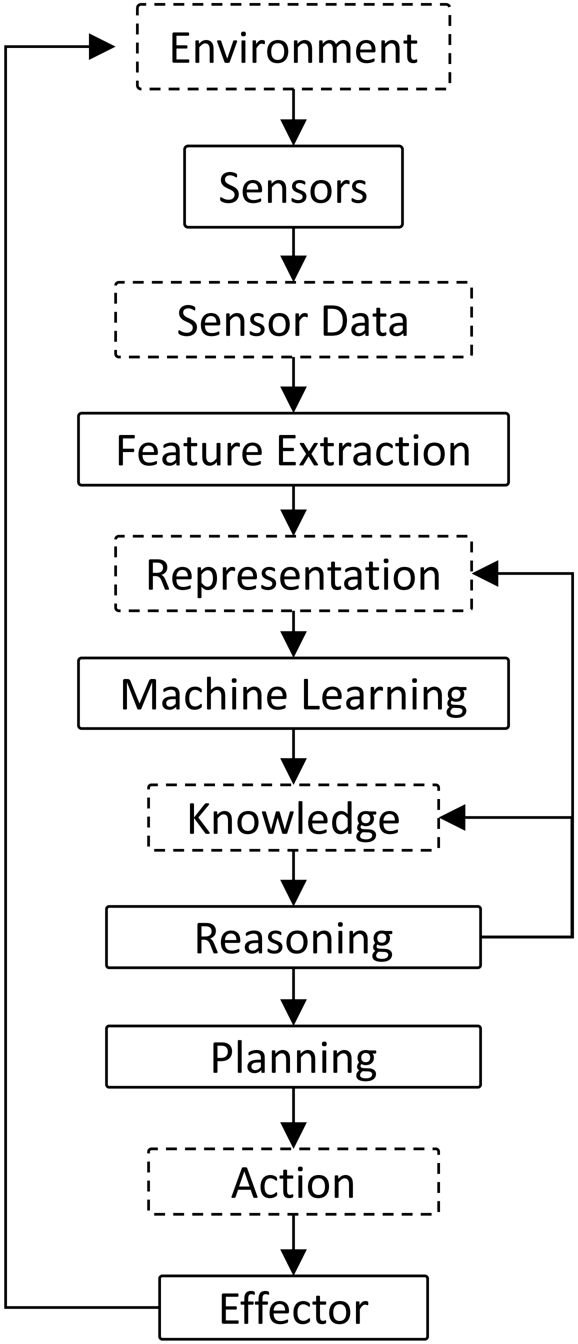
6.S094: Deep Learning for Self-Driving Cars 2018

https://selfdrivingcars.mit.edu

Lex Fridman

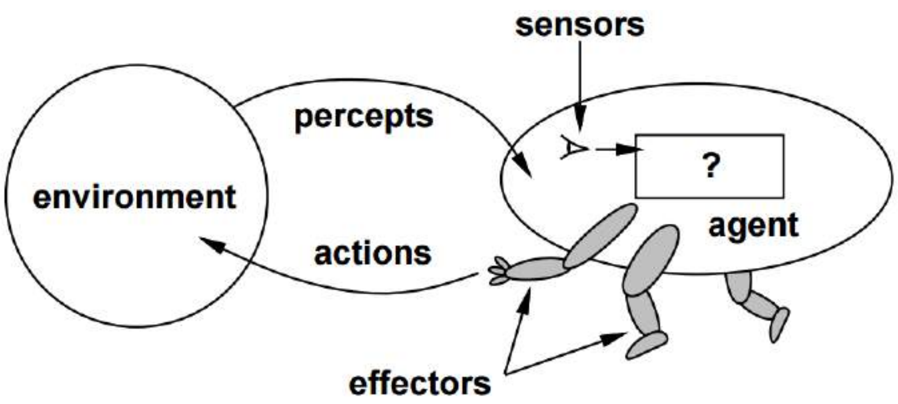
Lecture 3:

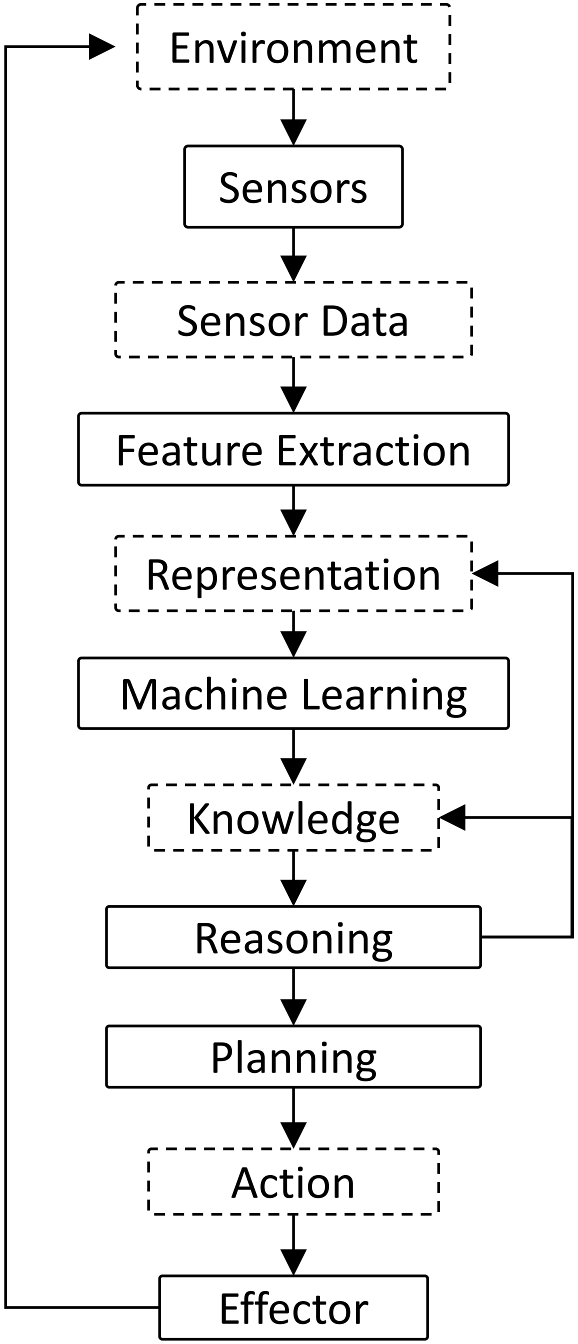
Deep Reinforcement Learning



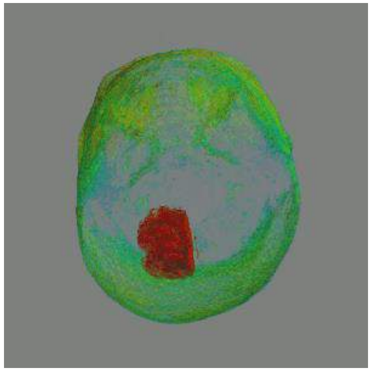
Open Question:

What can we **not** do with Deep Learning?





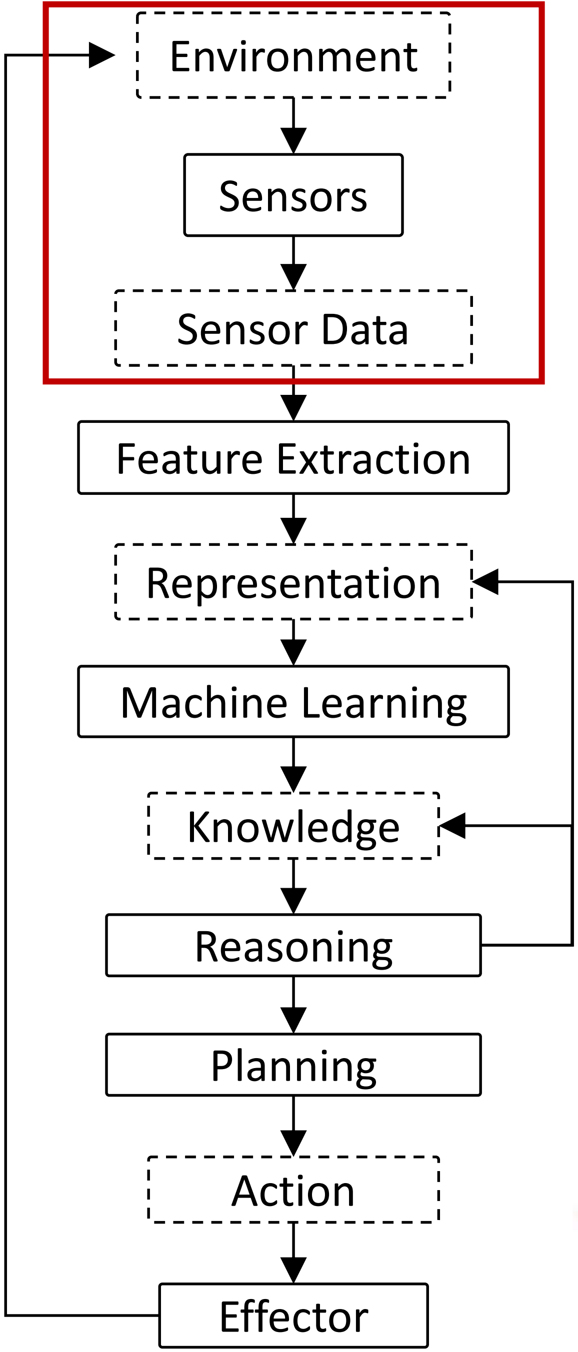
Formal tasks: Playing board games, card games. Solving puzzles, mathematical and logic problems.

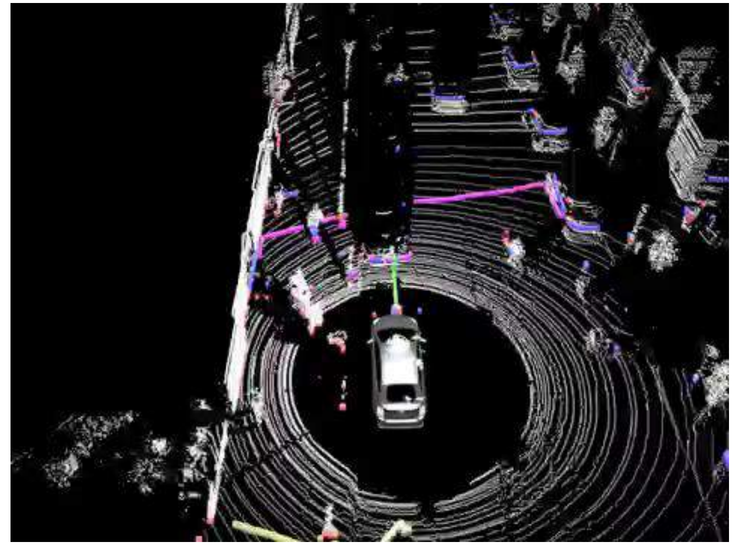


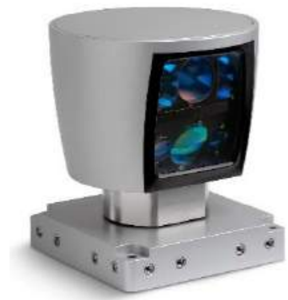
Expert tasks: Medical diagnosis, engineering, scheduling, computer hardware design.

Mundane tasks: Everyday speech, written language, perception, walking, object manipulation.

Human tasks: Awareness of self, emotion, imagination, morality, subjective experience, high-level-reasoning, consciousness.





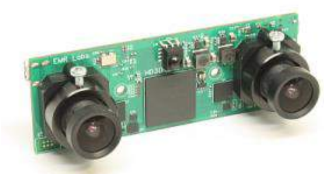


Lidar

Camera
(Visible, Infrared)

Radar

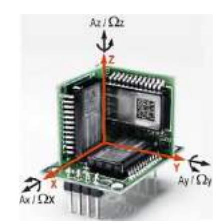
GPS



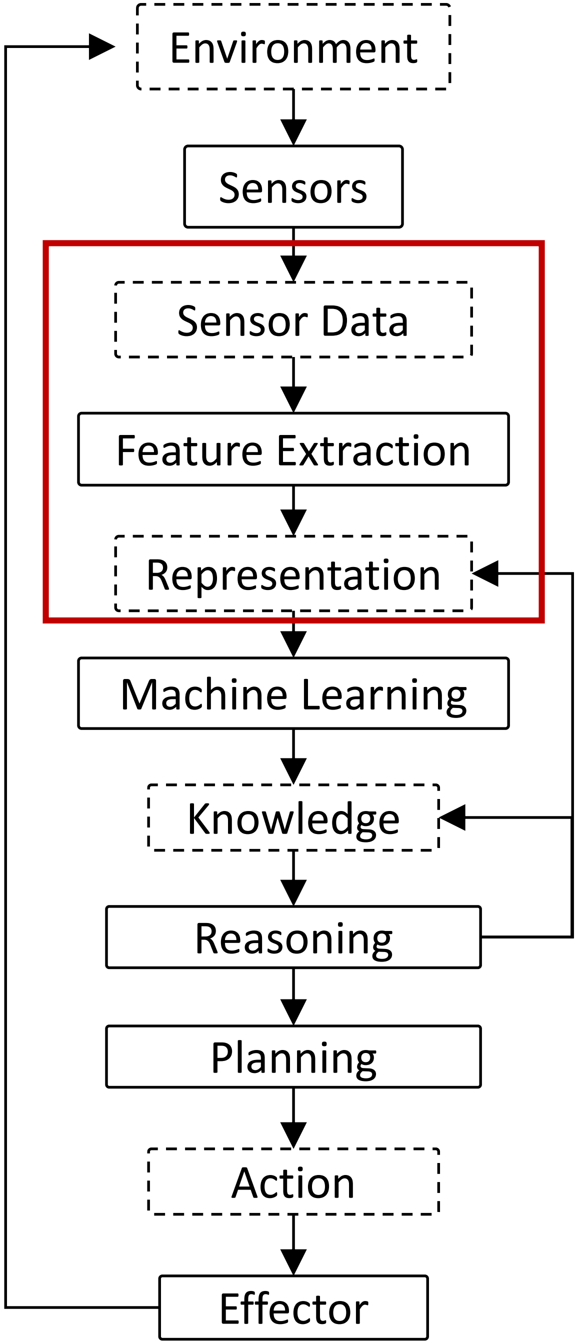
Stereo Camera

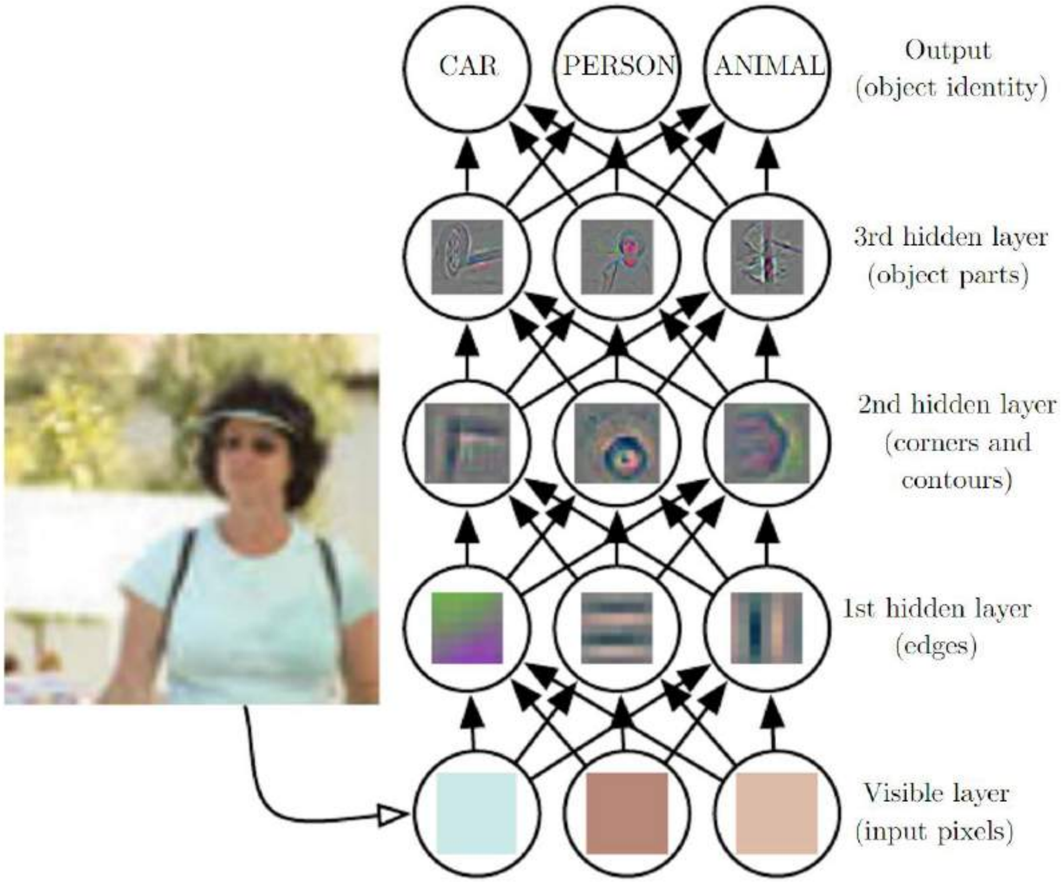
Microphone

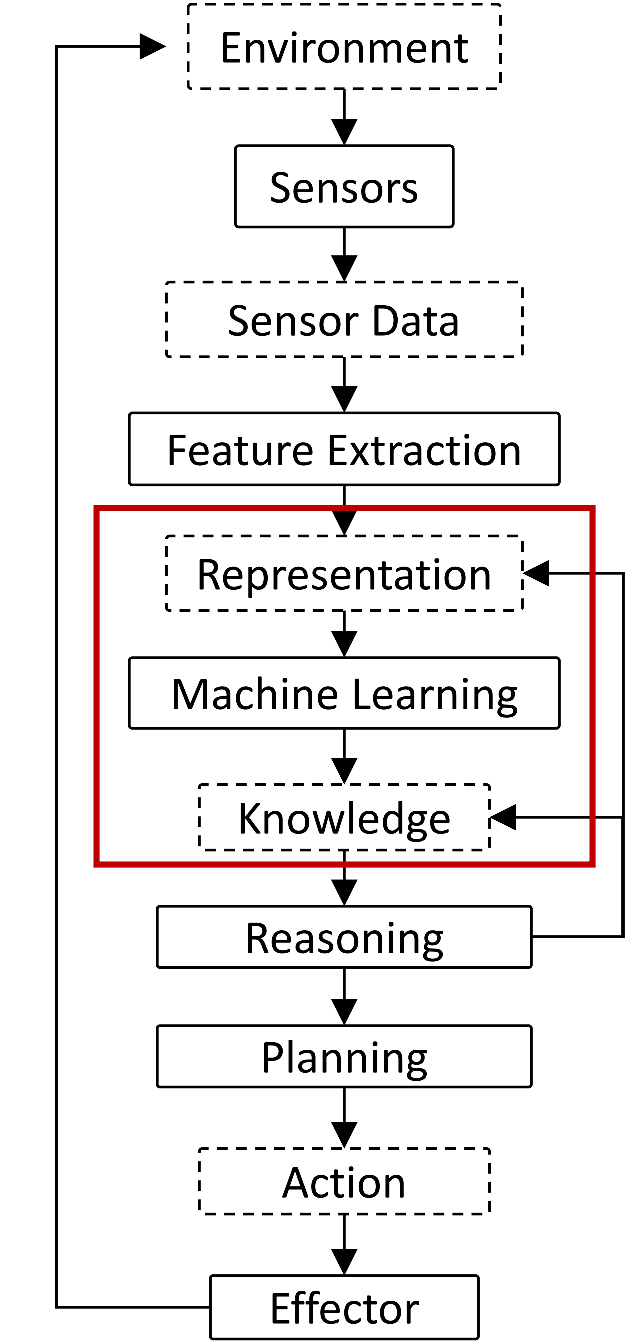
Networking
(Wired, Wireless)

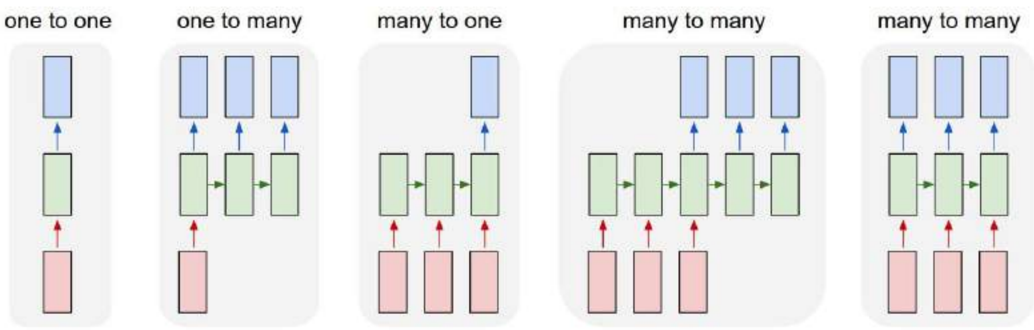


IMU


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


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

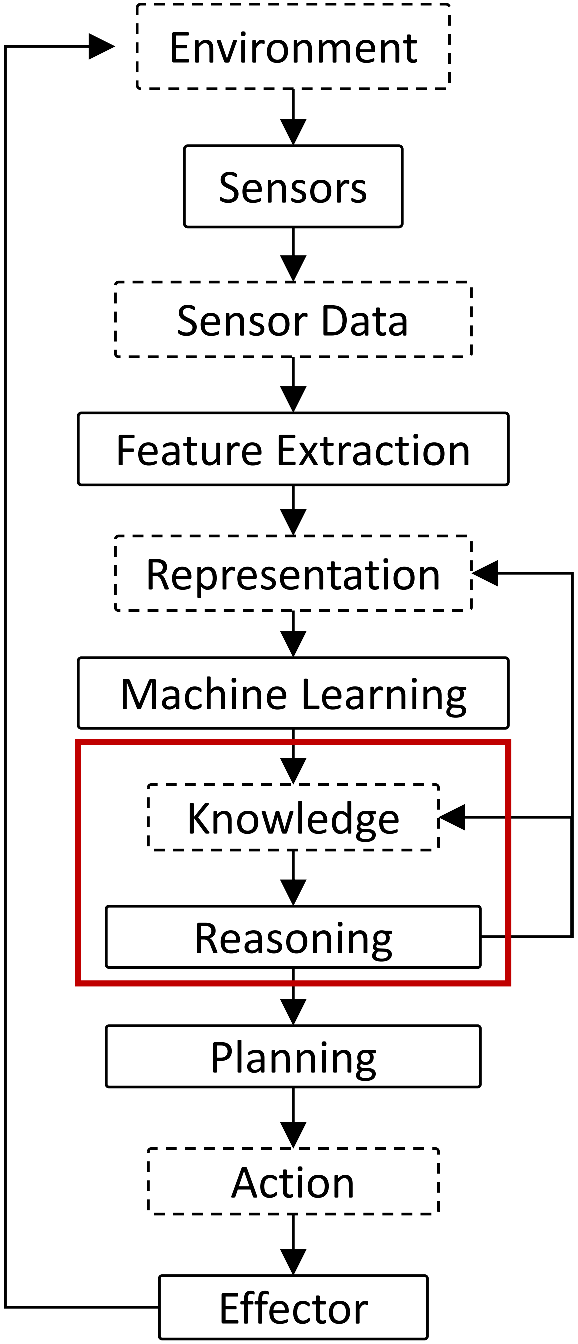
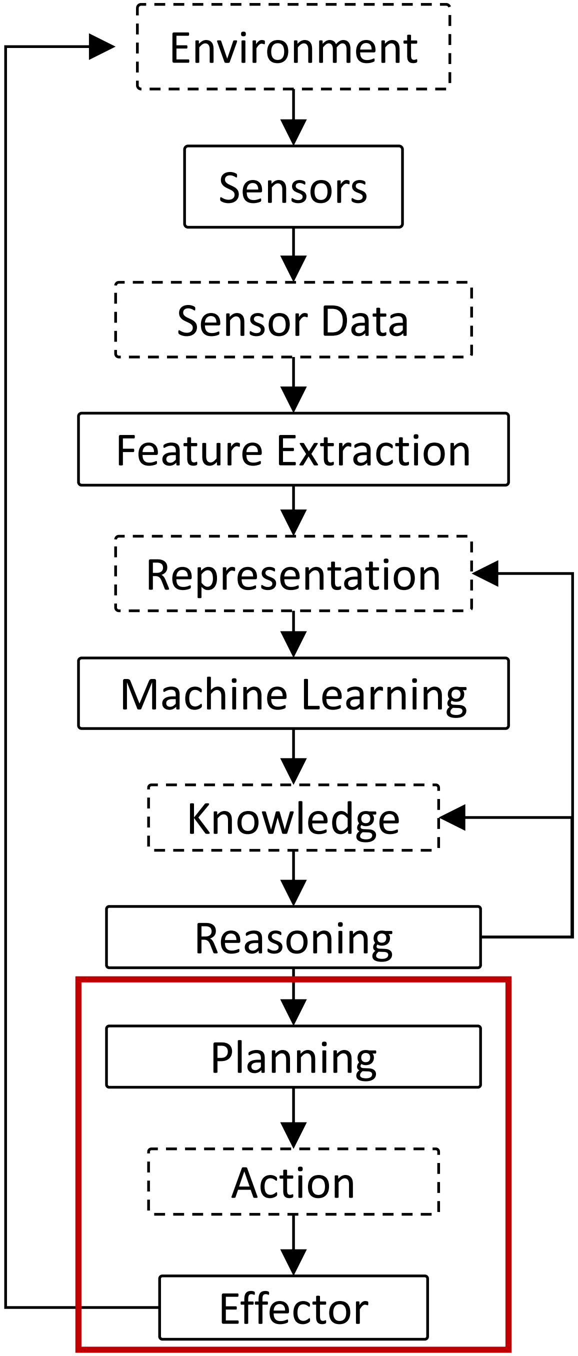
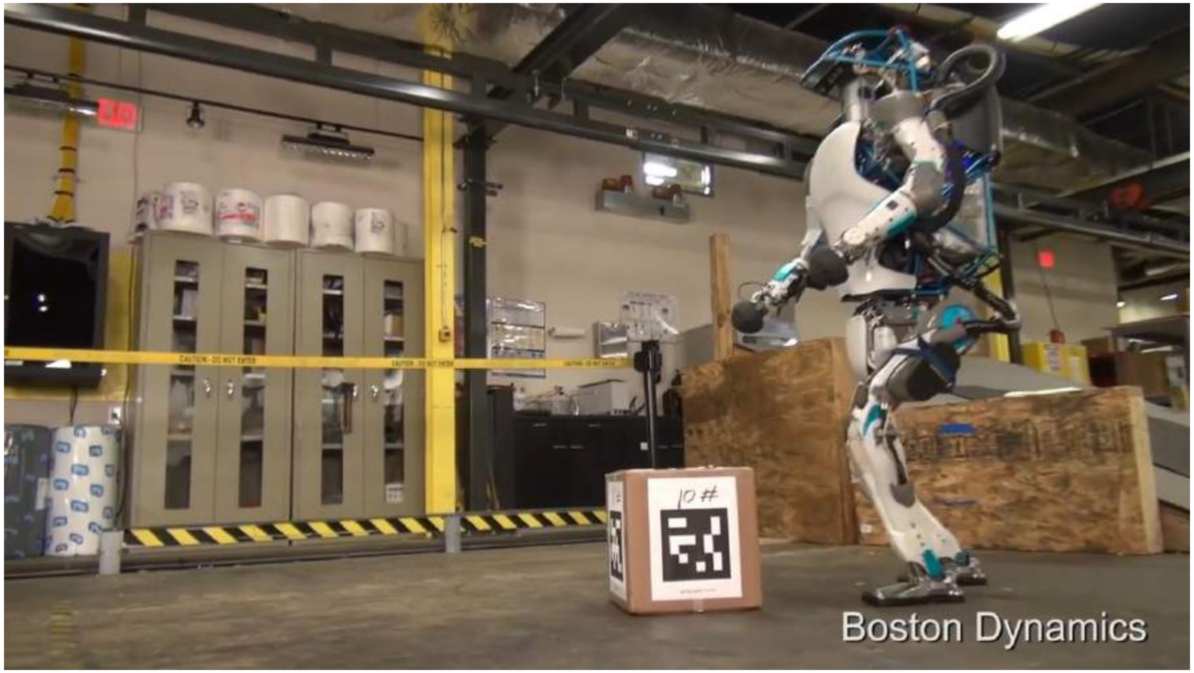


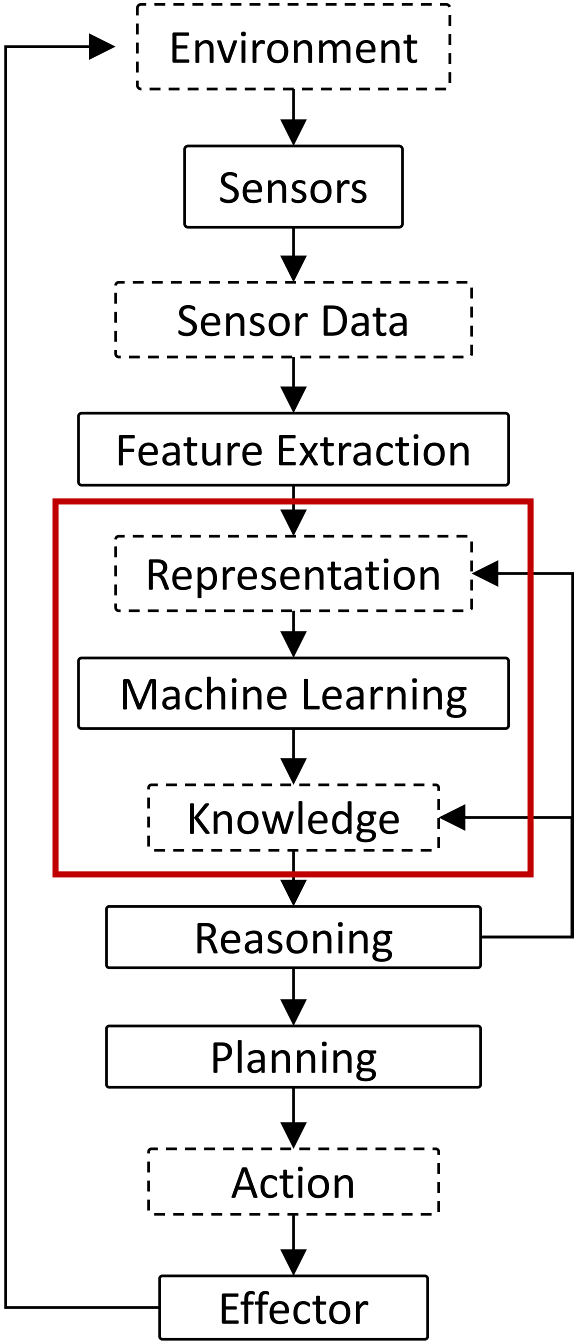
Image Recognition:
If it looks like a duck

Audio Recognition:
Quacks like a duck

Activity Recognition:
Swims like a duck

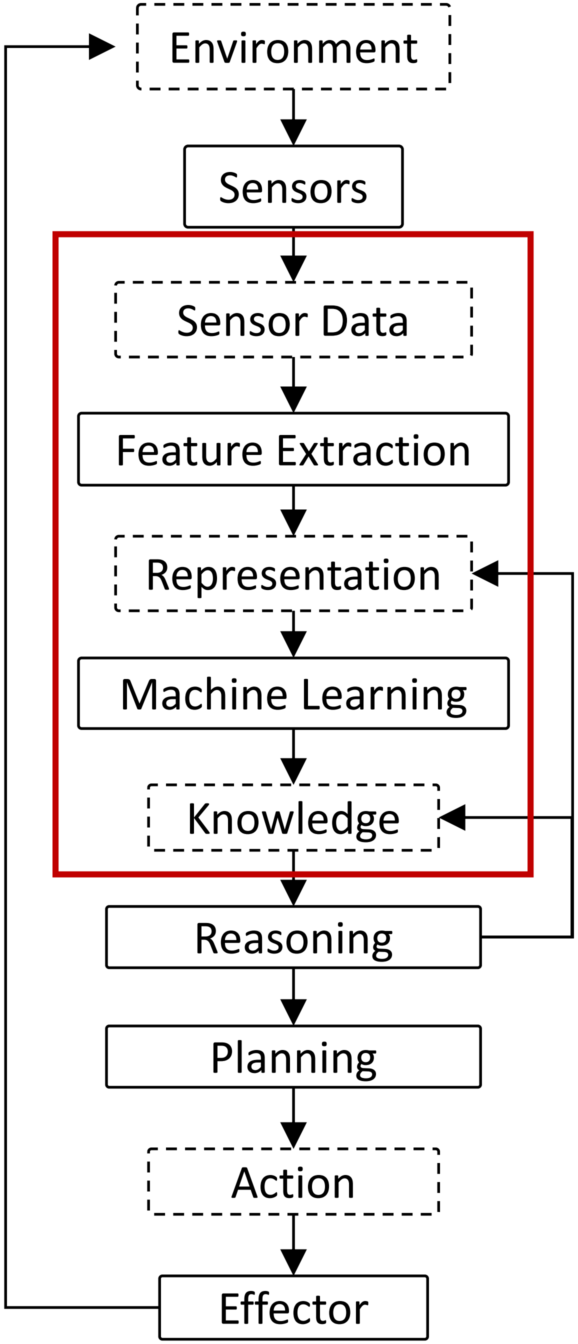

Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector


Boston Dynamics



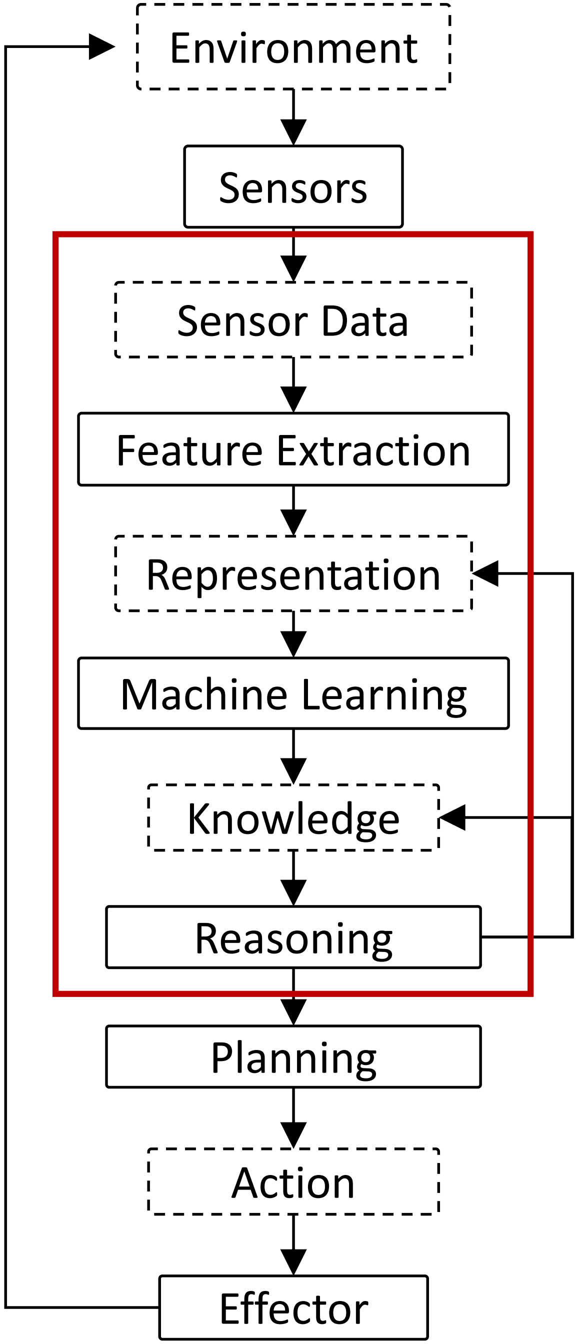
Open Question:

How much of this AI stack can be **learned**?



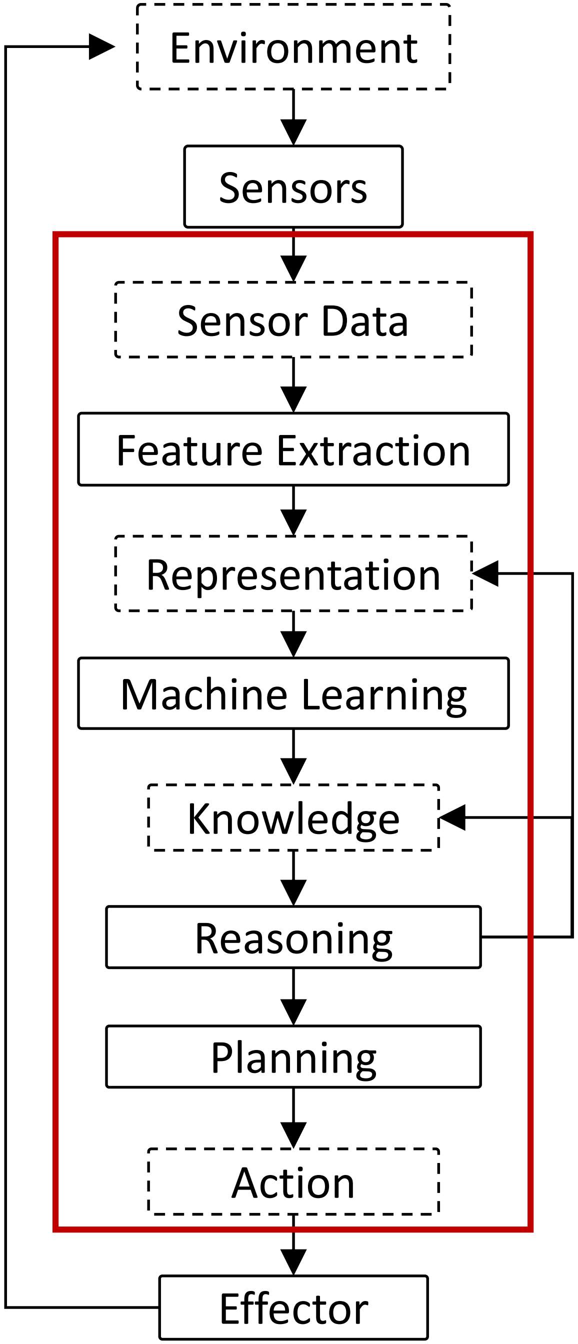
Open Question:

How much of this AI stack can be **learned**?



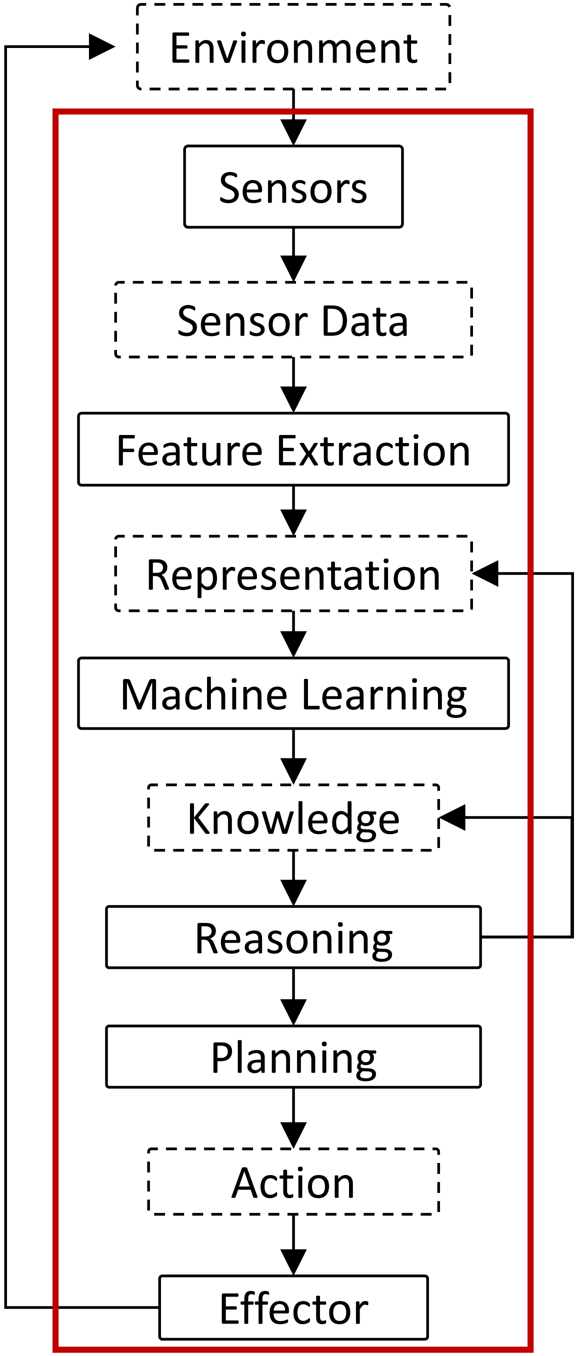
Open Question:

How much of this AI stack can be **learned**?



Open Question:

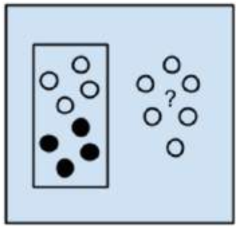
How much of this AI stack can be **learned**?



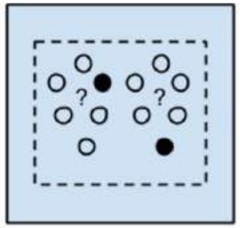
Open Question:

How much of this AI stack can be **learned**?

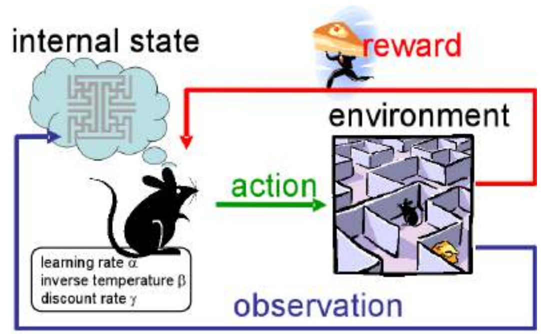
Types of Deep Learning



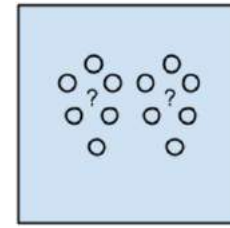
Supervised Learning



Semi-Supervised Learning

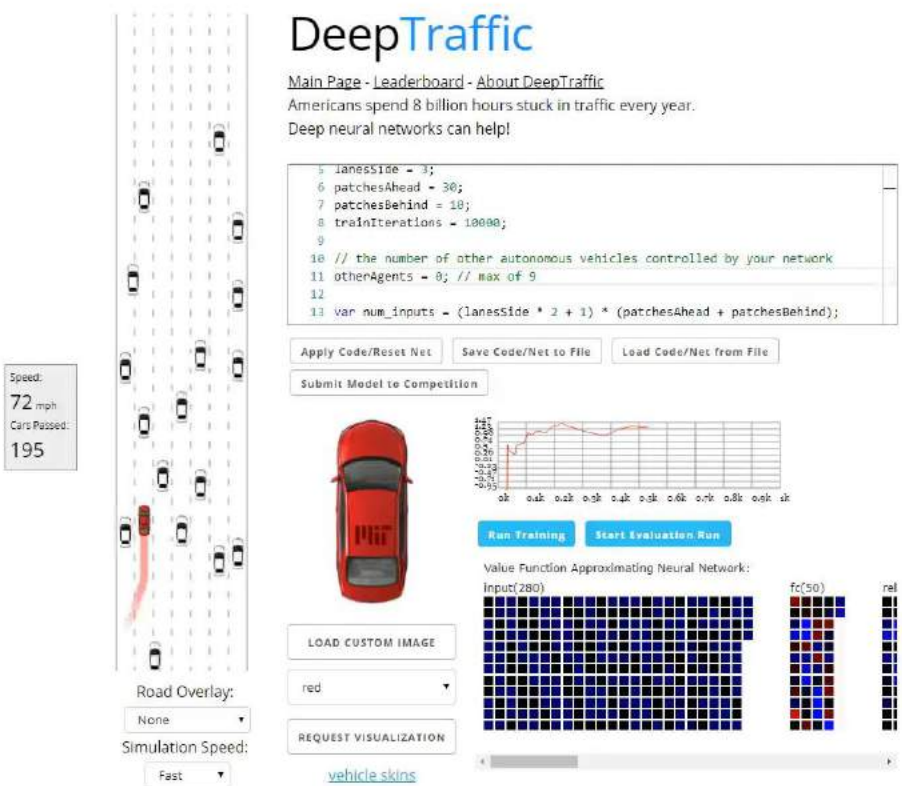


Reinforcement Learning



Unsupervised Learning

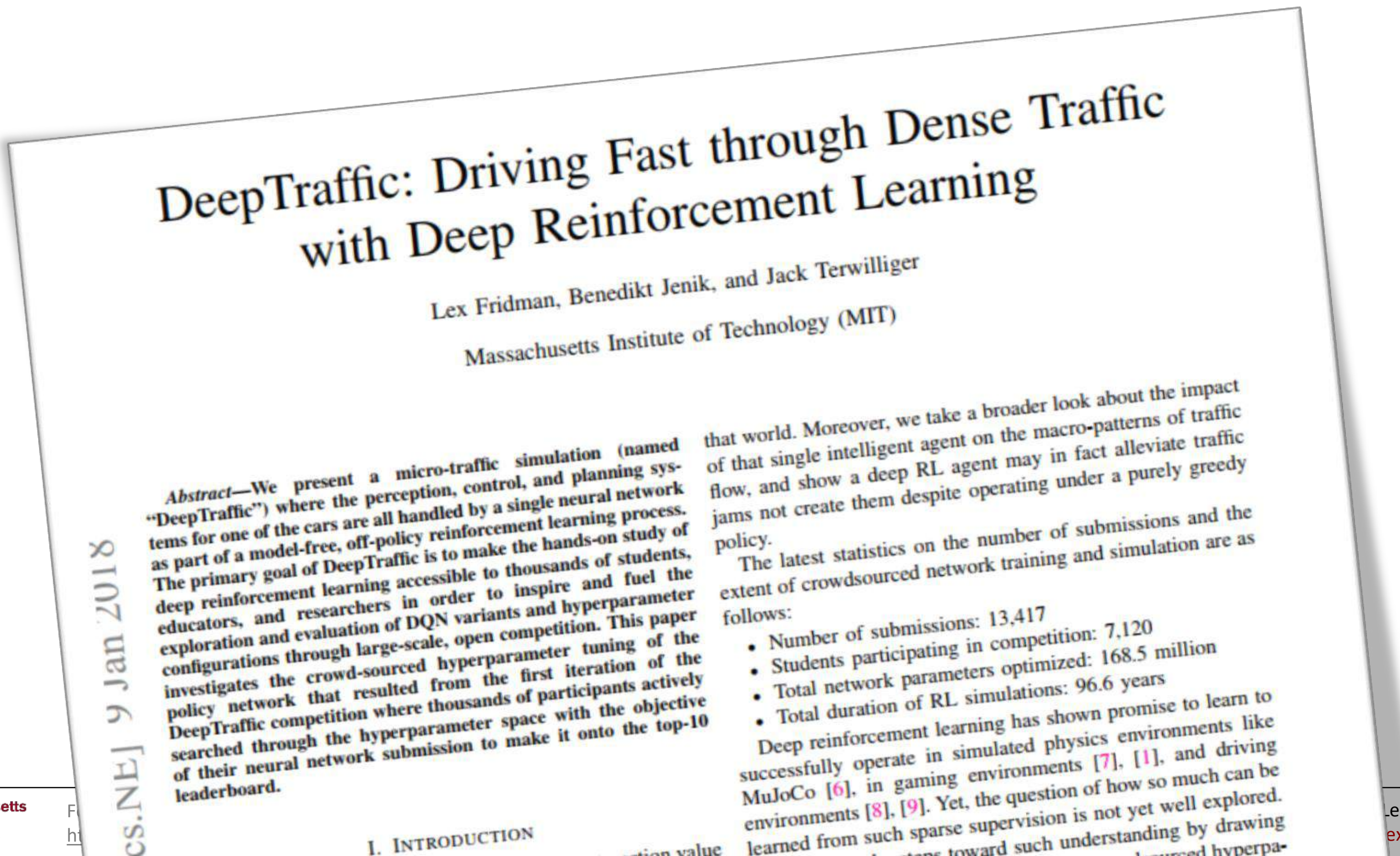
DeepTraffic: Deep Reinforcement Learning Competition



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

DeepTraffic: Deep Reinforcement Learning Competition

- **Competition:** https://github.com/lexfridman/deeptraffic
- **GitHub:** https://github.com/lexfridman/deeptraffic
- **Paper on arXiv:** https://arxiv.org/abs/1801.02805



DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning

Lex Fridman, Benedikt Jenik, and Jack Terwilliger

Massachusetts Institute of Technology (MIT)

Abstract—We present a micro-traffic simulation (named "DeepTraffic") where the perception, control, and planning systems for one of the cars are all handled by a single neural network as part of a model-free, off-policy reinforcement learning process. The primary goal of DeepTraffic is to make the hands-on study of deep reinforcement learning accessible to thousands of students, educators, and researchers in order to inspire and fuel the exploration and evaluation of DQN variants and hyperparameter configurations through large-scale, open competition. This paper investigates the crowd-sourced hyperparameter tuning of the policy network that resulted from the first iteration of the DeepTraffic competition where thousands of participants actively searched through the hyperparameter space with the objective of their neural network submission to make it onto the top-10 leaderboard.

I. INTRODUCTION

that world. Moreover, we take a broader look about the impact of that single intelligent agent on the macro-patterns of traffic flow, and show a deep RL agent may in fact alleviate traffic jams not create them despite operating under a purely greedy policy.

The latest statistics on the number of submissions and the extent of crowdsourced network training and simulation are as follows:

- Number of submissions: 13,417
- Students participating in competition: 7,120
- Total network parameters optimized: 168.5 million
- Total duration of RL simulations: 96.6 years

Deep reinforcement learning has shown promise to learn to successfully operate in simulated physics environments like MuJoCo [6], in gaming environments [7], [1], and driving environments [8], [9]. Yet, the question of how so much can be learned from such sparse supervision is not yet well explored.

Philosophical Motivation for Reinforcement Learning

Takeaway from Supervised Learning:

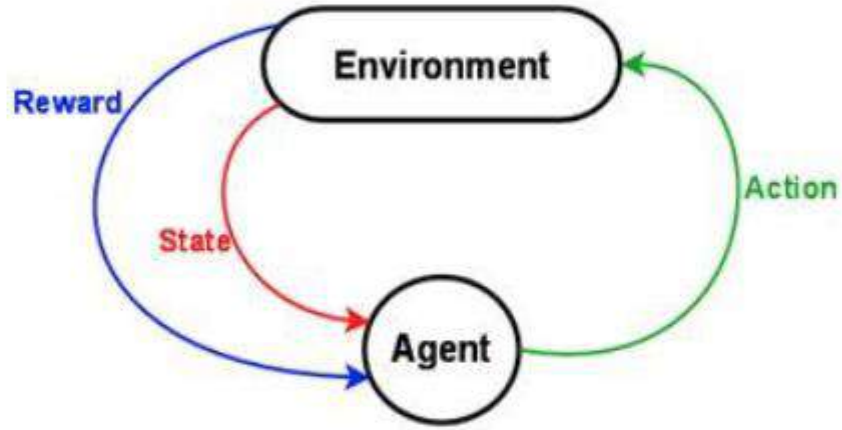
Neural networks are great at memorization and not (yet) great at reasoning.

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

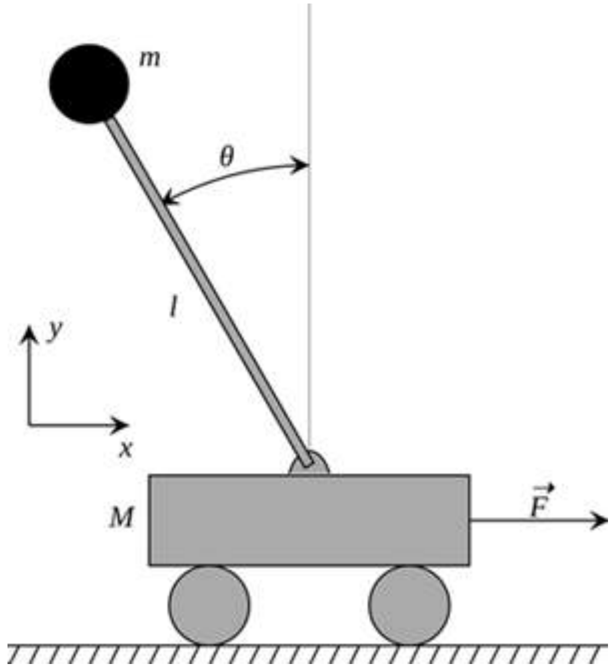


Examples of Reinforcement Learning

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** — angle, angular speed, 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

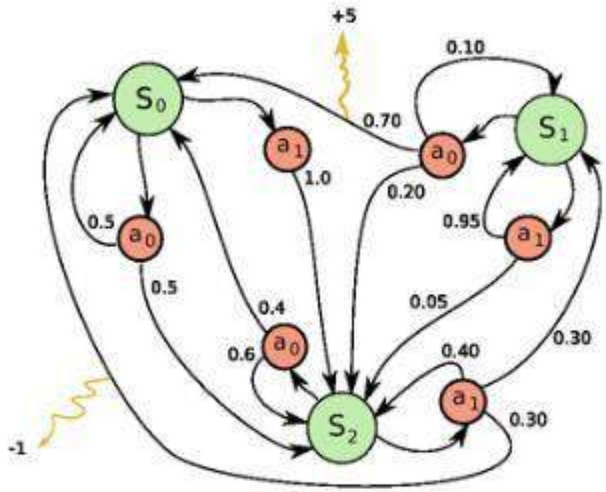
- **Goal** — Eliminate all opponents
- **State** — Raw game pixels of the game
- **Actions** — Up, Down, Left, Right etc
- **Reward** — Positive when eliminating an opponent, negative when the agent is eliminated

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

Markov Decision Process



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

state

action

reward

Terminal state

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

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

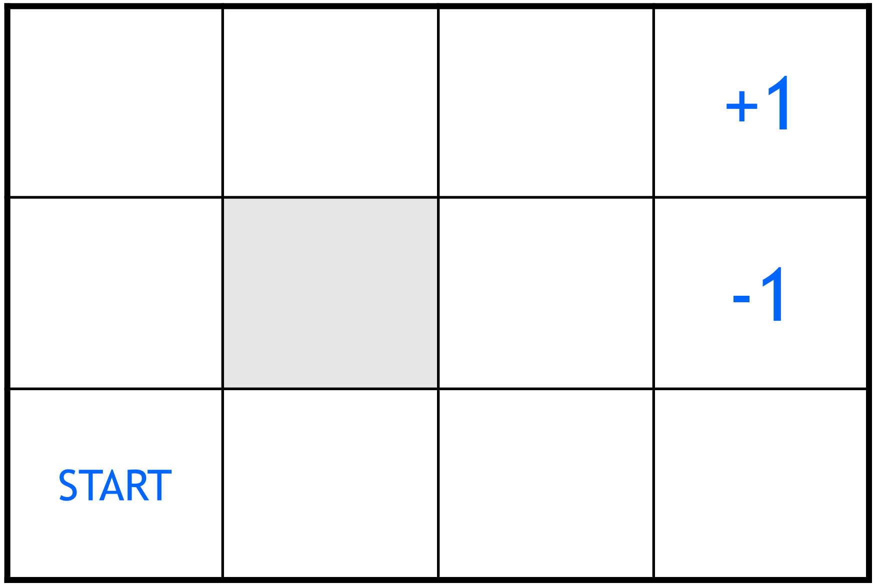
state

action

reward

Terminal state

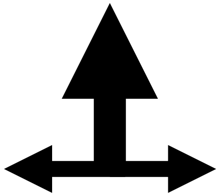
Robot in a Room



actions: UP, DOWN, LEFT, RIGHT

UP

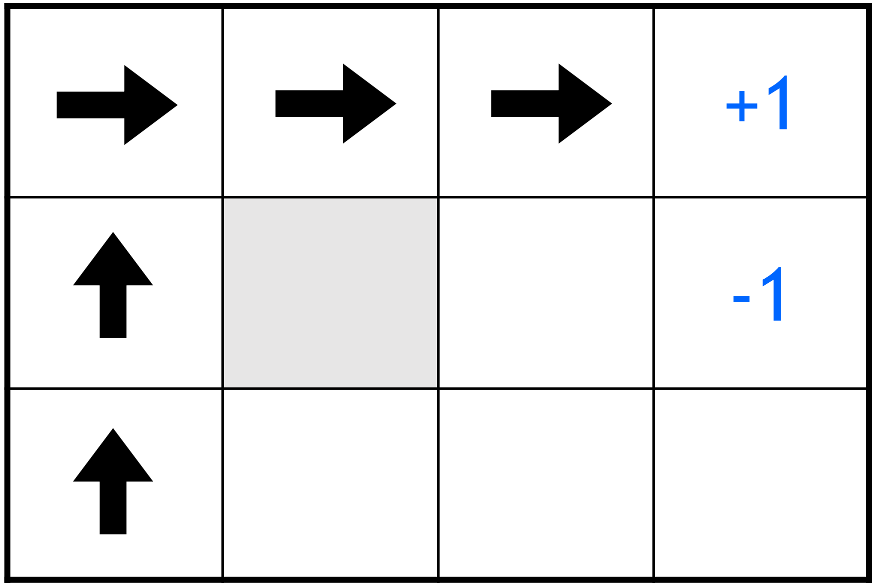
80%	move UP
10%	move LEFT
10%	move RIGHT



- 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?

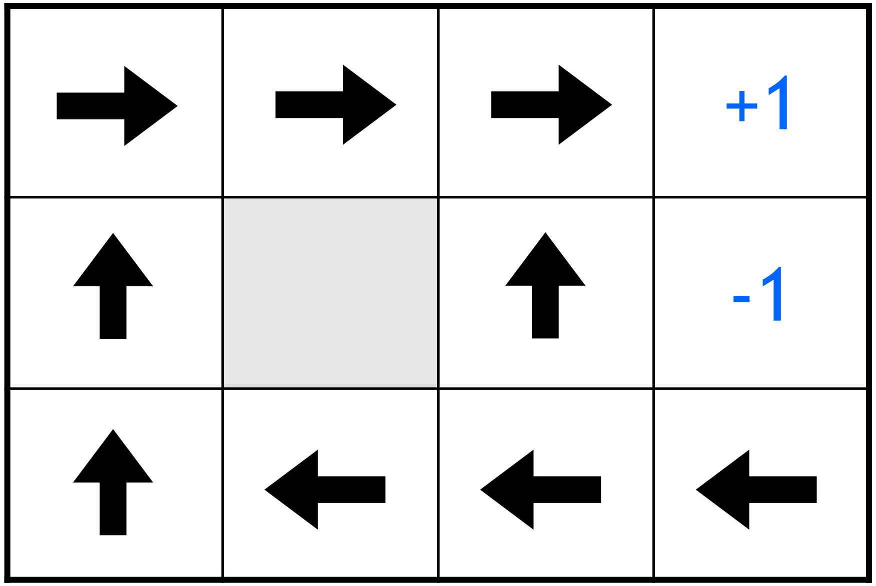
Is this a solution?



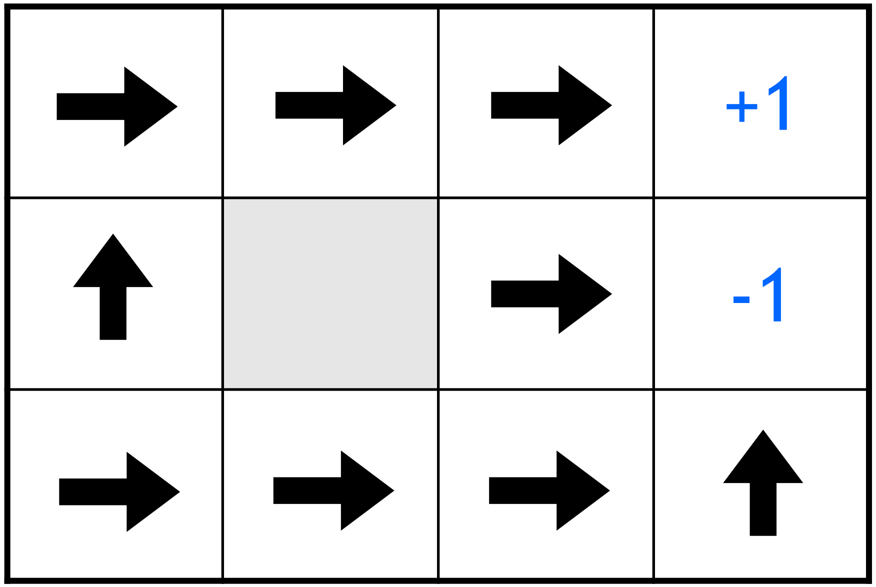
- only if actions deterministic
  - not in this case (actions are stochastic)

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

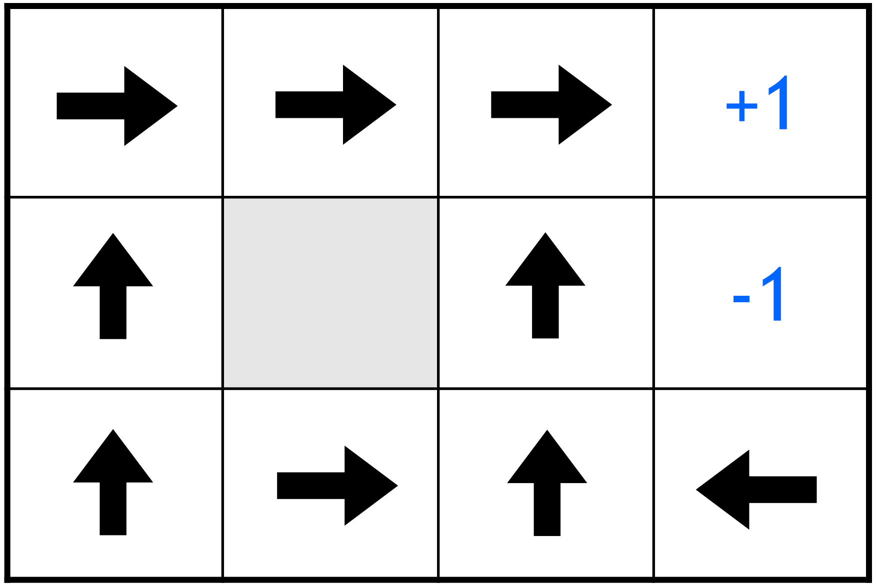
Optimal policy



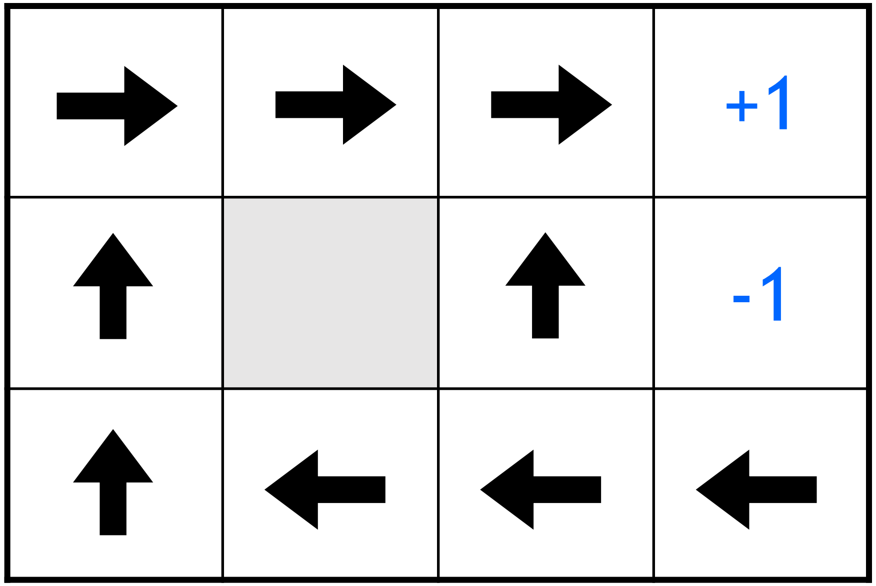
Reward for each step -2



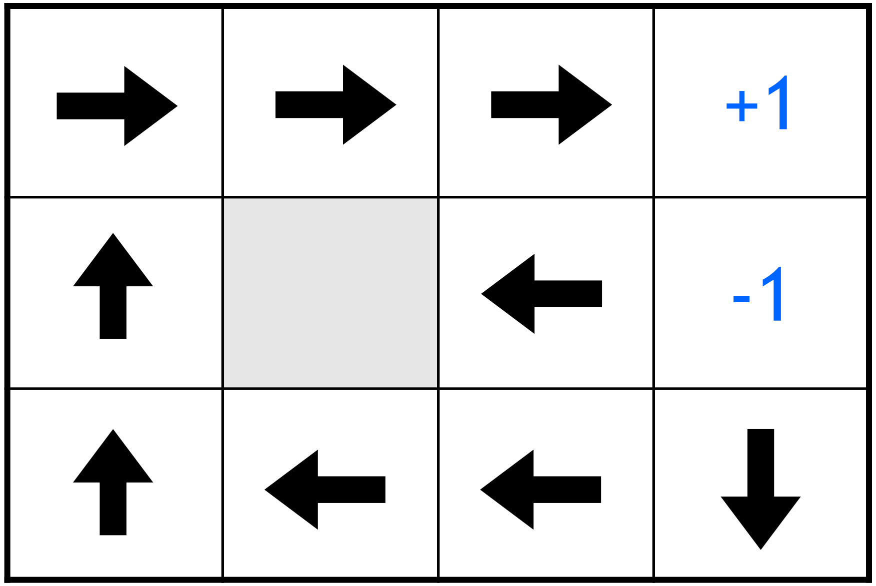
Reward for each step: -0.1



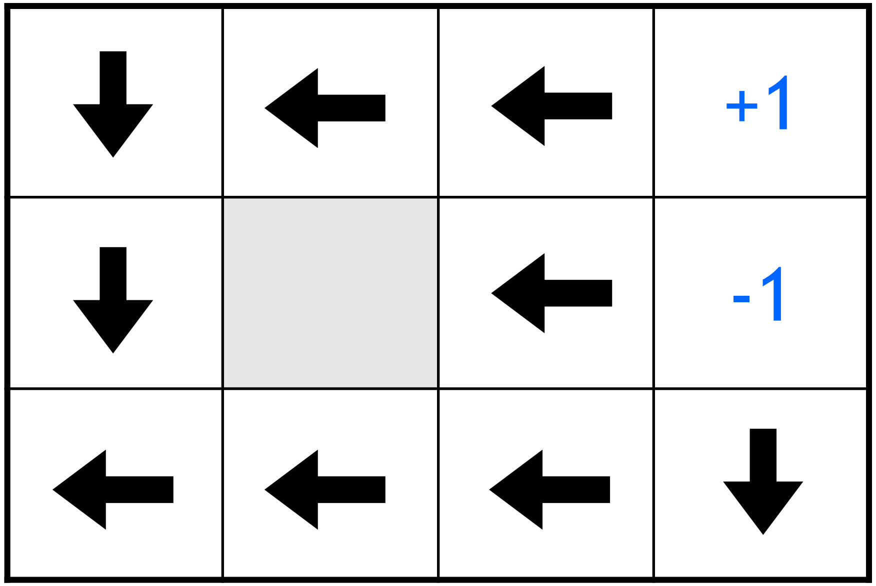
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01



Value Function

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

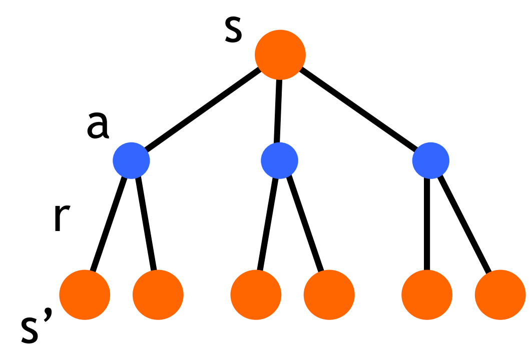
$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

- 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

- Key ingredient of Reinforcement Learning

- 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

- Maintain exploration
  - Use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - With probability 1-ε perform the optimal/greedy action
  - With probability ε perform a random action

  - Will keep exploring the environment
  - 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

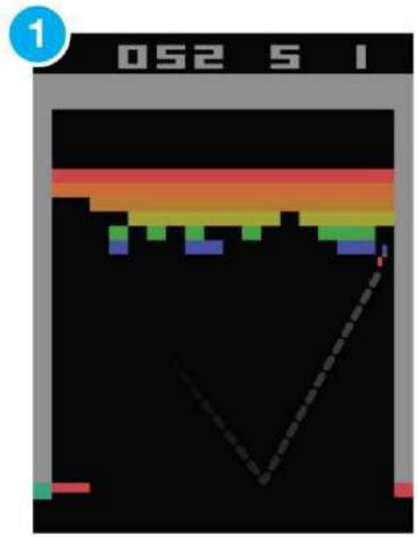
Reward

	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
```

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!

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:

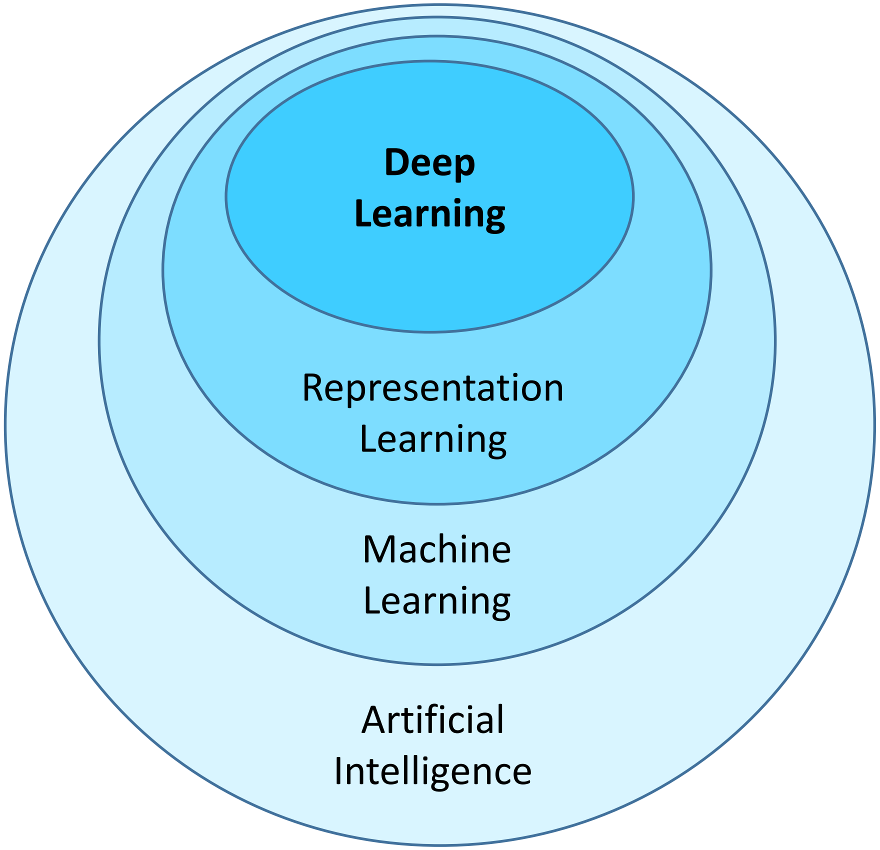
Brute-force propagation of outcomes to knowledge about states and actions. This is a kind of brute-force "reasoning".

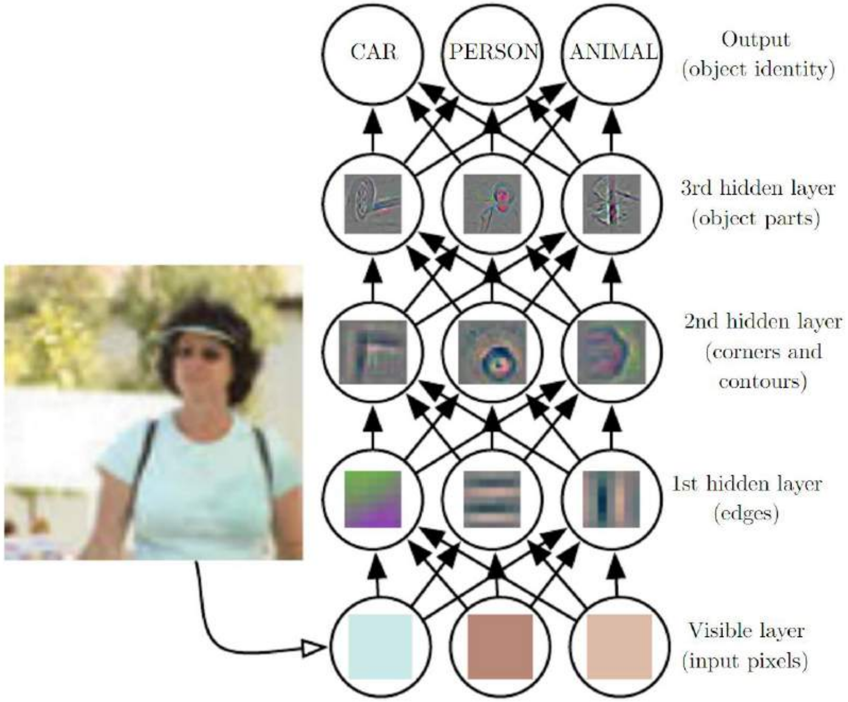
Hope for Deep Learning + Reinforcement Learning:

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)





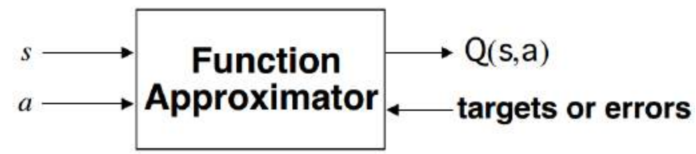
Intelligence: Ability to accomplish complex goals.

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

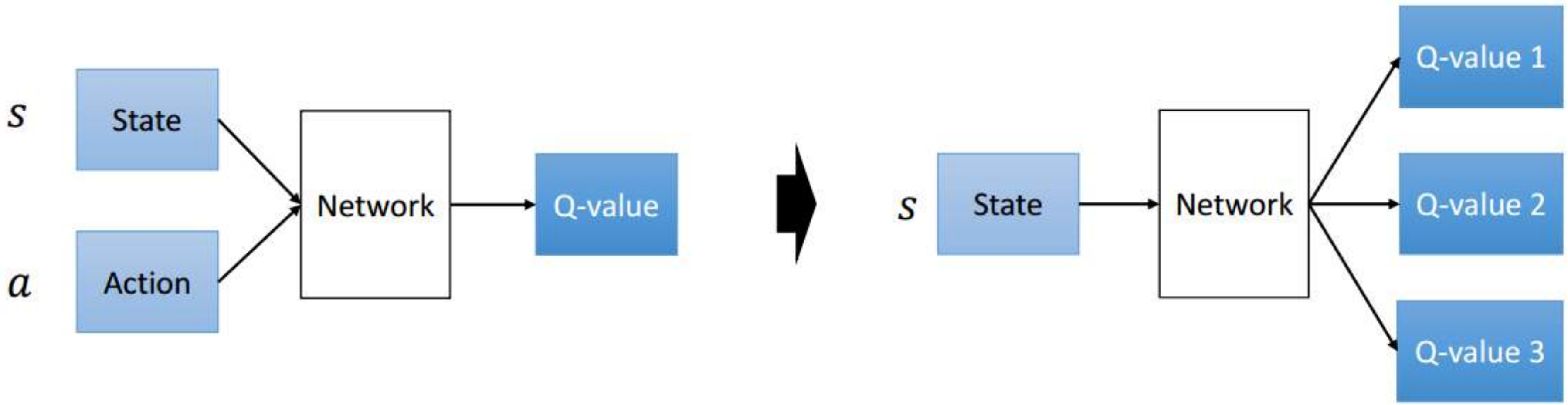
Deep Q-Learning

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

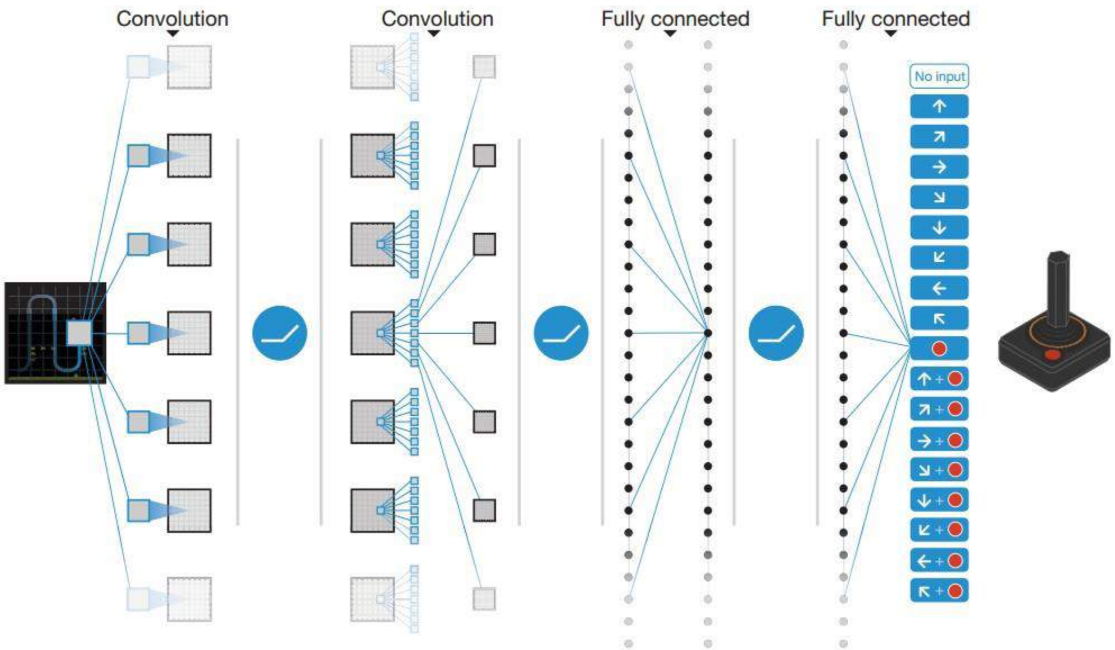
- Linear
- Non-linear: **Q-Network**



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



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.

Deep Q-Network Training

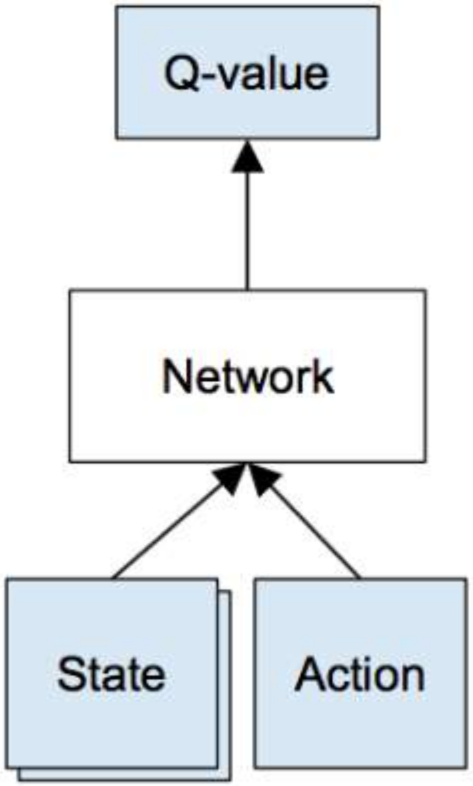
- Bellman Equation:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

- Loss function (squared error):

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

DQN Training



Given a transition < *s, a, r, s'* >, the Q-table update rule in the previous algorithm must be replaced with the following:

- Do a feedforward pass for the current state *s* to get **predicted Q-values for all actions**
- Do a feedforward pass for the next state *s'* and calculate maximum overall network outputs $\max_{a'} Q(s', a')$
- Set Q-value target for action to $r + \gamma \max_{a'} Q(s', a')$ (use the max calculated in step 2).
    - For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
- Update the weights using backpropagation.

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.

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

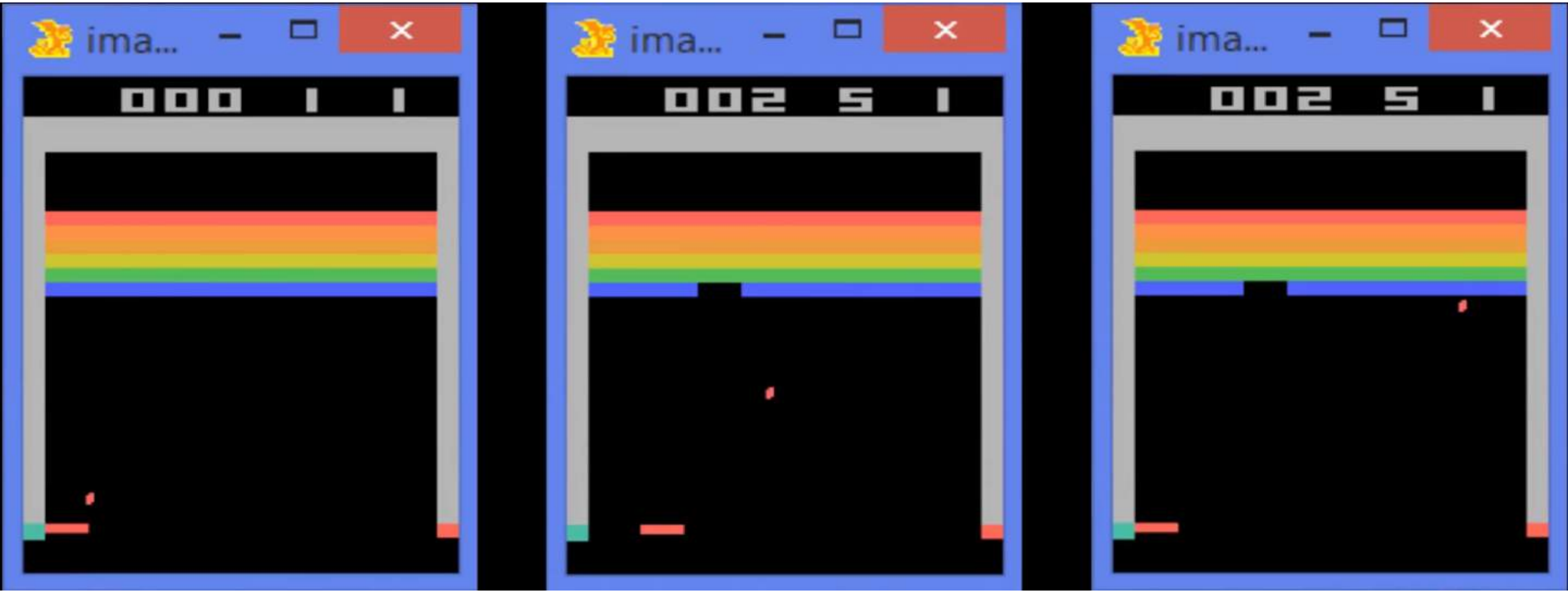
target Q function in the red rectangular is fixed

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

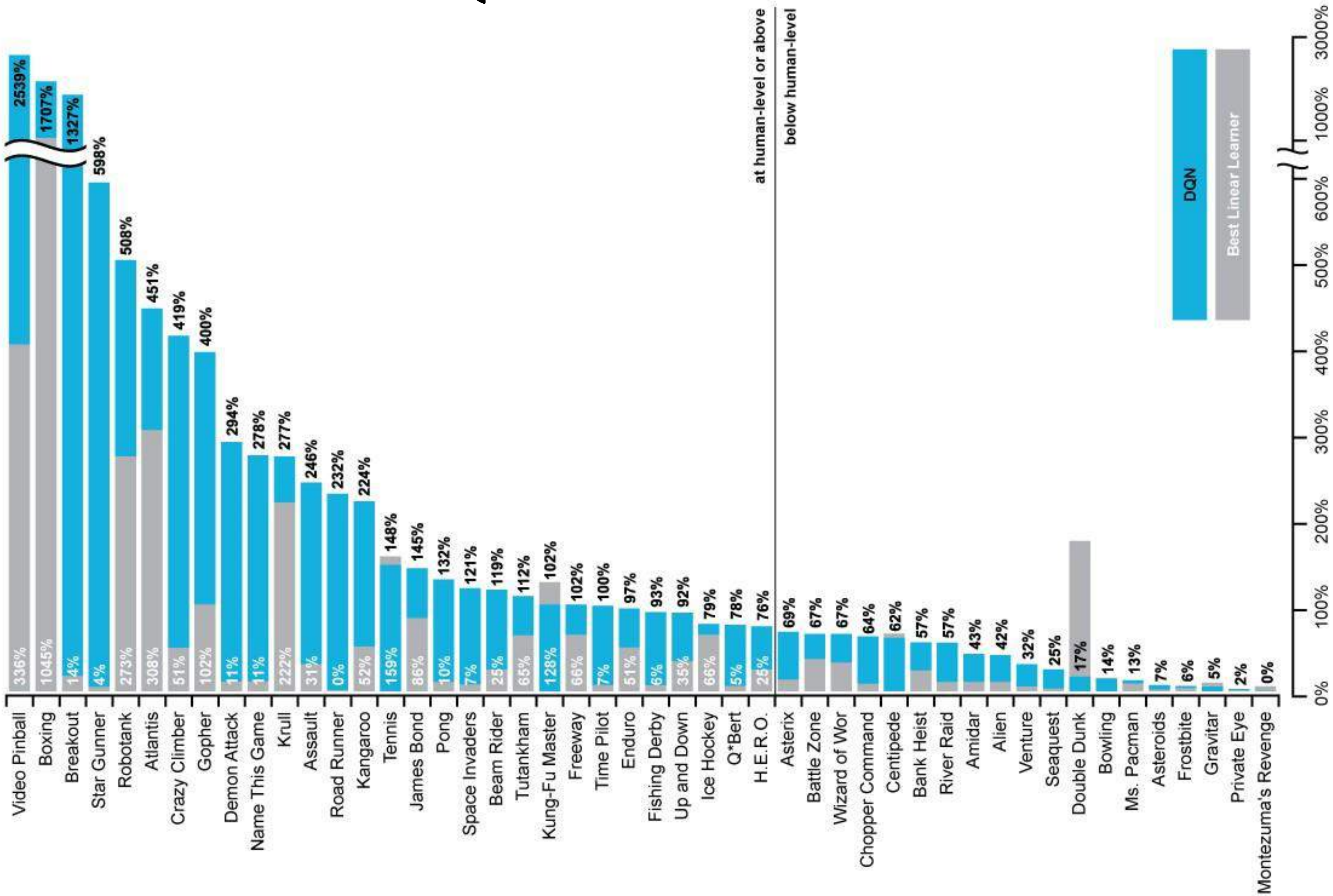


After **10 Minutes** of Training

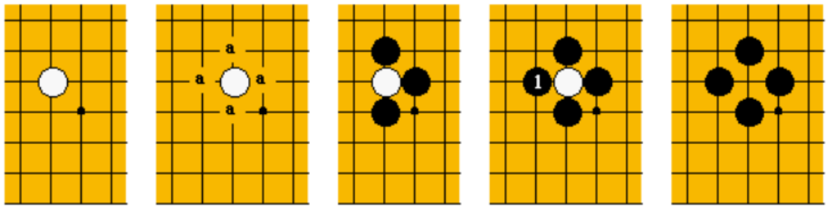
After **120 Minutes** of Training

After **240 Minutes** of Training

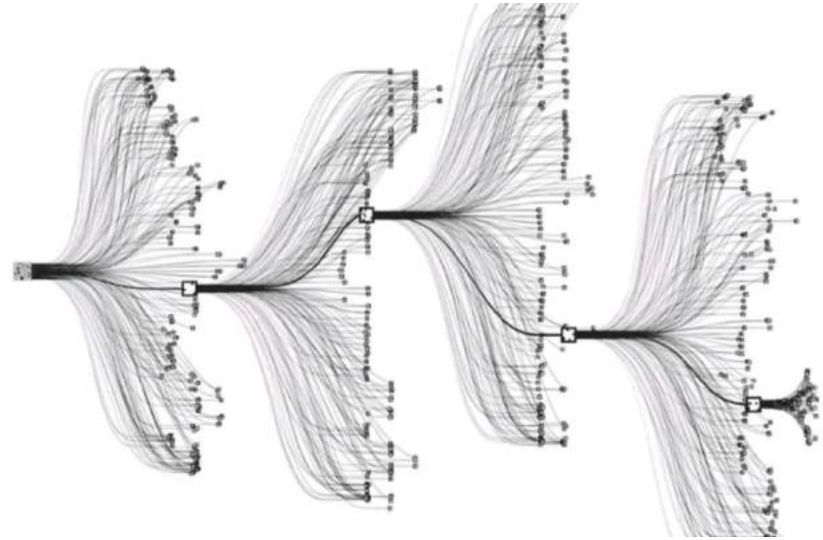
DQN Results in Atari



Game of Go

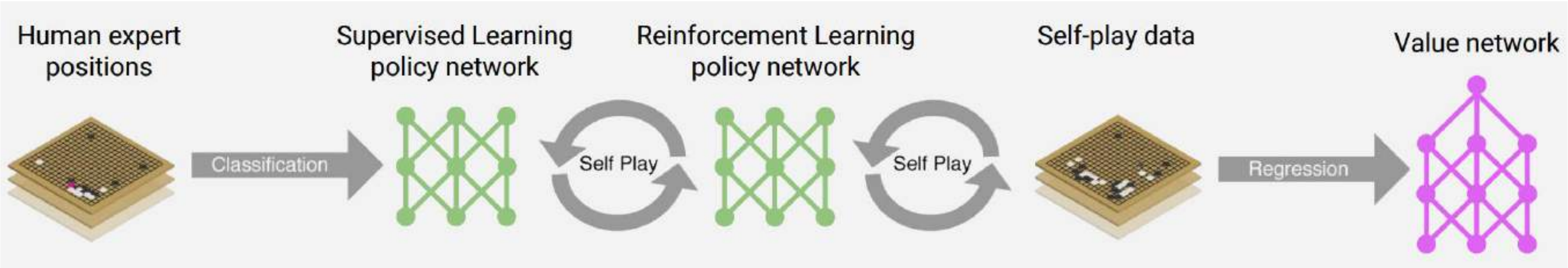


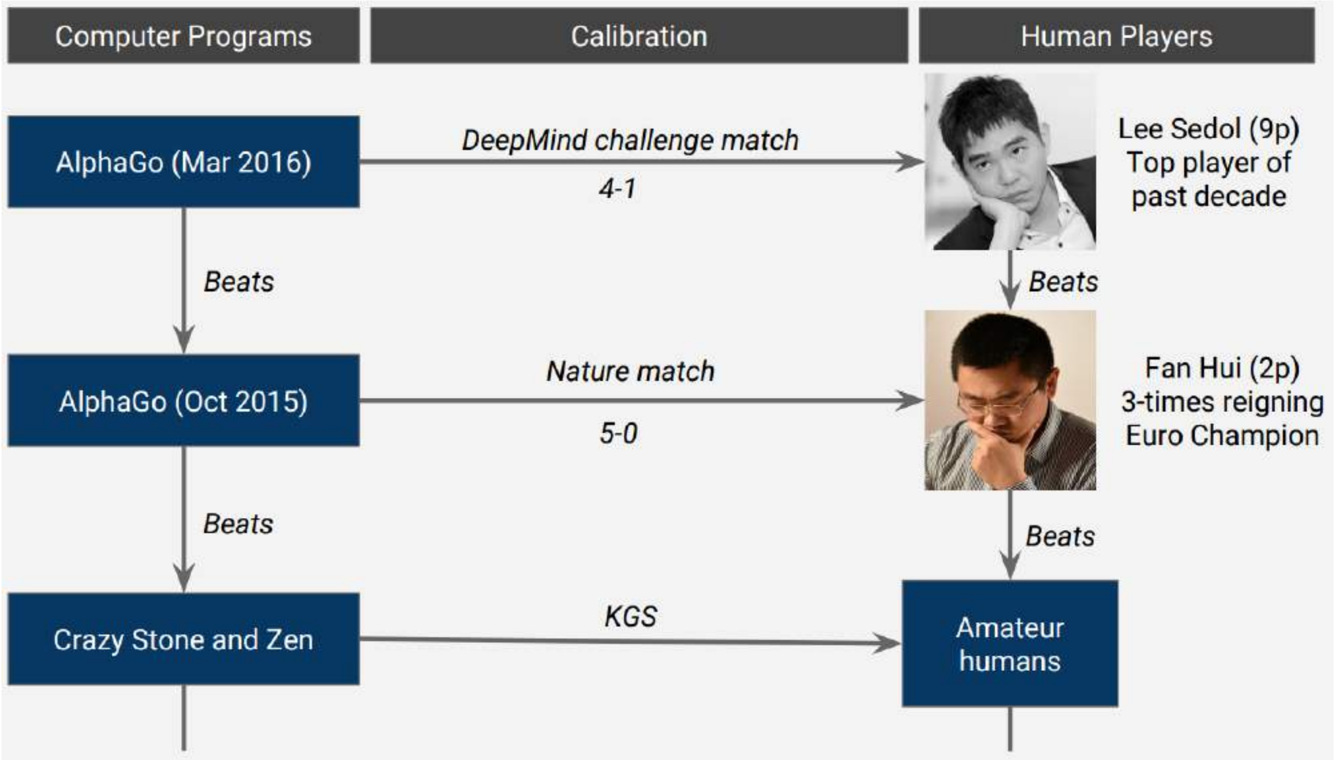
Dia. 3 | Dia. 4 liberties | Dia. 5 atari | Dia. 6 capture | Dia. 7 result



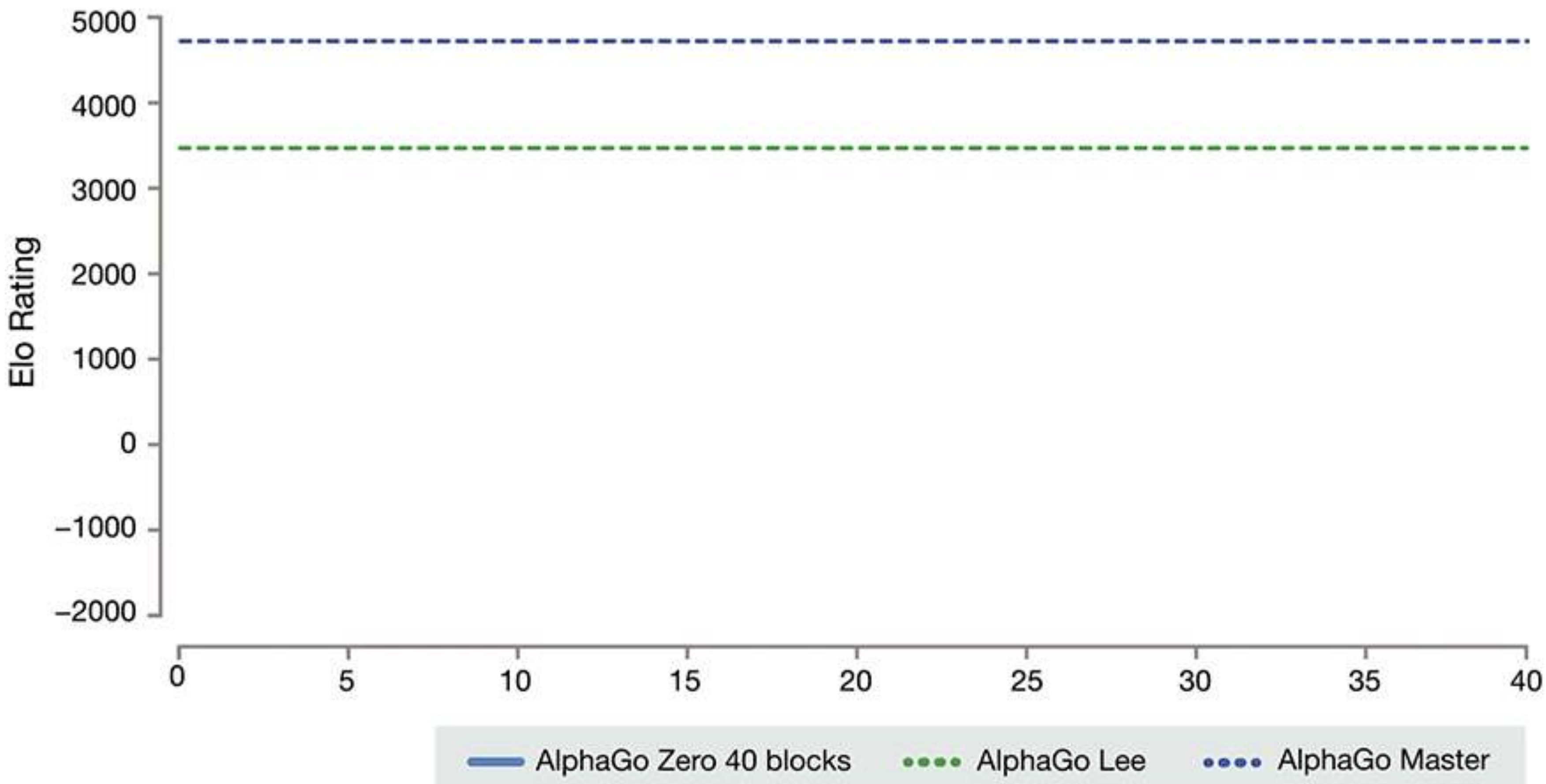
Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	$3.72497923 \times 10^{79}$
19×19	361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

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



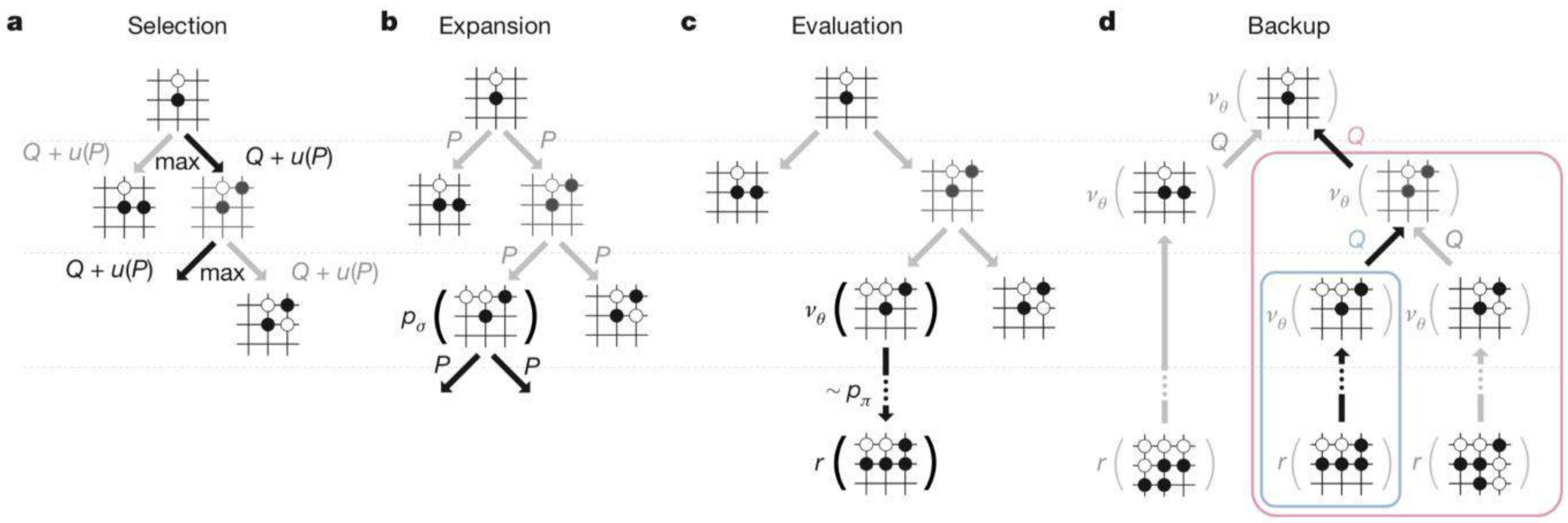


AlphaGo Zero (2017): Beats AlphaGo



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)

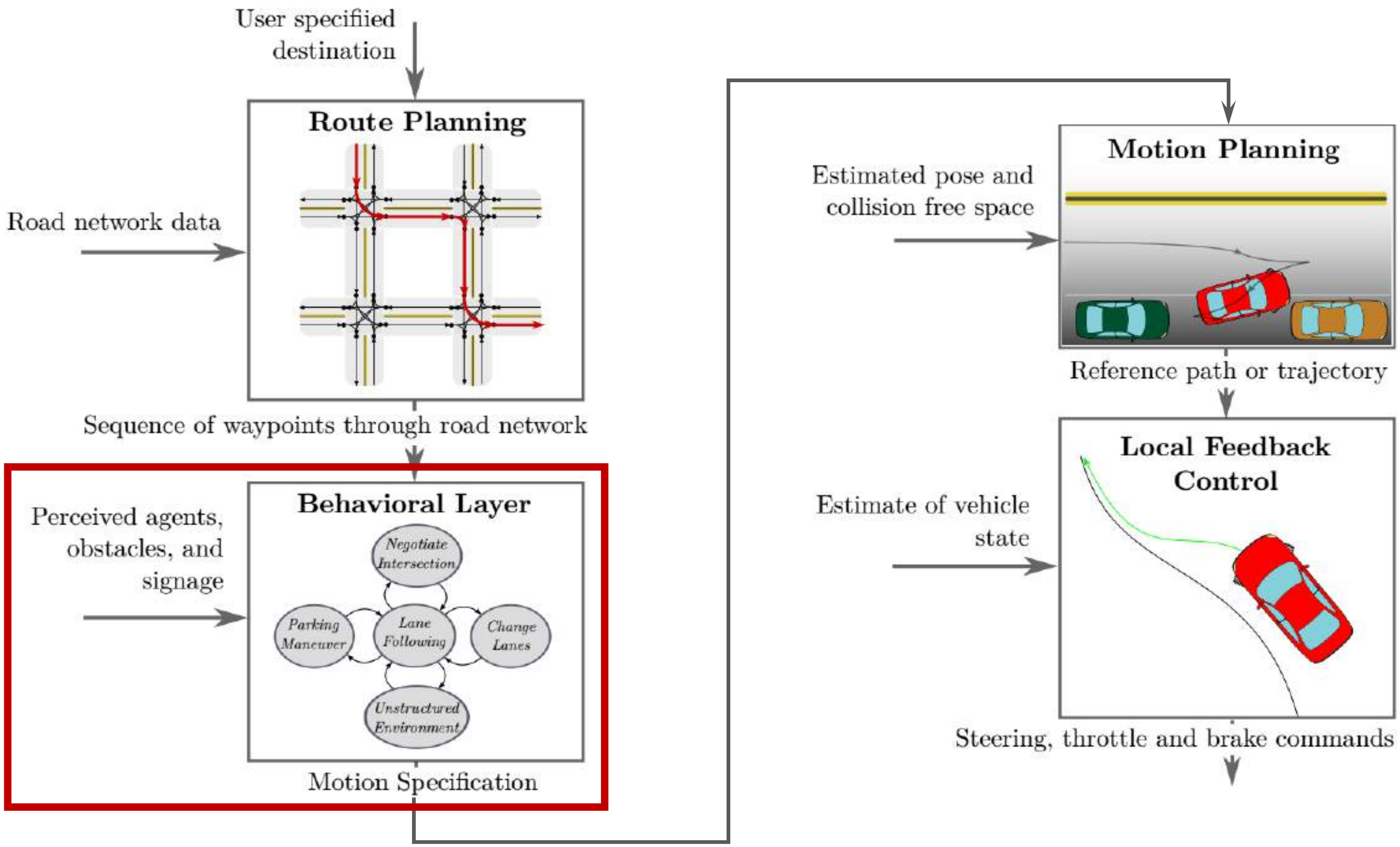


AlphaGo Zero Approach

- 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

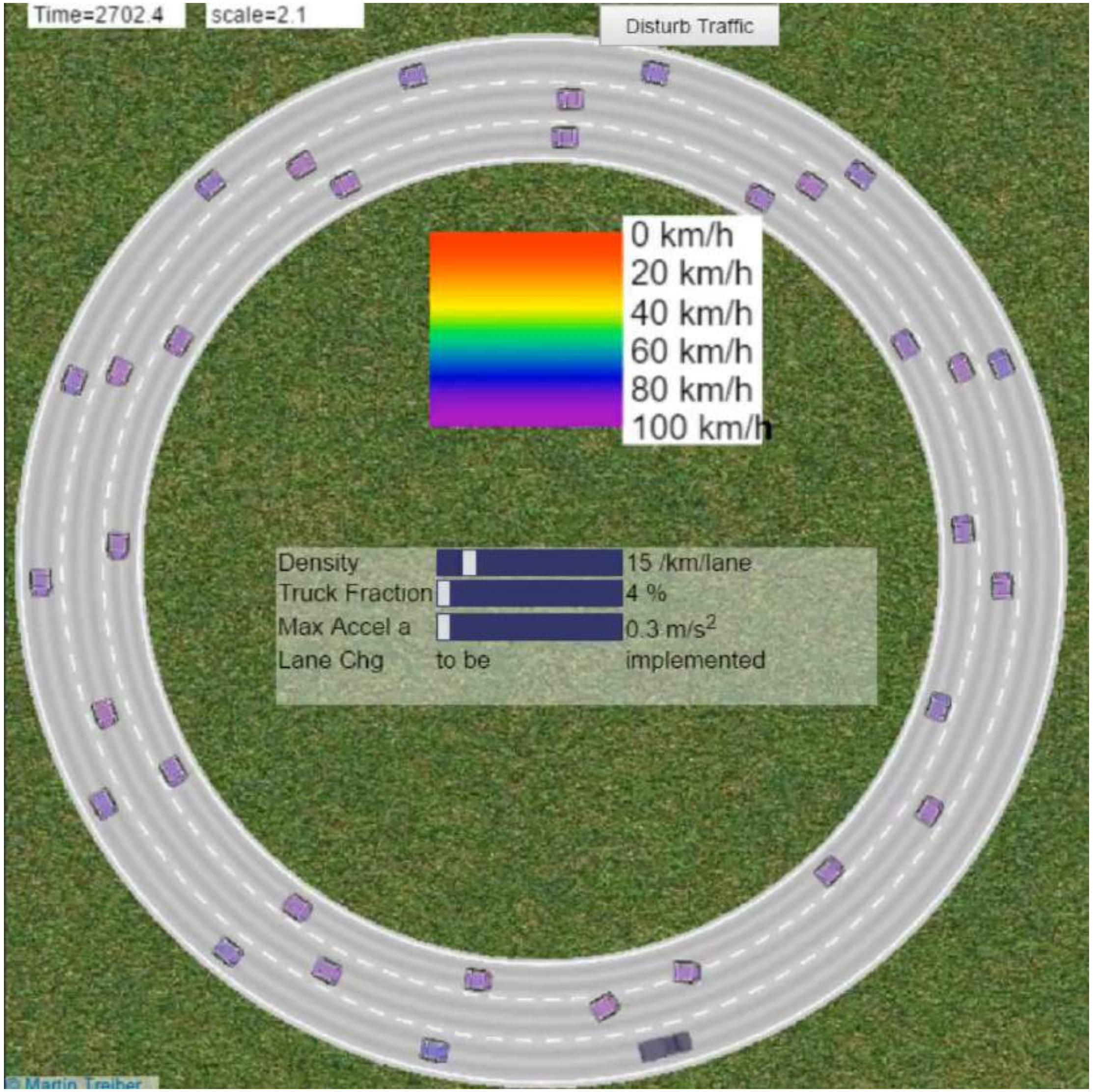
Americans spend 8 billion hours stuck in traffic every year.

Autonomous Driving: A Hierarchical View



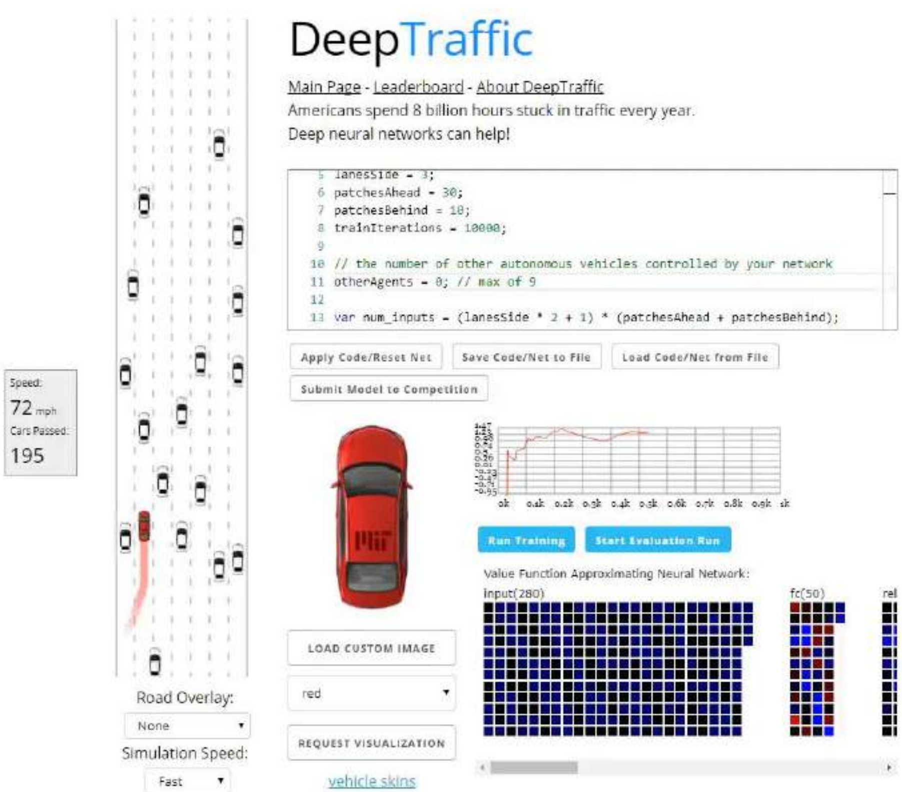
Paden B, Čáp M, Yong SZ, Yershov D, Frazzoli E. "A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles." IEEE Transactions on Intelligent Vehicles 1.1 (2016): 33-55.

Applying Deep Reinforcement Learning to Micro-Traffic Simulation



Reference: http://www.traffic-simulation.de

DeepTraffic: Deep Reinforcement Learning Competition



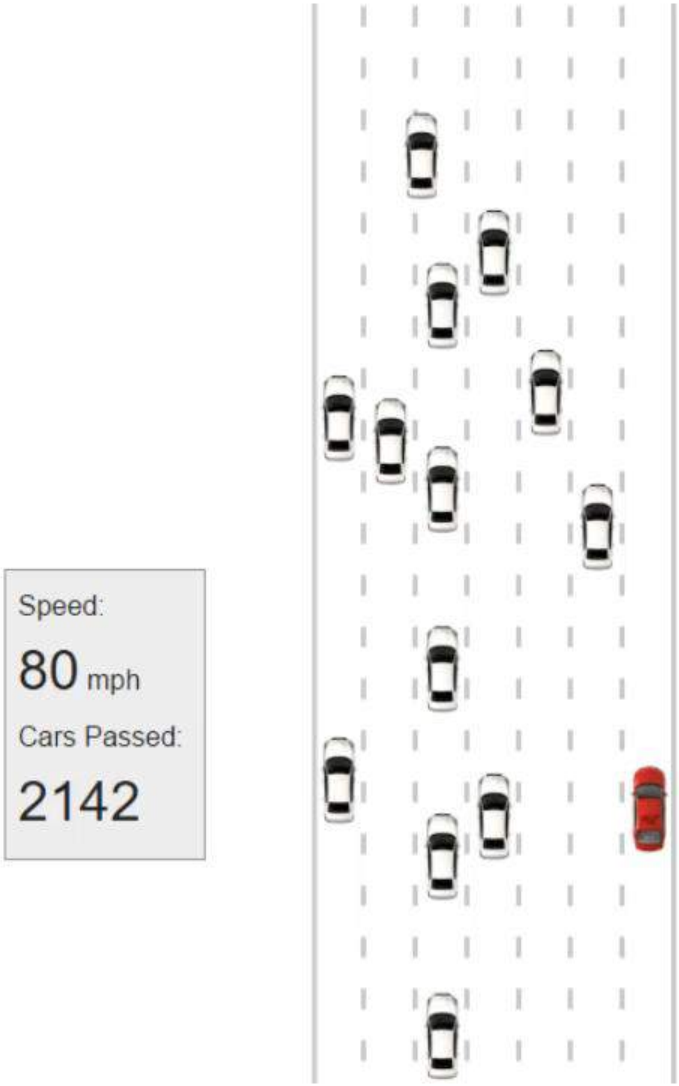
https://selfdrivingcars.mit.edu/**deeptraffic**

- **Goal:** Achieve the highest average speed over a long period of time.
- **Requirement for Students:** Follow tutorial to achieve a speed of 65mph

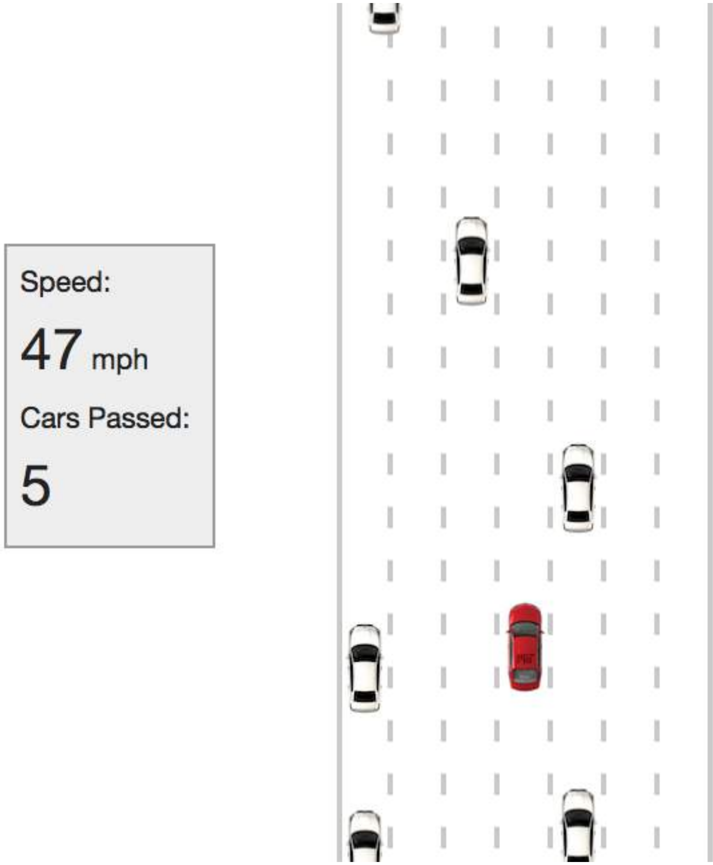
What You Should Do

- To compete:
  - Read the tutorial: https://selfdrivingcars.mit.edu/deeptraffic-about
  - Change parameters in the code box.
  - Click "Apply Code" white button. Apply Code/Reset Net
  - Click "Run Training" blue button. Run Training
  - Click "Submit Model to Competition". Submit Model to Competition
- And to visualize your submission for sharing with others:
  - Customize your image vehicle. Load Custom Image
  - Customize your color scheme. Red ▾
  - Click "Request Visualization". Request Visualization

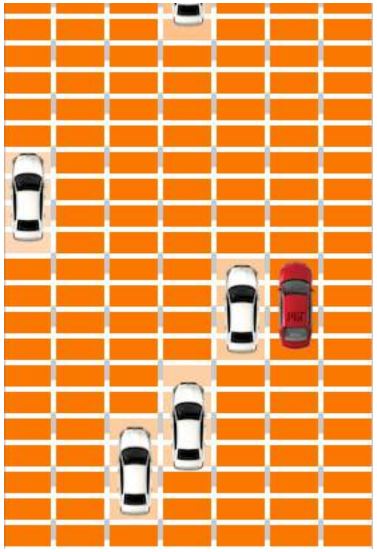
The Road, The Car, The Speed



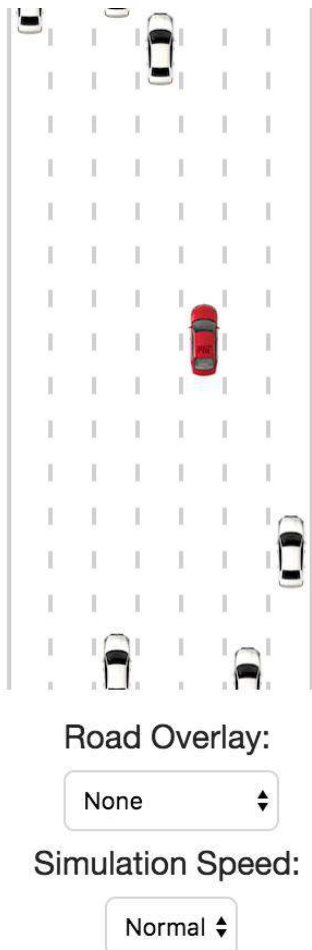
The Road, The Car, The Speed

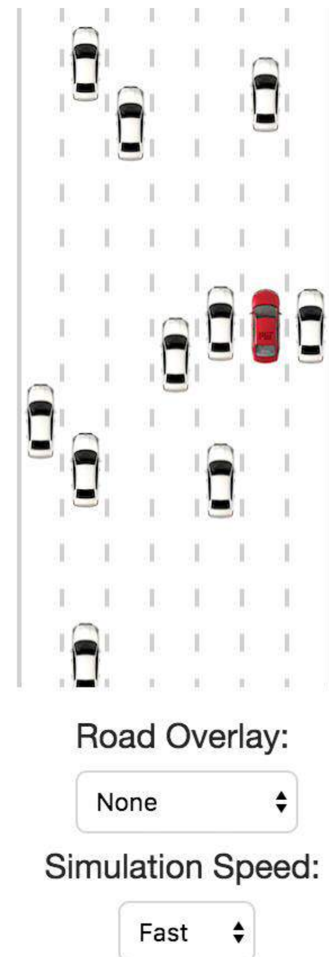


State Representation:

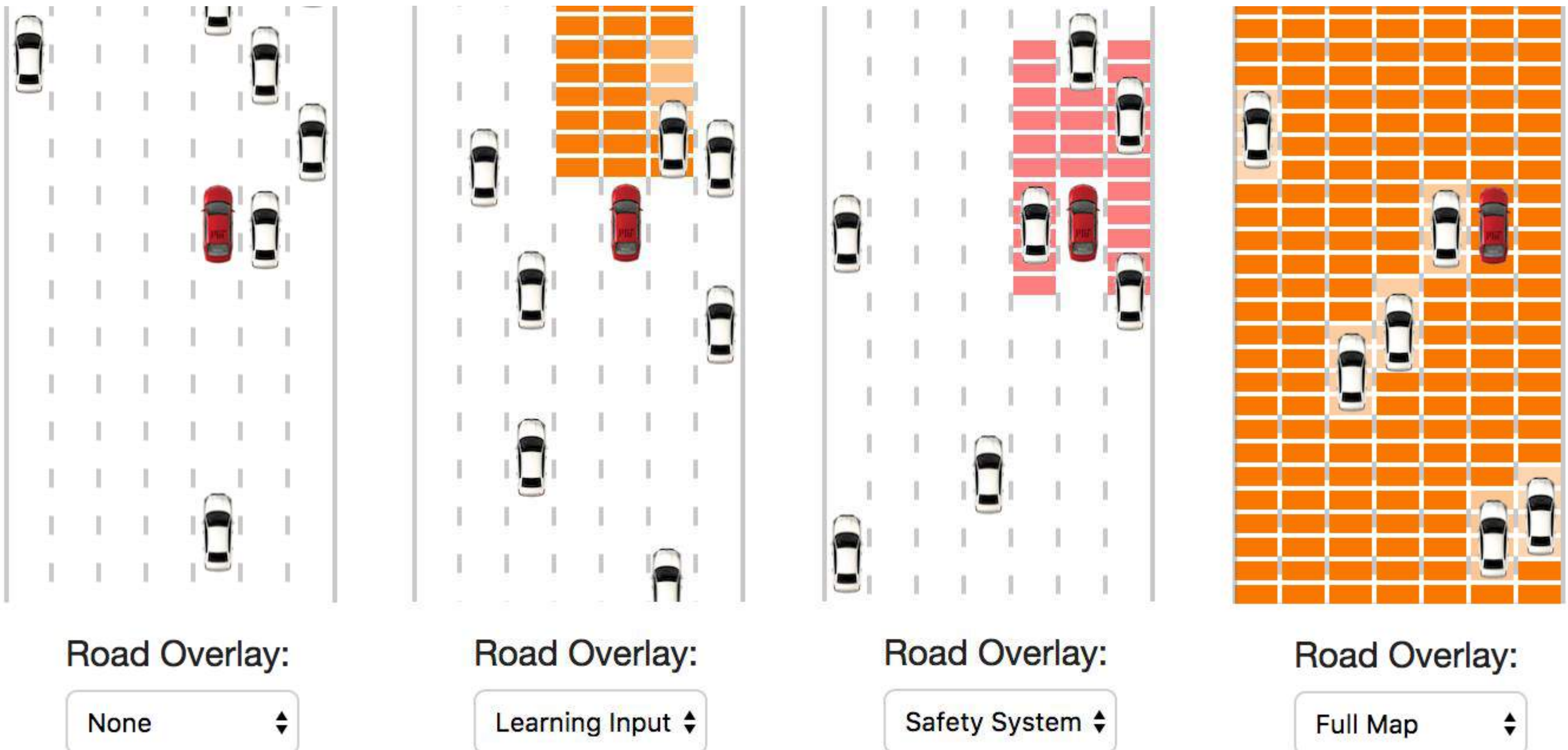


Simulation Speed

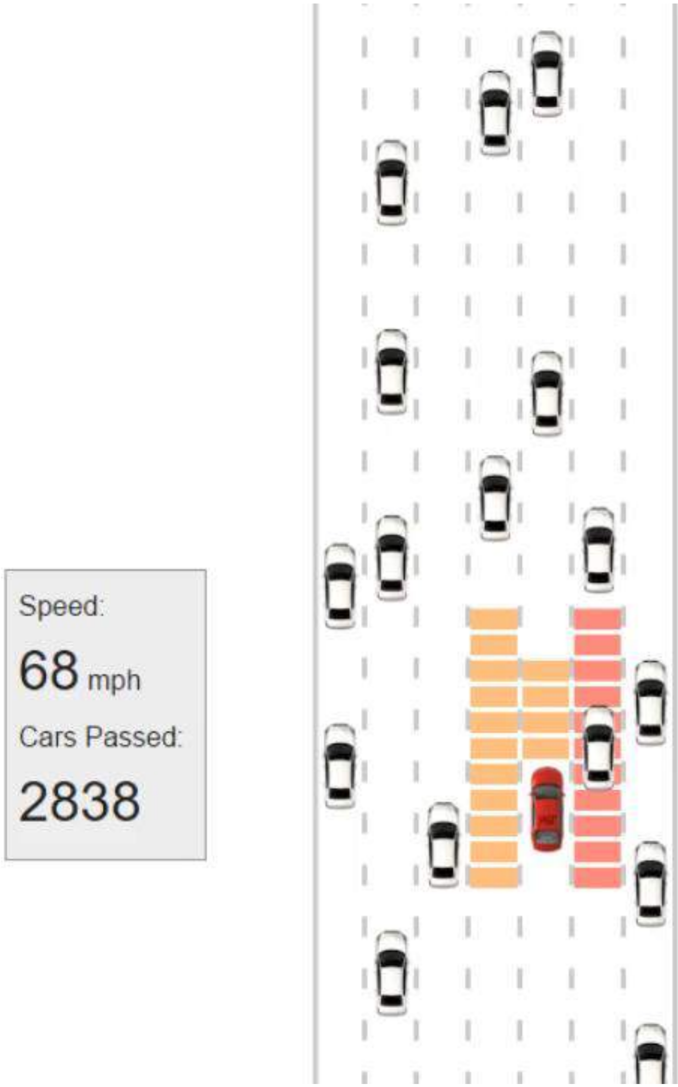


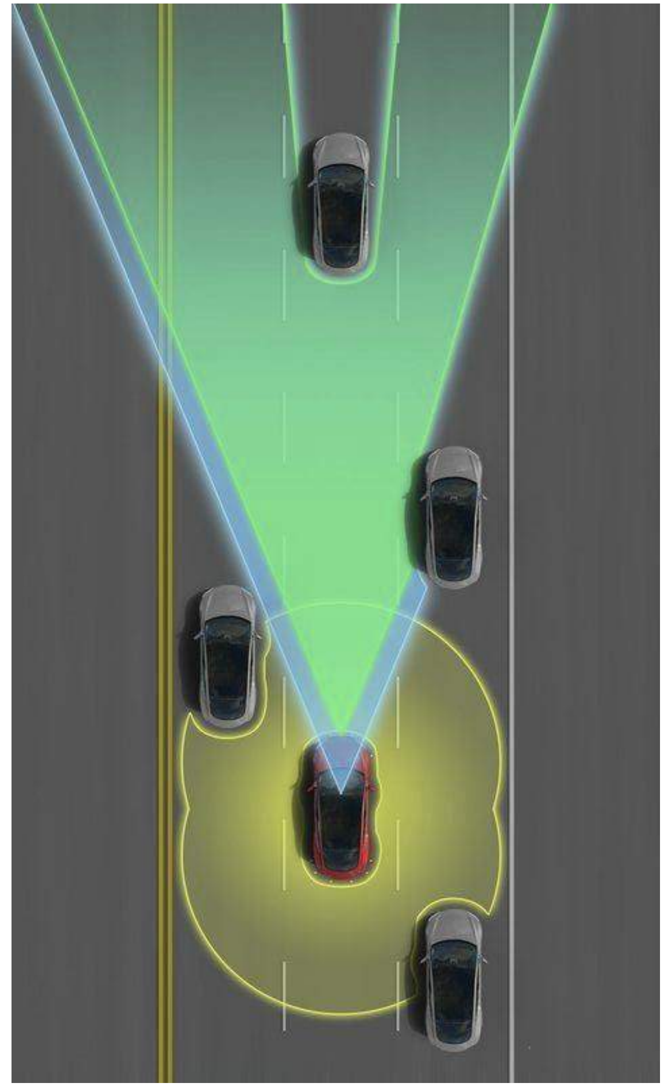


Display Options

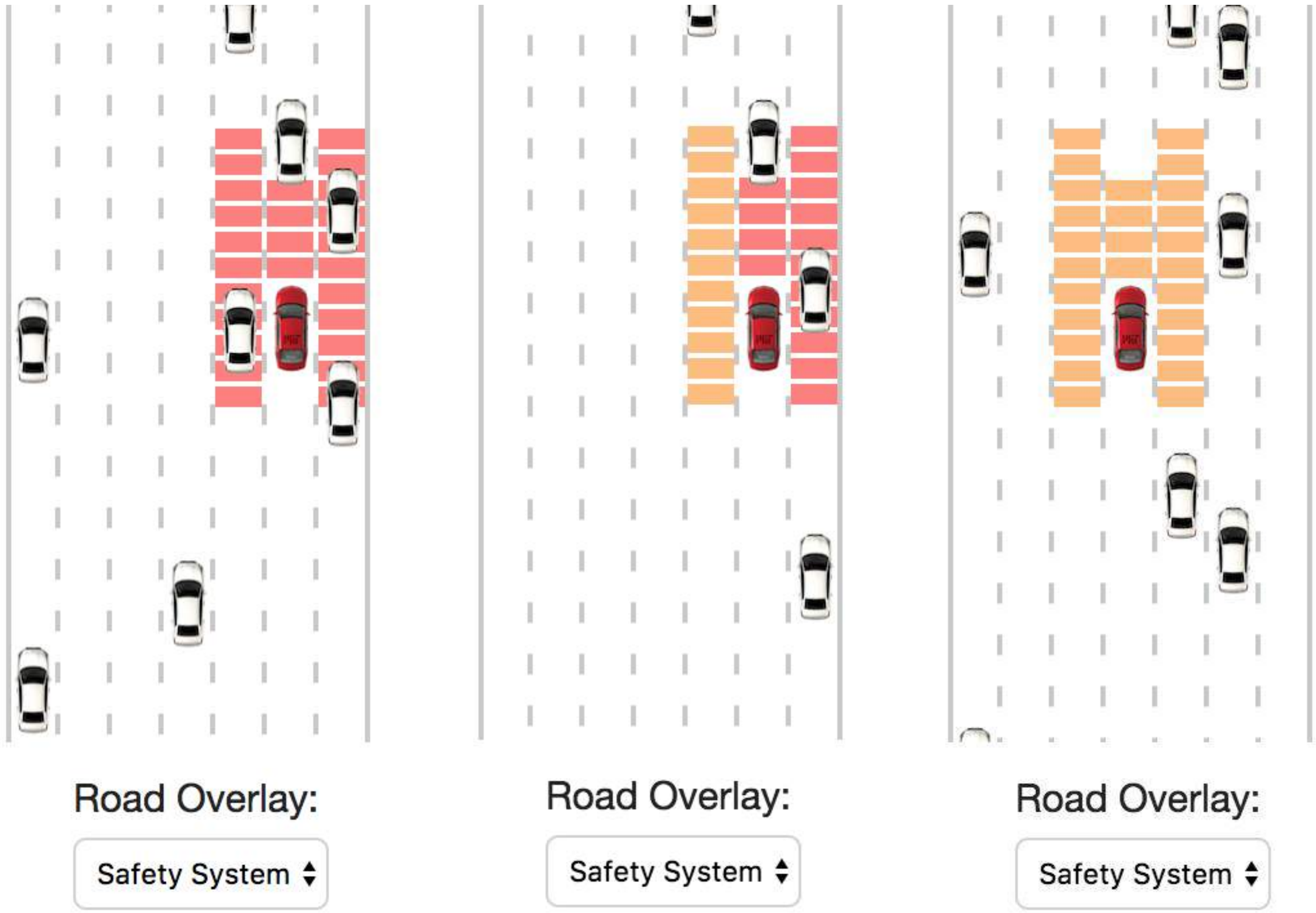


“Safety System”: Motion and Control are Given

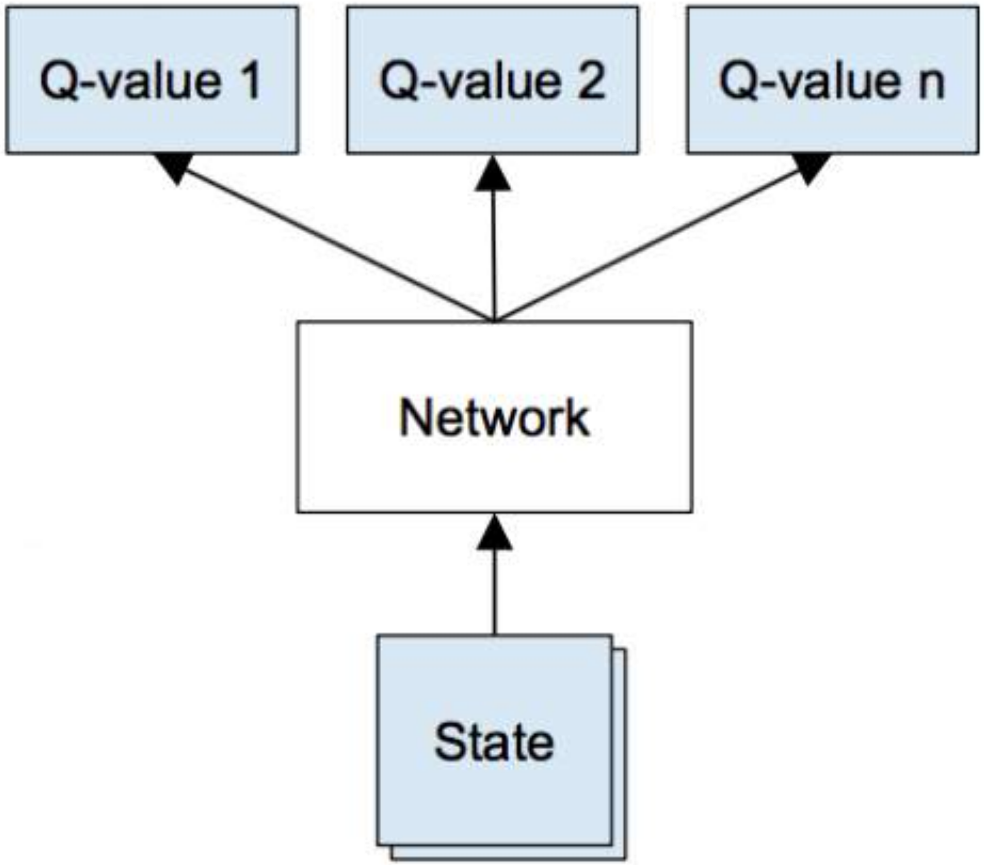




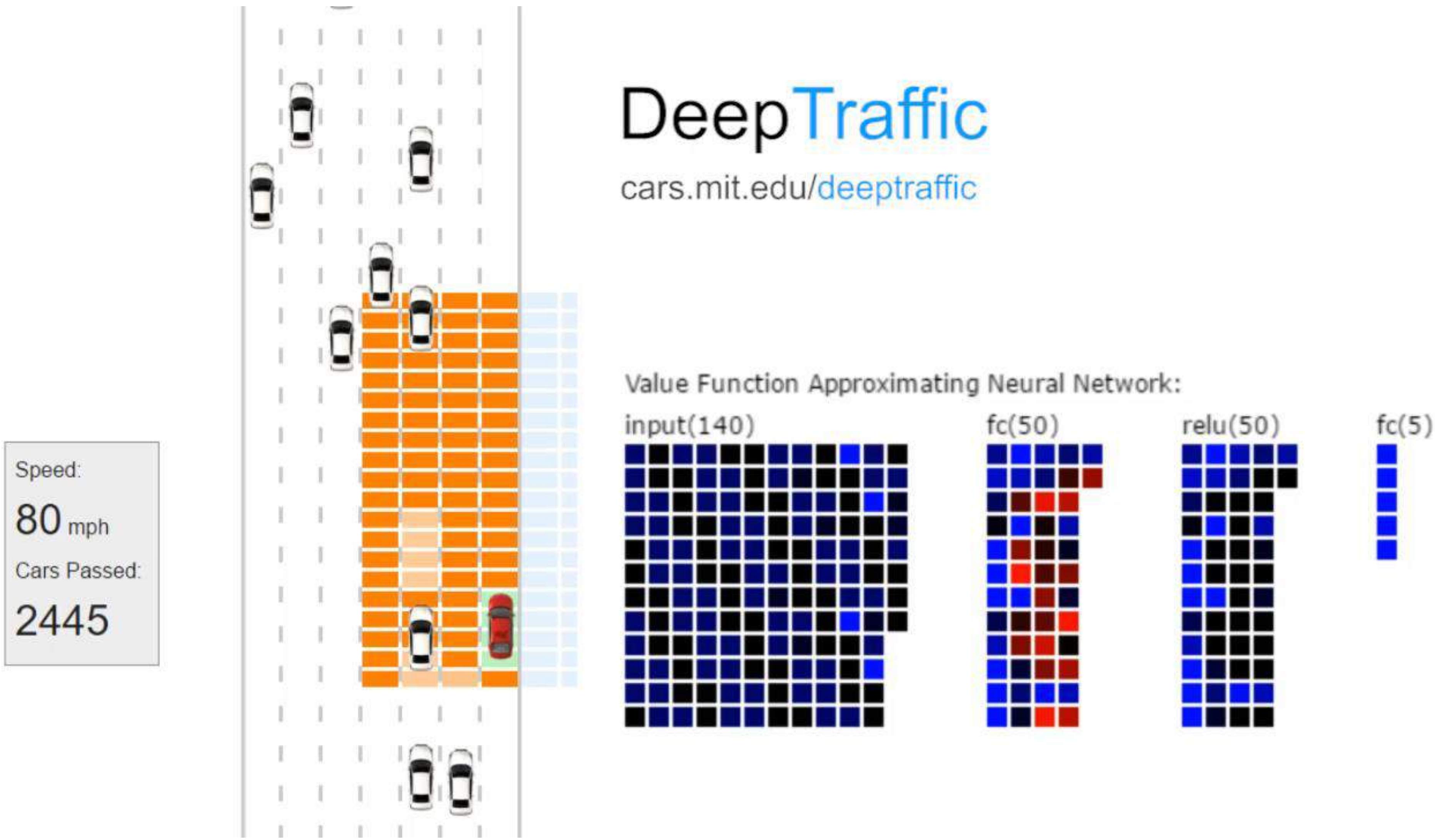
Safety System



Learning the "Behavioral Layer" Task



Learning the "Behavioral Layer" Task



Action Space


```
var noAction = 0;
var accelerateAction = 1;
var decelerateAction = 2;
var goLeftAction = 3;
var goRightAction = 4;
```

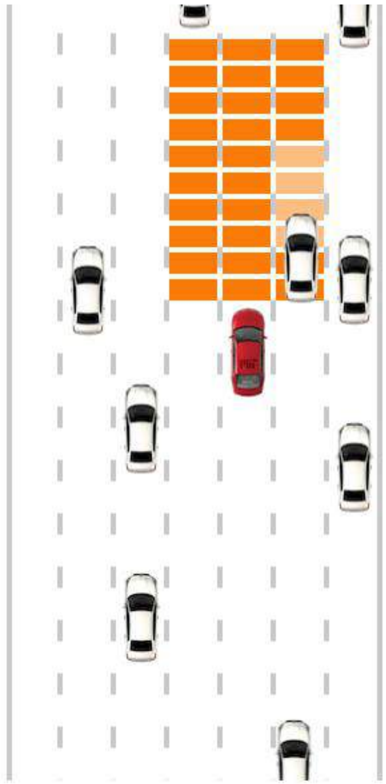
Driving / Learning

Road Overlay:

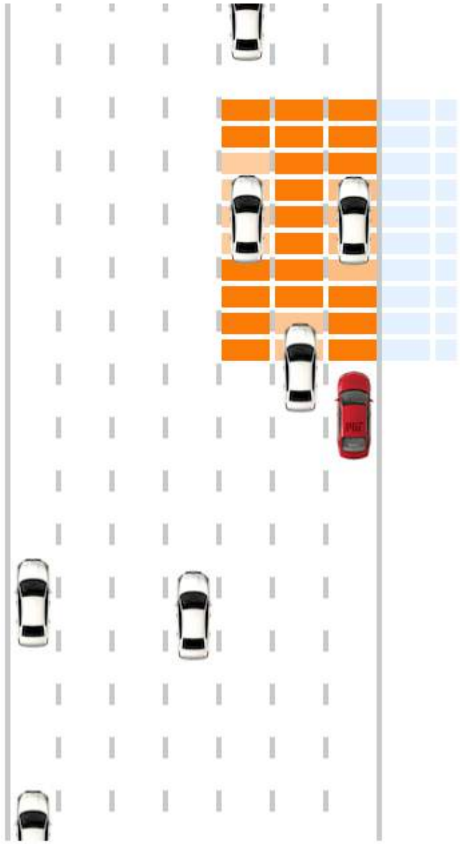
Learning Input

```
learn = function (state, lastReward) {
    brain.backward(lastReward);
    var action = brain.forward(state);
    return action;
}
```

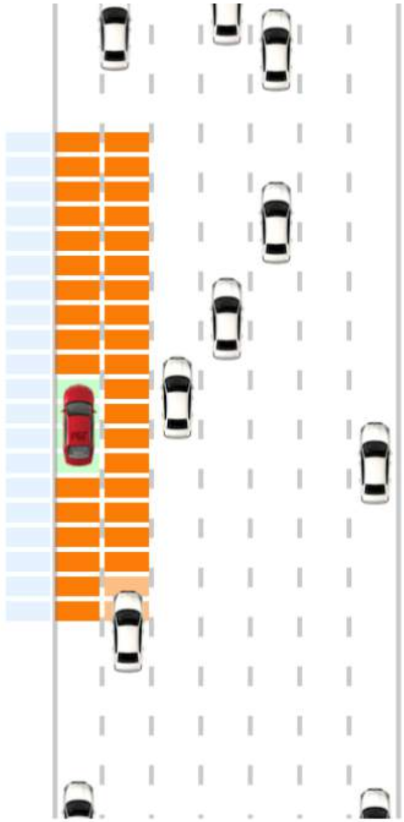
Learning Input



```
lanesSide = 1;
patchesAhead = 10;
patchesBehind = 0;
```



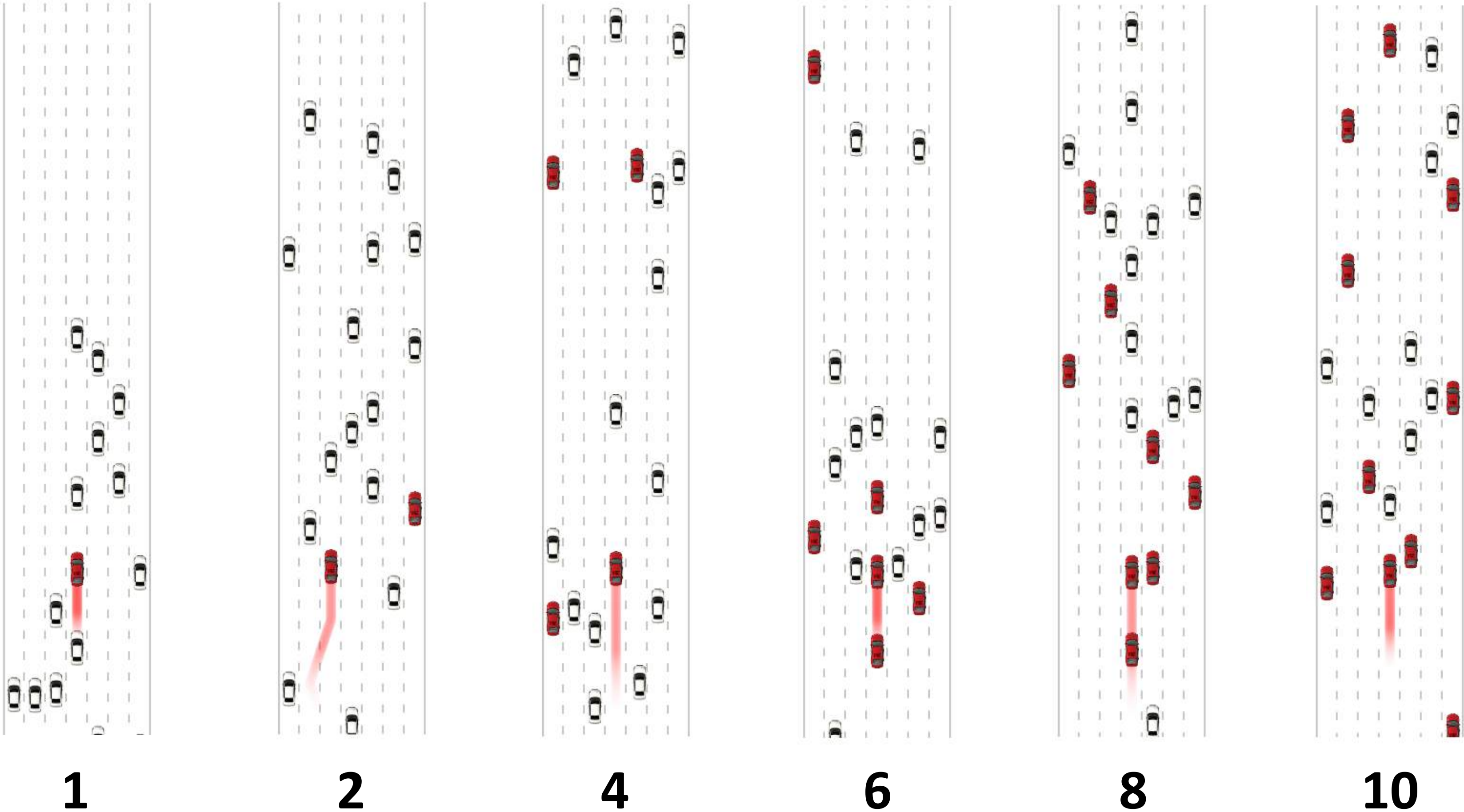
```
lanesSide = 2;
patchesAhead = 10;
patchesBehind = 0;
```



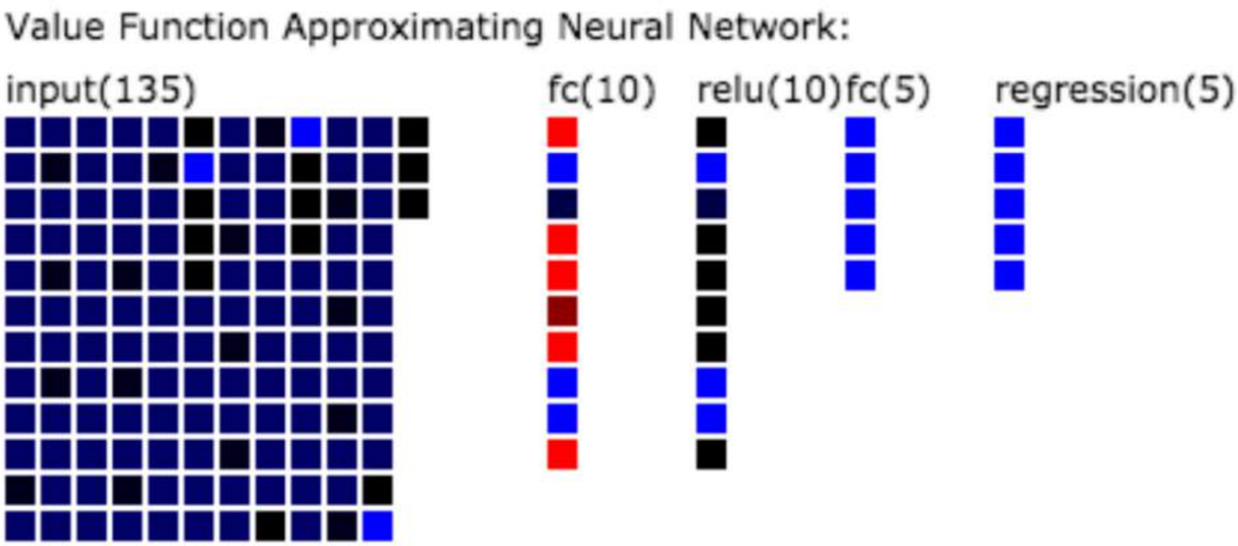
```
lanesSide = 1;
patchesAhead = 10;
patchesBehind = 10;
```

Multiple Agents

```
// the number of other autonomous vehicles controlled by your network
otherAgents = 0; // max of 9
```



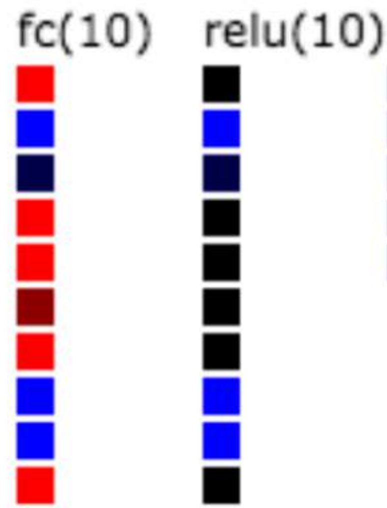
Deep RL: Q-Function Learning Parameters



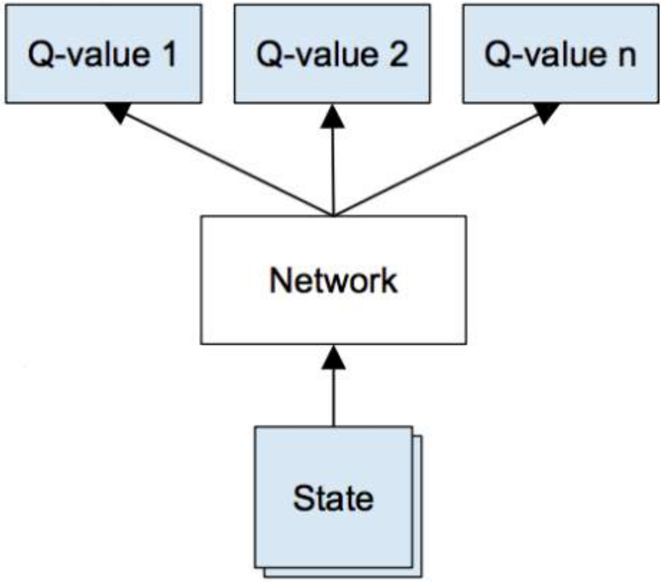

```
var num_inputs = (lanesSide * 2 + 1) * (patchesAhead + patchesBehind);
var num_actions = 5;
var temporal_window = 3;
var network_size = num_inputs * temporal_window + num_actions *
temporal_window + num_inputs;
```

Deep RL: Layers

```
layer_defs.push({
    type: 'fc',
    num_neurons: 10,
    activation: 'relu'
});
```



Deep RL: Output (Actions)




```
layer_defs.push({
    type: 'regression',
    num_neurons: num_actions
});
```

ConvNetJS: Options

```
var opt = {};
opt.temporal_window = temporal_window;
opt.experience_size = 3000;
opt.start_learn_threshold = 500;
opt.gamma = 0.7;
opt.learning_steps_total = 10000;
opt.learning_steps_burnin = 1000;
opt.epsilon_min = 0.0;
opt.epsilon_test_time = 0.0;
opt.layer_defs = layer_defs;
opt.tdtrainer_options = {
    learning_rate: 0.001, momentum: 0.0, batch_size: 64, l2_decay: 0.01
};

brain = new deepqlearn.Brain(num_inputs, num_actions, opt);
```

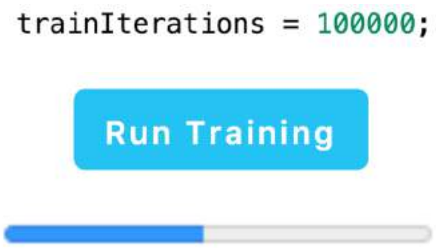
Coding/Changing the Net Layout

```

//<![CDATA[
// a few things don't have var in front of them - they update already existing variables the game needs
lanesSide = 1;
patchesAhead = 10;
patchesBehind = 10;
trainIterations = 100000;

// begin from convnetjs example
var num_inputs = (lanesSide * 2 + 1) * (patchesAhead + patchesBehind);
var num_actions = 5;
var temporal_window = 3; //1 // amount of temporal memory. 0 = agent lives in-the-moment :)
var network_size = num_inputs * temporal_window + num_actions *
```

Apply Code/Reset Net

Watch out: kills trained state!

Training



- Done on separate thread (Web Workers)
  - Separate simulation, resets, state, etc.
  - A lot faster (1000 fps +)
- Network state gets shipped to the main simulation from time to time
  - You get to see the improvements/learning live

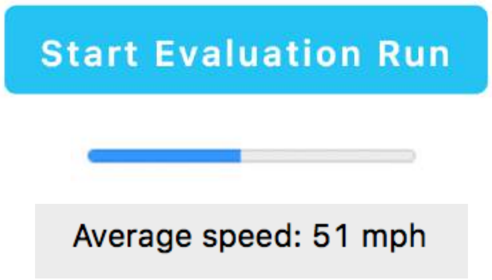
Training

```
trainIterations = 100000;
```

...

Evaluation

Local Evaluation
(Doesn't Count)



- Scoring: Average Speed
- Method:
  - Collect average speed
  - Ten runs, about 45 (simulated) minutes of game each
  - Result: median speed of the 500 runs
- Done server side after you submit
- You can try it locally to get an estimate
  - Uses exactly the same evaluation procedure/code
  - DeepTraffic 2.0: Significantly reduced the influence of randomness

Loading/Saving

Save Code/Net to File

- Danger: Overwrites all of your code and the trained net

Load Code/Net from File

Submitting Your Network

Submit Model to Competition

- Submits your code and the trained net state
    - **Make sure you ran training!**

- Adds your code to the end of a queue
    - Gets evaluated some time soon (no promises when)

- You can resubmit as often as you like
    - If your code wasn't evaluated yet it we still remove it from the queue (and move you to the end)
    - The highest score counts.

Customization and Visualization

Load Custom Image

Red ▼

Request Visualization

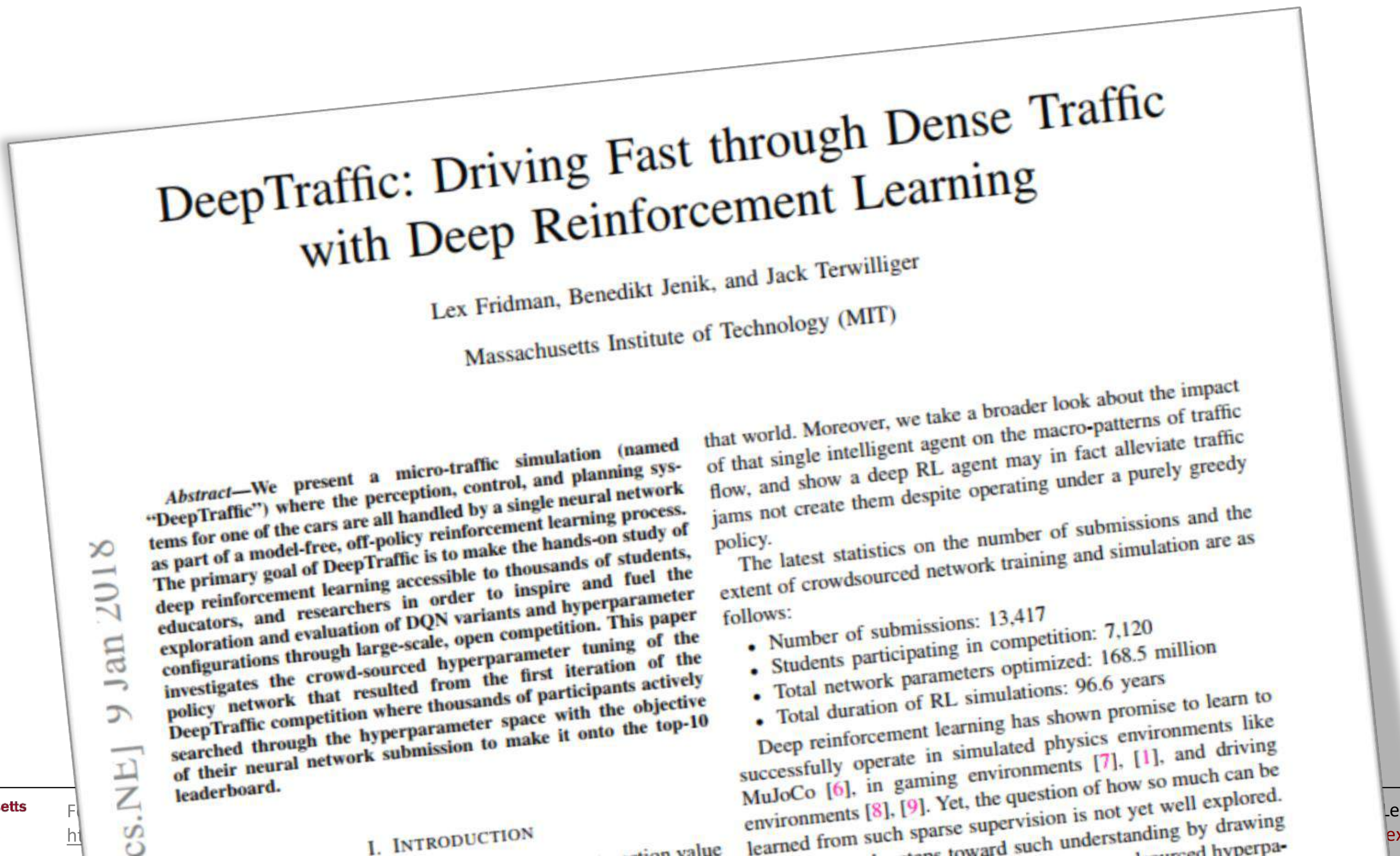
Vehicle Skins

What You Should Do

- To compete:
  - Read the tutorial: https://selfdrivingcars.mit.edu/deeptraffic-about
  - Change parameters in the code box.
  - Click "Apply Code" white button. Apply Code/Reset Net
  - Click "Run Training" blue button. Run Training
  - Click "Submit Model to Competition". Submit Model to Competition
- And to visualize your submission for sharing with others:
  - Customize your image vehicle. Load Custom Image
  - Customize your color scheme. Red ▾
  - Click "Request Visualization". Request Visualization

DeepTraffic: Deep Reinforcement Learning Competition

- **Competition:** https://github.com/lexfridman/deeptraffic
- **GitHub:** https://github.com/lexfridman/deeptraffic
- **Paper on arXiv:** https://arxiv.org/abs/1801.02805



DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning

Lex Fridman, Benedikt Jenik, and Jack Terwilliger

Massachusetts Institute of Technology (MIT)

arXiv:1801.02805v1 [cs.NE] 9 Jan 2018

Abstract—We present a micro-traffic simulation (named "DeepTraffic") where the perception, control, and planning systems for one of the cars are all handled by a single neural network as part of a model-free, off-policy reinforcement learning process. The primary goal of DeepTraffic is to make the hands-on study of deep reinforcement learning accessible to thousands of students, educators, and researchers in order to inspire and fuel the exploration and evaluation of DQN variants and hyperparameter configurations through large-scale, open competition. This paper investigates the crowd-sourced hyperparameter tuning of the policy network that resulted from the first iteration of the DeepTraffic competition where thousands of participants actively searched through the hyperparameter space with the objective of their neural network submission to make it onto the top-10 leaderboard.

I. Introduction

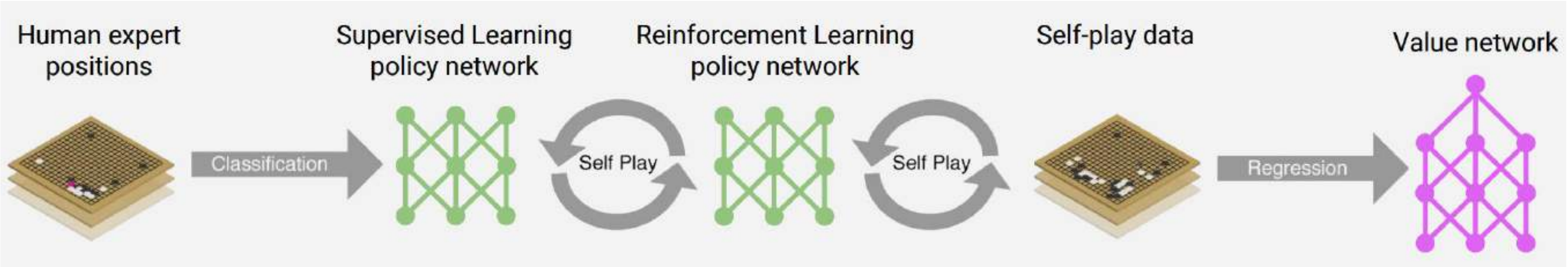
that world. Moreover, we take a broader look about the impact of that single intelligent agent on the macro-patterns of traffic flow, and show a deep RL agent may in fact alleviate traffic jams not create them despite operating under a purely greedy policy.

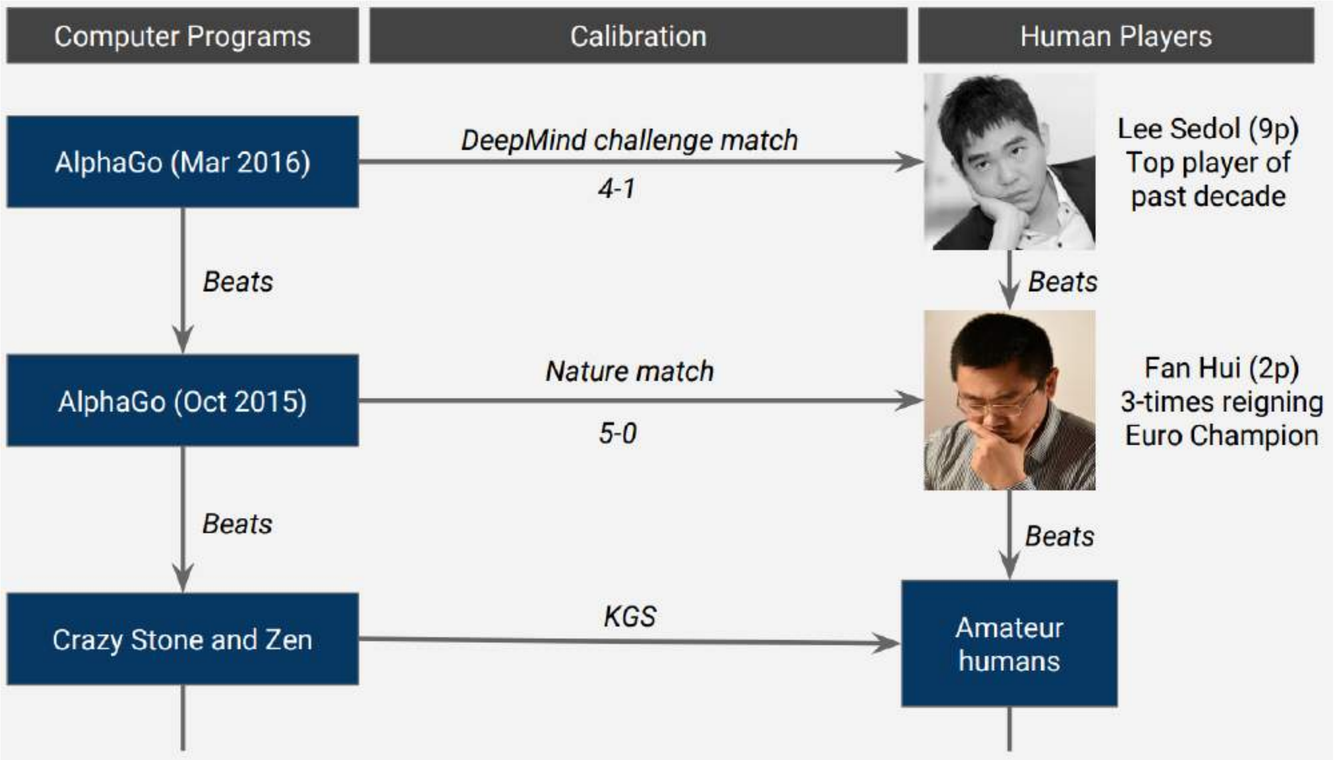
The latest statistics on the number of submissions and the extent of crowdsourced network training and simulation are as follows:

- Number of submissions: 13,417
- Students participating in competition: 7,120
- Total network parameters optimized: 168.5 million
- Total duration of RL simulations: 96.6 years

Deep reinforcement learning has shown promise to learn to successfully operate in simulated physics environments like MuJoCo [6], in gaming environments [7], [1], and driving environments [8], [9]. Yet, the question of how so much can be learned from such sparse supervision is not yet well explored.

Human-in-the-Loop Reinforcement Learning:
Driving Ready?

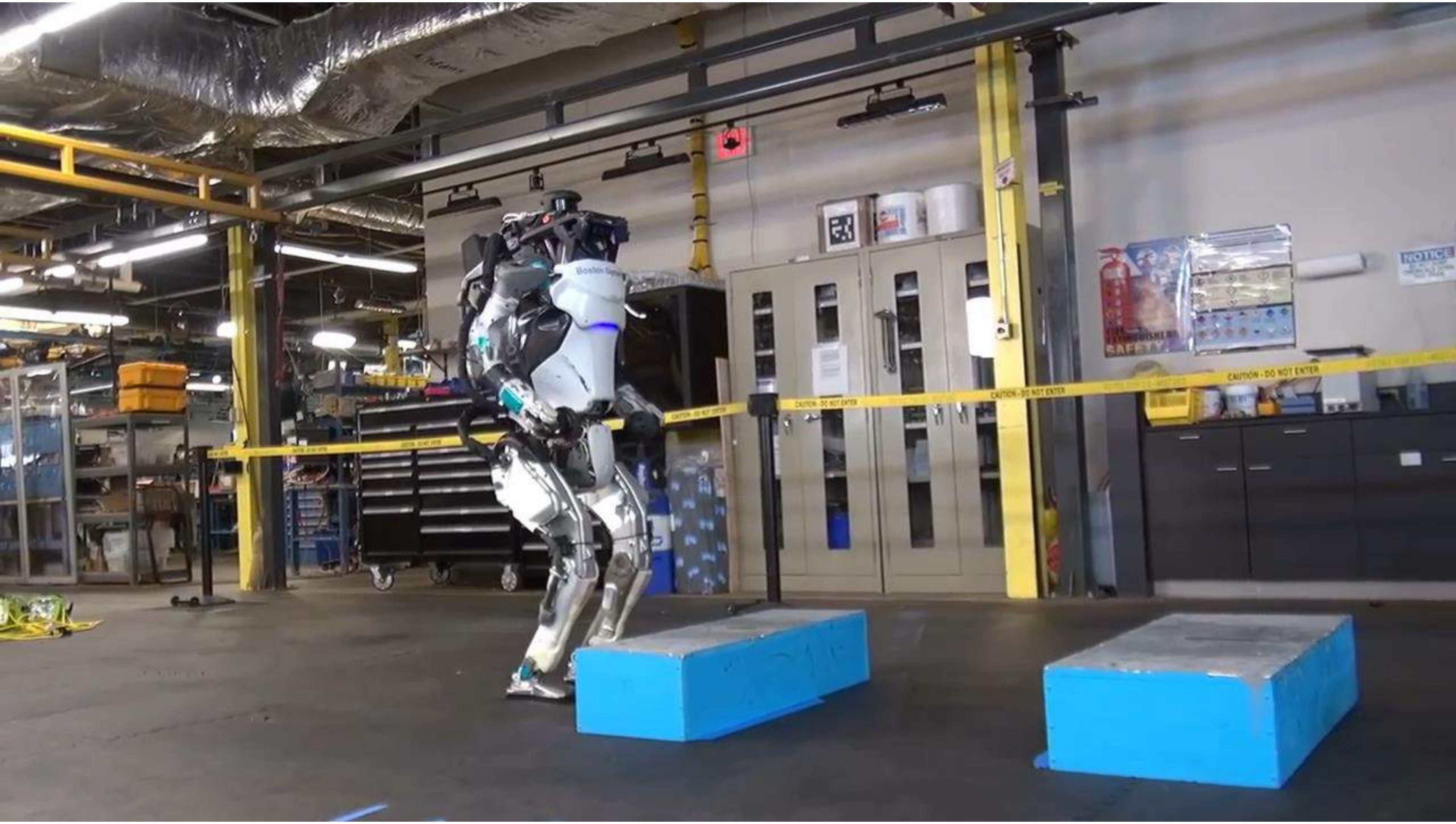




To date, for most successful robots operating in the real world:

Deep RL is not involved

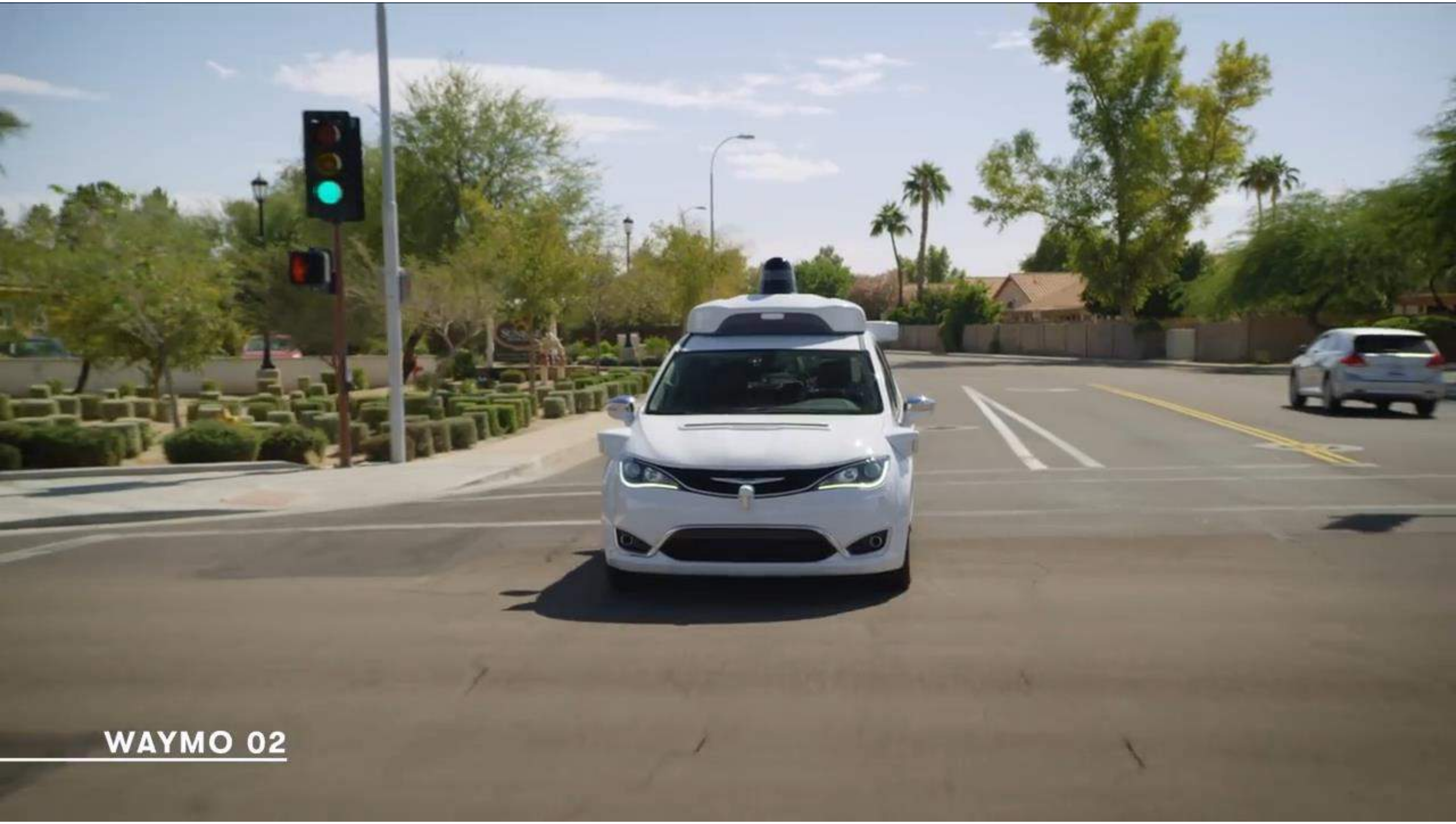
(to the best of our knowledge)



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

We will explore more about bias, safety, and ethics in:
MIT 6.S099 Artificial General Intelligence
https://agi.mit.edu

Thank You

Next lecture: ***Computer Vision***