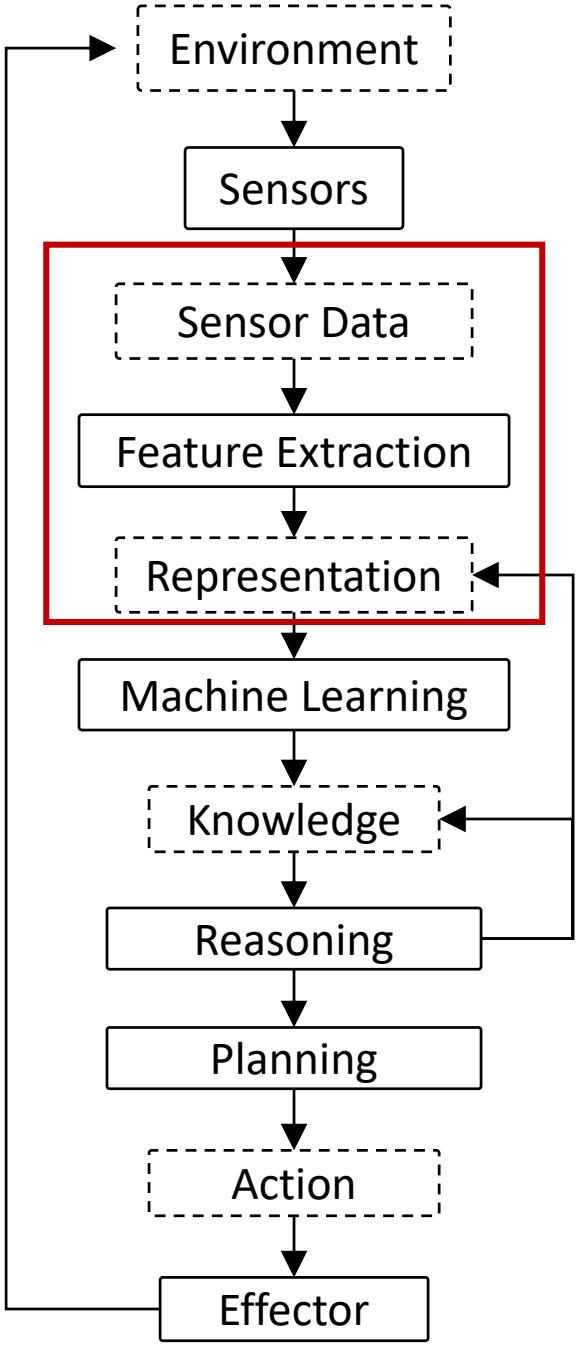
Stereo Camera

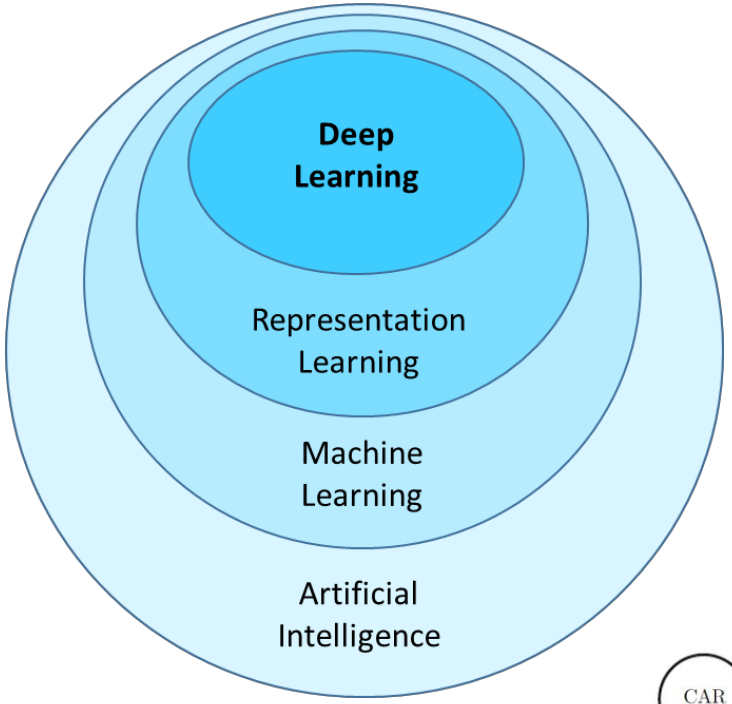
Microphone

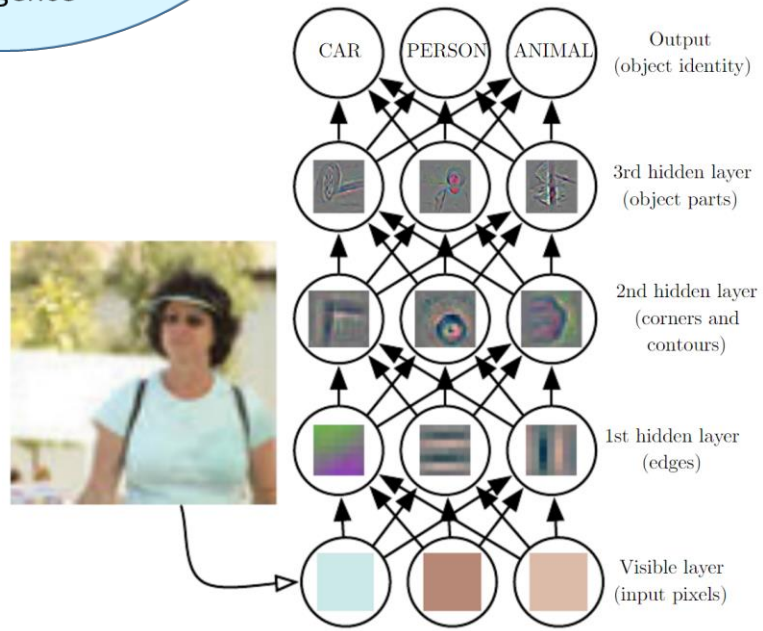
Networking
(Wired, Wireless)

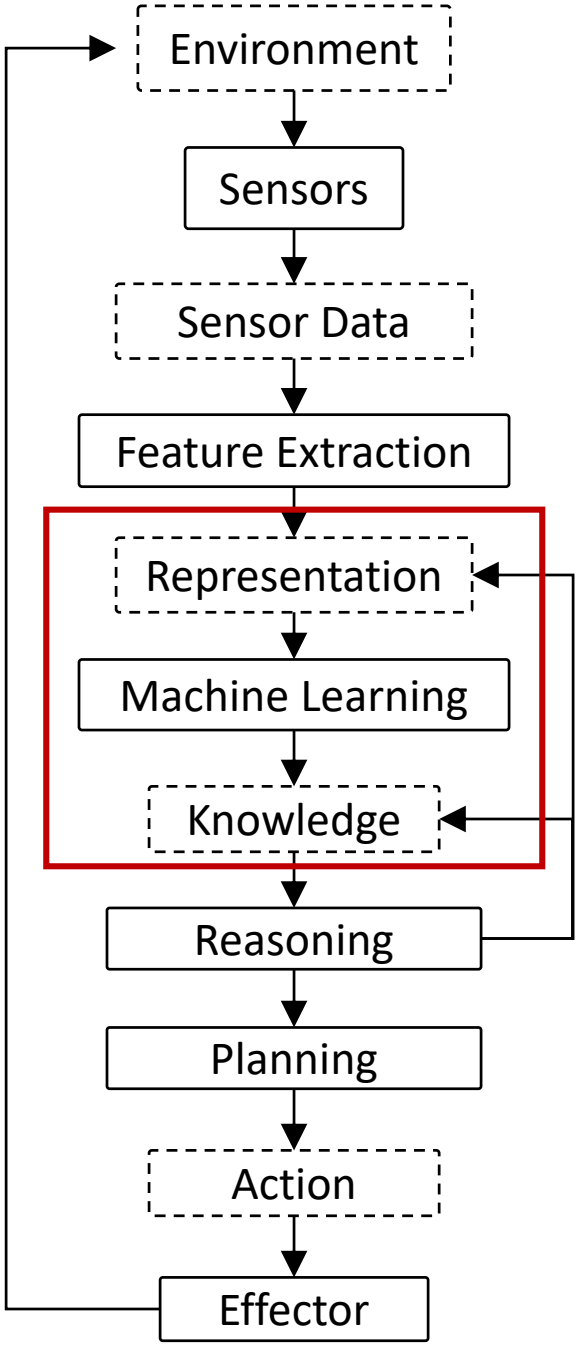


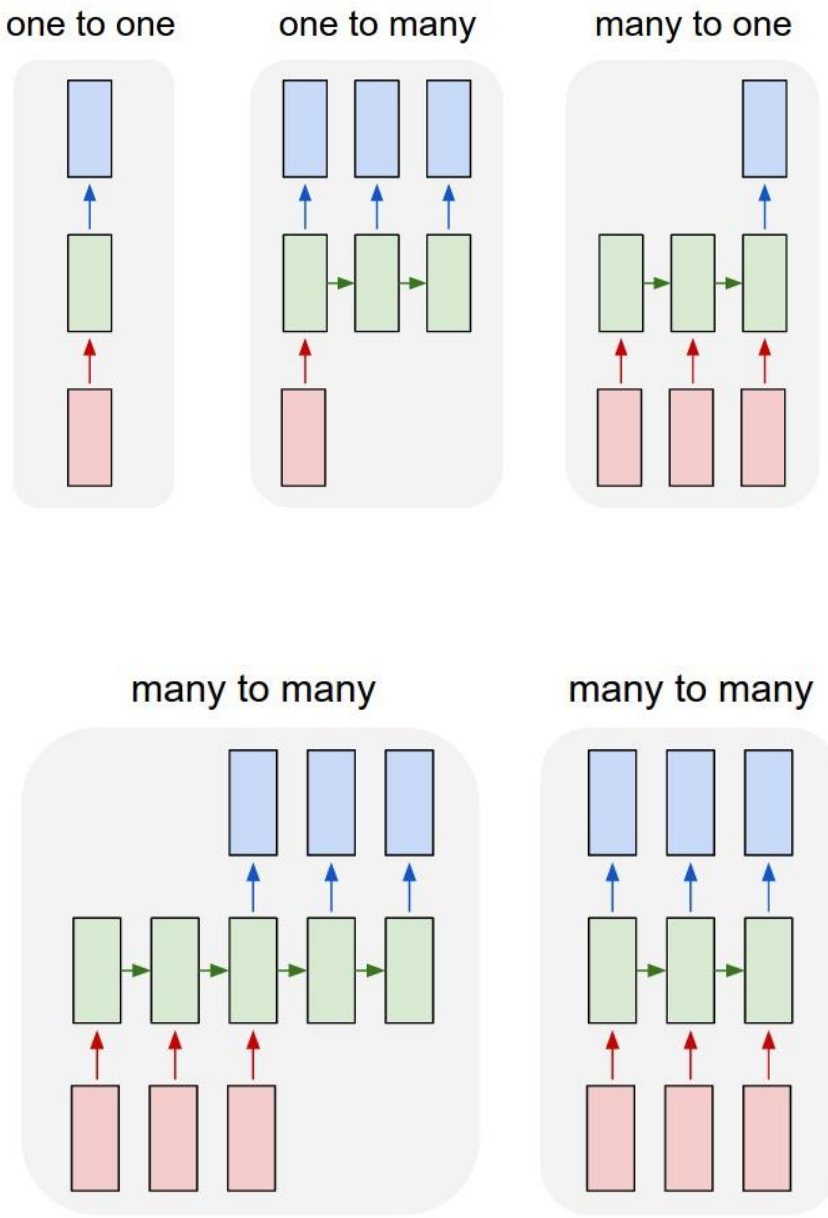
IMU


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

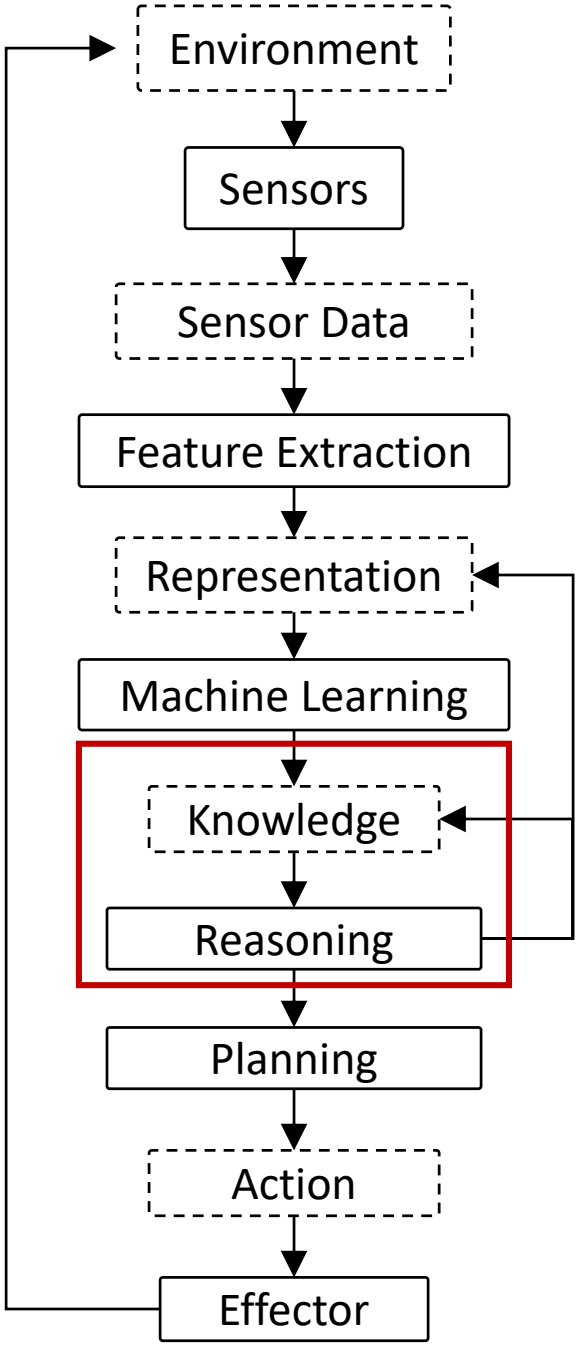
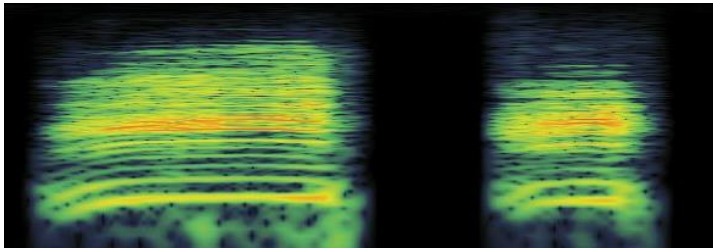
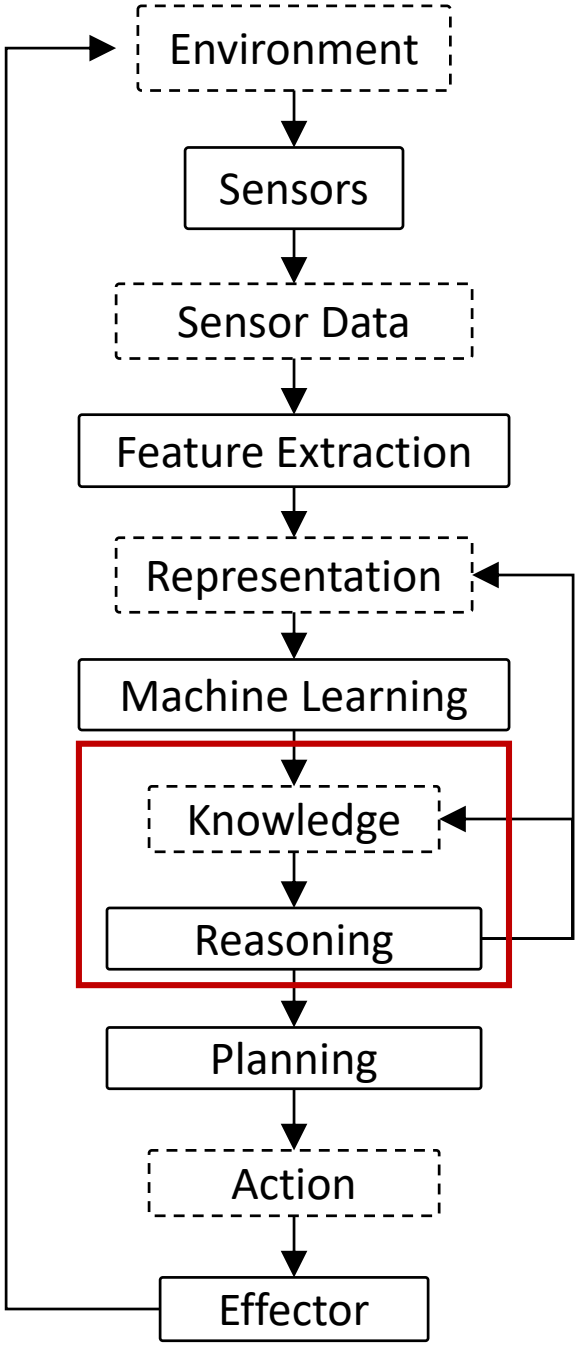


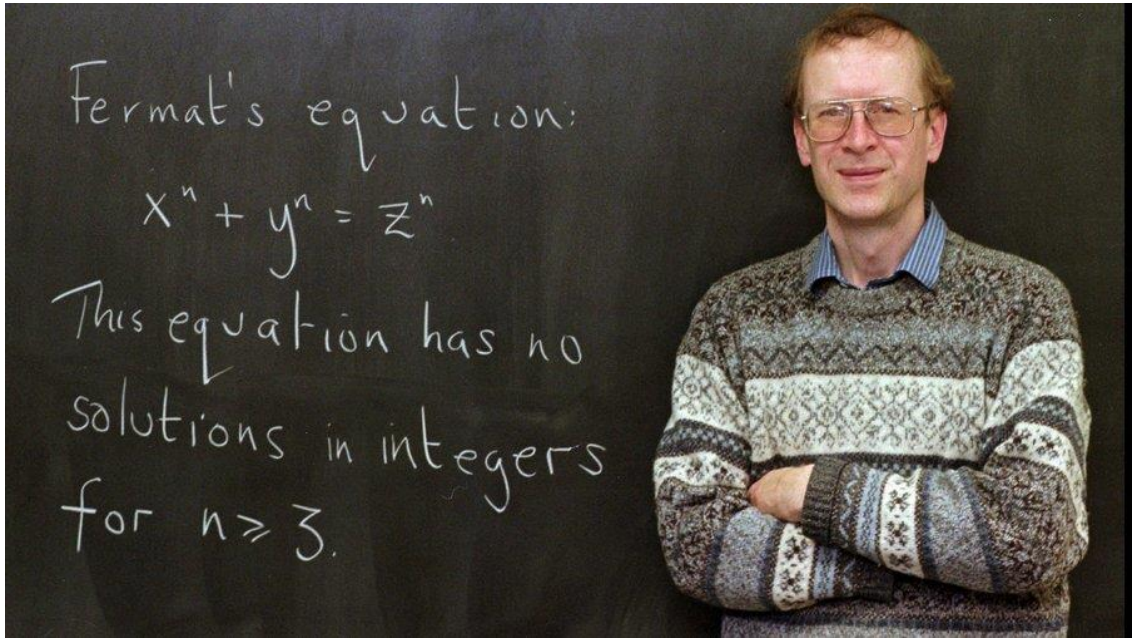
Image Recognition:
If it looks like a duck

Audio Recognition:
Quacks like a duck

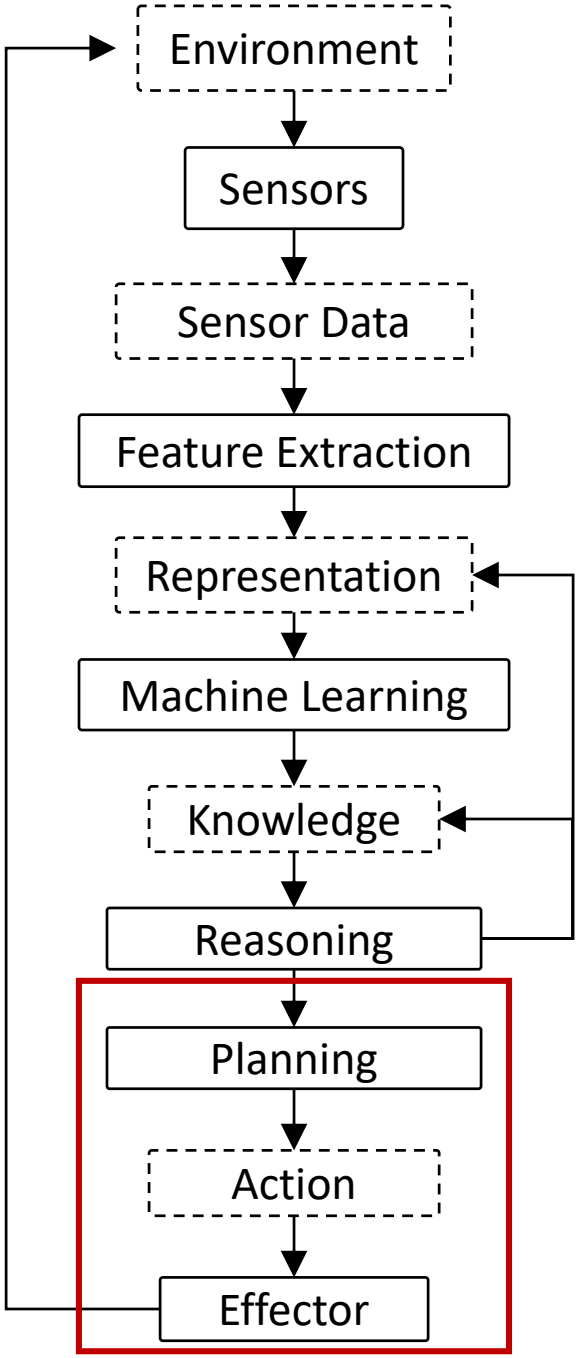


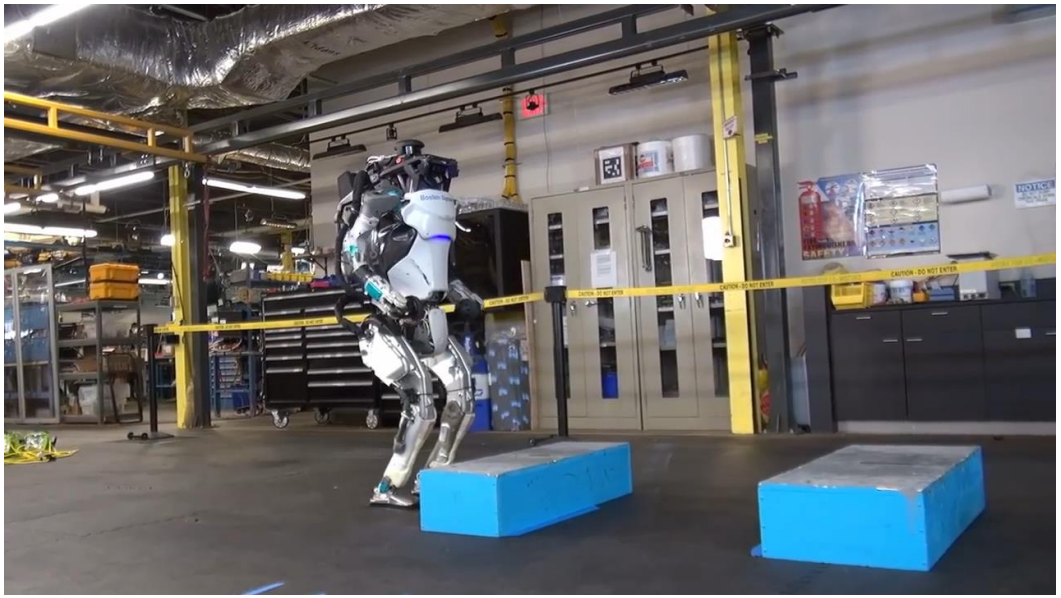
Activity Recognition:
Swims like a duck

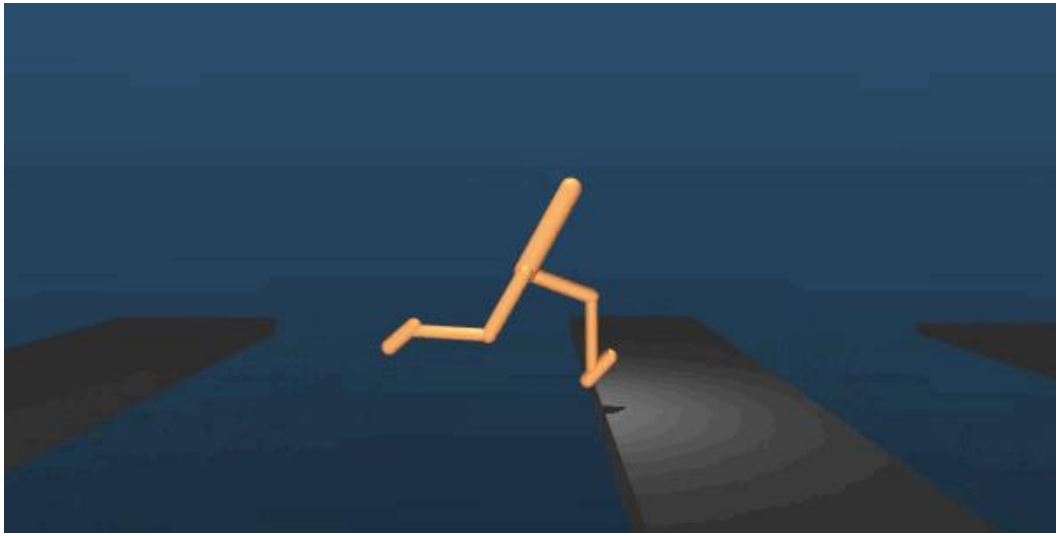


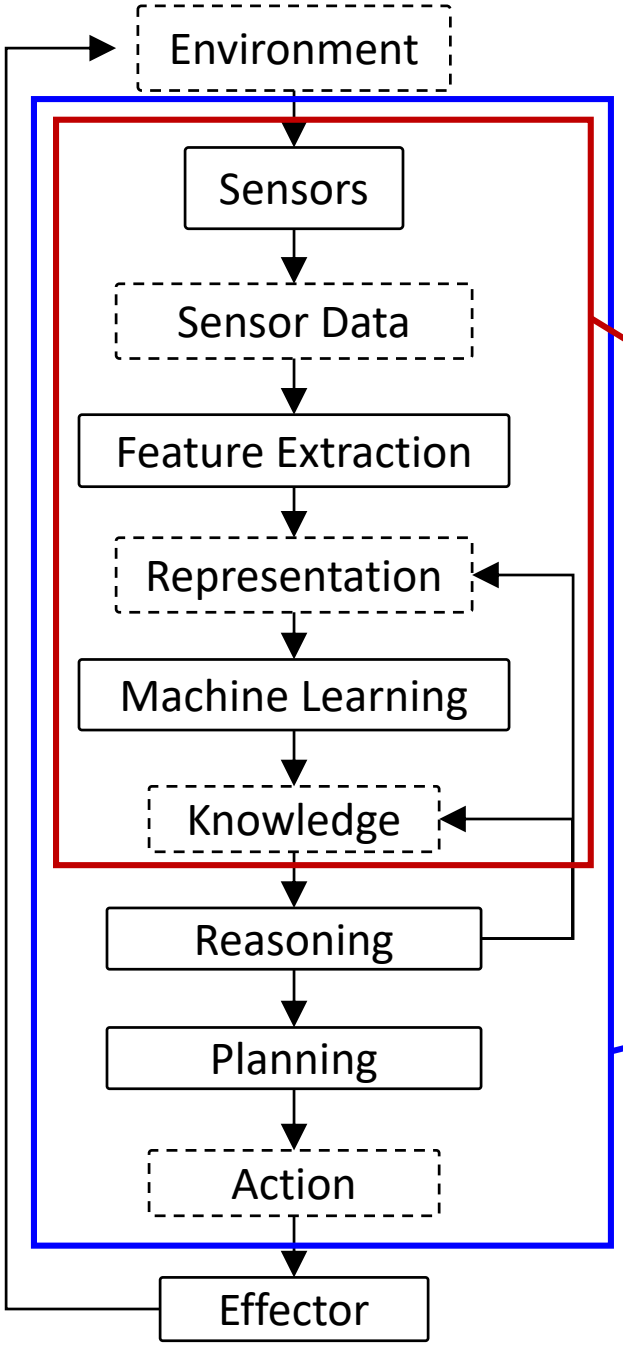


Final **breakthrough**, 358 years after its conjecture:
"It was so indescribably beautiful; it was so simple and so elegant. I couldn't understand how I'd missed it and I just stared at it in disbelief for twenty minutes. Then during the day I walked around the department, and I'd keep coming back to 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







The promise of
Deep Learning

The promise of
Deep Reinforcement Learning

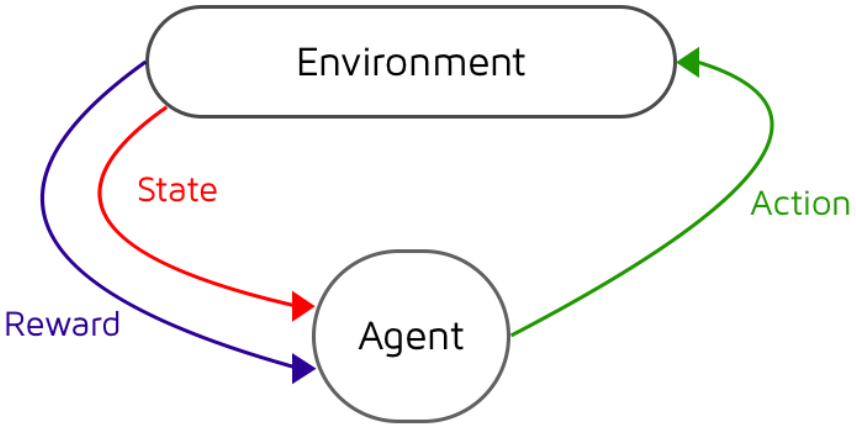
Reinforcement Learning Framework

At each step, the agent:

- Executes action
- Observe new state
- Receive reward

Open Questions:

- What cannot be modeled in this way?
- What are the challenges of learning in this framework?



The Challenge for RL in Real-World Applications

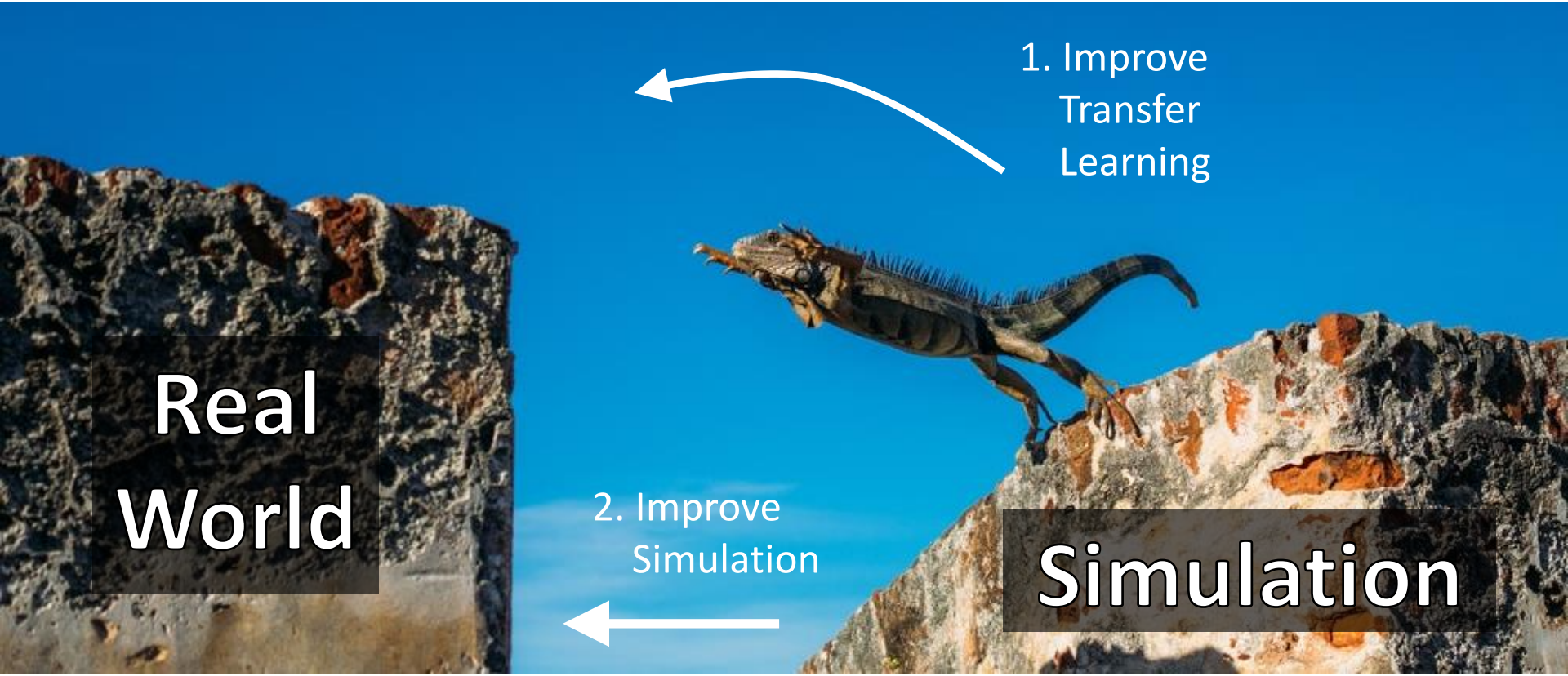
Reminder:

Supervised learning:
teach by example

Reinforcement learning:
teach by experience

Open **Challenges**. Two Options:

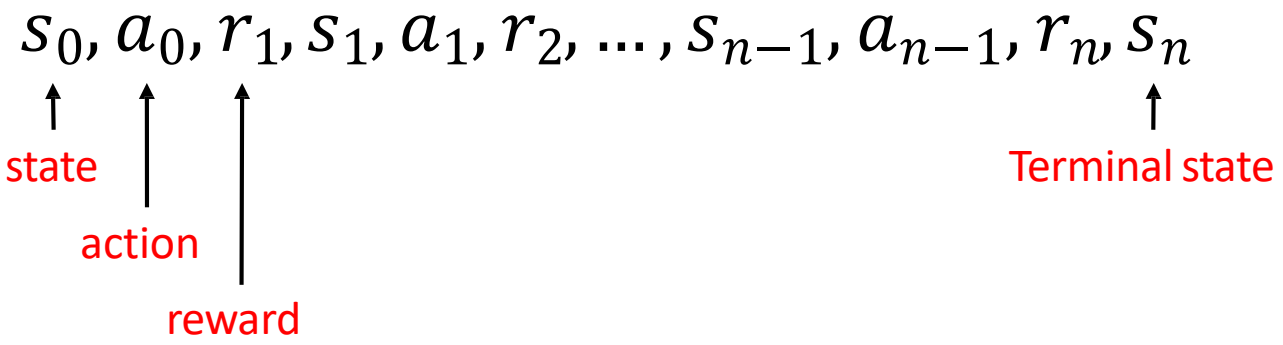
1. Real world observation + one-shot trial & error
2. Realistic simulation + transfer learning



Major Components of an RL Agent

An RL agent may be directly or indirectly trying to learn a:

- **Policy:** agent's behavior function
- **Value function:** how good is each state and/or action
- **Model:** agent's representation of the environment



Meaning of Life for RL Agent: **Maximize Reward**

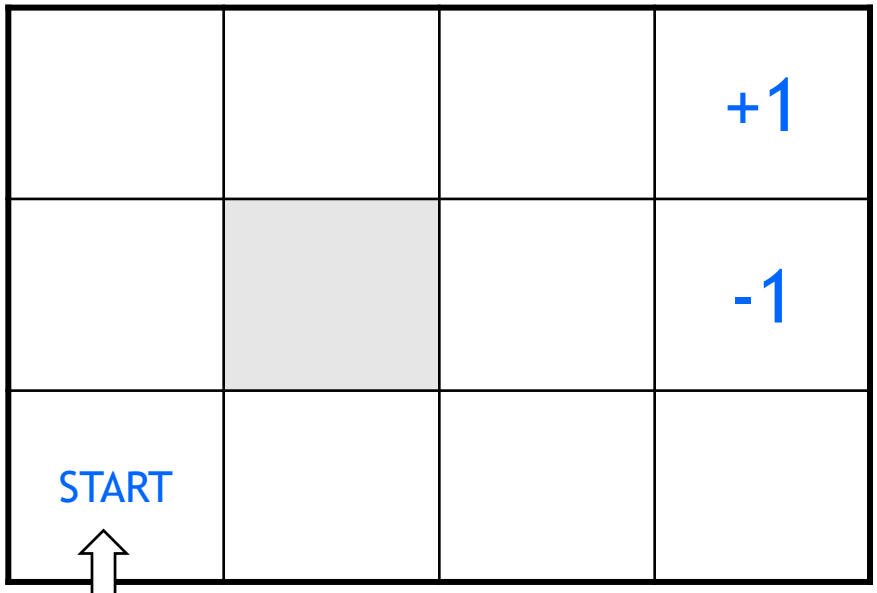
- Future reward: $R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$
- Discounted future reward:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$$

- A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward
- Why "discounted"?
  - Math trick to help analyze convergence
  - Uncertainty due to environment stochasticity, partial observability, or that life can end at any moment:

"If today were the last day of my life, would I want to do what I'm about to do today?" – Steve Jobs

Robot in a Room

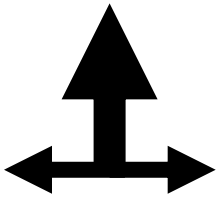


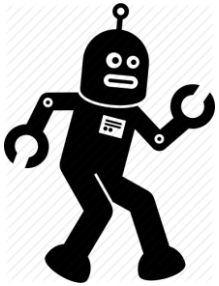
actions: UP, DOWN, LEFT, RIGHT

(Stochastic) model of the world:

Action: **UP**

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

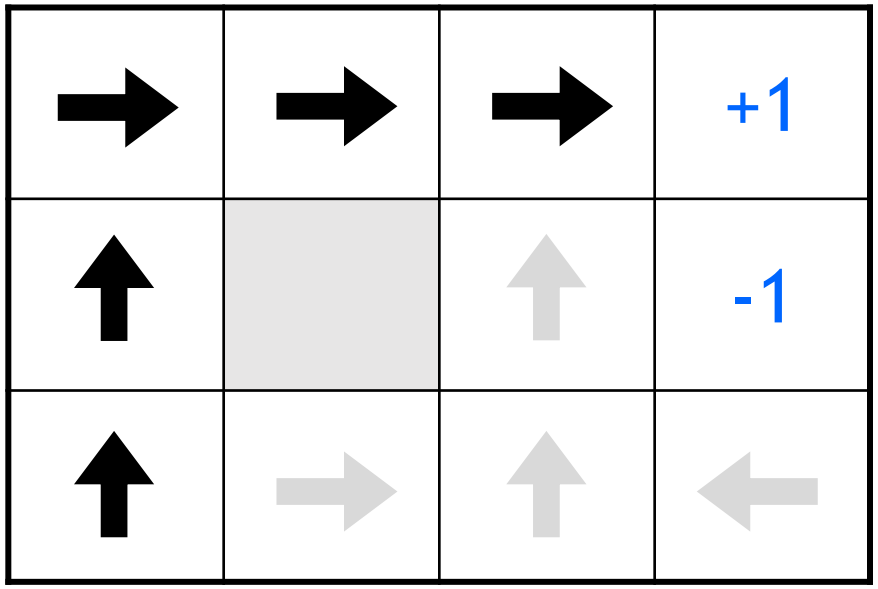




- Reward +1 at [4,3], -1 at [4,2]
- Reward -0.04 for each step
- What's the strategy to achieve max reward?
  - We can learn the model and plan
  - We can learn the value of (action, state) pairs and act greed/non-greedy
  - We can learn the policy directly while sampling from it

Optimal Policy for a Deterministic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are deterministic:

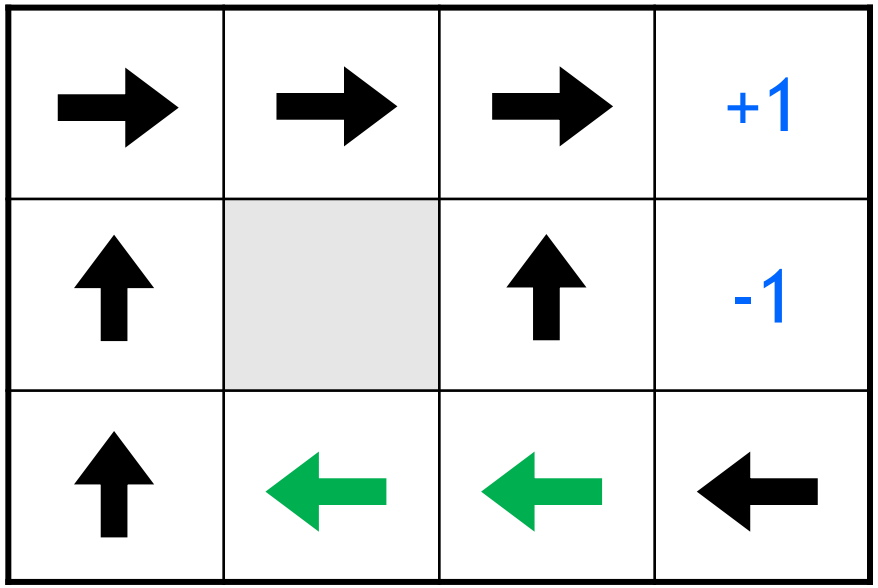
UP

100%	move UP
0%	move LEFT
0%	move RIGHT

Policy: Shortest path.

Optimal Policy for a Stochastic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

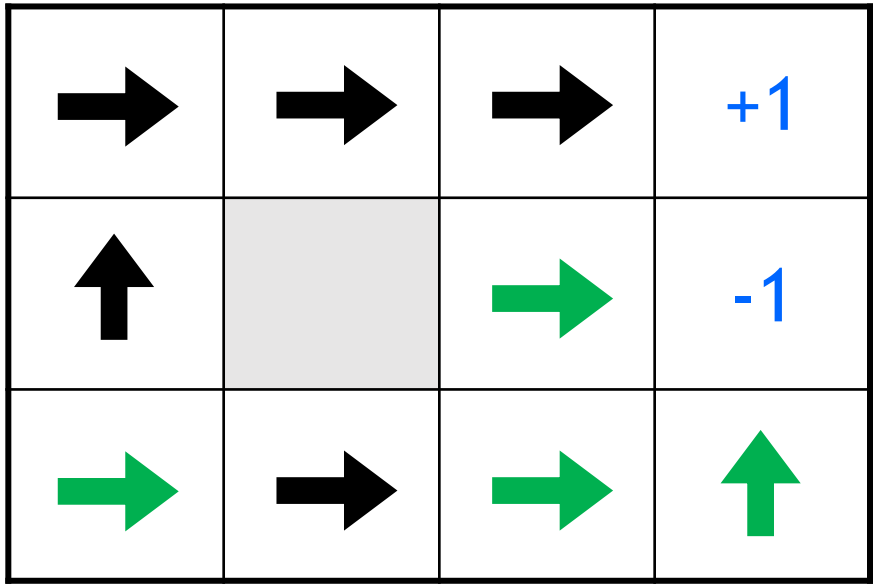
UP

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

Policy: Shortest path. Avoid -UP around -1 square.

Optimal Policy for a Stochastic World

Reward: **-2** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

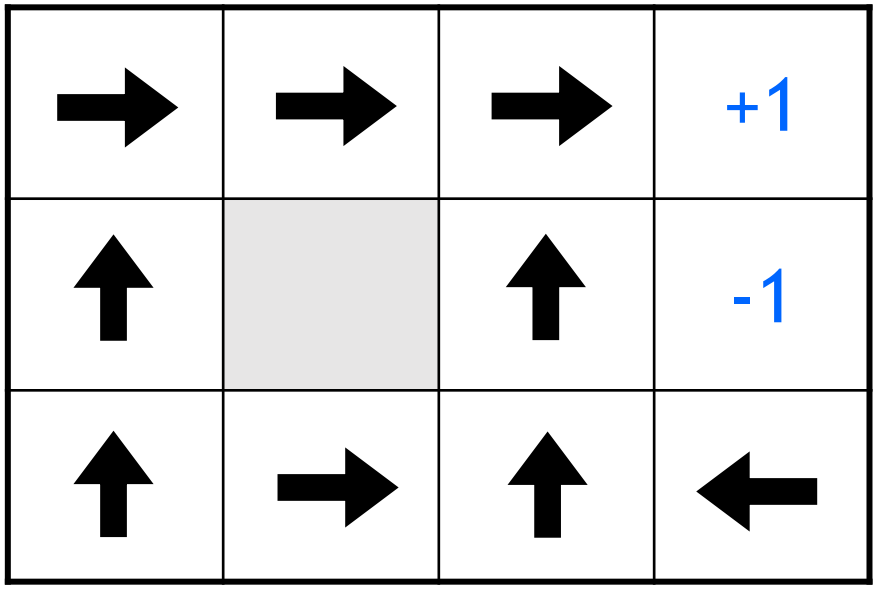
UP

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

Policy: Shortest path.

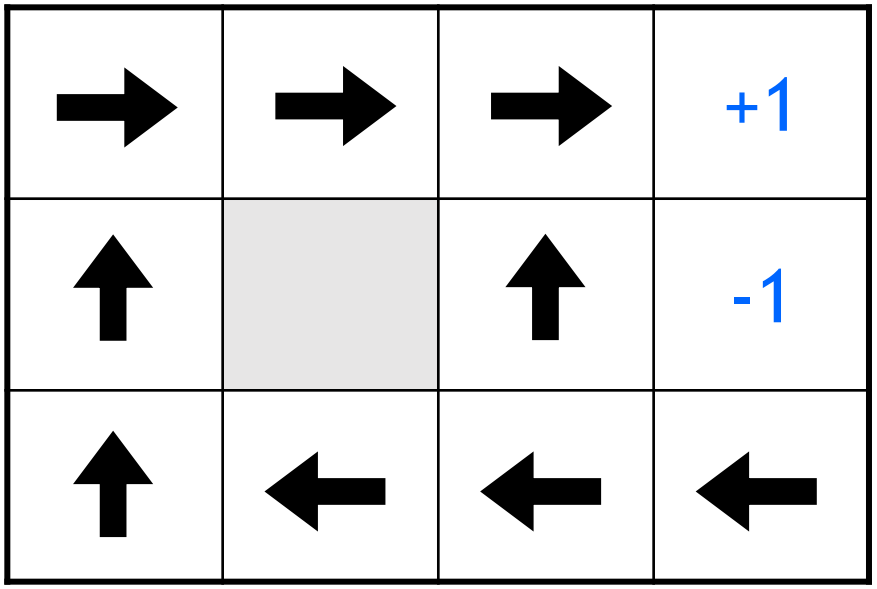
Optimal Policy for a Stochastic World

Reward: **-0.1** for each step



More urgent

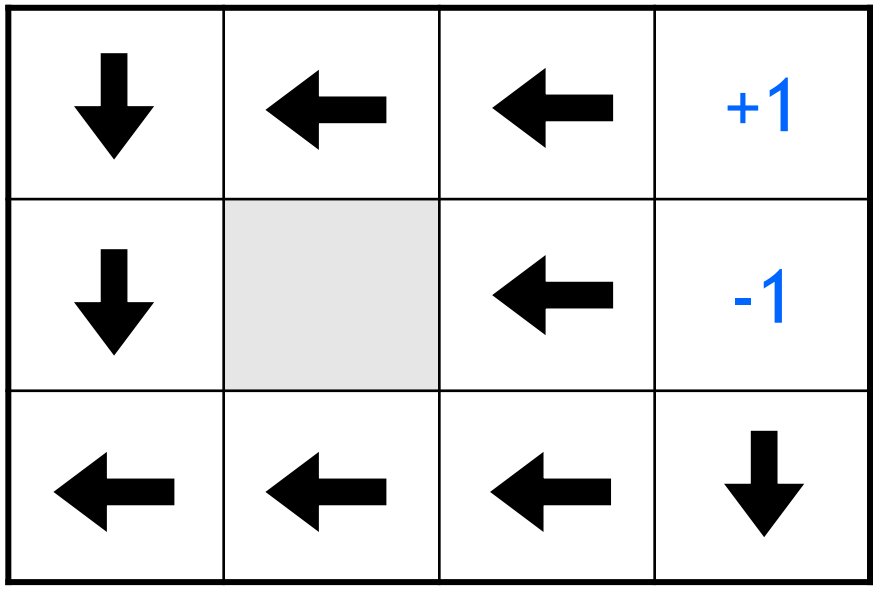
Reward: **-0.04** for each step



Less urgent

Optimal Policy for a Stochastic World

Reward: **+0.01** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

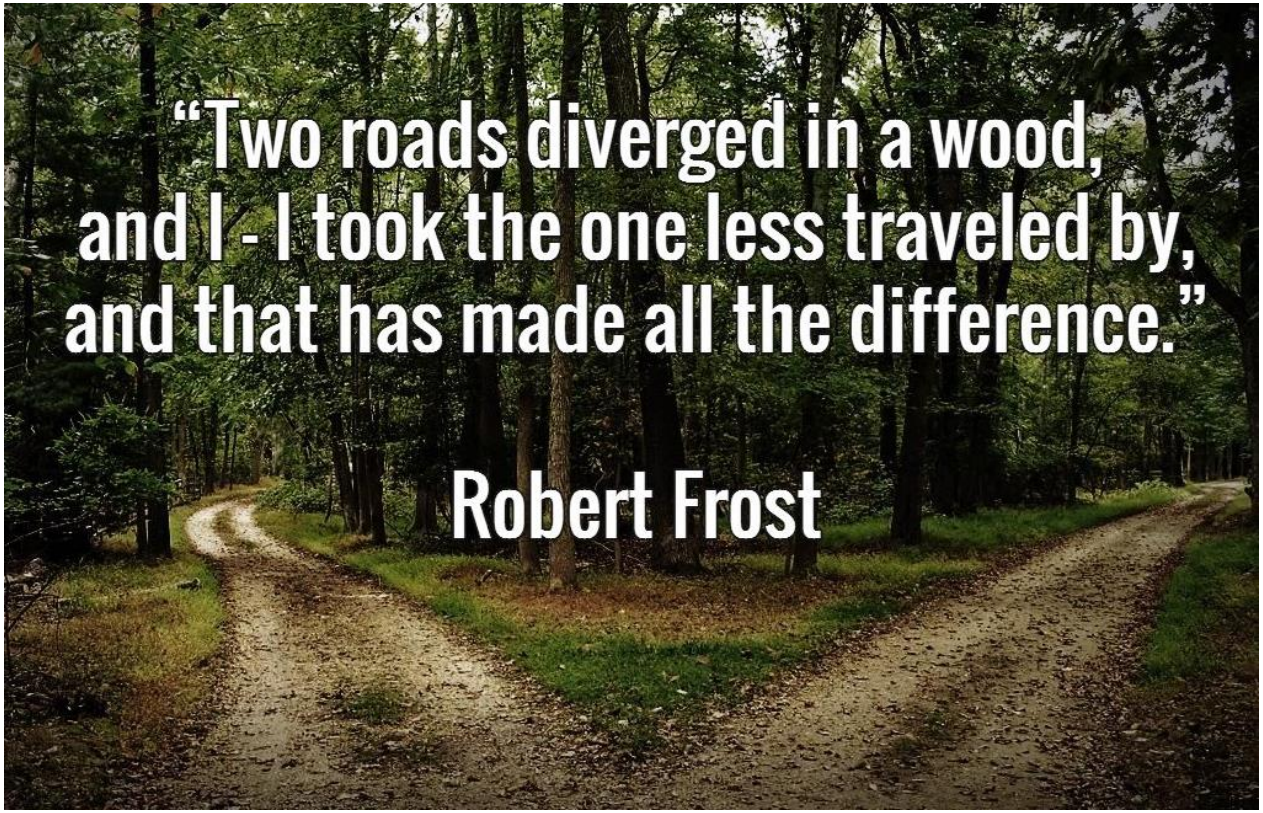
UP

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

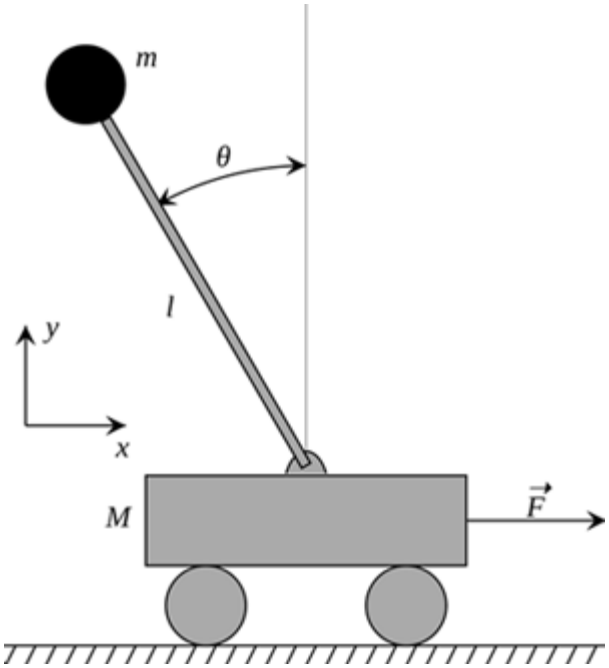
Policy: Longest path.

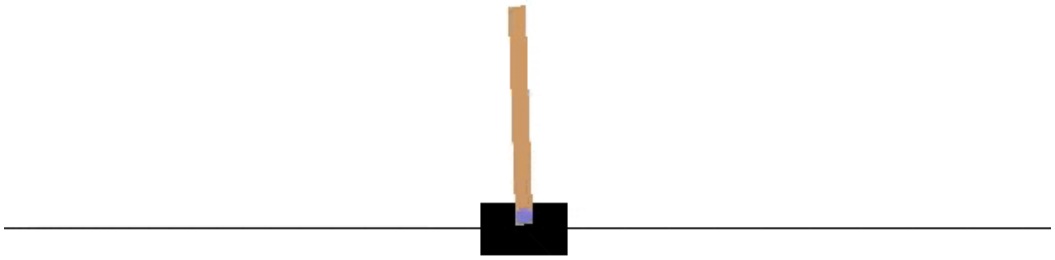
Lessons from Robot in Room

- Environment model has big impact on optimal policy
- Reward structure has big impact on optimal policy



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

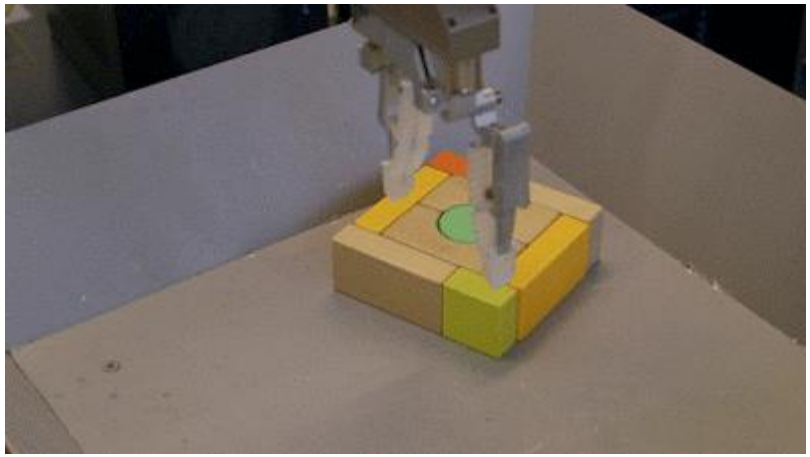
Examples of Reinforcement Learning

Doom*

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

** Added for important thought-provoking considerations of AI safety in the context of autonomous weapons systems (see AGI lectures on the topic).*

Examples of Reinforcement Learning



Grasping Objects with Robotic Arm

- **Goal** - Pick an object of different shapes
- **State** - Raw pixels from camera
- **Actions** – Move arm. Grasp.
- **Reward** - Positive when pickup is successful

Key Takeaways for Real-World Impact

- Deep Learning:
  - **Fun part:** Good algorithms that learn from data.
  - **Hard part:** Good questions, 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.

3 Types of Reinforcement Learning

Model-based

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

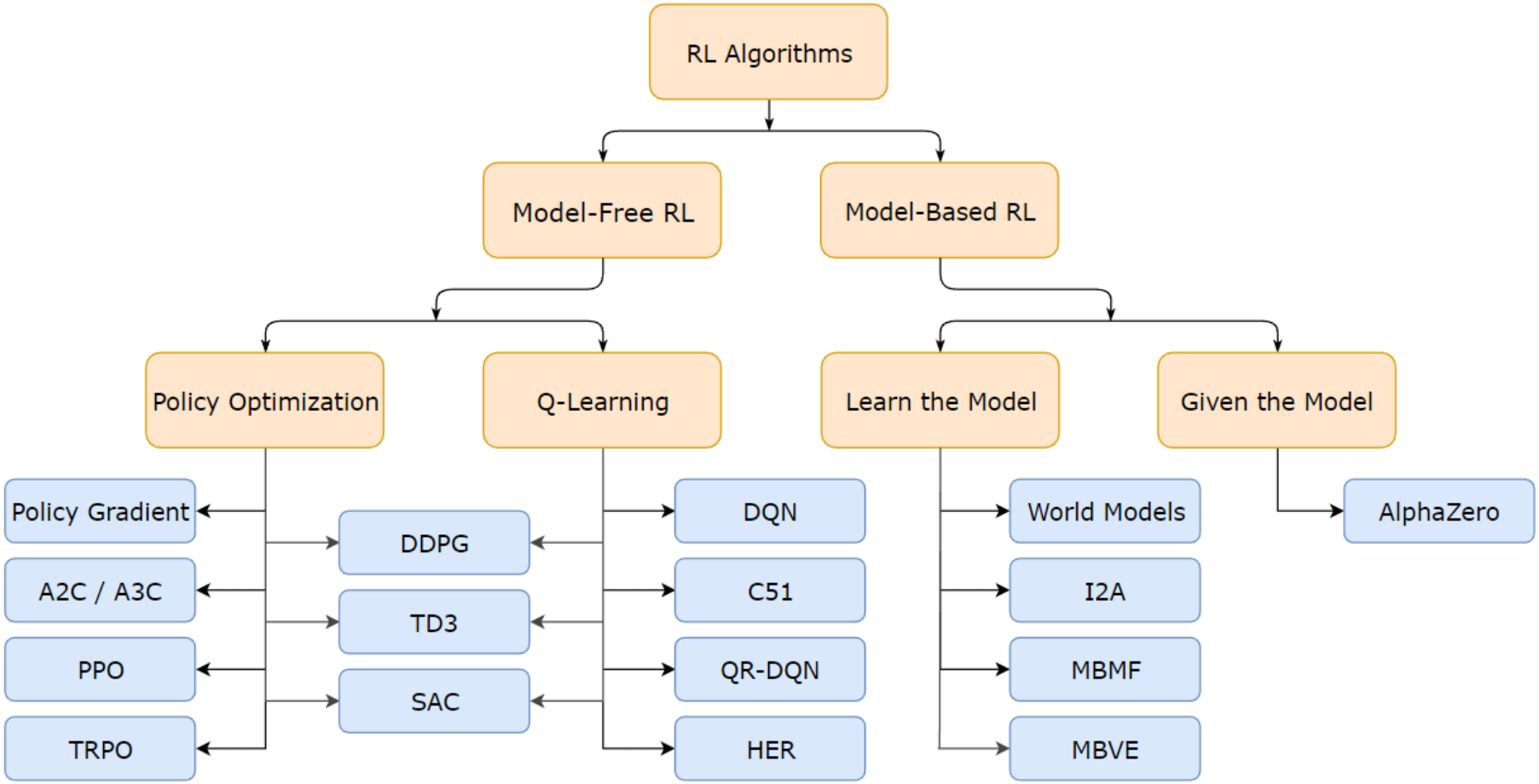
Value-based

- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

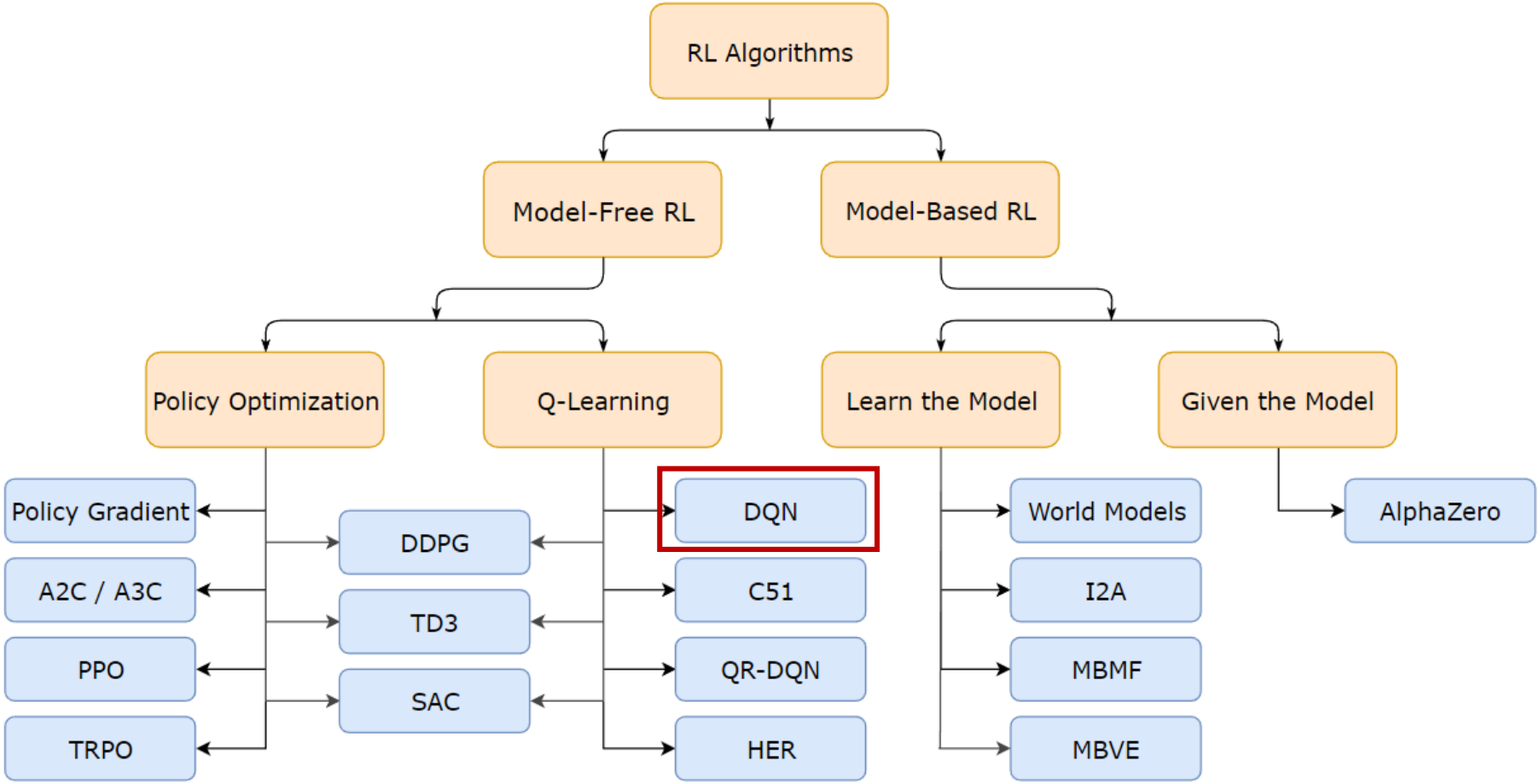
- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Taxonomy of RL Methods

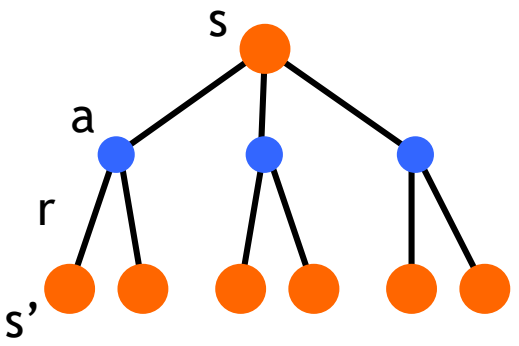


Link: https://spinningup.openai.com

Taxonomy of RL Methods



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

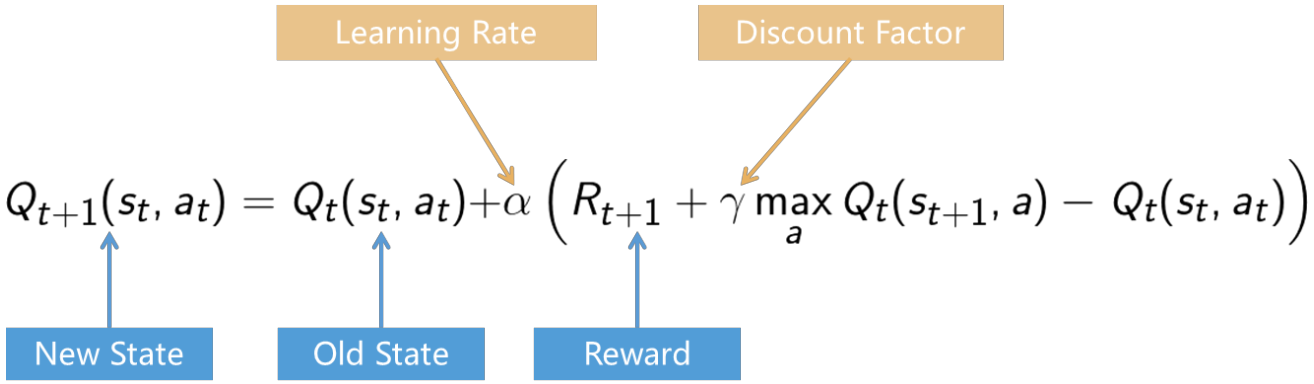
Old State

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



	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

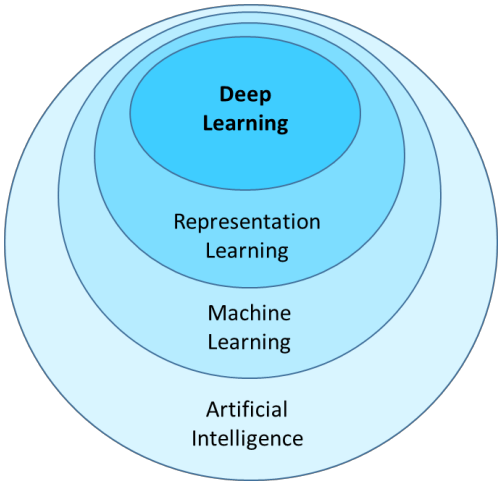
- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $\mathbf{84 \times 84}$ (resized)
    - Consecutive **4** images
    - Grayscale with **256** gray levels

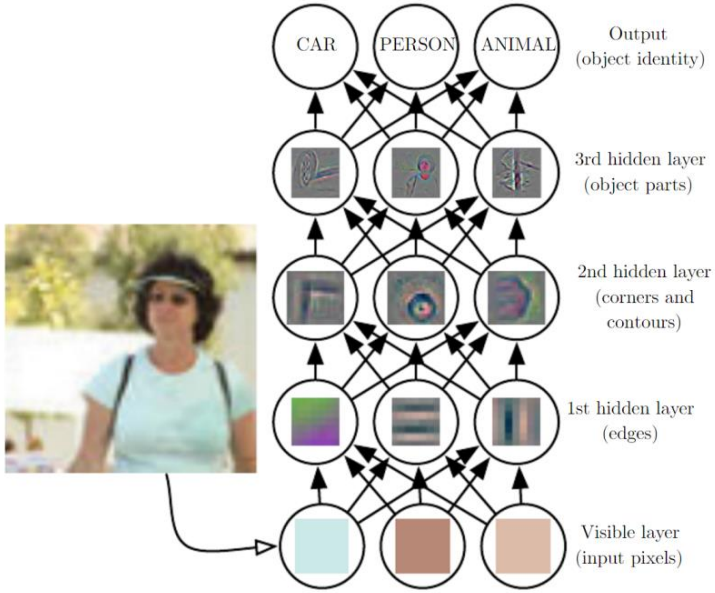
$\mathbf{256^{84 \times 84 \times 4}}$ rows in the Q-table!

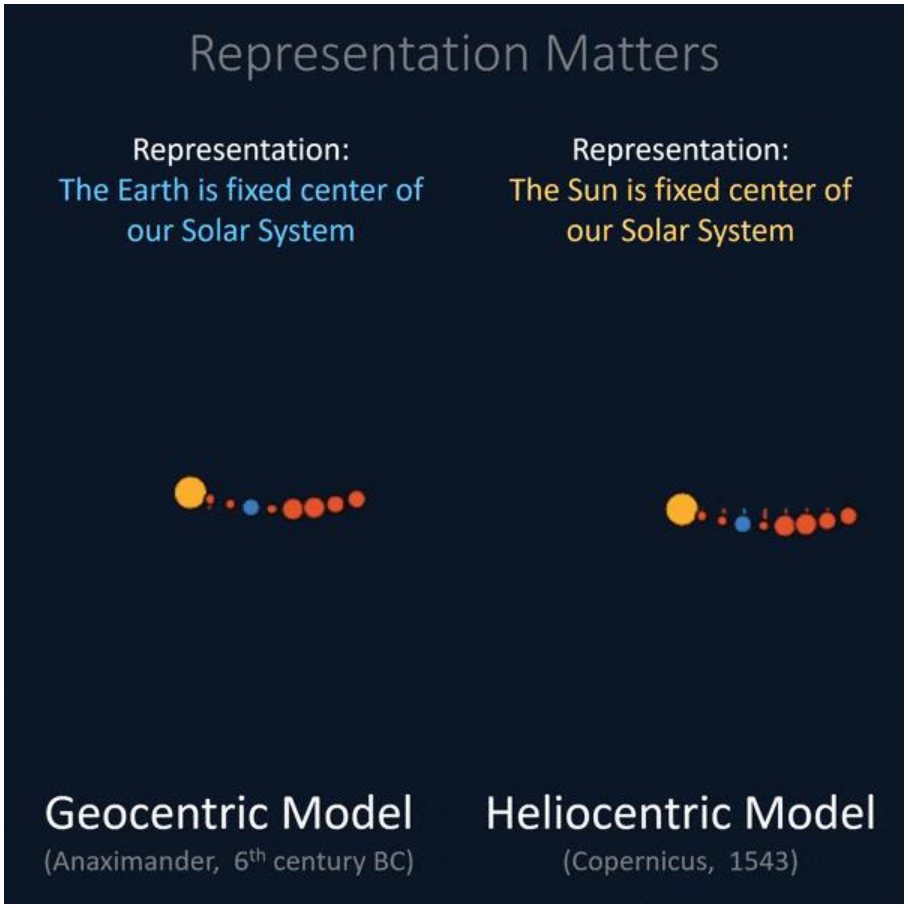
$= 10^{69,970} >> 10^{82}$ atoms in the universe

References: [83, 84]

Deep RL = RL + Neural Networks

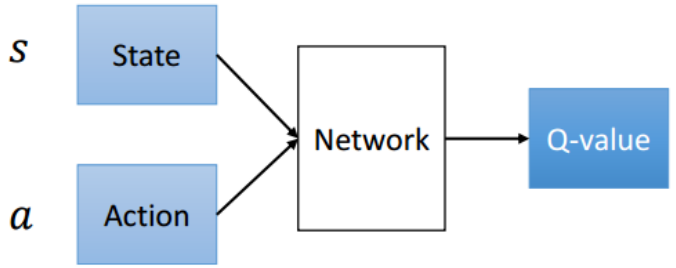




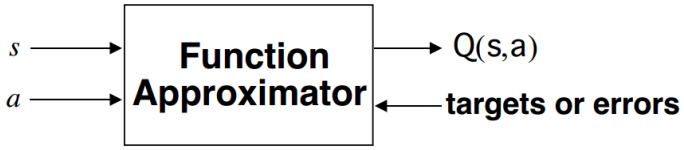


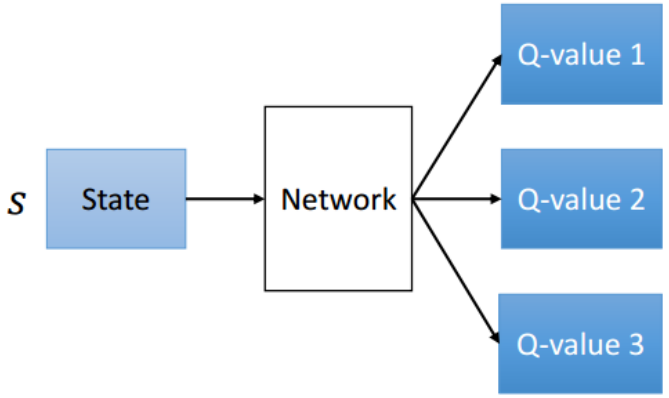
DQN: Deep Q-Learning

Use a neural network to approximate the Q-function:

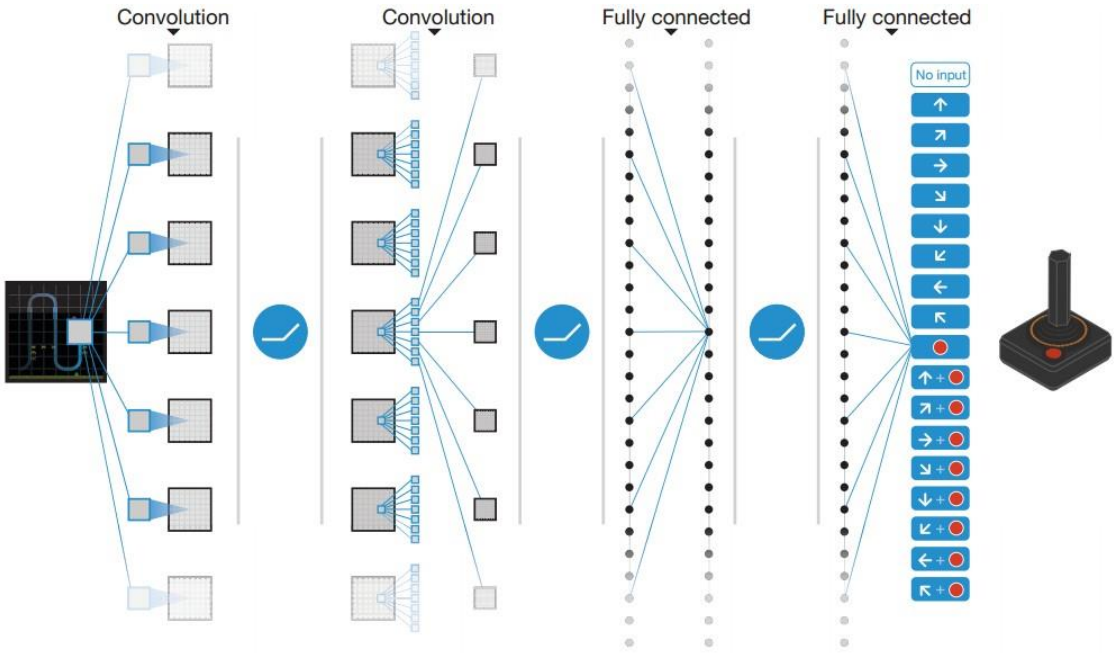


$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$





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

- Loss function (squared error):

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

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

DQN Tricks

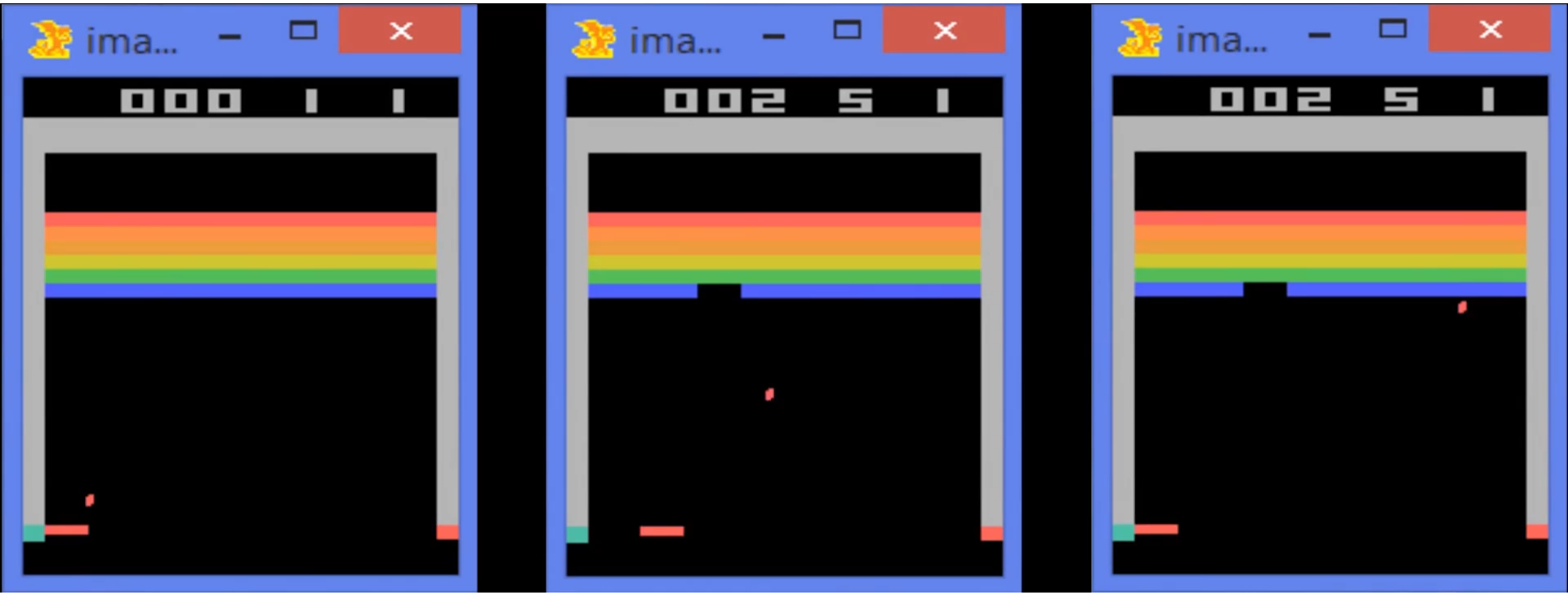
- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000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

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

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

Atari Breakout

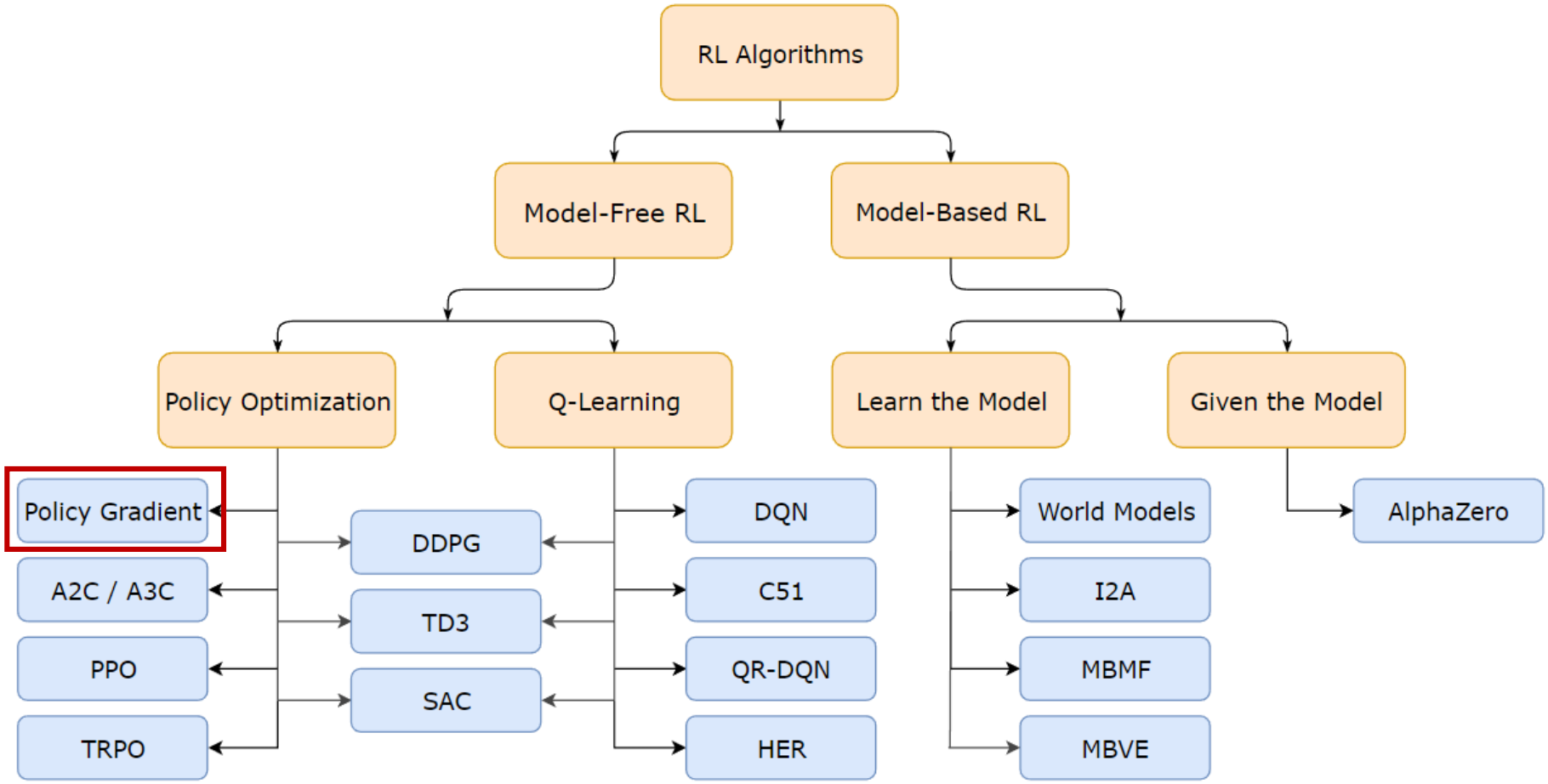


After
10 Minutes
of Training

After
120 Minutes
of Training

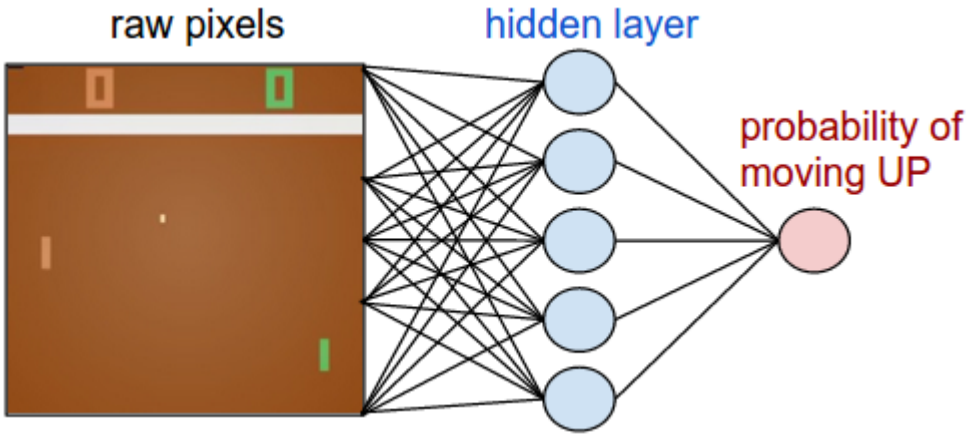
After
240 Minutes
of Training

Taxonomy of RL Methods



Policy Gradient (PG)

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



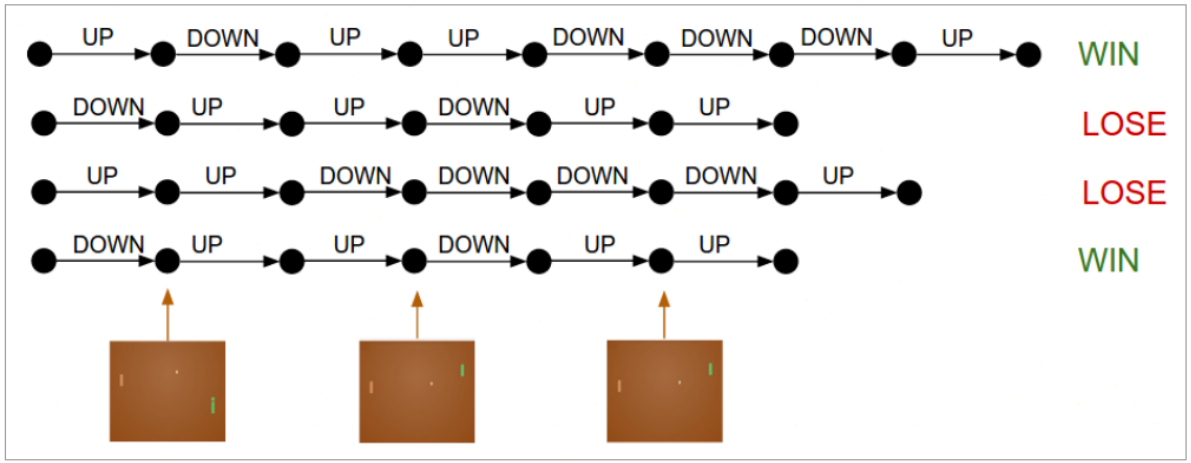
Policy Network

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

"Deep Reinforcement Learning: Pong from Pixels"

Policy Gradient – Training

Policy Gradients: Run a policy for a while. See what actions led to high rewards. Increase their probability.



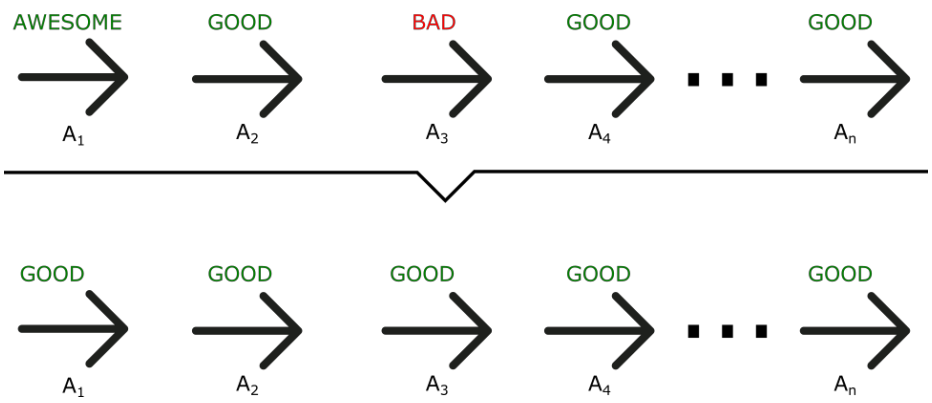
- **REINFORCE:** Policy gradient that increases probability of good actions and decreases probability of bad action:

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

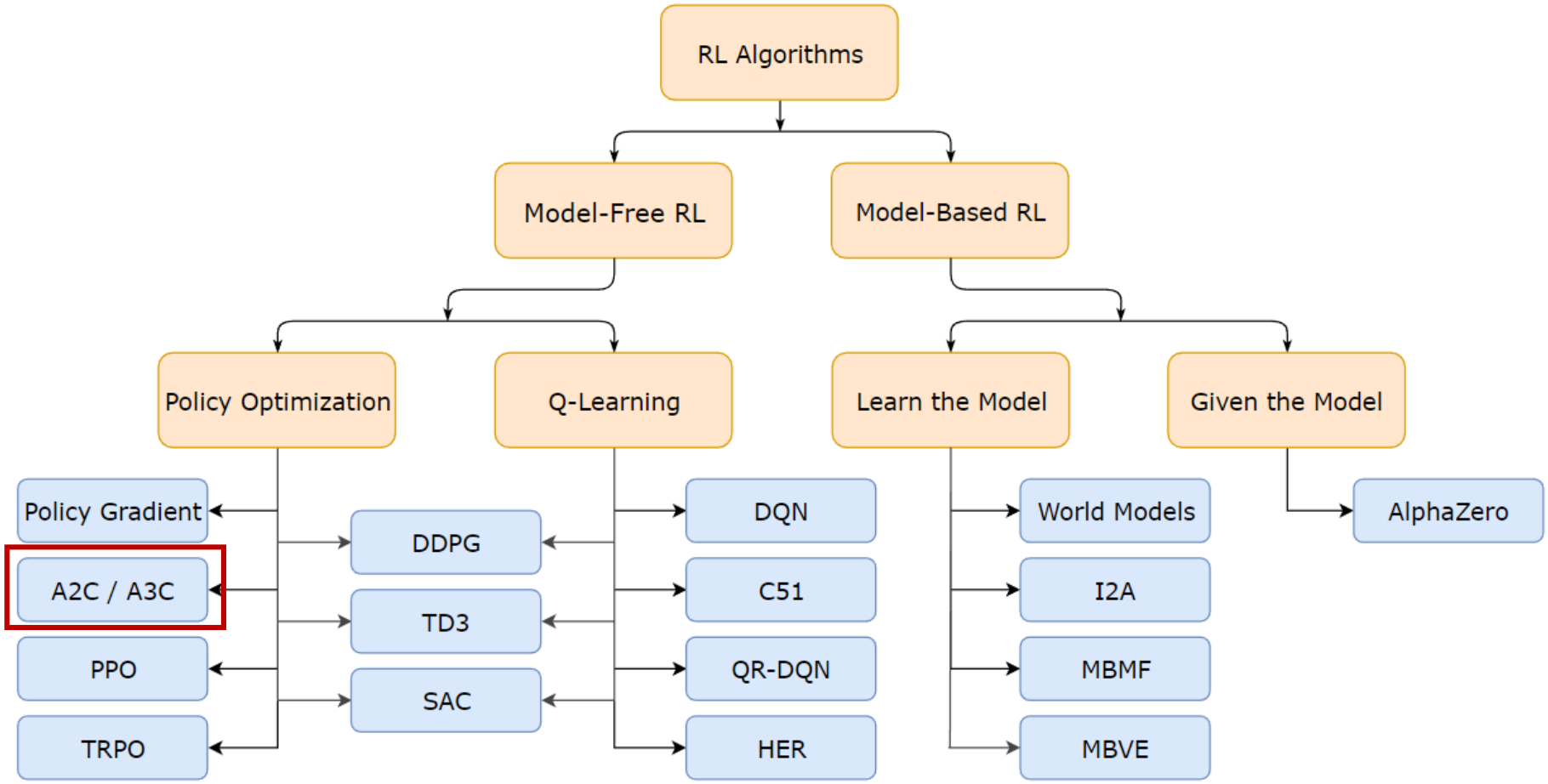
Policy Gradient

- Pros vs DQN:
  - **Messy World:** If Q function is too complex to be learned, DQN may fail miserably, while PG will still learn a good policy.
  - **Speed:** Faster convergence
  - **Stochastic Policies:** Capable of learning stochastic policies - DQN can't
  - **Continuous actions:** Much easier to model continuous action space
- Cons vs DQN:
  - **Data:** Sample inefficient (needs more data)
  - **Stability:** Less stable during training process.
  - Poor **credit assignment** to (state, action) pairs for delayed rewards

Problem with REINFORCE:
Calculating the reward at the end, means all the actions will be averaged as good because the total reward was high.



Taxonomy of RL Methods



Advantage Actor-Critic (A2C)

- Combine DQN (value-based) and REINFORCE (policy-based)
- Two neural networks (Actor and Critic):
  - **Actor** is policy-based: Samples the action from a policy
  - **Critic** is value-based: Measures how good the chosen action is

Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * R(t)$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * Q(S_t, A_t)$$

- Update at each time step - temporal difference (TD) learning

Symbol	Meaning
$s \in \mathcal{S}$	States.
$a \in \mathcal{A}$	Actions.
$r \in \mathcal{R}$	Rewards.
S_t, A_t, R_t	State, action, and reward at time step t of one trajectory. I may occasionally use s_t, a_t, r_t as well.
γ	Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$.
G_t	Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
$P(s', r \vert s, a)$	Transition probability of getting to the next state s' from the current state s with action a and reward r.
$\pi(a \vert s)$	Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by θ.
$\mu(s)$	Deterministic policy; we can also label this as $\pi(s)$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either π or μ is what a reinforcement learning algorithm aims to learn.
$V(s)$	State-value function measures the expected return of state s; $V_w(.)$ is a value function parameterized by w.
$V^\pi(s)$	The value of state s when we follow a policy π; $V^\pi(s) = \mathbb{E}_{a \sim \pi}[G_t \vert S_t = s]$.
$Q(s, a)$	Action-value function is similar to $V(s)$, but it assesses the expected return of a pair of state and action (s, a); $Q_w(.)$ is a action value function parameterized by w.
$Q^\pi(s, a)$	Similar to $V^\pi(.)$, the value of (state, action) pair when we follow a policy π; $Q^\pi(s, a) = \mathbb{E}_{a \sim \pi}[G_t \vert S_t = s, A_t = a]$.

Policy Gradient Theorem

The reward function is defined as:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a)$$

where $d^\pi(s)$ is the stationary distribution of Markov chain for π_θ (on-policy state distribution under

$$\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s)] && ; \text{Because } (\ln x)' = 1/x
\end{aligned}$$

REINFORCE (linear approach)

The process is pretty straightforward:

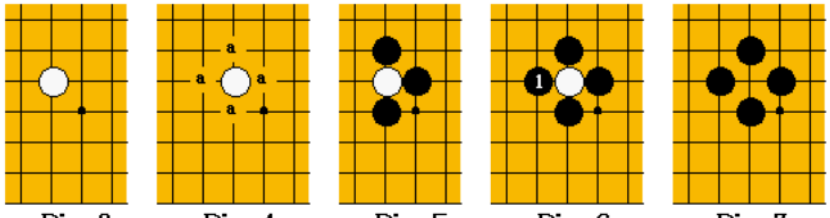
1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_θ: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$.
3. For t=1, 2, ... , T:
    1. Estimate the the return G_t;
    2. Update policy parameters: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

Источники

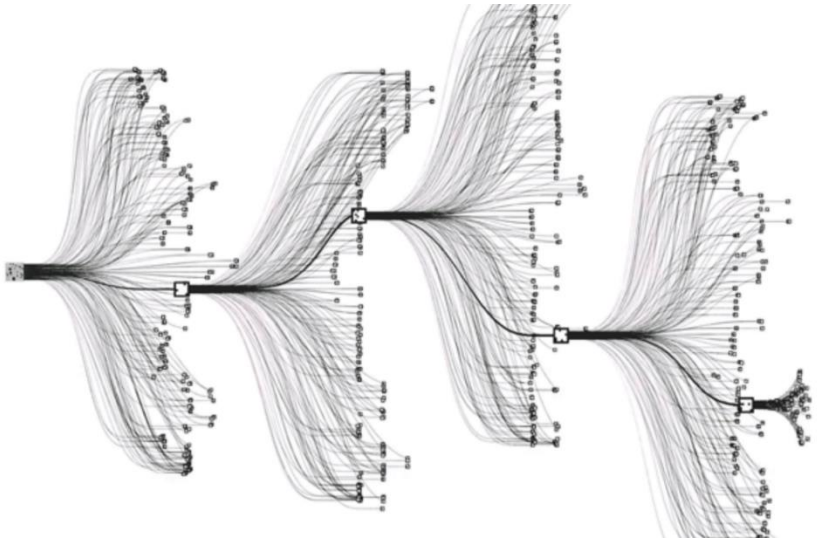
- Лекции Лекса Фридмана (философия и обзорные state of the art рассказы) — https://deeplearning.mit.edu/

- Заметки Lilian Weng по поводу Policy Gradient (хардкорная математика)— https://lilianweng.github.io/posts/2018-04-08-policy-gradient/#reinforce

Game of Go

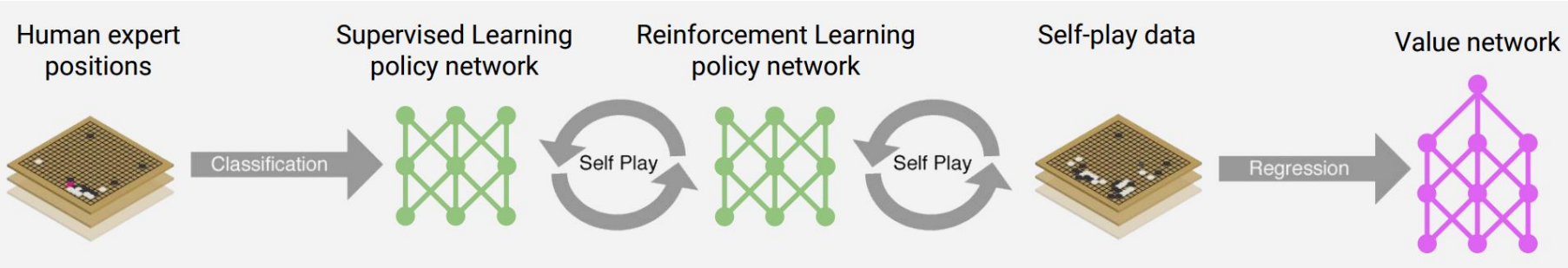


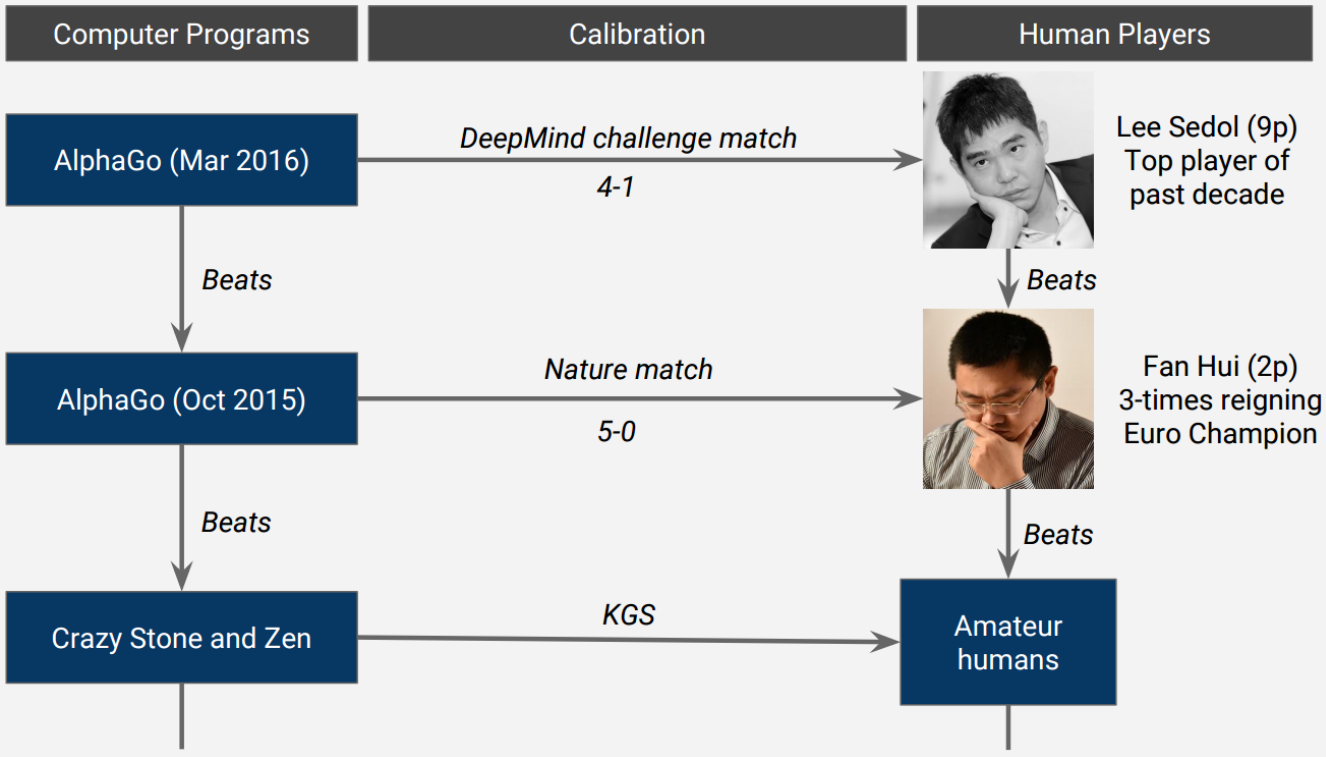
Dia. 3 | Dia. 4 liberties | Dia. 5 atari | Dia. 6 capture | Dia. 7 result



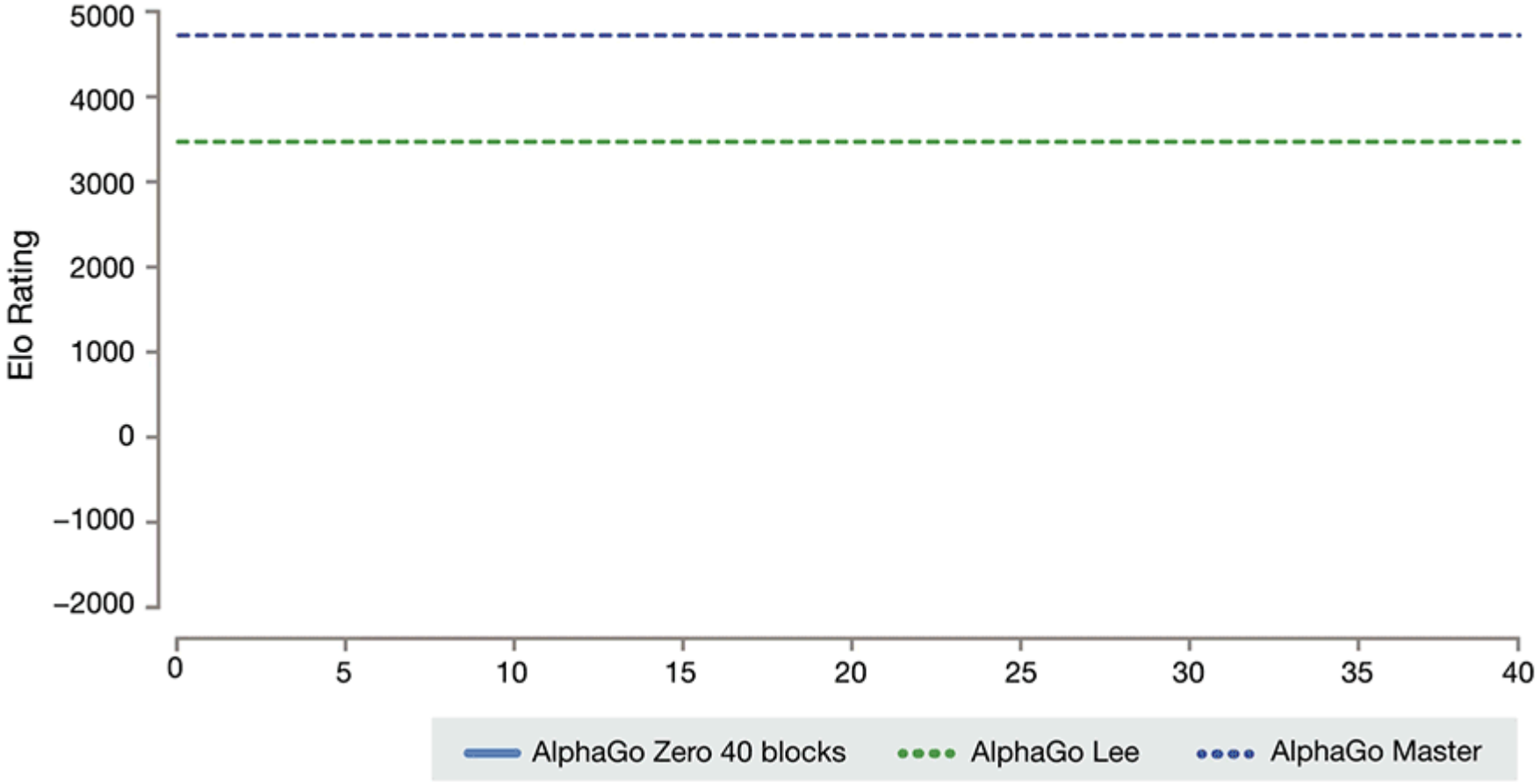
Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}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

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



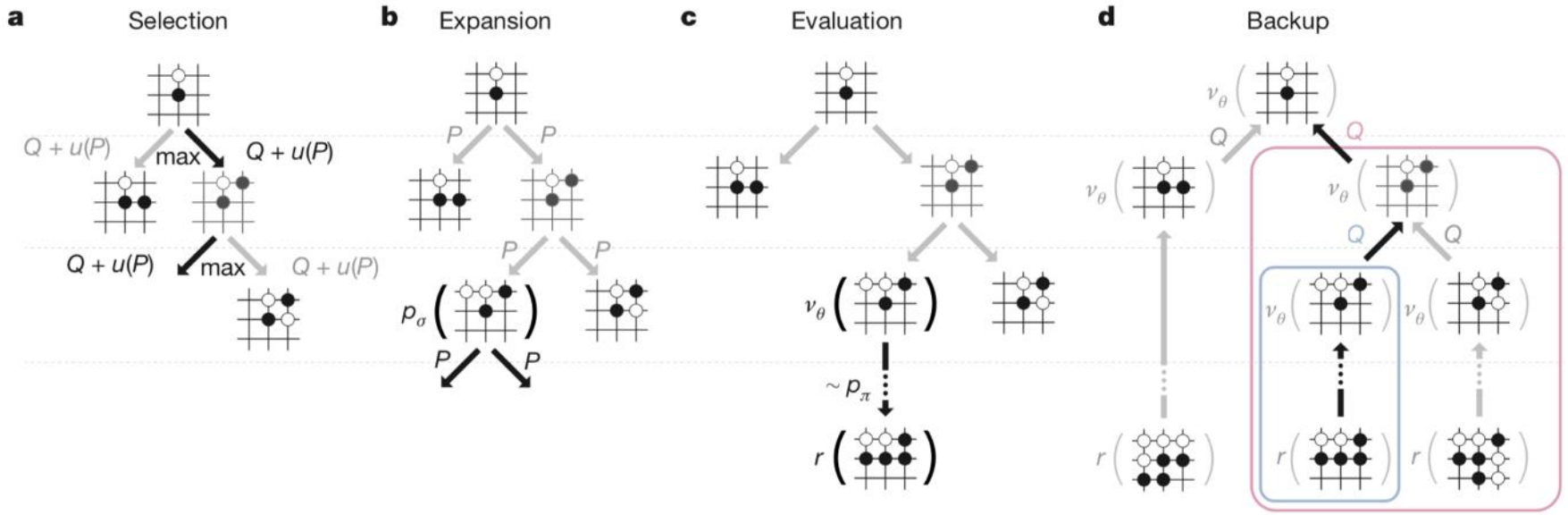


AlphaGo Zero (2017): Beats AlphaGo



AlphaGo Zero Approach

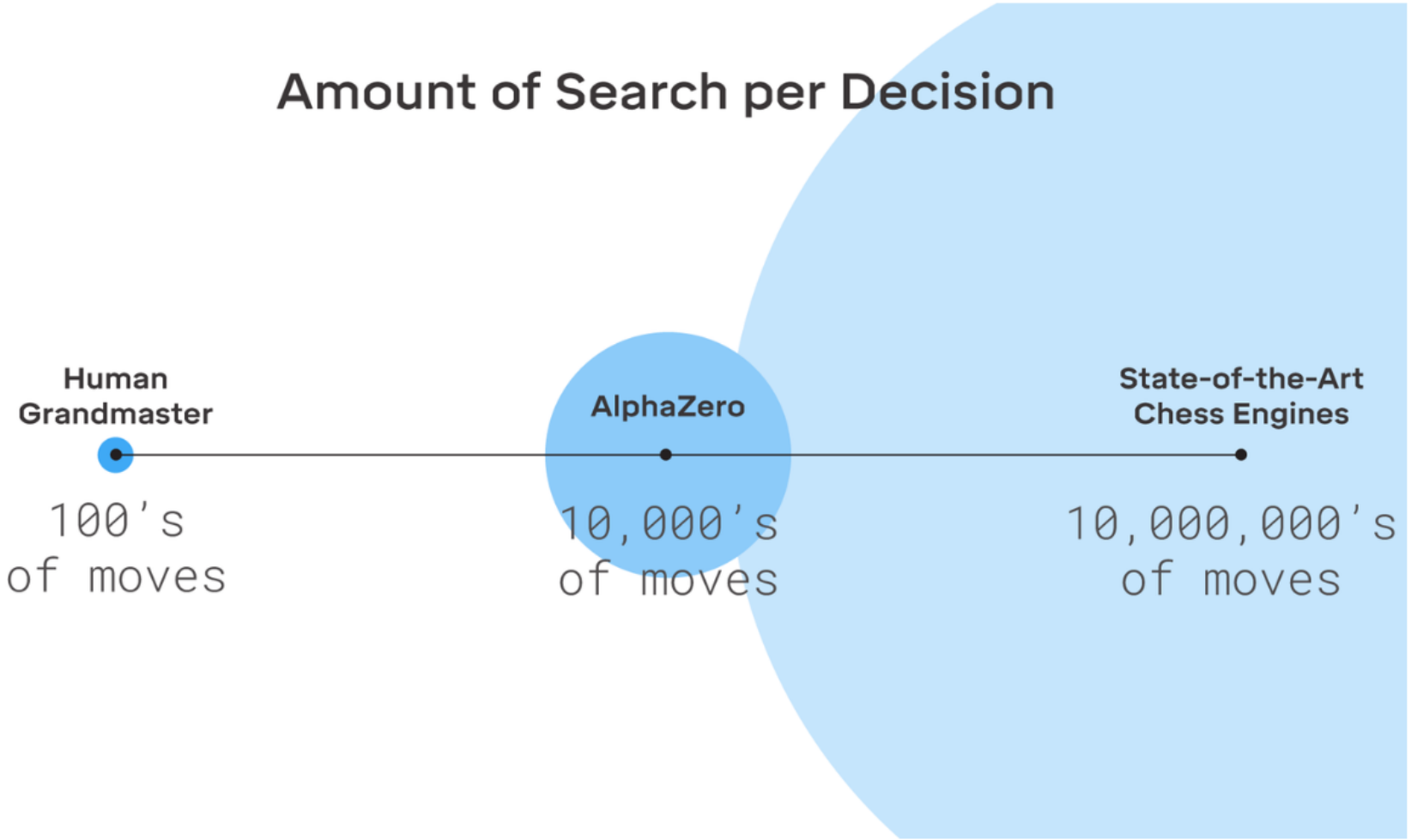
- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)



AlphaGo Zero Approach

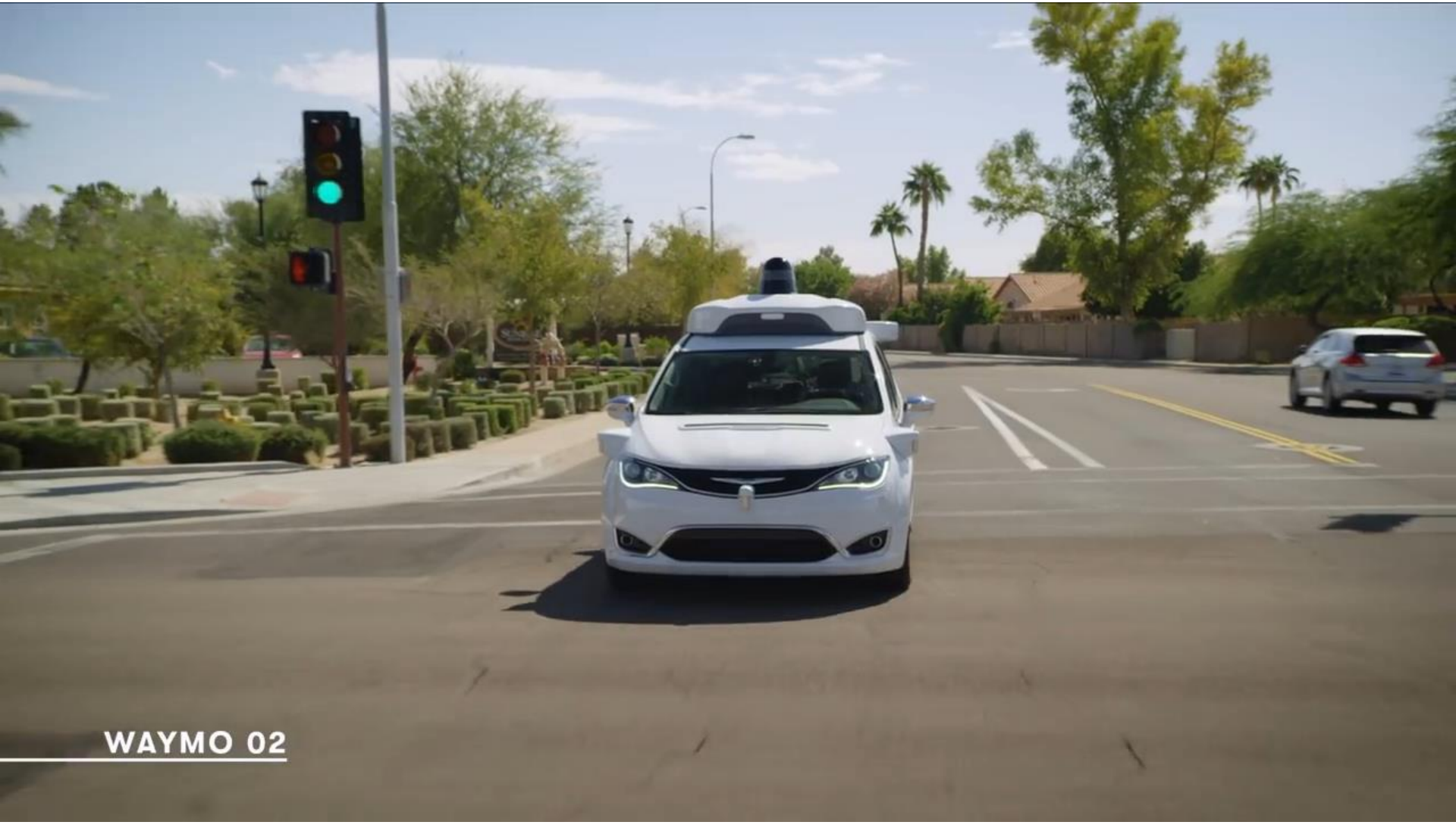
- Same as the best before: Monte Carlo Tree Search (MCTS)
  - Balance exploitation/exploration (going deep on promising positions or exploring new underplayed positions)
- Use a neural network as "intuition" for which positions to expand as part of MCTS (same as AlphaGo)
- "Tricks"
  - Use MCTS intelligent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

AlphaZero vs Chess, Shogi, Go



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

Deep RL is not involved

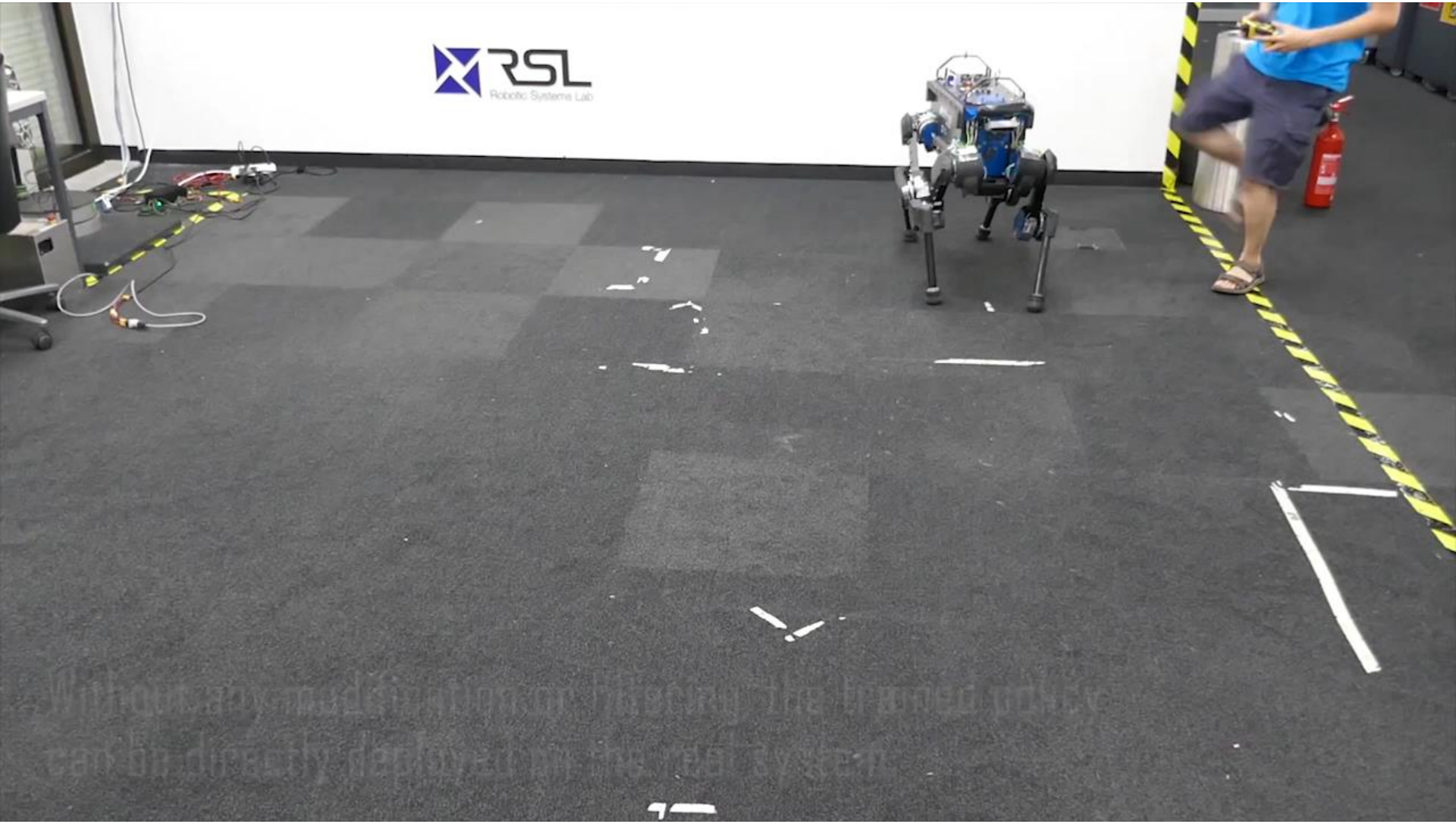


To date, for most successful robots operating in the real world:

Deep RL is not involved

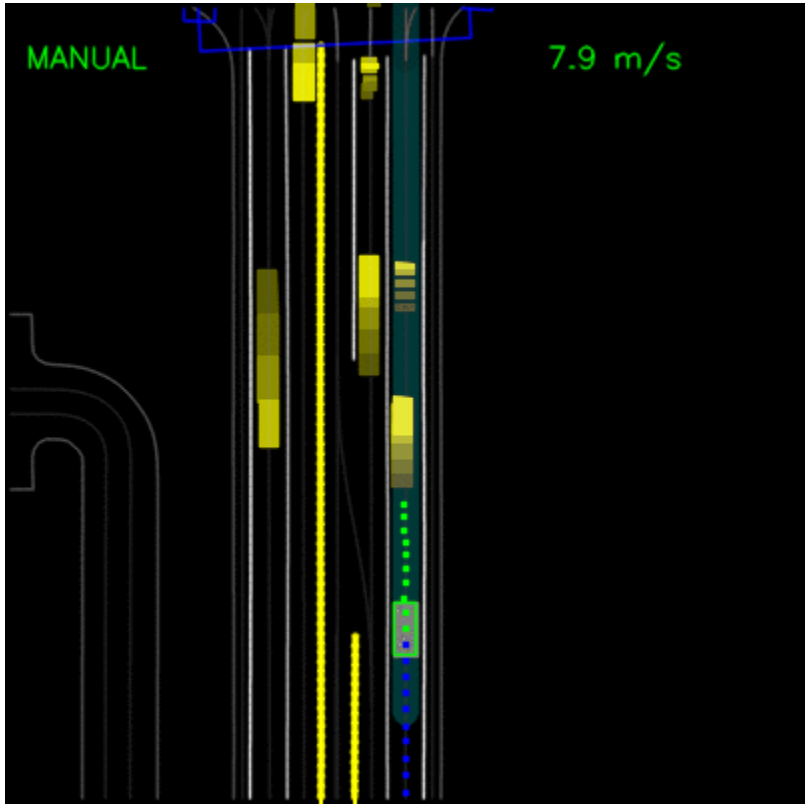
But… that's slowly changing:

Learning Control Dynamics



But… that's slowly changing:

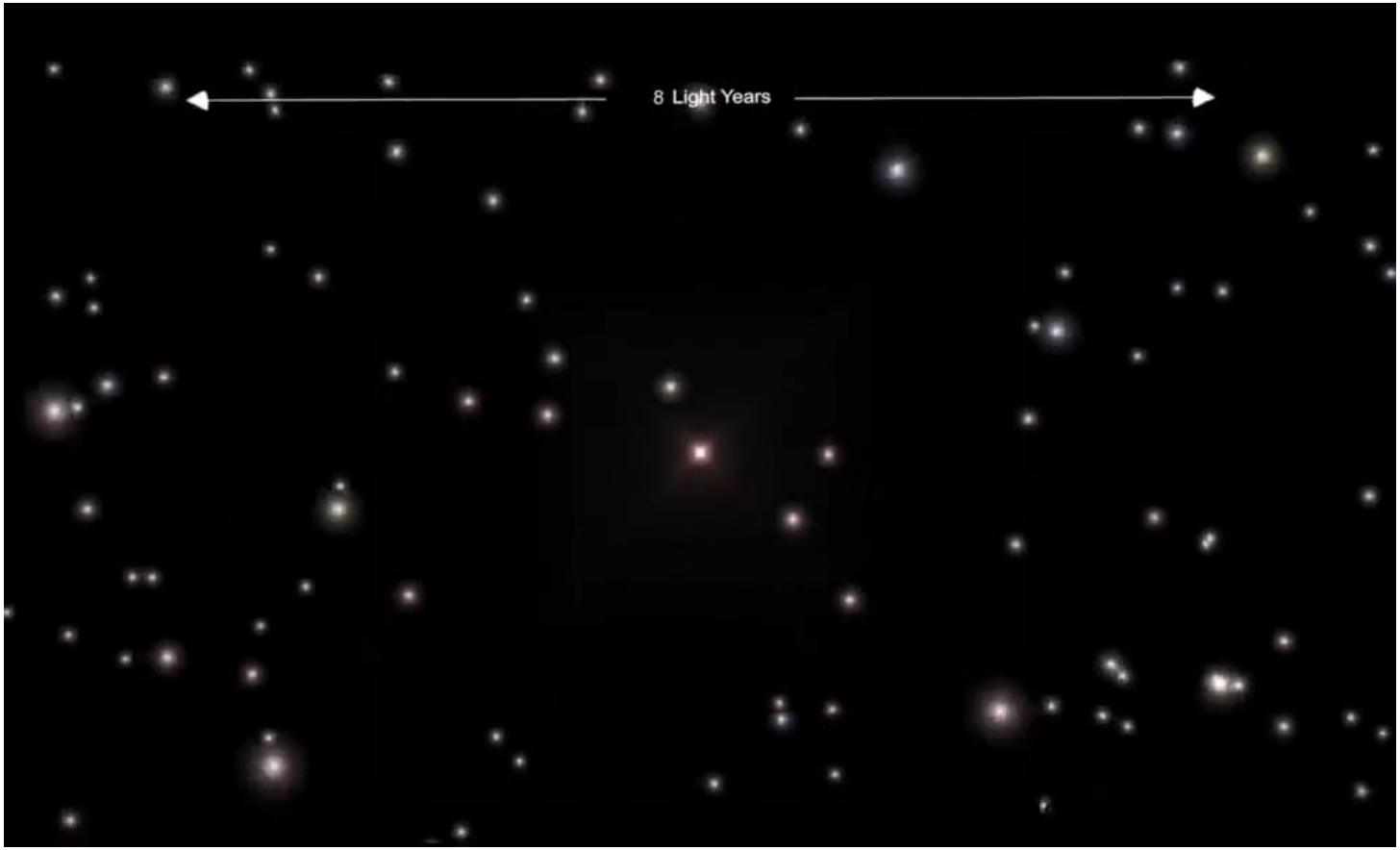
Learning to Drive: Beyond Pure Imitation (Waymo)



Thinking Outside the Box:

Multiverse Theory and the Simulation Hypothesis

- Create an (infinite) set of simulation environments to learn in so that our reality becomes just another sample from the set.



Next Steps in Deep RL

- Lectures: https://deeplearning.mit.edu
- Tutorials: https://github.com/lexfridman/mit-deep-learning
- Advice for Research (from Spinning Up as a Deep RL Researcher by Joshua Achiam)
  - Background
    - Fundamentals in probability, statistics, multivariate calculus.
    - Deep learning basics
    - Deep RL basics
    - TensorFlow (or PyTorch)
  - Learn by doing
    - Implement core deep RL algorithms (discussed today)
    - Look for tricks and details in papers that were key to get it to work
    - Iterate fast in simple environments (see Einstein quote on simplicity)
  - Research
    - Improve on an existing approach
    - Focus on an unsolved task / benchmark
    - Create a new task / problem that hasn't been addressed with RL