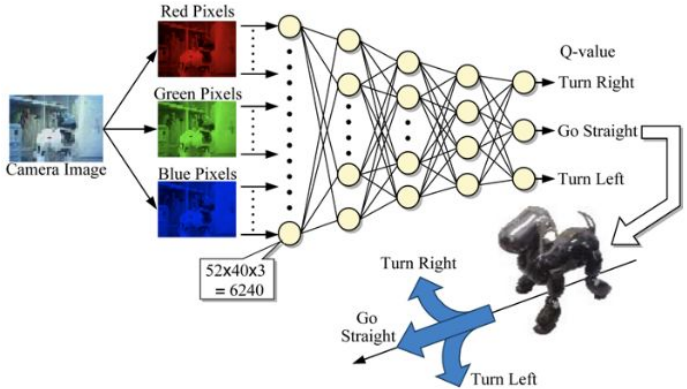
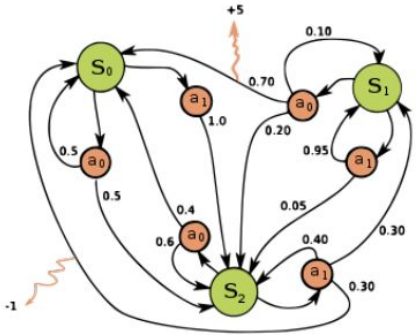
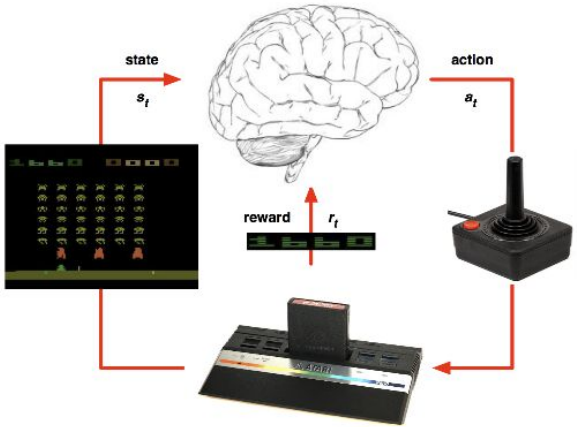
Reinforcement Learning in Unity

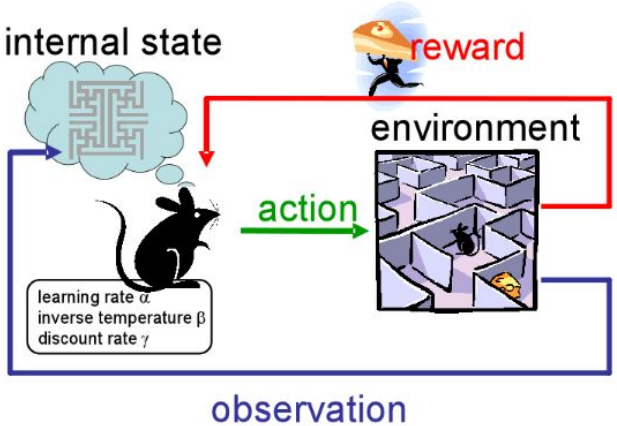
William Edward Hahn, PhD

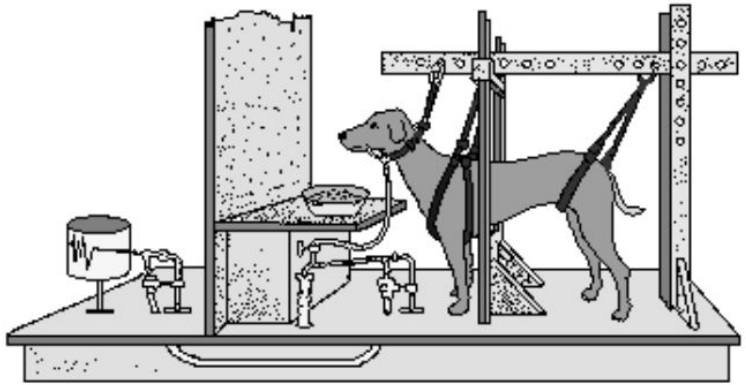
Machine Perception and Cognitive Robotics Laboratory
Florida Atlantic University


Red Pixels
Green Pixels
Blue Pixels
Camera Image
52x40x3
= 6240
Q-value
Turn Right
Go Straight
Turn Left
Turn Right
Go
Straight
Turn Left

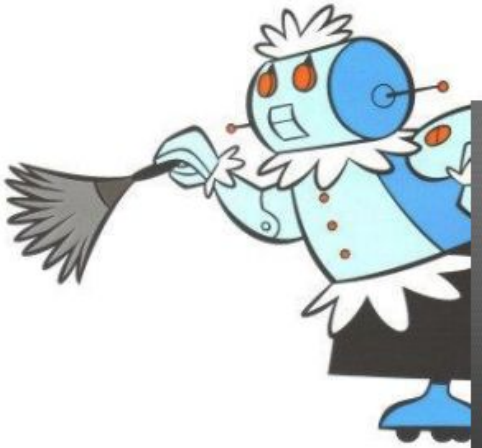

+5
-1
S0
S1
S2
a0
a1
0.10
0.70
1.0
0.20
0.95
0.5
0.5
0.4
0.6
0.05
0.40
0.30
0.30


state
s_t
action
a_t
reward
r_t


internal state
reward
environment
action
learning rate α
inverse temperature β
discount rate γ
observation



How do we build intelligent machines?



Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain.

- Alan Turing

Electronic Brain

One way of setting about our task of building a 'thinking machine' would be to take a man as a whole and to try to replace all the parts of him by machinery. He would include television cameras, microphones, loudspeakers, wheels and 'handling servo-mechanisms' as well as some sort of 'electronic brain'. This would of course be a tremendous undertaking. The object if produced by present techniques would be of immense size, even if the 'brain' part were stationary and controlled the body from a distance. In order that the machine should have a chance of finding things out for itself it should be allowed to roam the countryside...

- Alan Turing 1948

TYPES OF MACHINE LEARNING

Supervised Learning

Train an algorithm to perform classifcation and regression with a labelled data set.

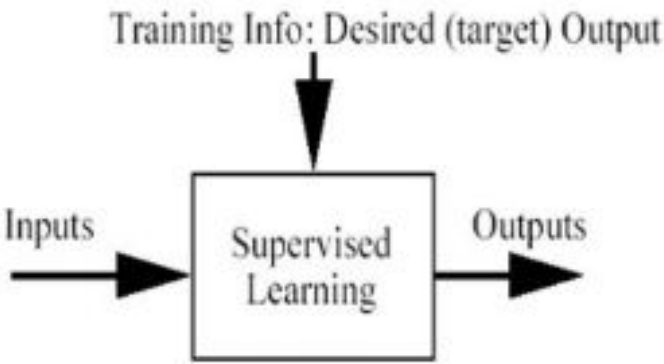
Unsupervised Learning

Train an algorithm to find clusters and associations in an unlabelled data set.

Reinforcement Learning

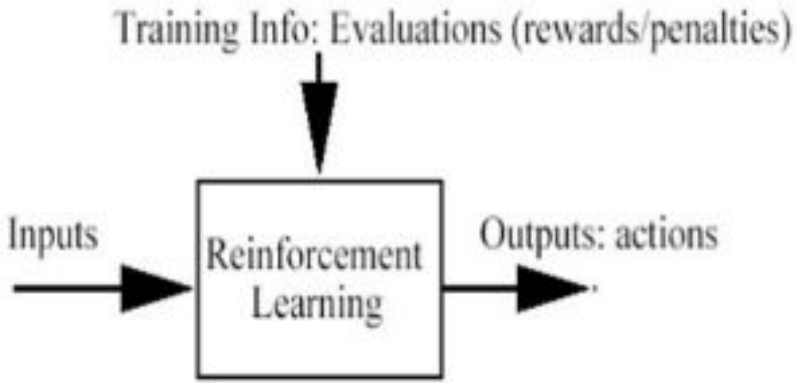
Train an agent to take certain actions in an environment without a data set.

Supervised Learning

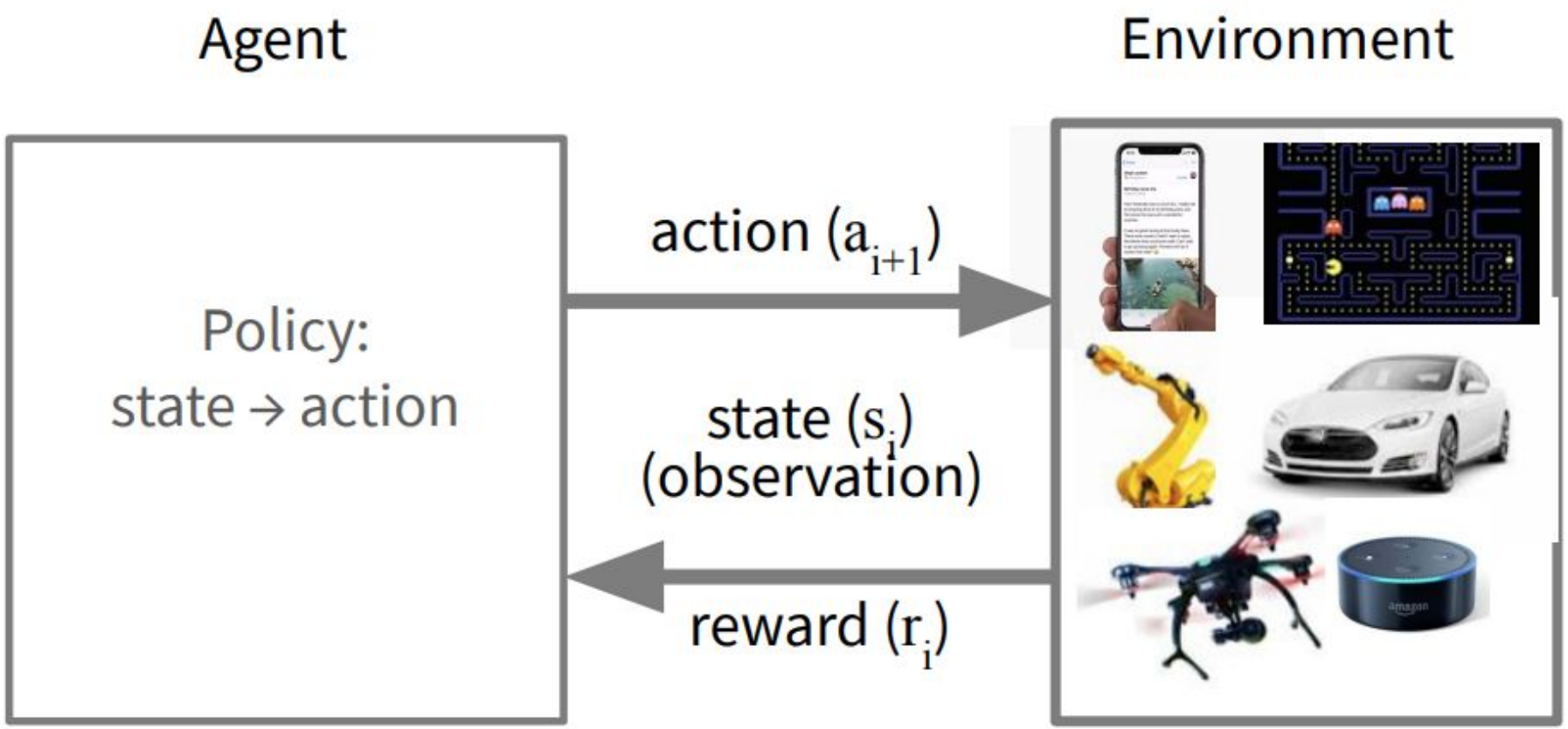


Error = (target output - actual output)

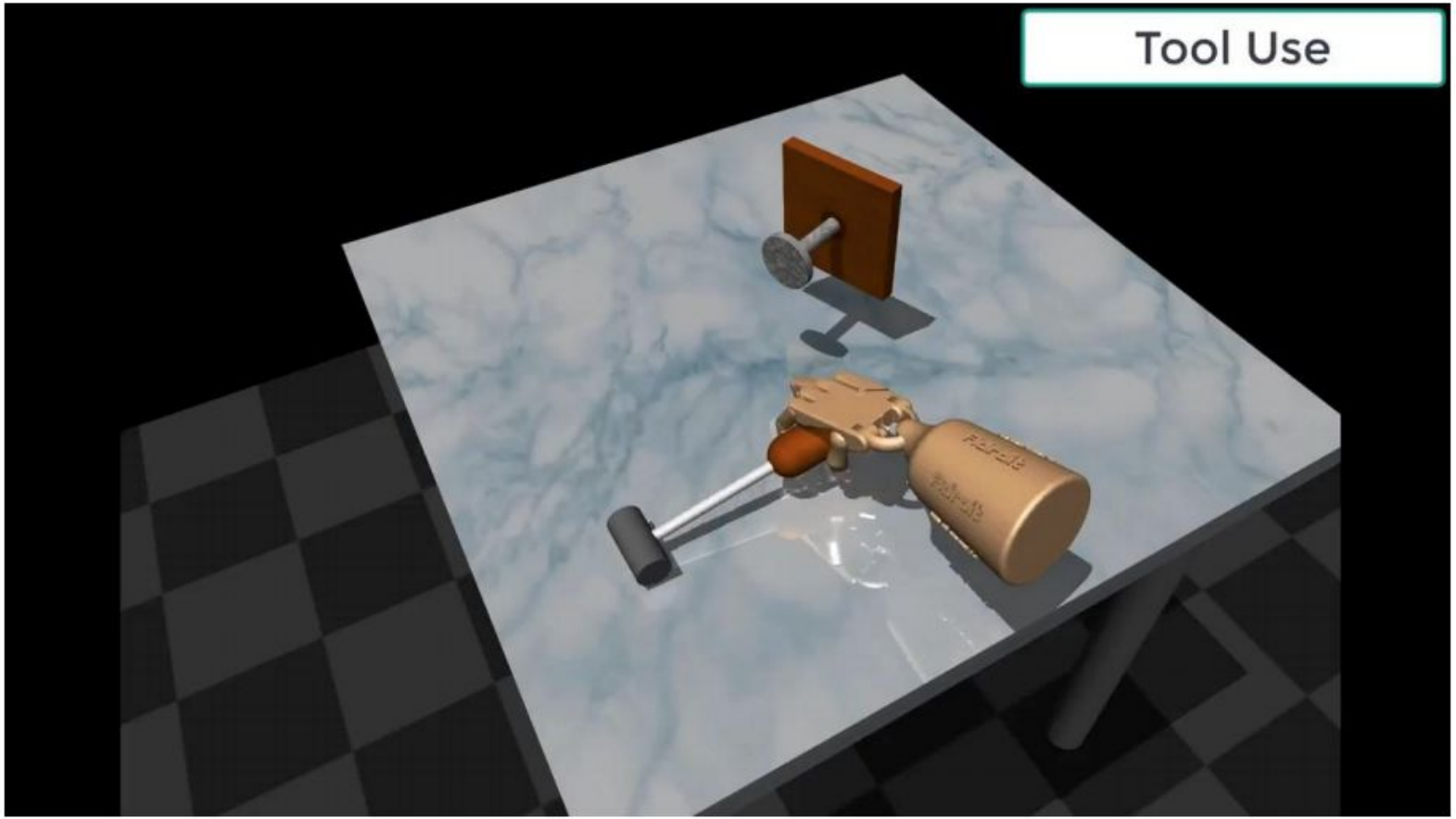
Reinforcement Learning

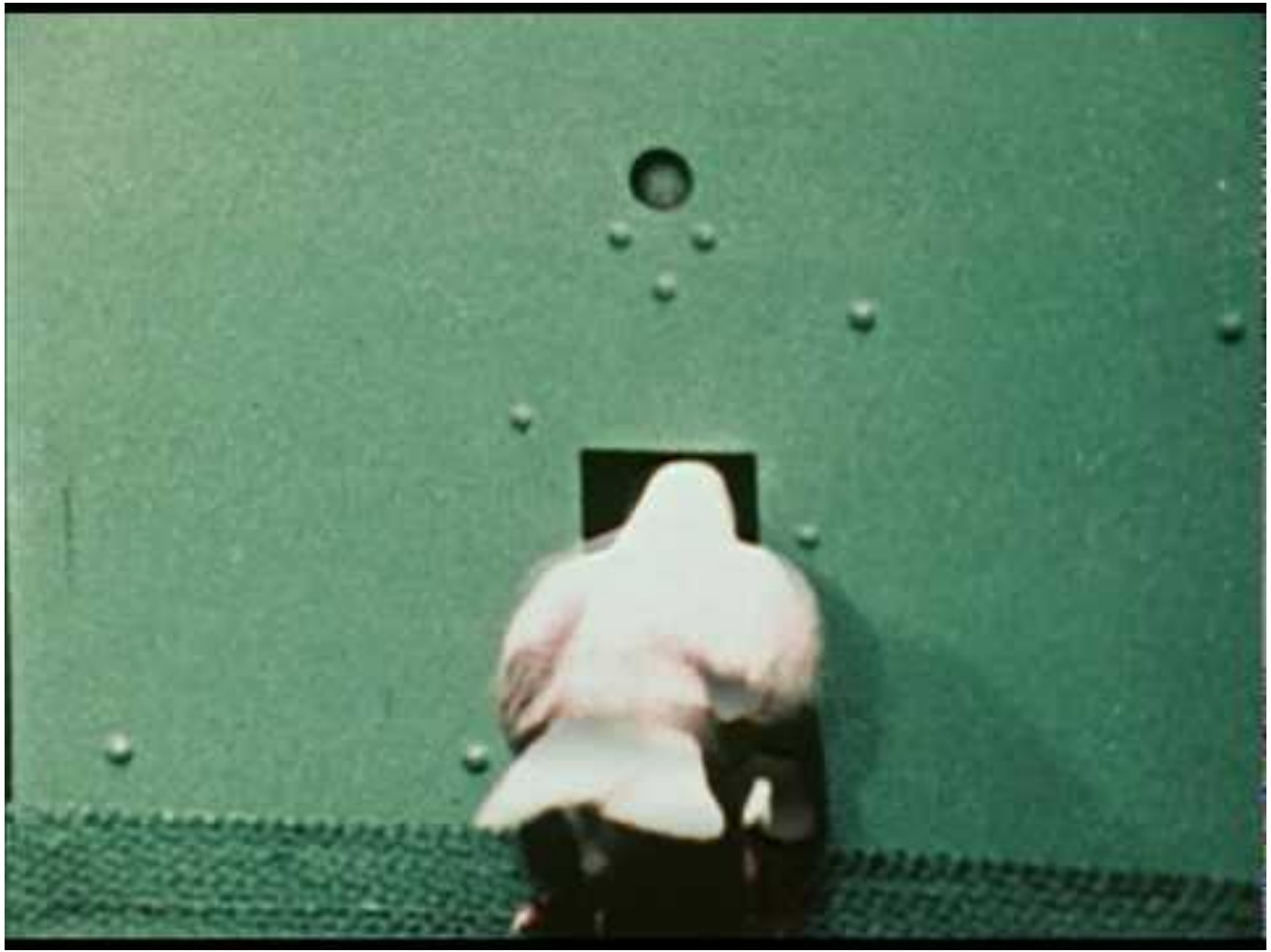


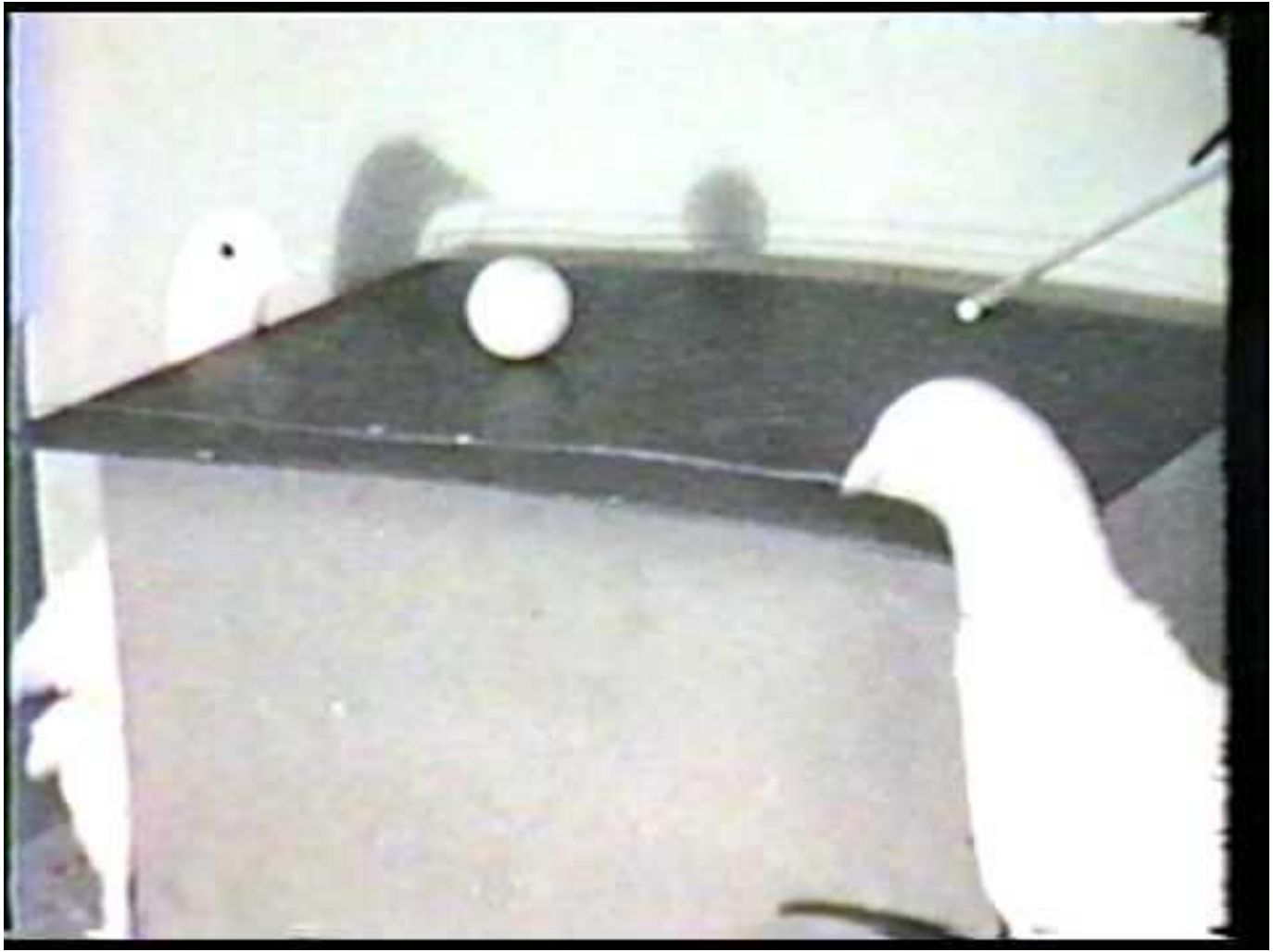
Objective: Get as much reward as possible

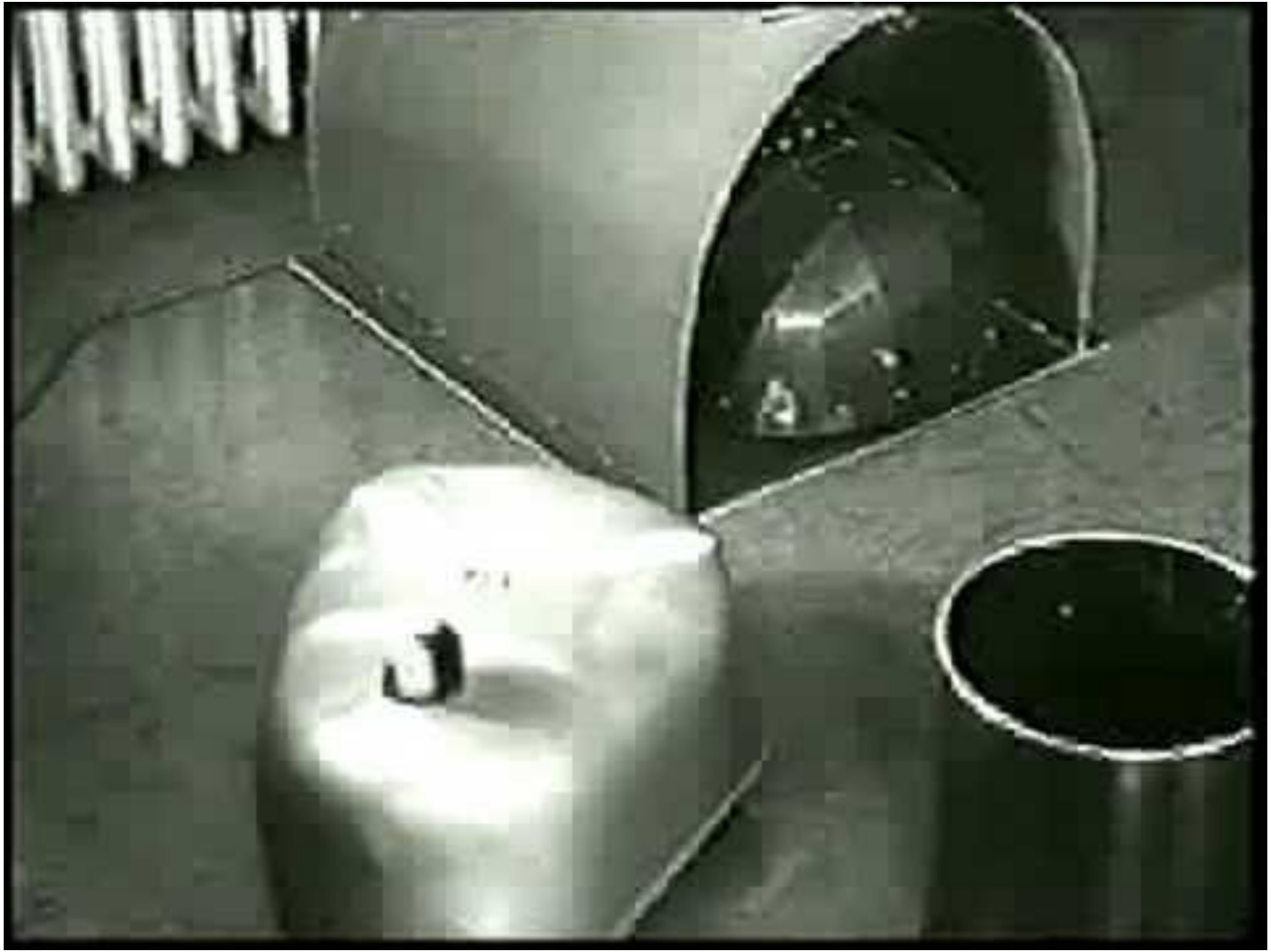

Agent
Environment
Policy:
state → action
action (a_{i+1})
state (s_i)
(observation)
reward (r_i)

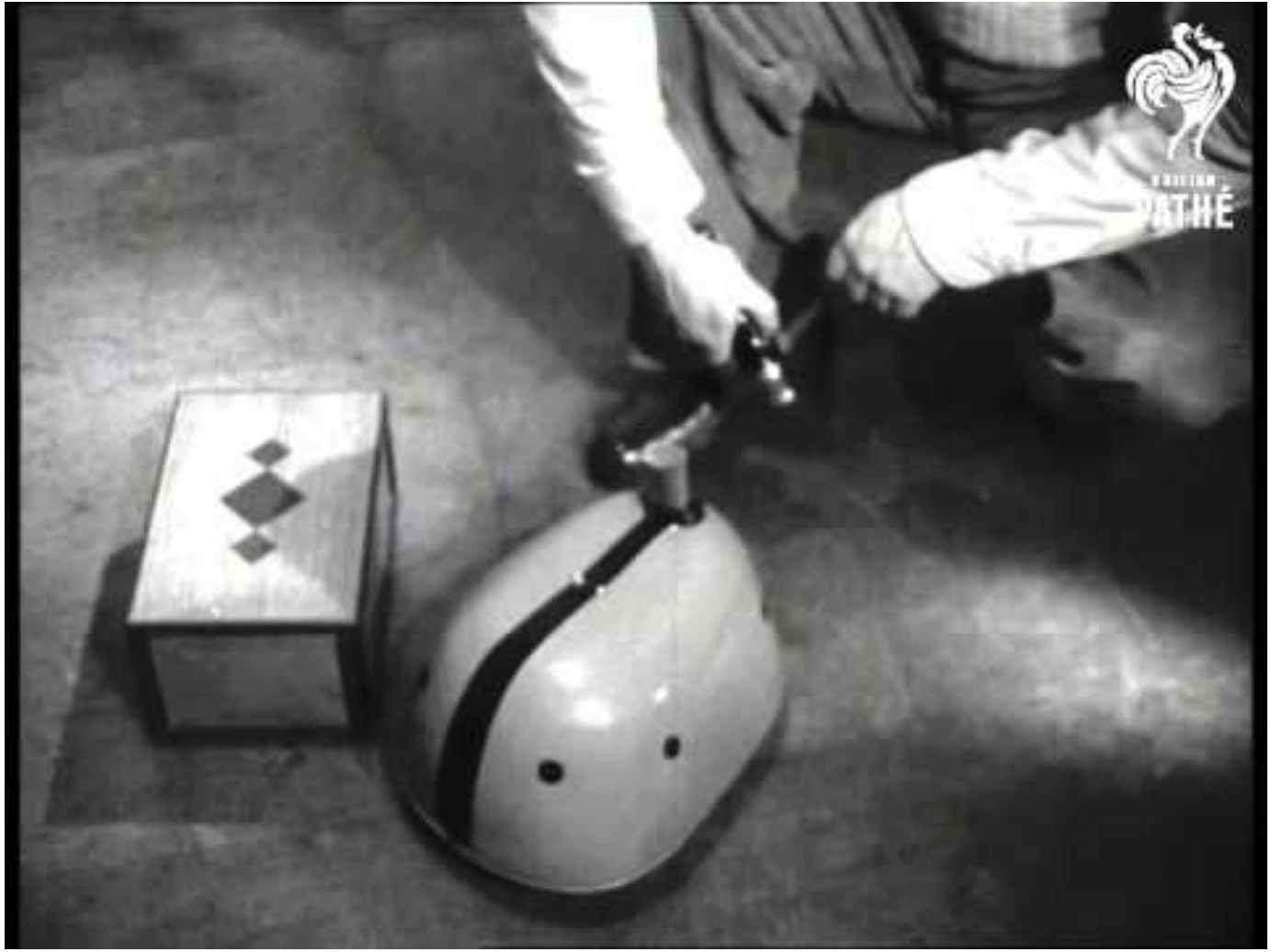
Complex physical tasks…

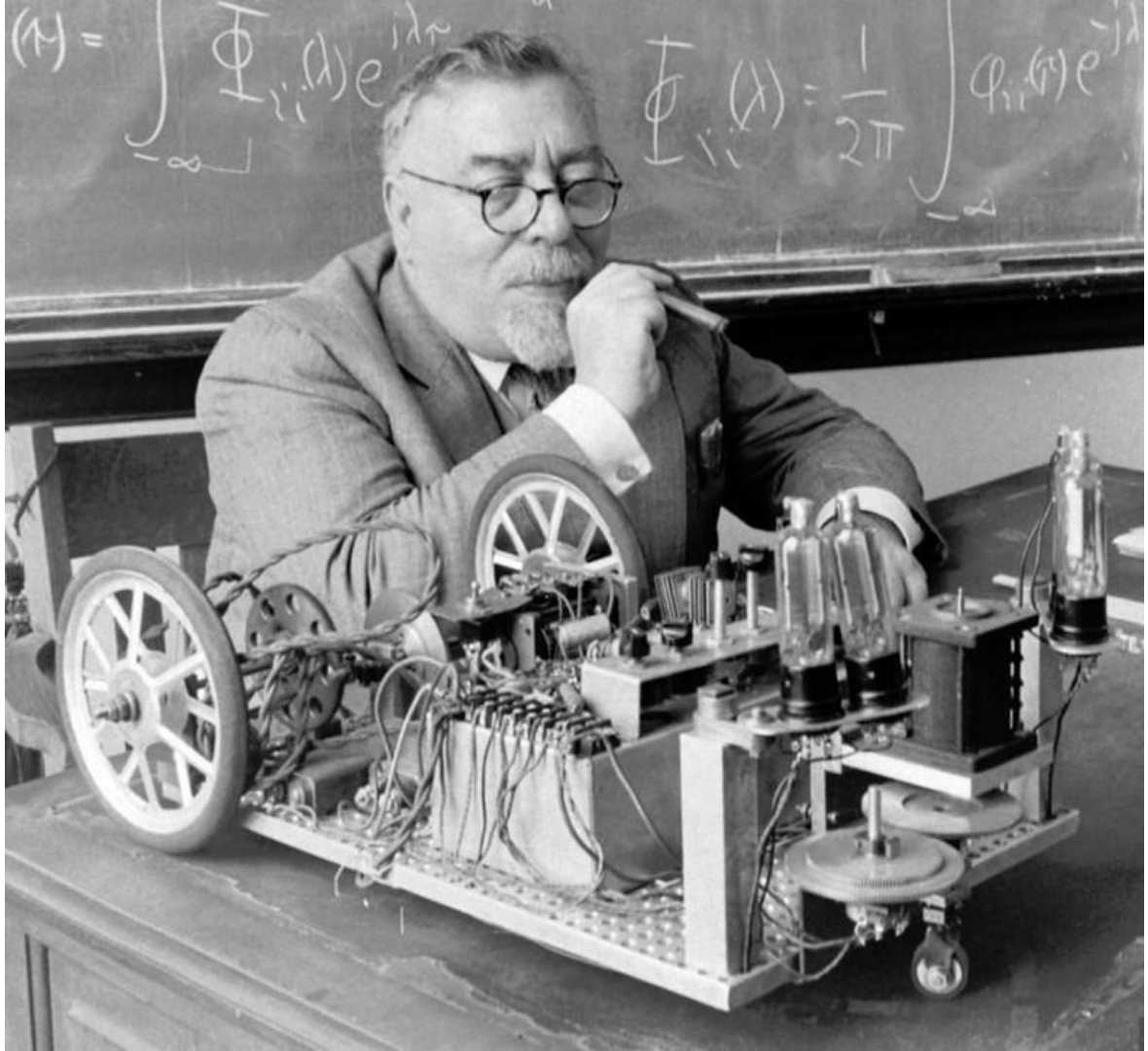









PATHÉ



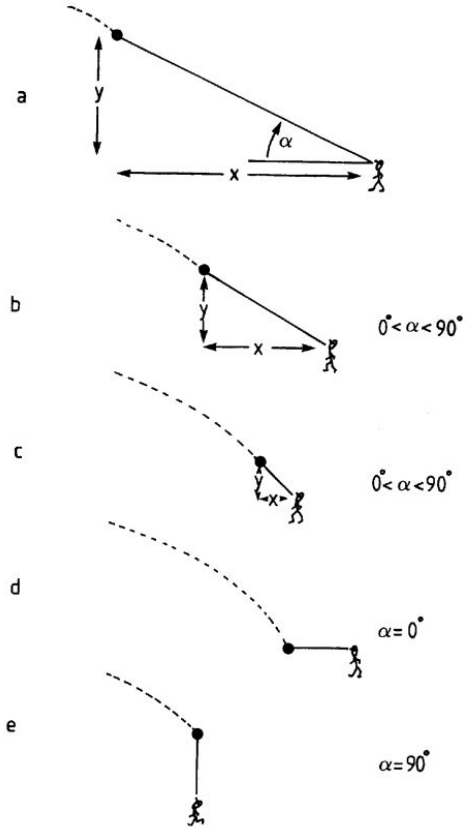


Figure 6. The conditions for intercepting or missing the ball. (a) α is the angle of elevation of gaze as the fielder watched the ball. If the fielder ran at a speed that kept α positive but less than 90° as the ball fell (b and c), it would be intercepted. If α fell to 0° or reached 90°, the ball would not be intercepted because it would have fallen in front (d) or gone over the fielder's head (e).

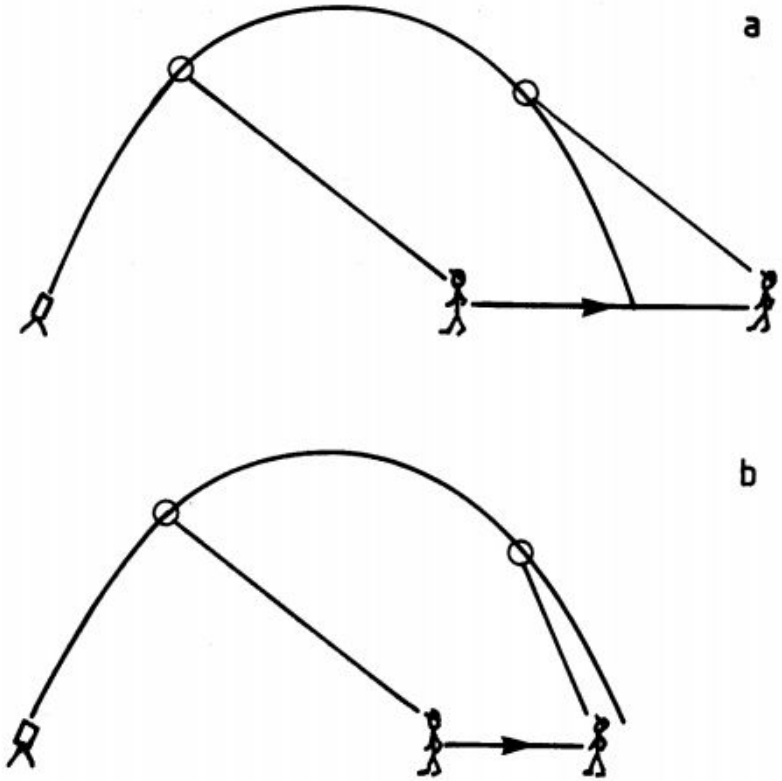
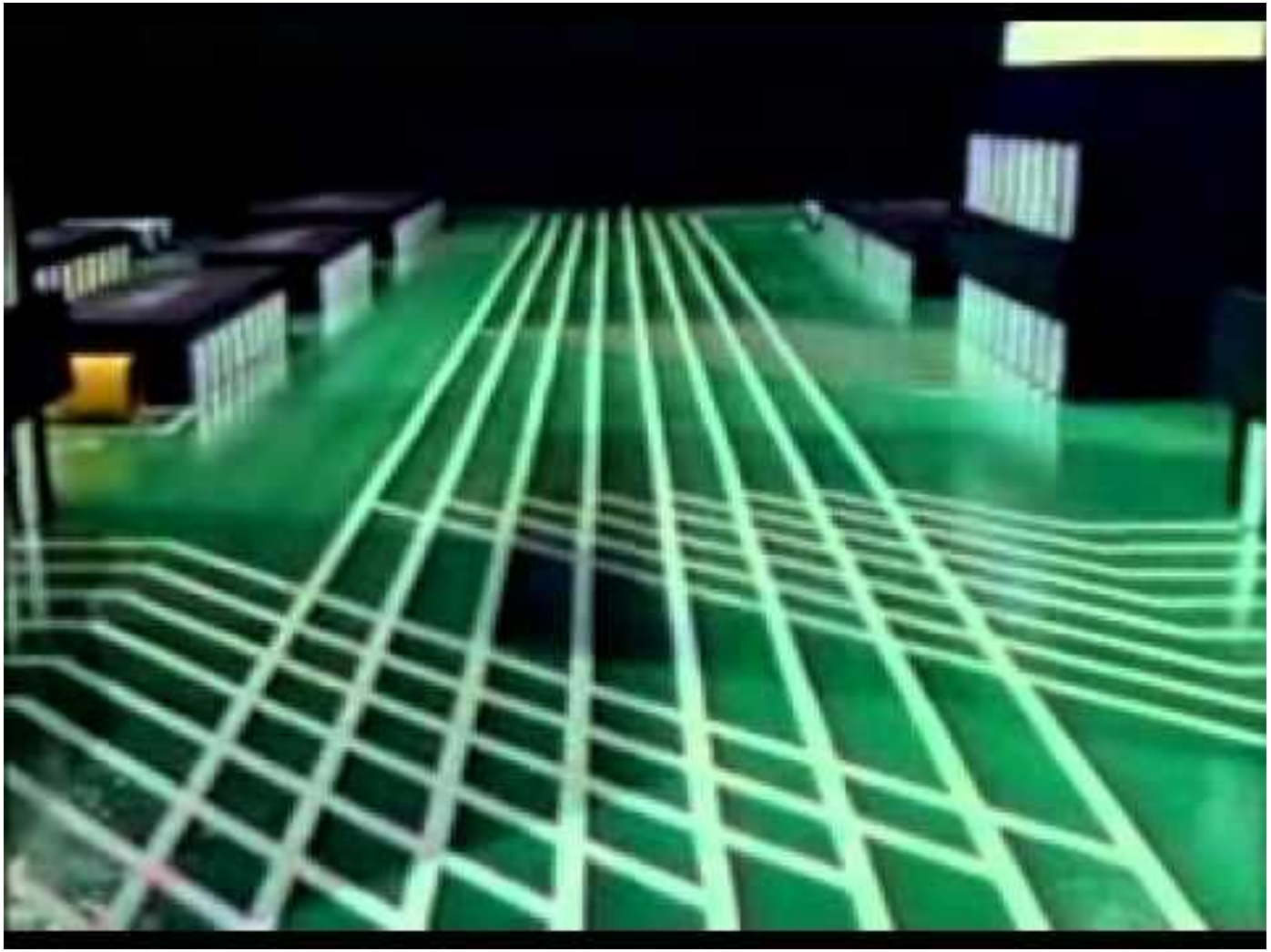
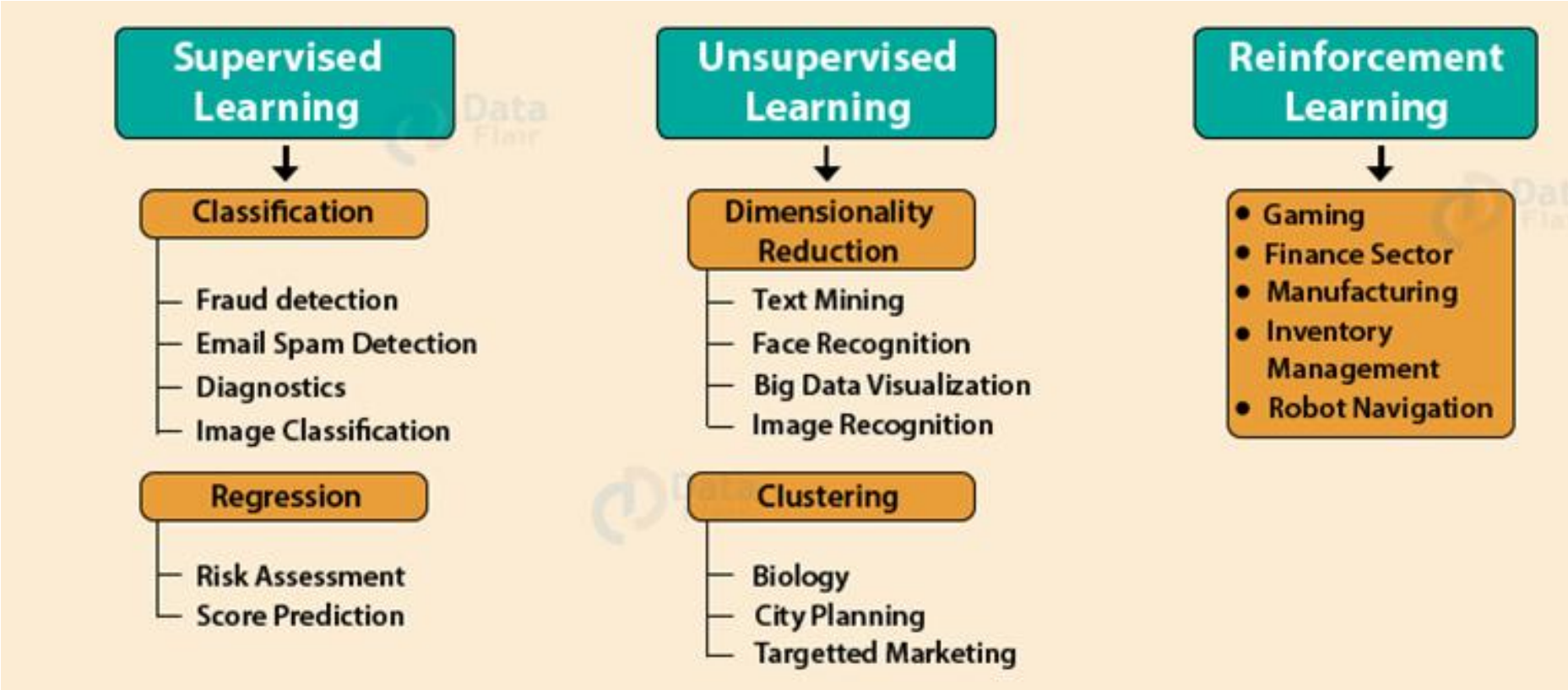


Figure 9. Two strategies for interception when the fielder is running backward. (a) Keeping the angle of elevation of gaze constant is an inefficient strategy because it will send the fielder past the point where the catch can be made. (b) Allowing the angle of gaze to increase avoids this problem.

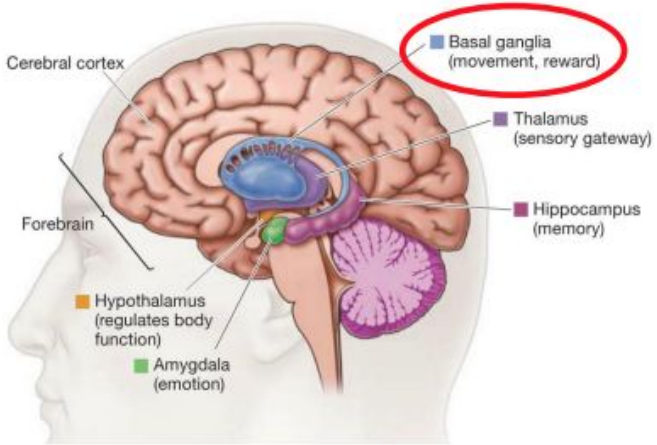



Supervised Learning
Classification
Fraud detection
Email Spam Detection
Diagnostics
Image Classification
Regression
Risk Assessment
Score Prediction
Unsupervised Learning
Dimensionality Reduction
Text Mining
Face Recognition
Big Data Visualization
Image Recognition
Clustering
Biology
City Planning
Targetted Marketing
Reinforcement Learning
Gaming
Finance Sector
Manufacturing
Inventory Management
Robot Navigation

Where do rewards come from?

Mnih et al. '15

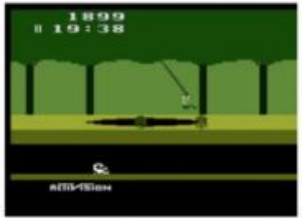
reinforcement learning agent

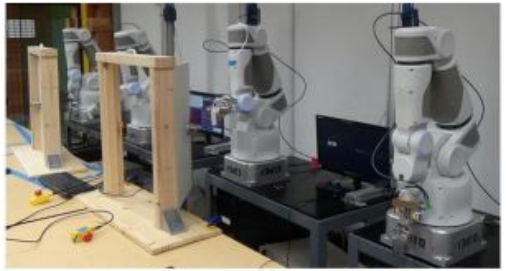


[~] LazyOptimist 32 points 5 days ago

As human agents, we are accustomed to operating with rewards that are so sparse that we only experience them once or twice in a lifetime, if at all.

Why should we study this **now**?





Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". (2015).
V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016).

Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. "End-to-end training of deep visuomotor policies". (2015).

Q-learning:

D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". (2018).

Beating Go champions:

Supervised learning + policy gradients + value functions + Monte Carlo tree search:

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. "Mastering the game of Go with deep neural networks and tree search". Nature (2016).

Why should we study this **now**?

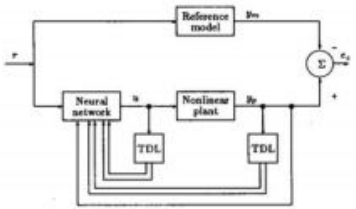
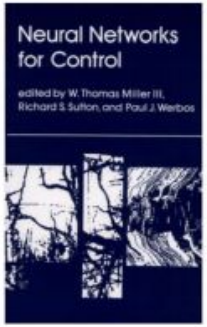
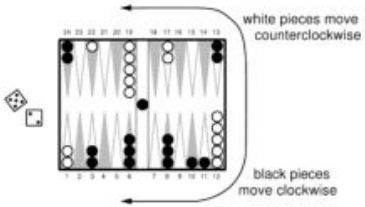


Fig. 21. Direct adaptive control of nonlinear plants using neural networks.





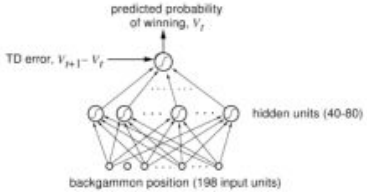


Table 11.1: Summary of TD-Gammon Results

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, ...	−13 pts / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	−7 pts / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 pt / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

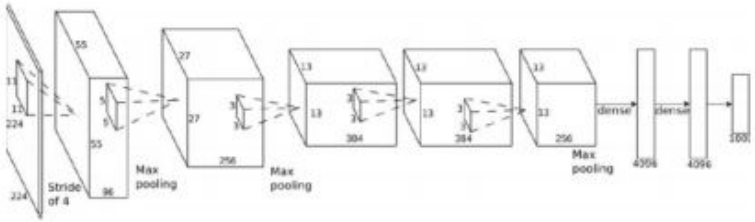
Tesauro, 1995

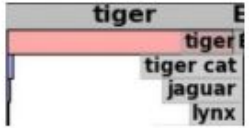
This dissertation demonstrates how we can possibly overcome the slow learning problem and tackle non-Markovian environments, making reinforcement learning more practical for realistic robot tasks:

- Reinforcement learning can be naturally integrated with artificial neural networks to obtain high-quality generalization, resulting in a significant learning speedup. Neural networks are used in this dissertation, and they generalize effectively even in the presence of noise and a large number of binary and real-valued inputs.
- Reinforcement learning agents can save many learning trials by using an action model, which can be learned on-line. With a model, an agent can mentally experience the effects of its actions without actually executing them. Experience replay is a simple technique that implements this idea, and is shown to be effective in reducing the number of action executions required.
- Reinforcement learning agents can take advantage of instructive training instances provided by human teachers, resulting in a significant learning speedup. Teaching can also help learning agents avoid local optima during the search for optimal control. Simulation experiments indicate that even a small amount of teaching can save agents many learning trials.
- Reinforcement learning agents can significantly reduce learning time by hierarchical learning— they first solve elementary learning problems and then combine solutions to the elementary problems to solve a complex problem. Simulation experiments indicate that a robot with hierarchical learning can solve a complex problem, which otherwise is hardly solvable within a reasonable time.
- Reinforcement learning agents can deal with a wide range of non-Markovian environments by having a memory of their past. Three memory architectures are discussed. They work reasonably well for a variety of simple problems. One of them is also successfully applied to a nontrivial non-Markovian robot task.

L.-J. Lin, “Reinforcement learning for robots using neural networks.” 1993

Why should we study this **now**?



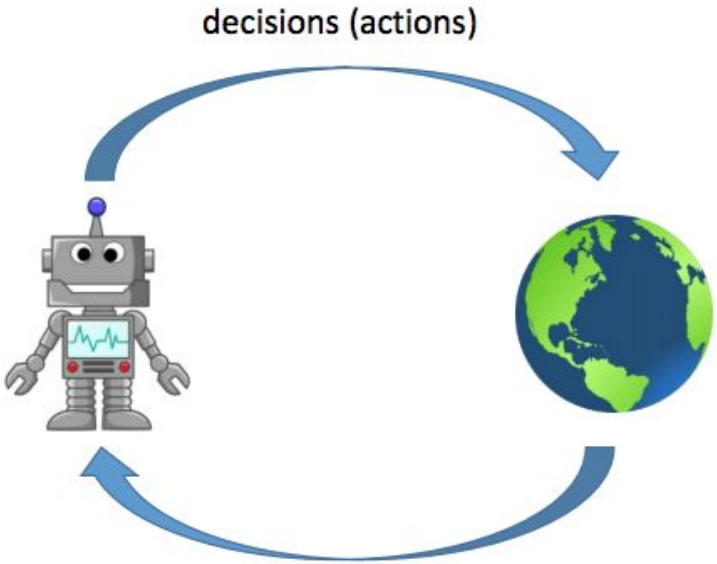


1. Advances in deep learning

2. Advances in reinforcement learning

3. Advances in computational capability

Reinforcement learning provides a formalism for *behavior*



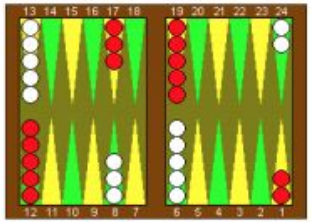
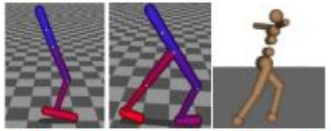


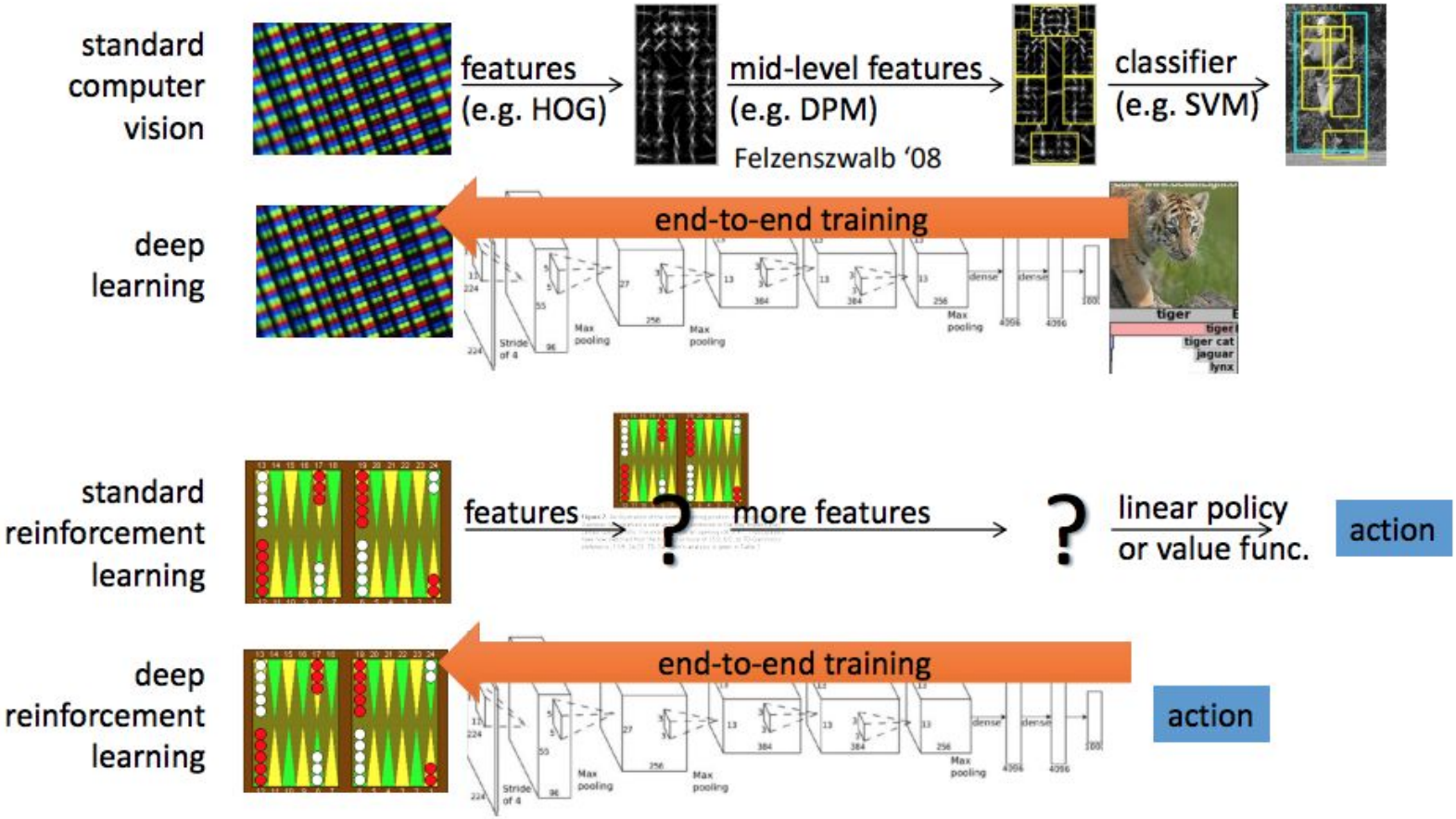
Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

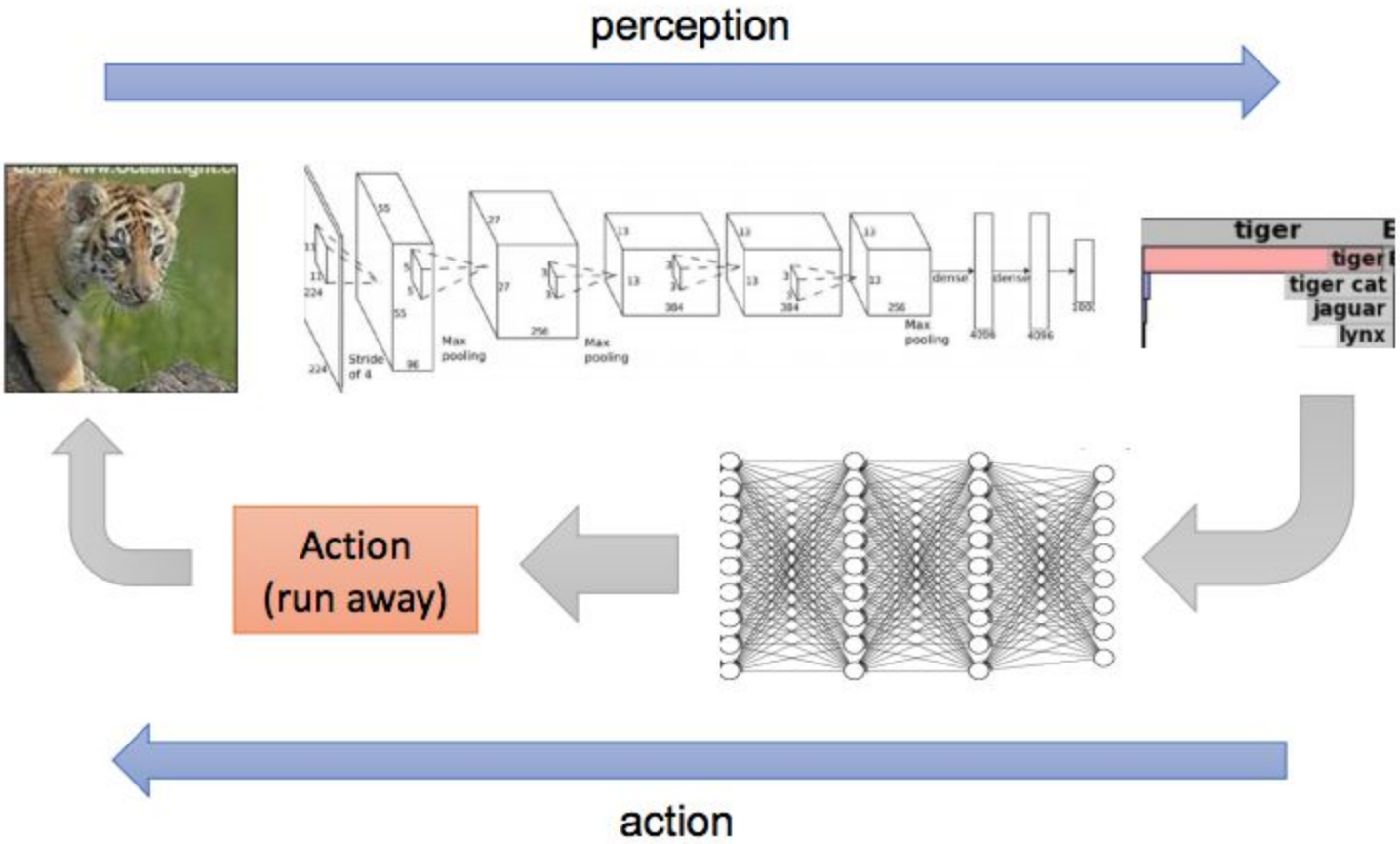


Schulman et al. '14 & '15

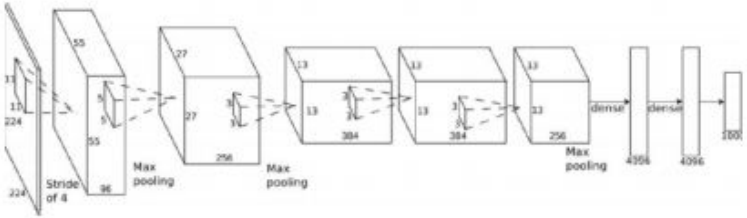
Mnih et al. '13

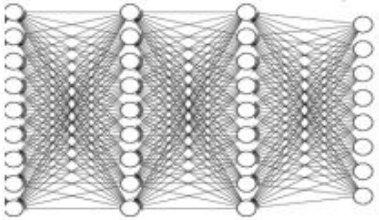
Levine*, Finn*, et al. '16


standard computer vision
features (e.g. HOG)
mid-level features (e.g. DPM)
Felzenszwalb '08
classifier (e.g. SVM)
deep learning
end-to-end training
tiger
tiger cat
jaguar
lynx
standard reinforcement learning
features
more features
linear policy or value func.
action
deep reinforcement learning
end-to-end training
action


perception
Stride of 4
Max pooling
Max pooling
Max pooling
dense
dense
tiger
tiger cat
jaguar
lynx
Action
(run away)
action

sensorimotor loop

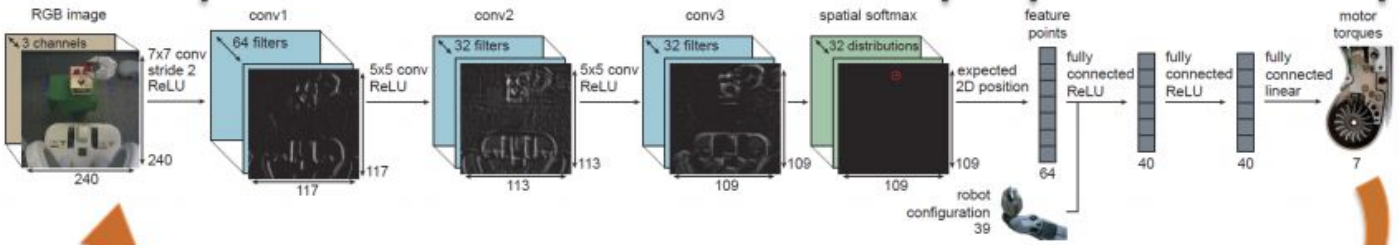




Action
(run away)

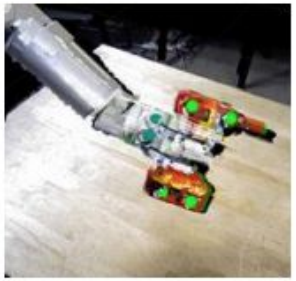
tiny, highly specialized "visual cortex"

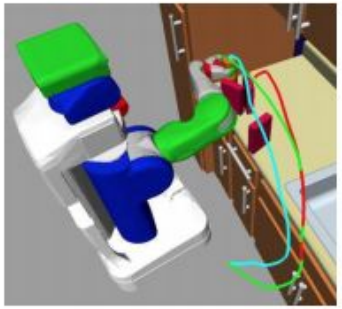
tiny, highly specialized "motor cortex"



sensorimotor loop

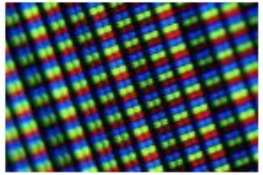
Example: robotics

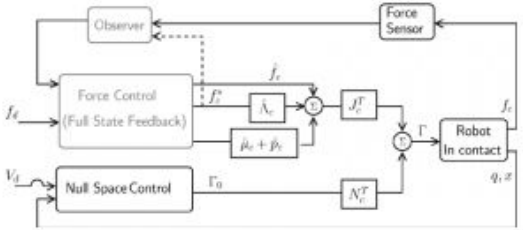




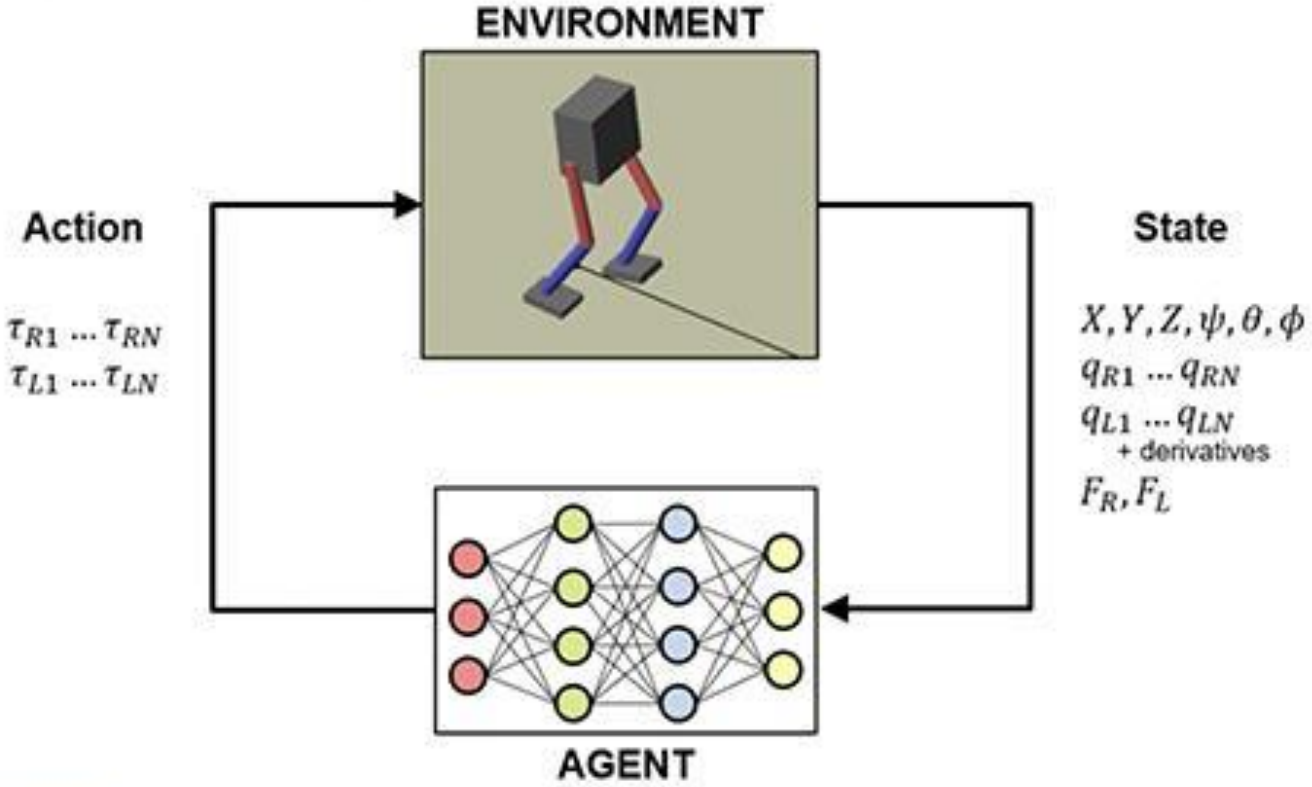
robotic control pipeline

observations → state estimation (e.g. vision) → modeling & prediction → planning → low-level control → controls

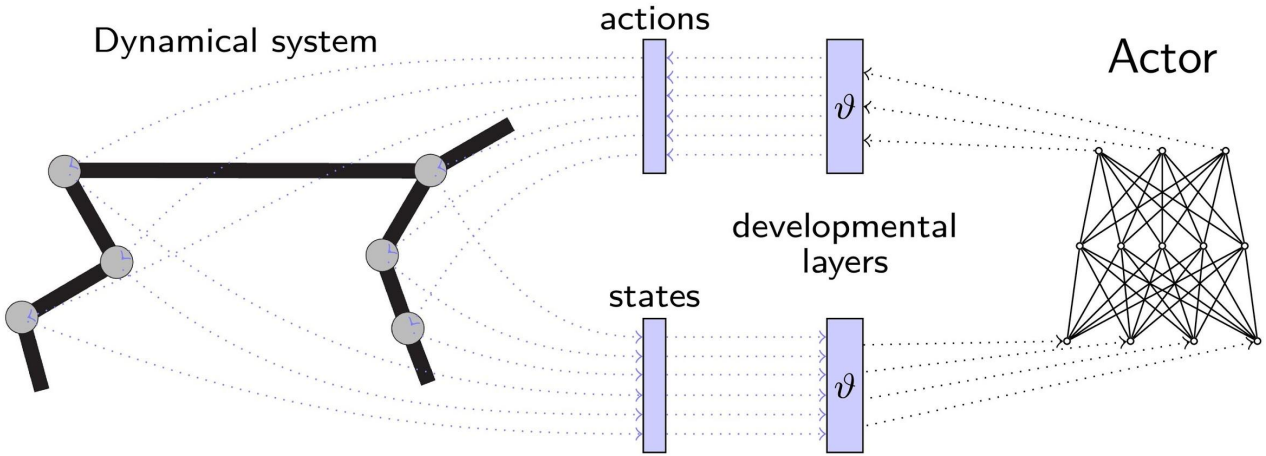




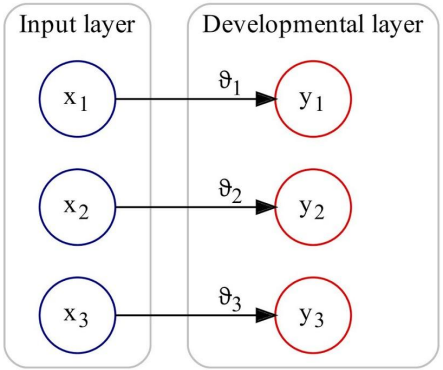
Walking Robot Example

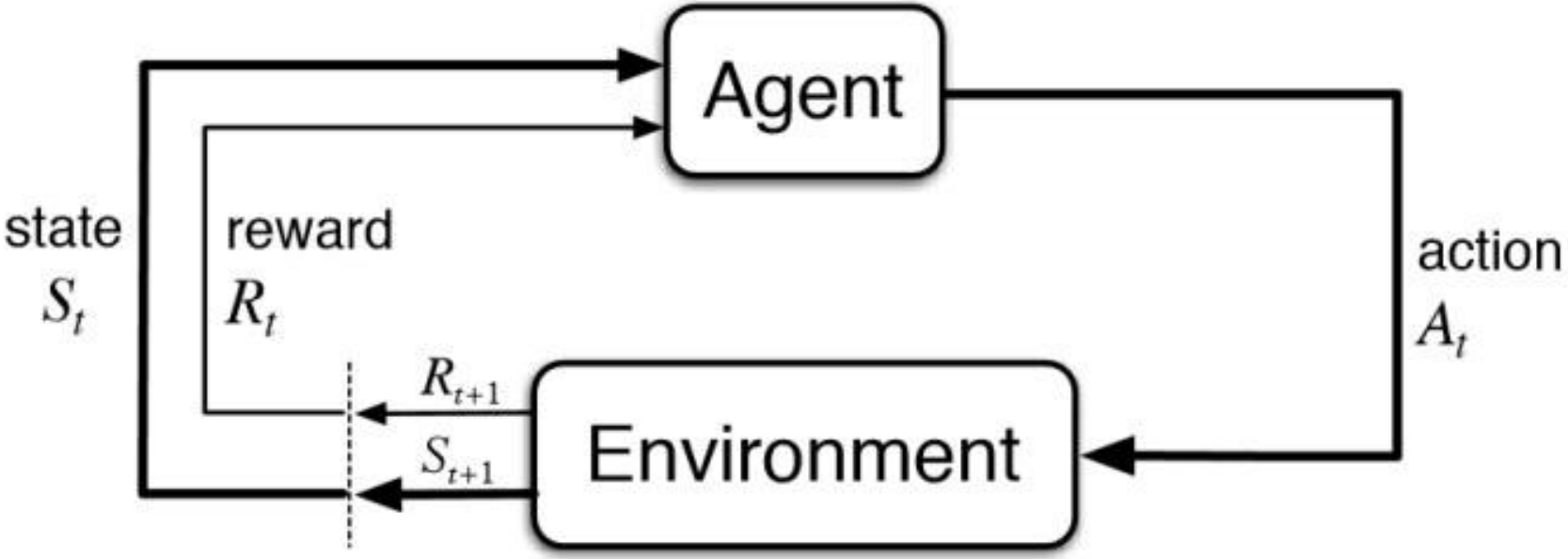


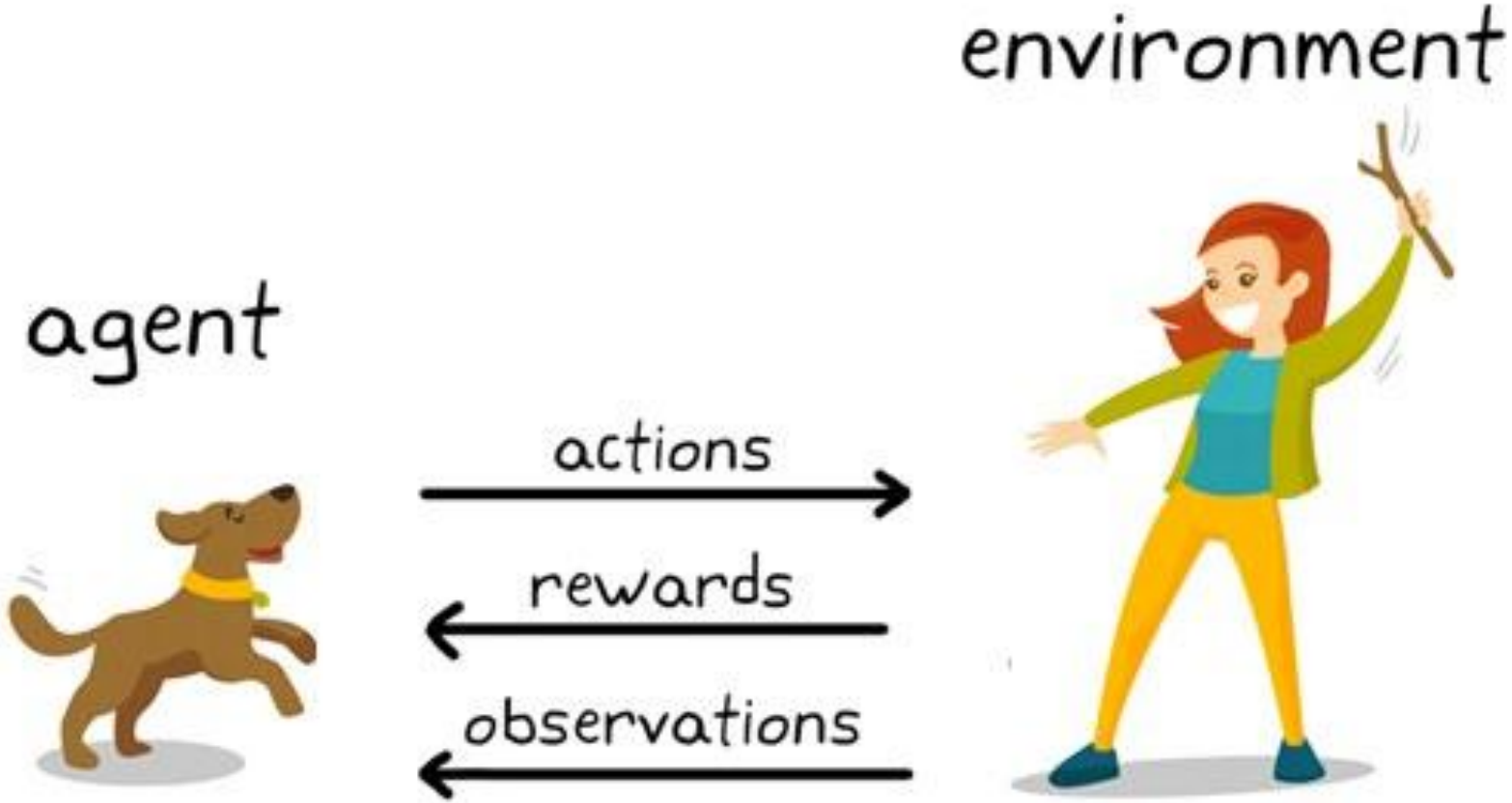
Developmental layers

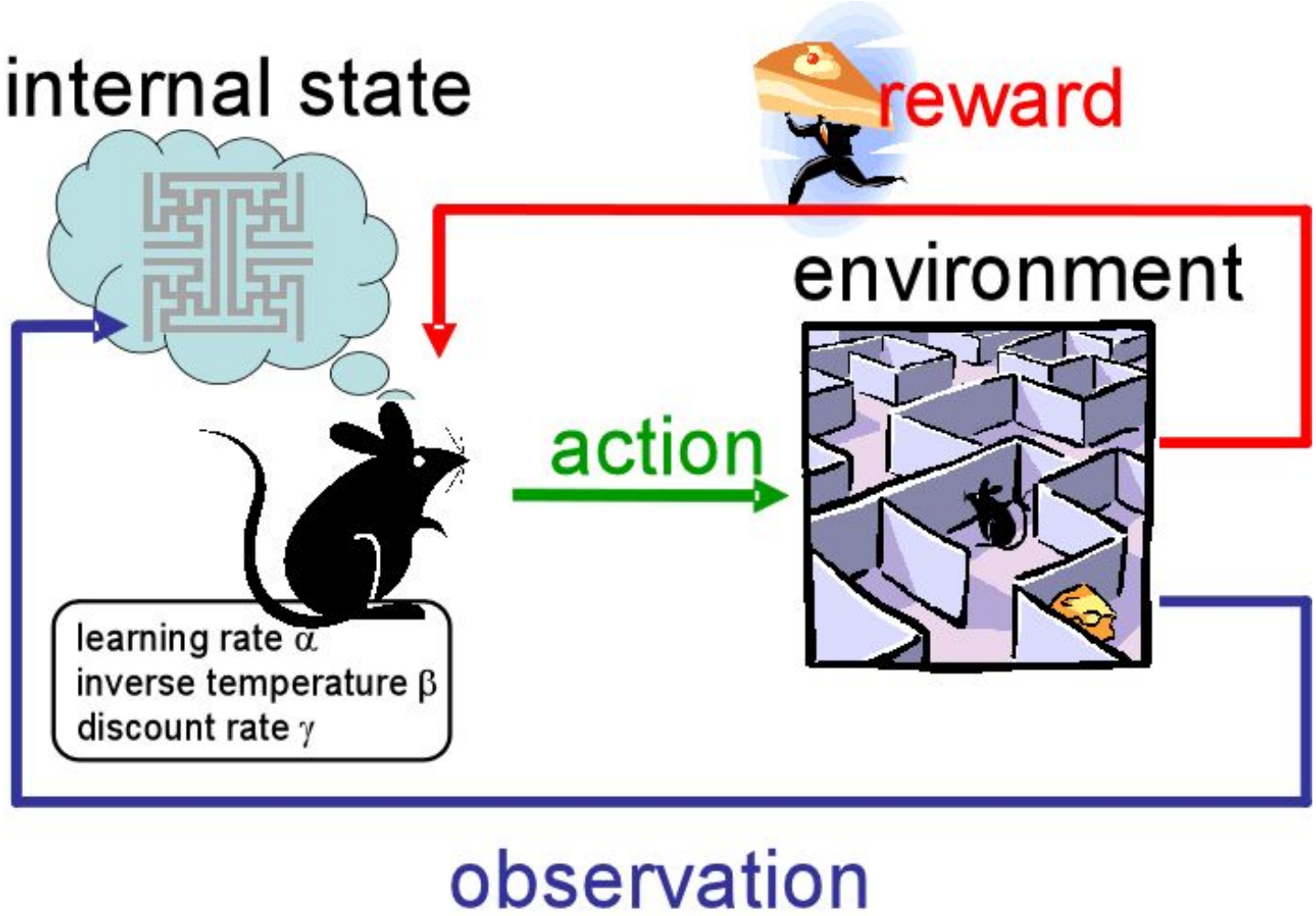


deterministic $y_i = \begin{cases} x_i & \text{if } \vartheta_i \geq \text{threshold} \\ 0 & \text{else} \end{cases}$

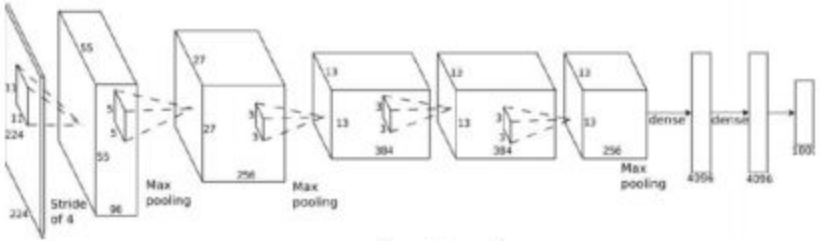



Agent
state
S_t
reward
R_t
action
A_t
R_{t+1}
S_{t+1}
Environment


environment
agent
actions
rewards
observations


internal state
reward
environment
action
learning rate α
inverse temperature β
discount rate γ
observation

$\mathbf{o}_t$



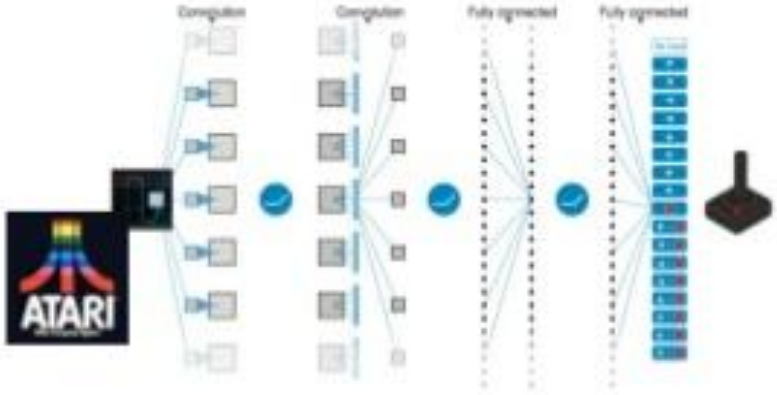
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

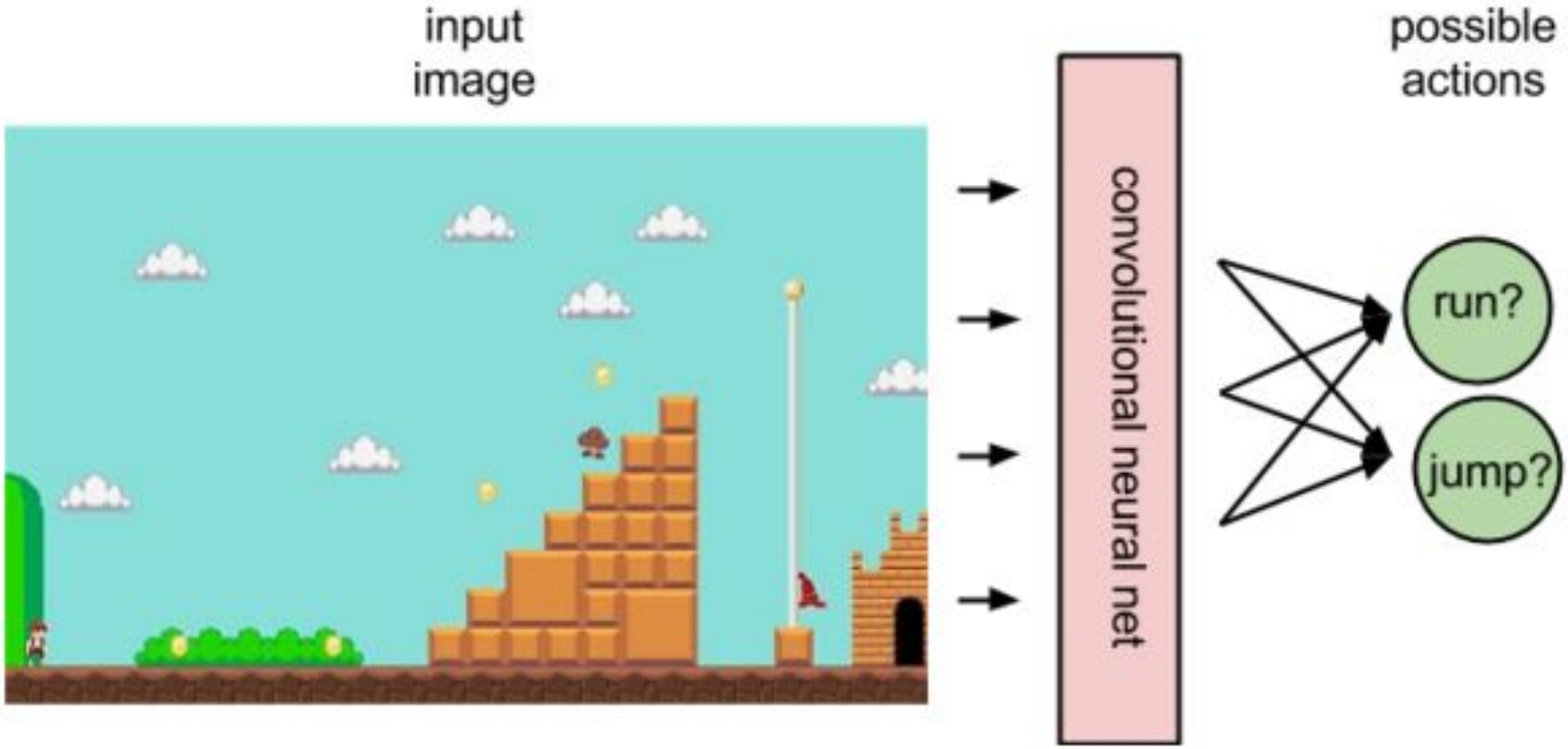
Deep Reinforcement Learning

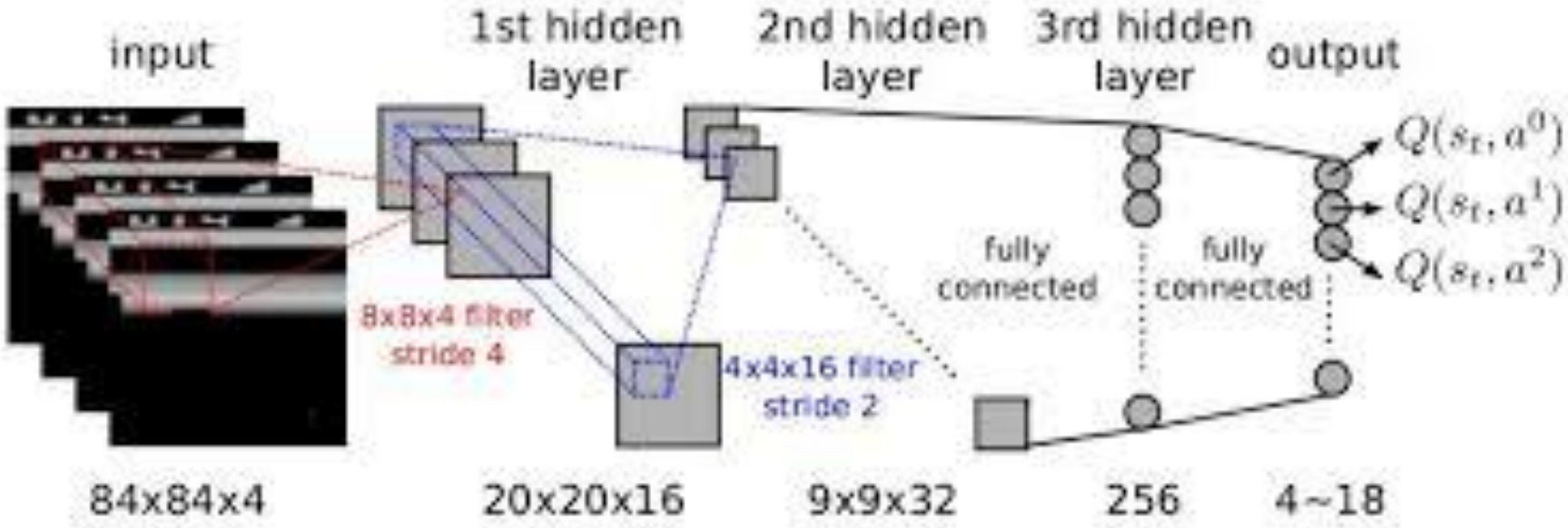


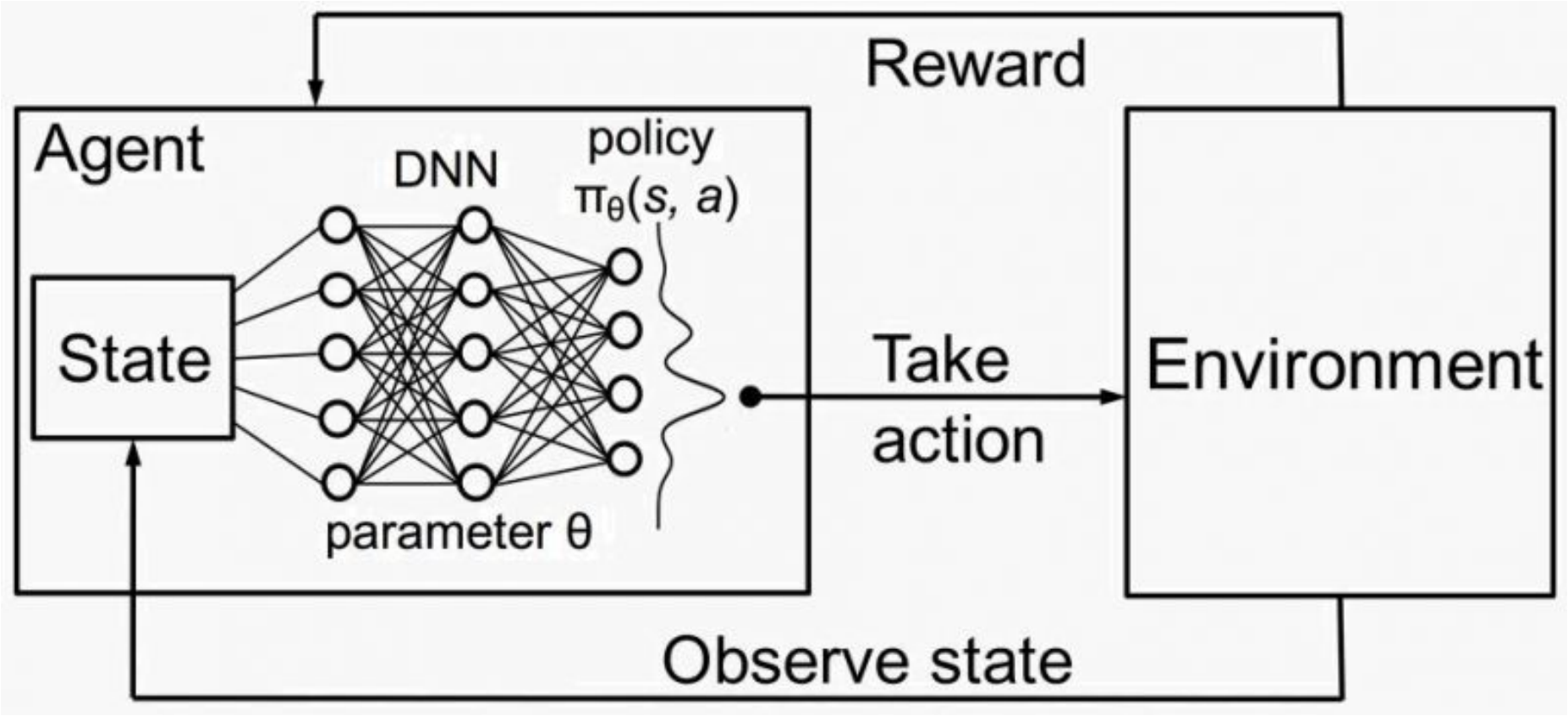
- Reinforcement Learning + Deep Learning
  - From Pixel to Action
    - General game play in ATARI domain
    - Car Driver
    - (Go Expert)
- Deep Reinforcement Learning in Continuous Action Domain: DDPG
  - Lillicrap, Timothy P., et al.
    "Continuous control with deep reinforcement learning." , *ICLR 2016*

Vision-based Car agent in TORCS
Steering + Accel/Blake
= 2 dim continuous actions

Convolutional Agent



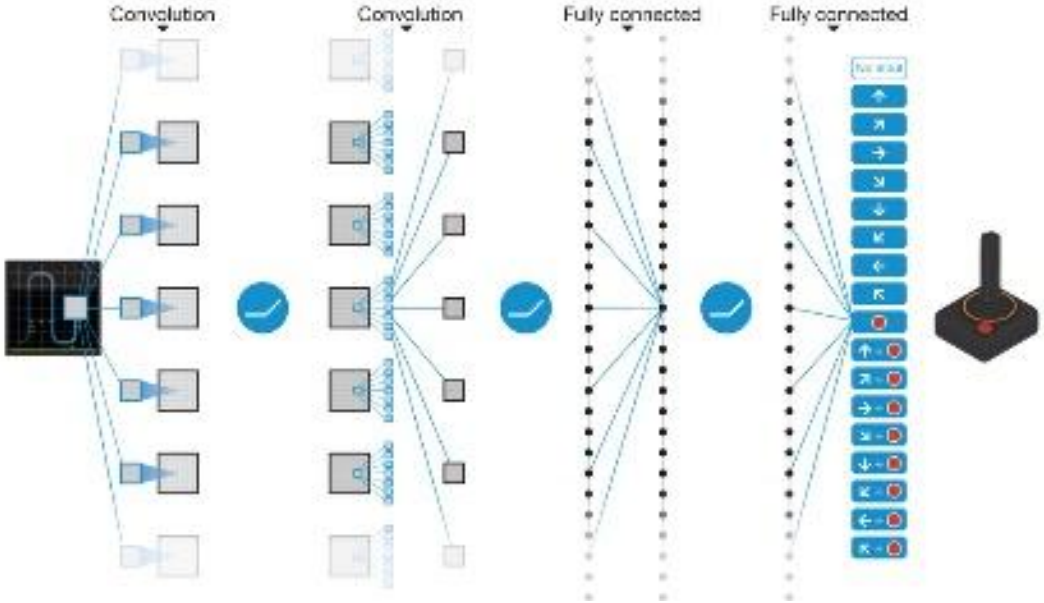

input
1st hidden layer
2nd hidden layer
3rd hidden layer
output
$Q(s_t, a^0)$
$Q(s_t, a^1)$
$Q(s_t, a^2)$
fully connected
fully connected
8x8x4 filter stride 4
4x4x16 filter stride 2
84x84x4
20x20x16
9x9x32
256
4~18


Reward
Agent
DNN
policy
$\pi_\theta(s, a)$
State
Take
action
Environment
parameter θ
Observe state

Human-level control through deep reinforcement learning

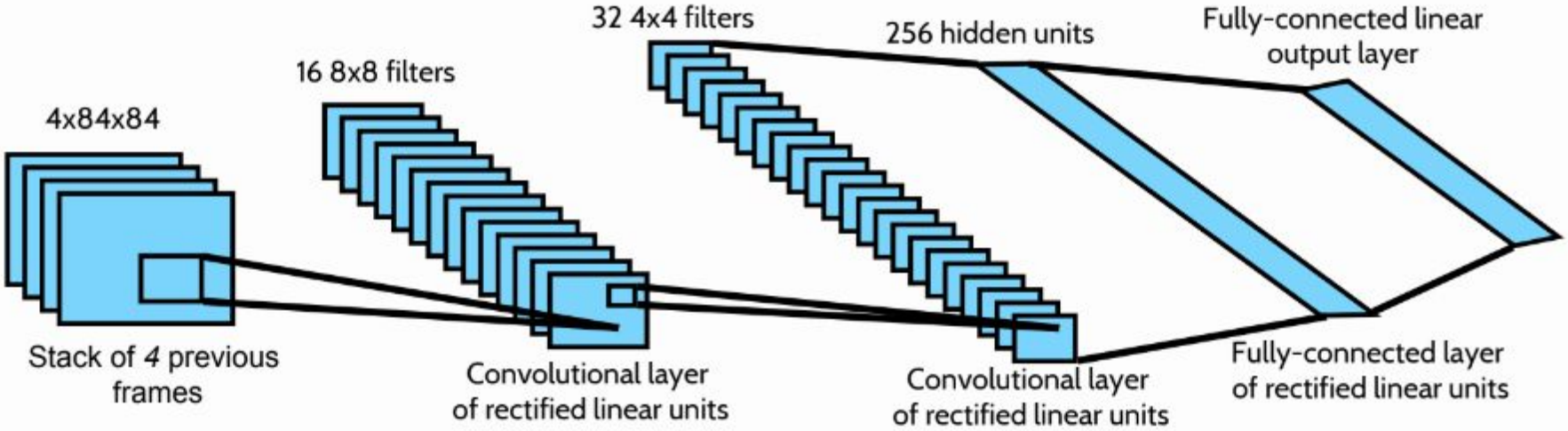
Volodymyr Mnih, Koray Kavukcuoglu, David Silver et. al.

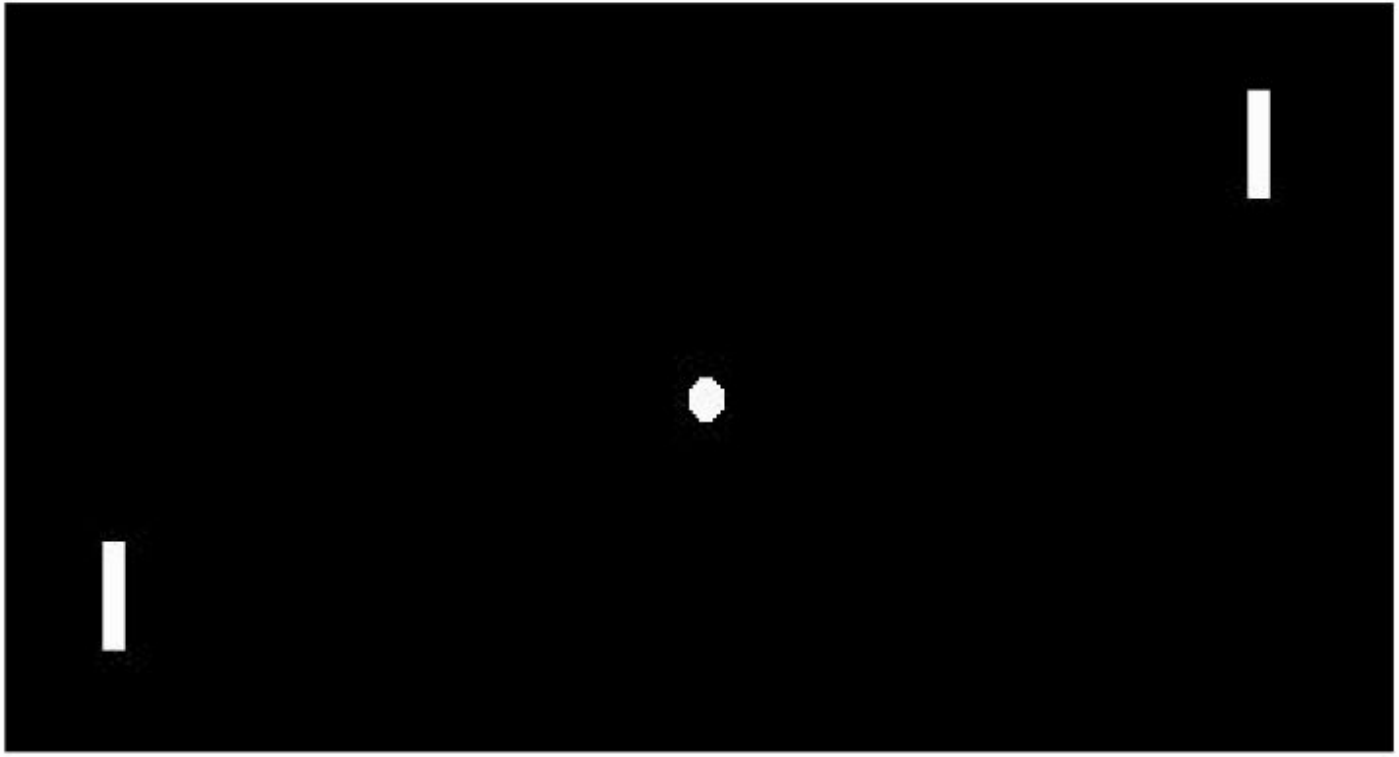
Google Deepmind



ATARI Network Architecture

- Convolutional neural network architecture:
  - History of frames as input.
  - One output per action - expected reward for that action $Q(s, a)$.
  - Final results used a slightly bigger network (3 convolutional + 1 fully-connected hidden layers).



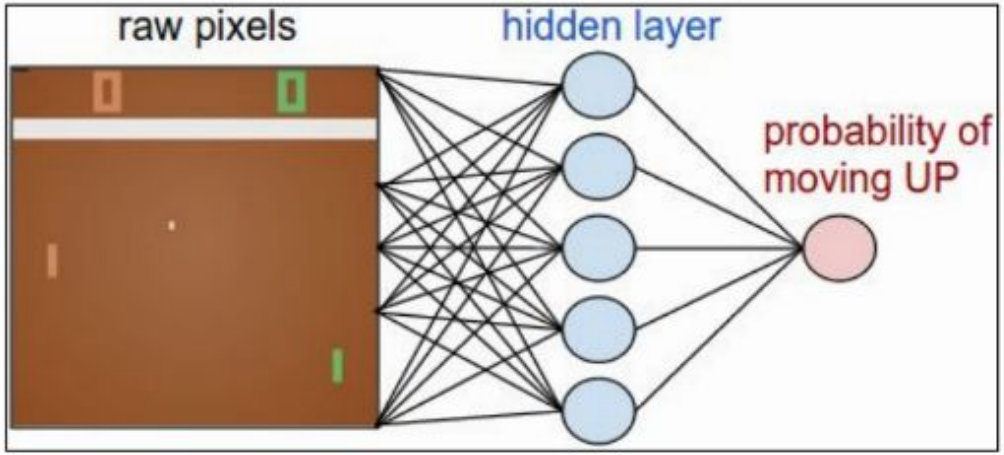


Network does not see this. Network sees 80*80 = 6,400 numbers.
It gets a reward of +1 or -1, some of the time.
Q: How do we efficiently find a good setting of the 1.3M parameters?

Policy network

height width

[80 x 80]
array



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

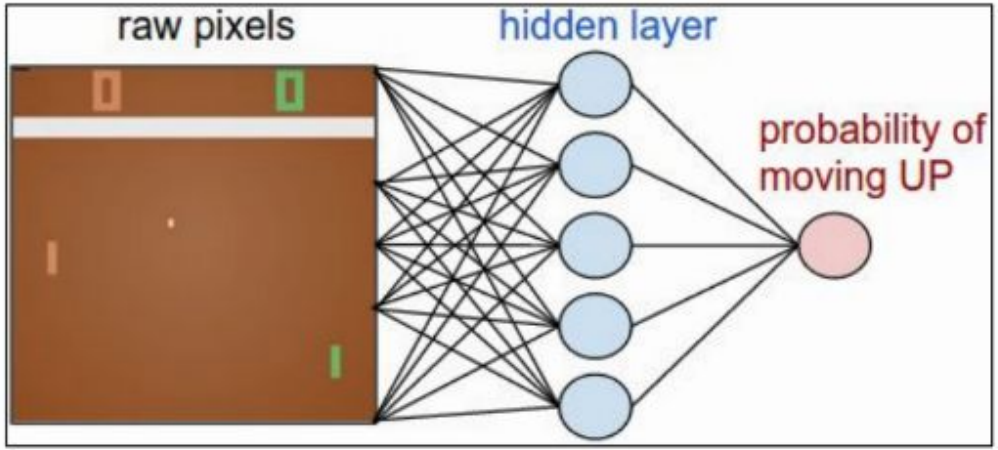
[(80*80)*200 + 200] + [200*1 + 1] = **~1.3M parameters**

Layer 1 Layer 2

Policy network

height width

[80 x 80]
array



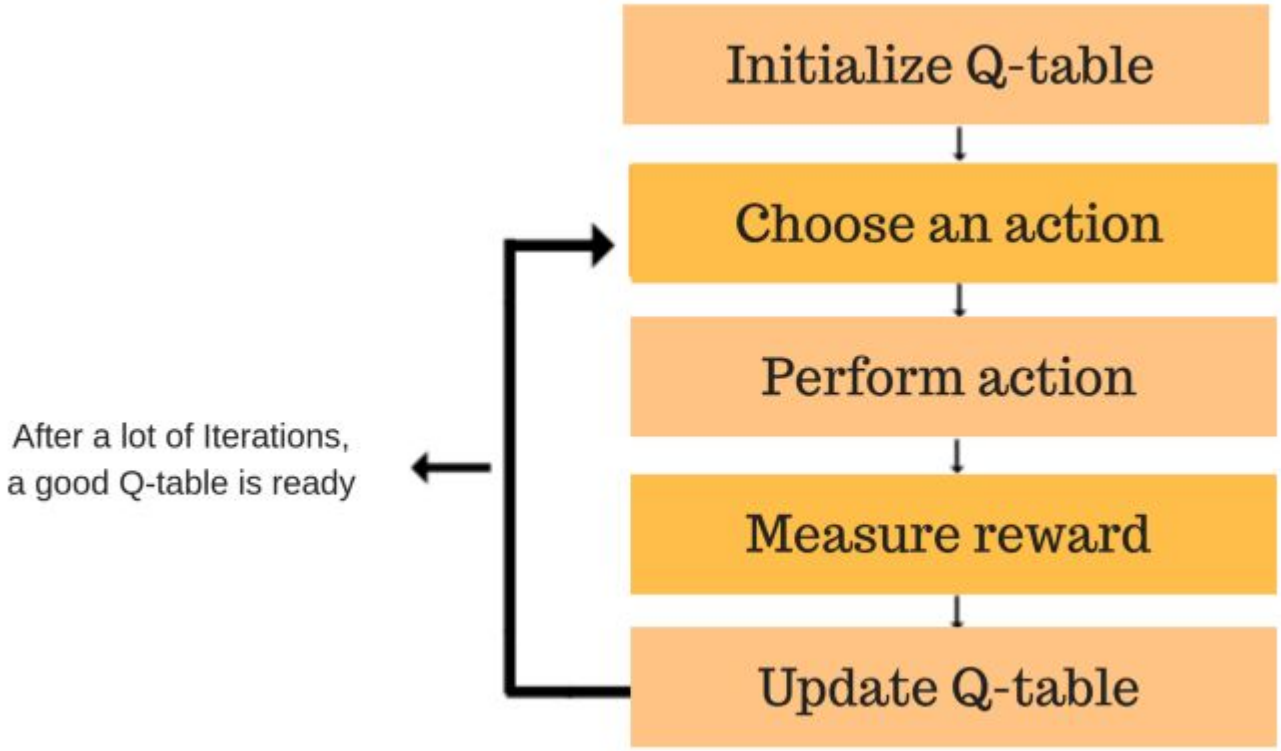

```
h = np.dot(W1, x) # compute hidden layer neuron activations
h[h<0] = 0 # ReLU nonlinearity: threshold at zero
logp = np.dot(W2, h) # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp)) # sigmoid function (gives probability of going up)
```

Problem is easy if you want to be inefficient...

1. **Repeat Forever:**
2. Sample 1.3M random numbers
3. Run the policy for a while
4. If the performance is best so far, save it
5. Return the best policy

Problem is easy if you want to be inefficient...

1. **Repe**
2. Sa
3. Ru
4. If th
5. Retur


Initialize Q-table
Choose an action
Perform action
Measure reward
Update Q-table
After a lot of Iterations,
a good Q-table is ready

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α' Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

State

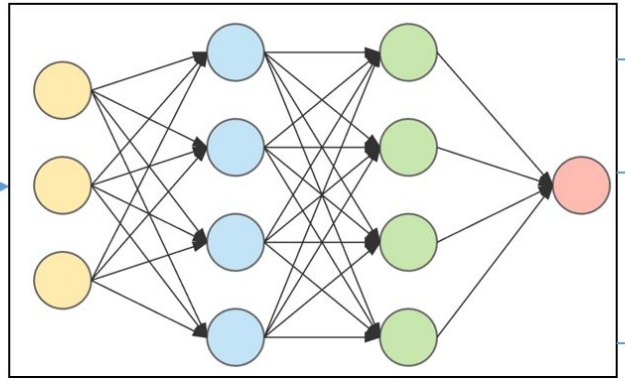
Action

Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q-Value

Q Learning

State



Q-Value Action 1

Q-Value Action 2

-

-

Q-Value Action N

Deep Q Learning

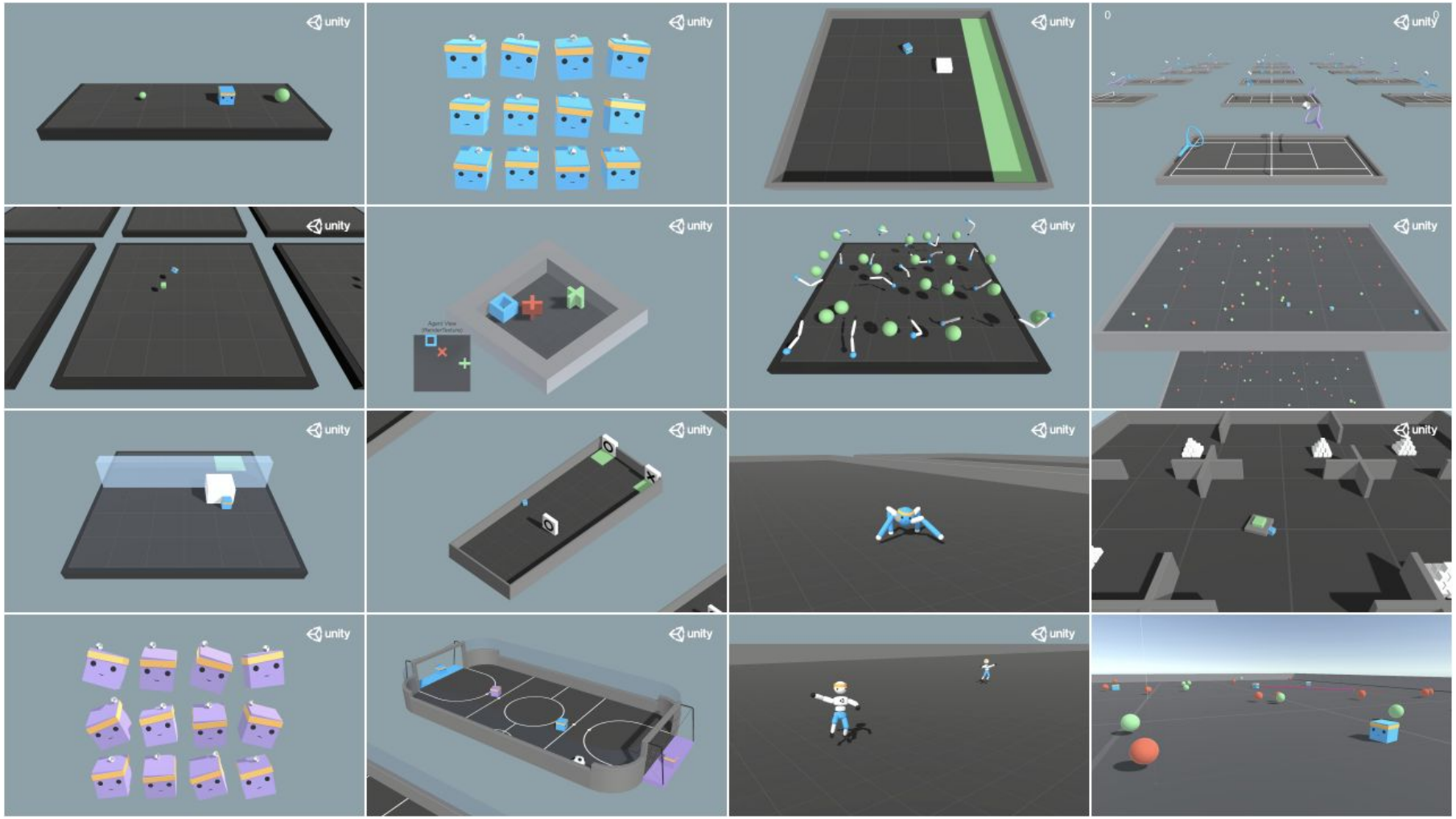
$$\text{New } Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

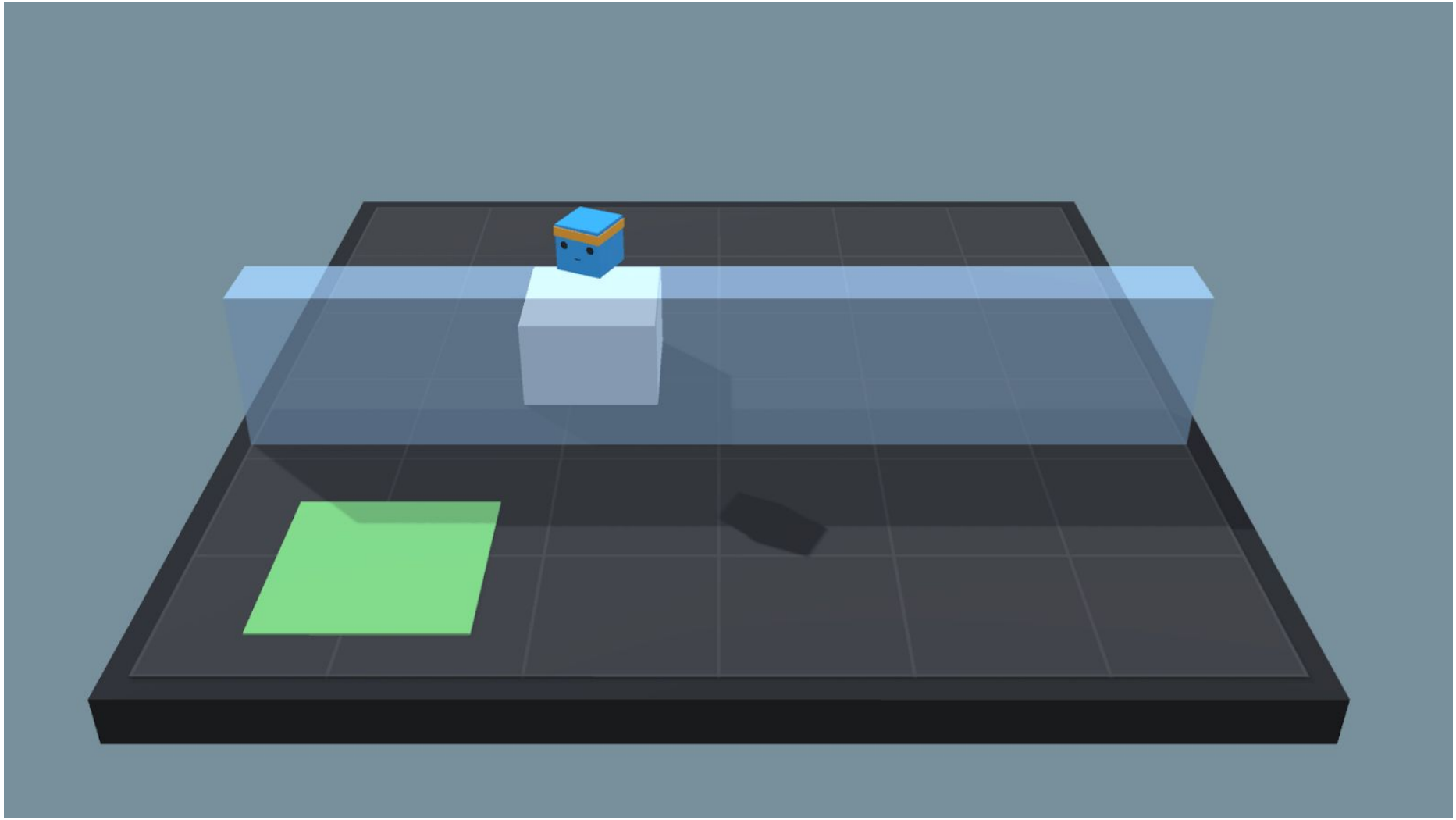
- New Q Value for that state and the action
- Learning Rate
- Reward for taking that action at that state
- Current Q Values
- Maximum expected future reward given the new state (s') and all possible actions at that new state.
- Discount Rate

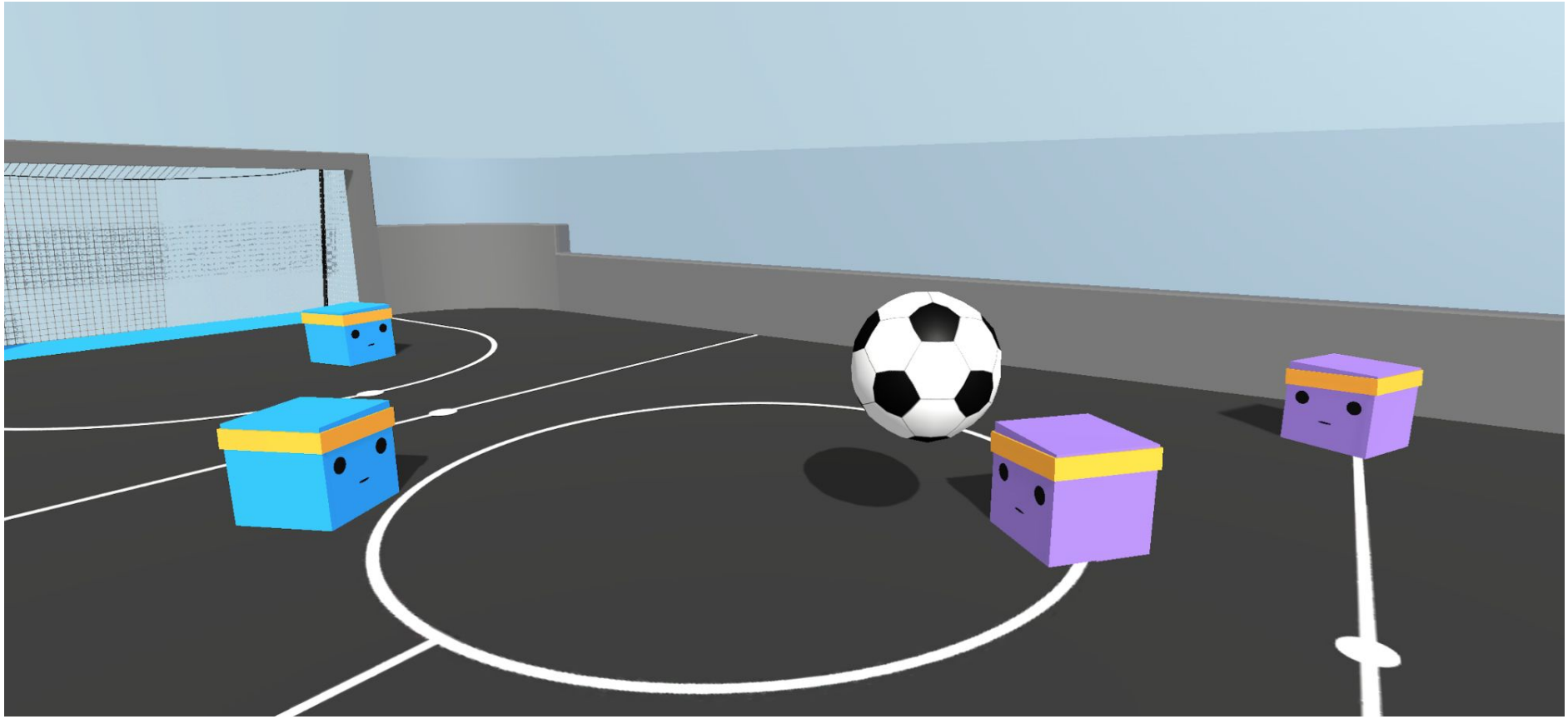
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import gym
import huskarl as hk


# Setup gym environment
create_env = lambda: gym.make('CartPole-v0').unwrapped
dummy_env = create_env()

# Build a simple neural network with 3 fully connected layers as our model
model = Sequential([
        Dense(16, activation='relu', input_shape=dummy_env.observation_space.shape),
        Dense(16, activation='relu'),
        Dense(16, activation='relu'),
])

# Create Deep Q-Learning Network agent
agent = hk.agent.DQN(model, actions=dummy_env.action_space.n, nsteps=2)

def plot_rewards(episode_rewards, episode_steps, done=False):
        plt.clf()
        plt.xlabel('Step')
        plt.ylabel('Reward')
        for ed, steps in zip(episode_rewards, episode_steps):
                plt.plot(steps, ed)
        plt.show() if done else plt.pause(0.001) # Pause a bit so that the graph is updated

# Create simulation, train and then test
sim = hk.Simulation(create_env, agent)
sim.train(max_steps=3000, visualize=True, plot=plot_rewards)
sim.test(max_steps=1000)
```

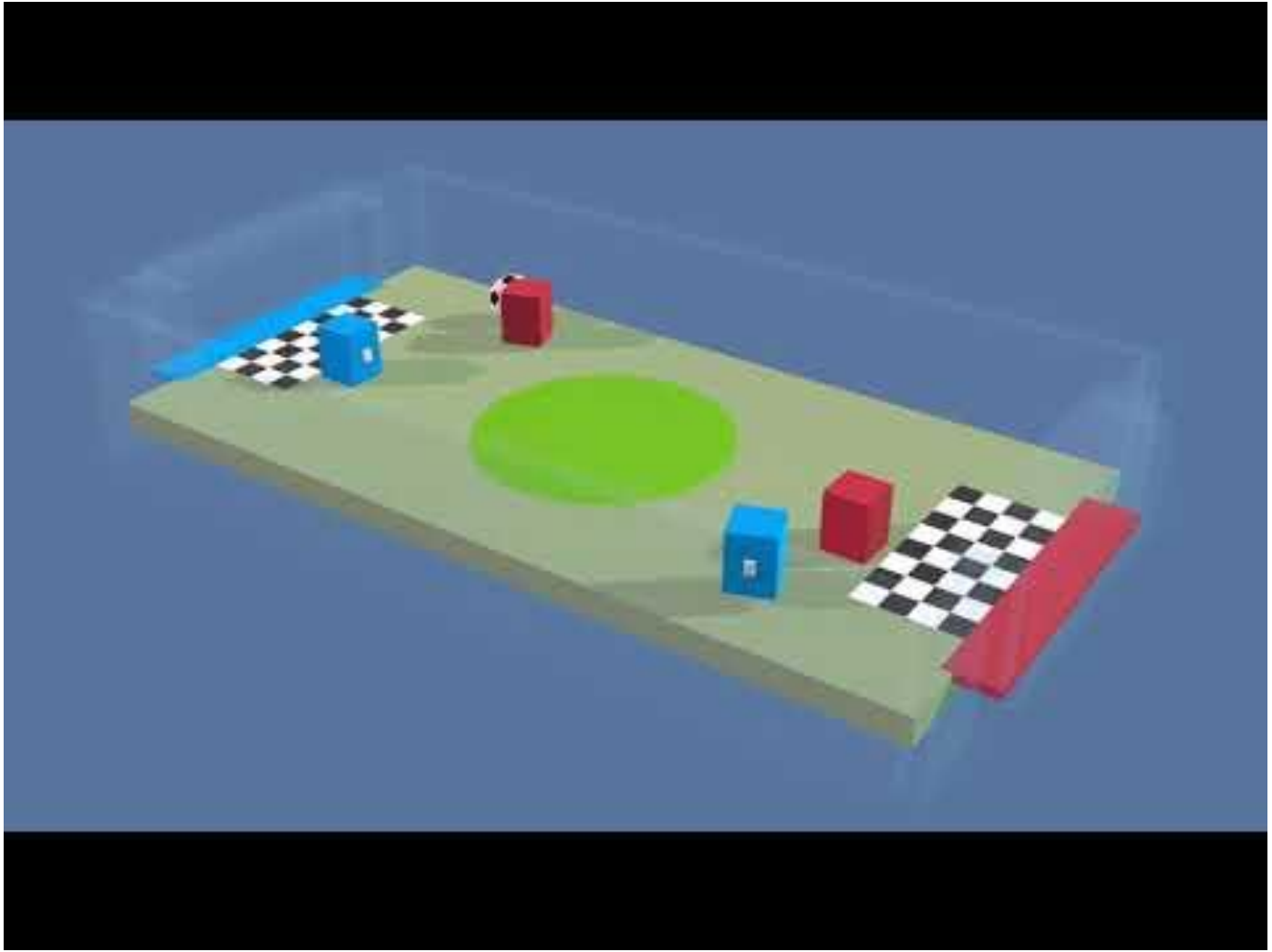
unity

Machine Learning Agents

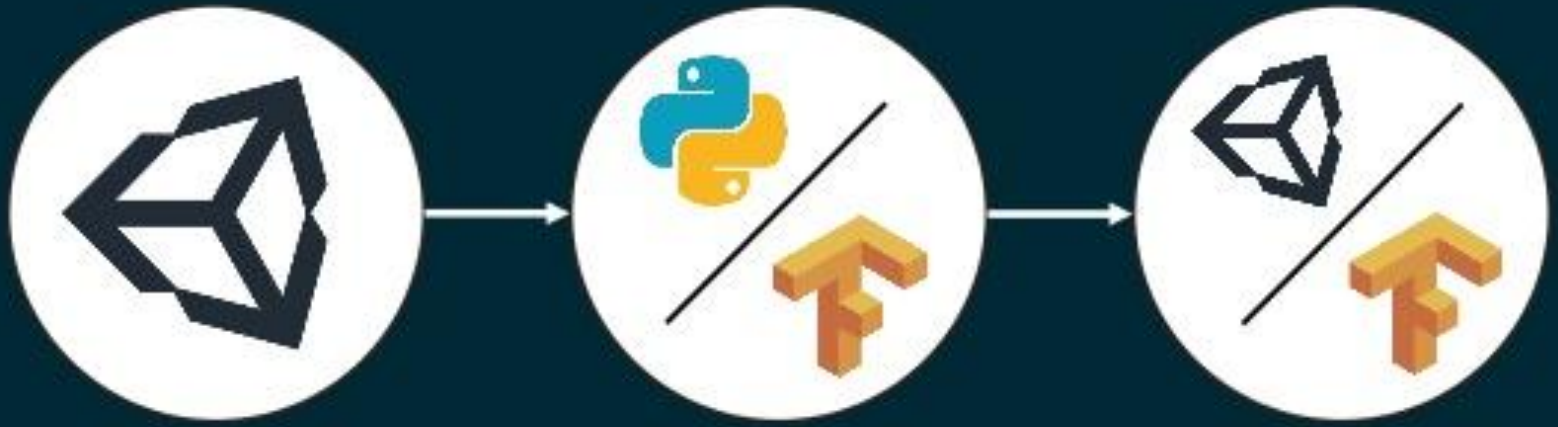

unity
Agent View
(RenderTexture)







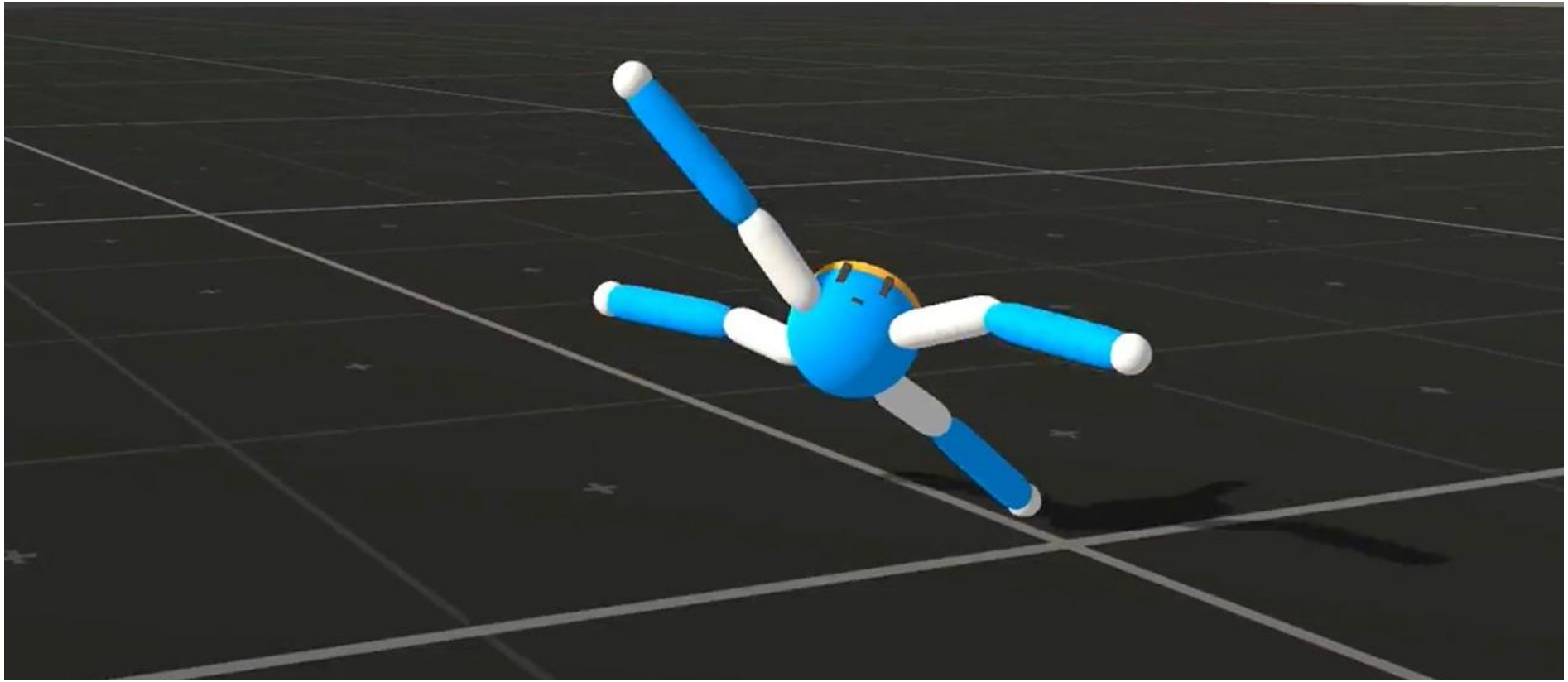
Unity ML-Agents Workflow



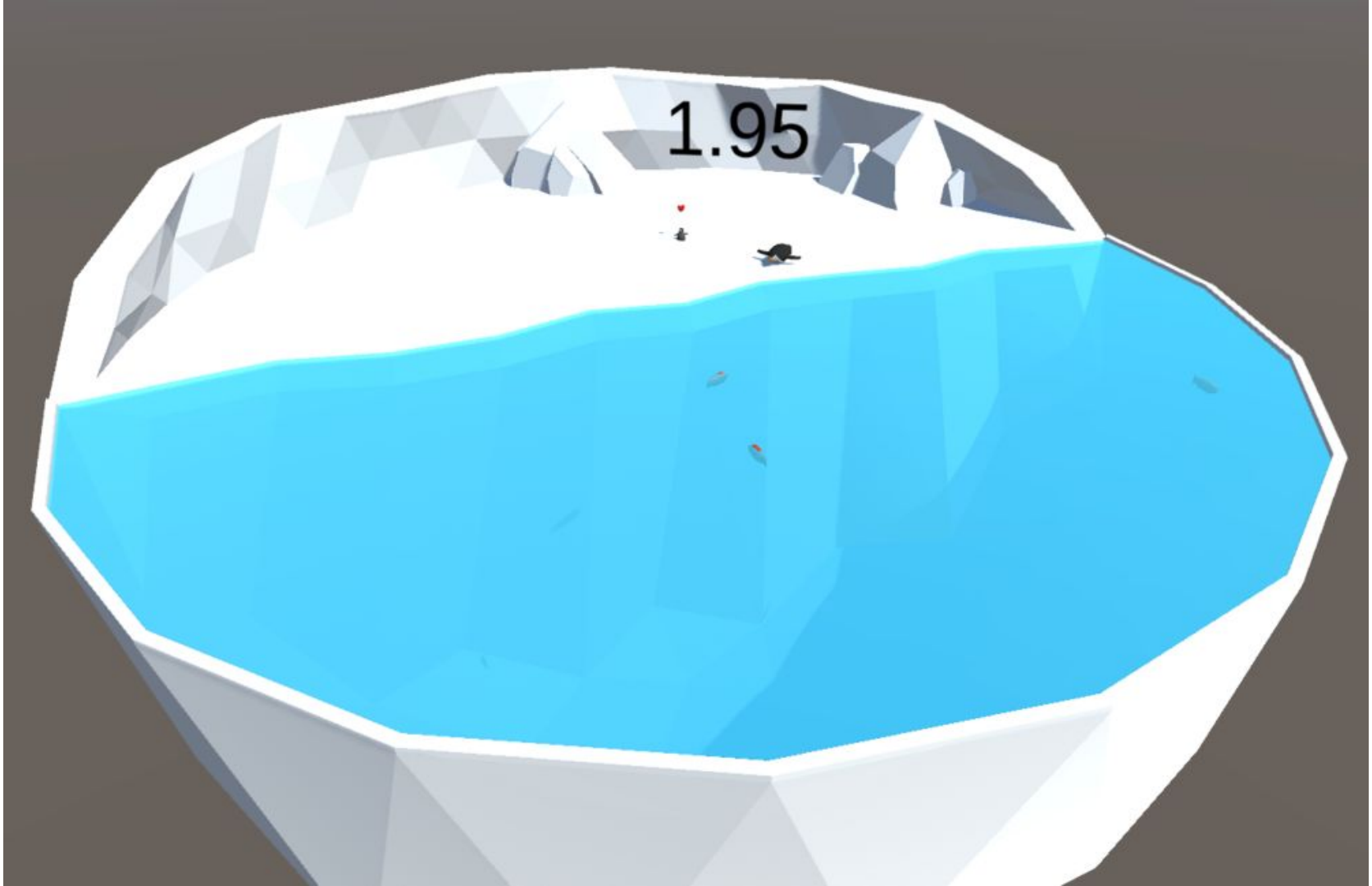
Create Environment

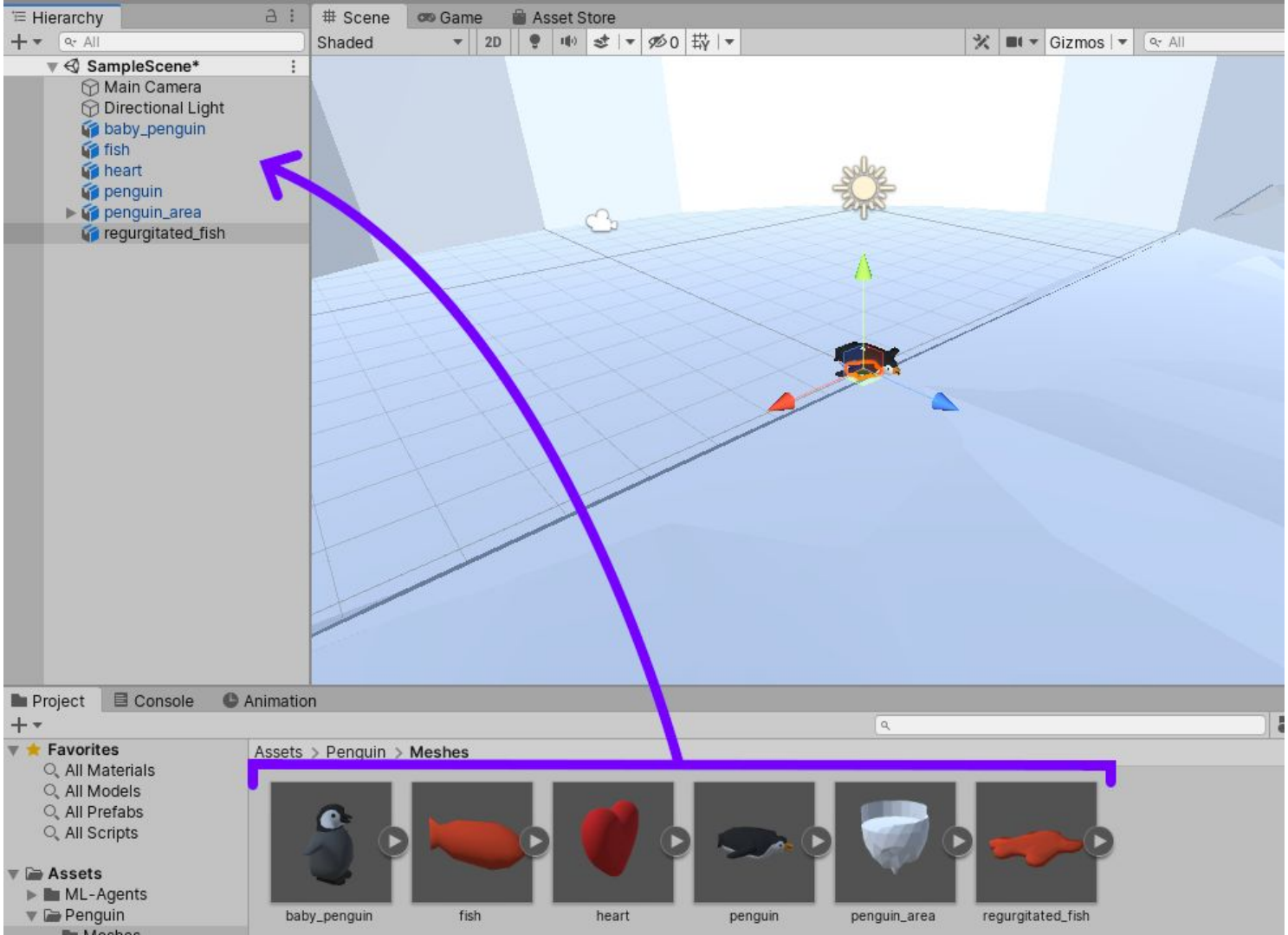
Train Agents

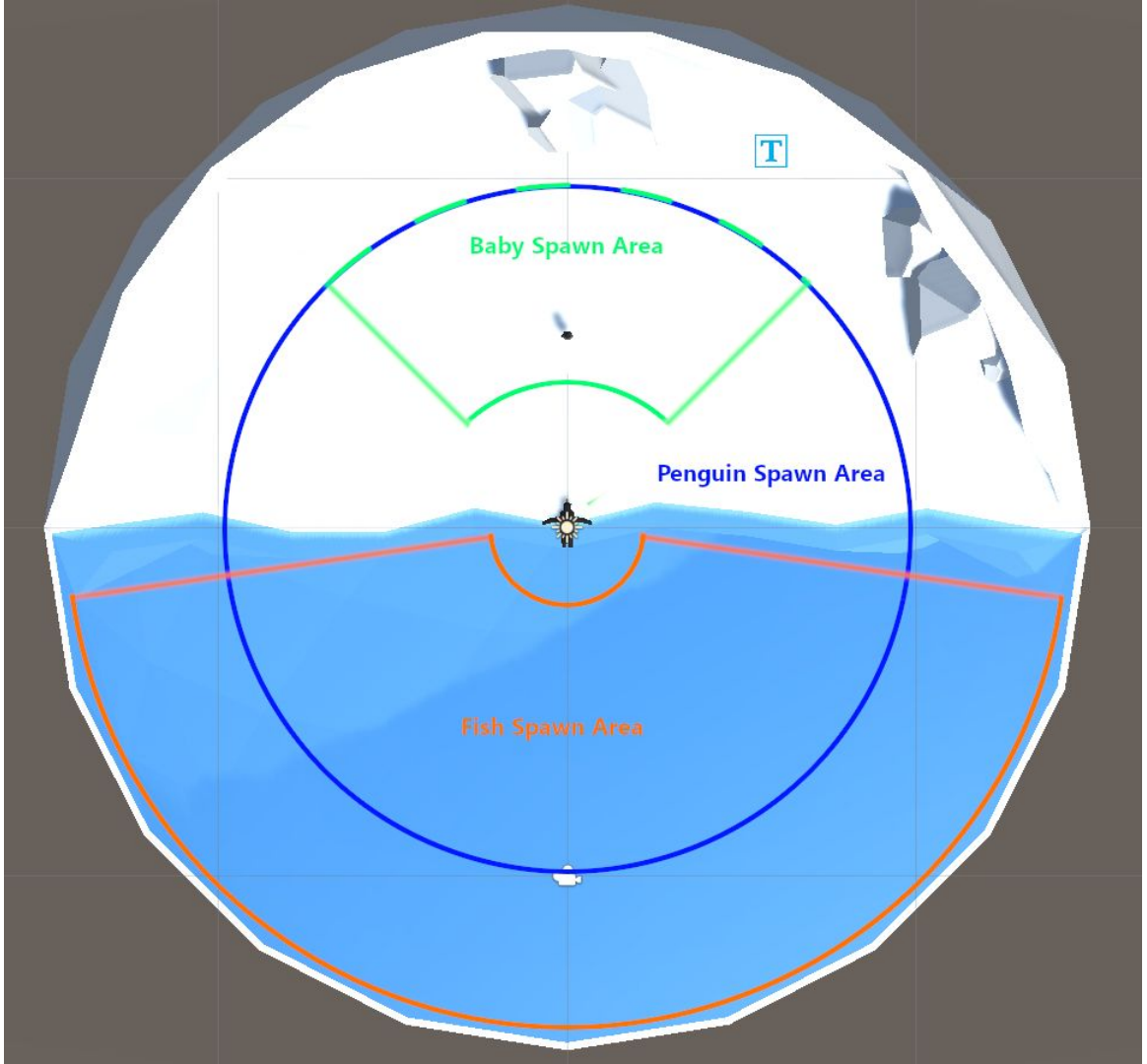
Embed Agents

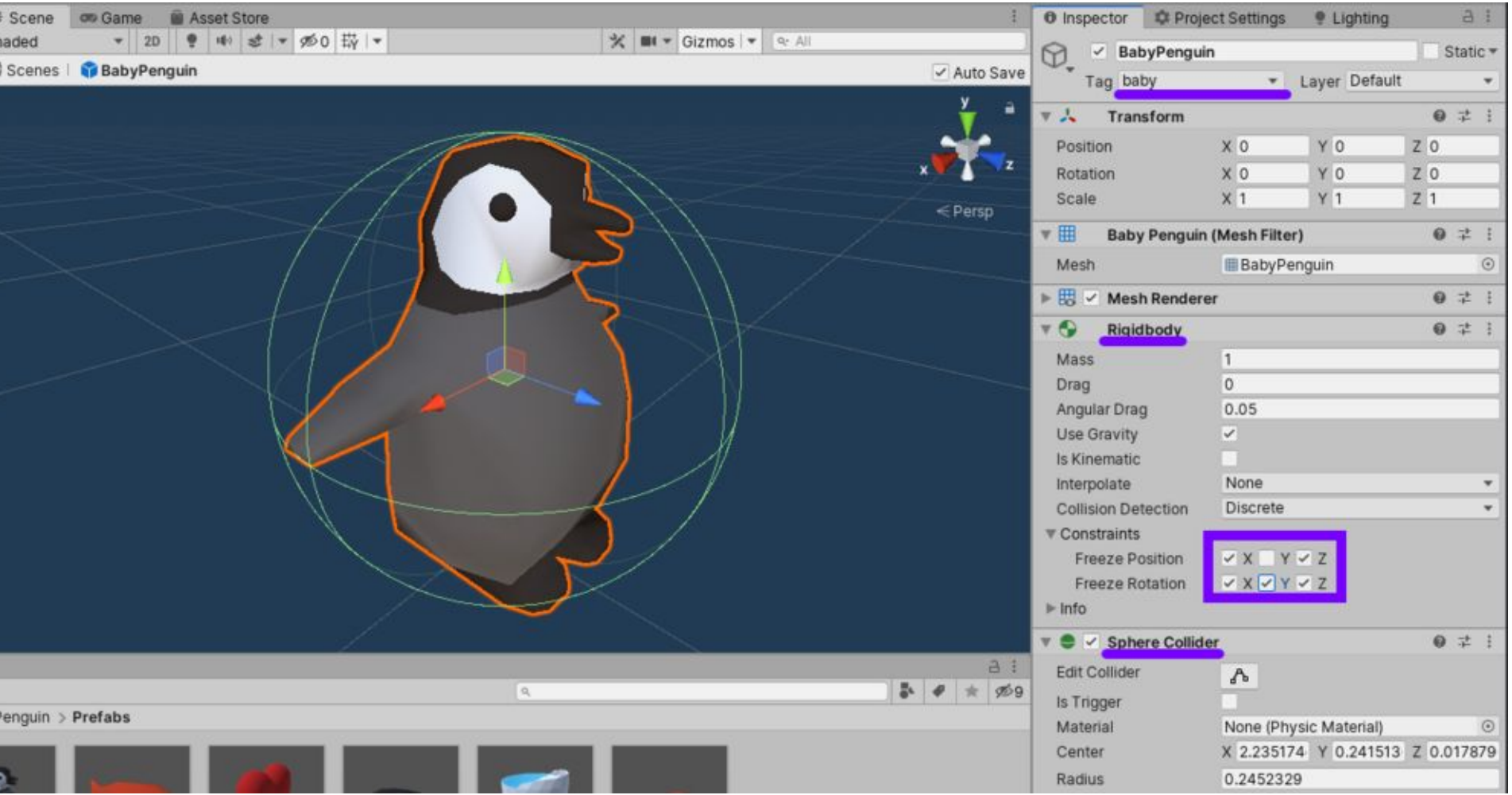


Policy Gradients


1.95


Hierarchy
Scene
Game
Asset Store
Shaded
2D
Gizmos
SampleScene*
Main Camera
Directional Light
baby_penguin
fish
heart
penguin
penguin_area
regurgitated_fish
Project
Console
Animation
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
ML-Agents
Penguin
Assets > Penguin > Meshes
baby_penguin
fish
heart
penguin
penguin_area
regurgitated_fish


T
Baby Spawn Area
Penguin Spawn Area
Fish Spawn Area


Scene
Game
Asset Store
2D
Gizmos
Scenes
BabyPenguin
Auto Save
Persp
Prefabs
Inspector
Project Settings
Lighting
BabyPenguin
Static
Tag baby
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Baby Penguin (Mesh Filter)
Mesh BabyPenguin
Mesh Renderer
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z
Info
Sphere Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 2.235174 Y 0.241513 Z 0.017879
Radius 0.2452329

Scene | Game | Asset Store

Shaded | 2D | Gizmos | All

Scenes | Penguin | Auto Save

y z

Right

Penguin > Prefabs

Inspector | Project Settings | Lighting

Penguin | Static

Tag: Untagged | Layer: Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Penguin (Mesh Filter)

Mesh: Penguin

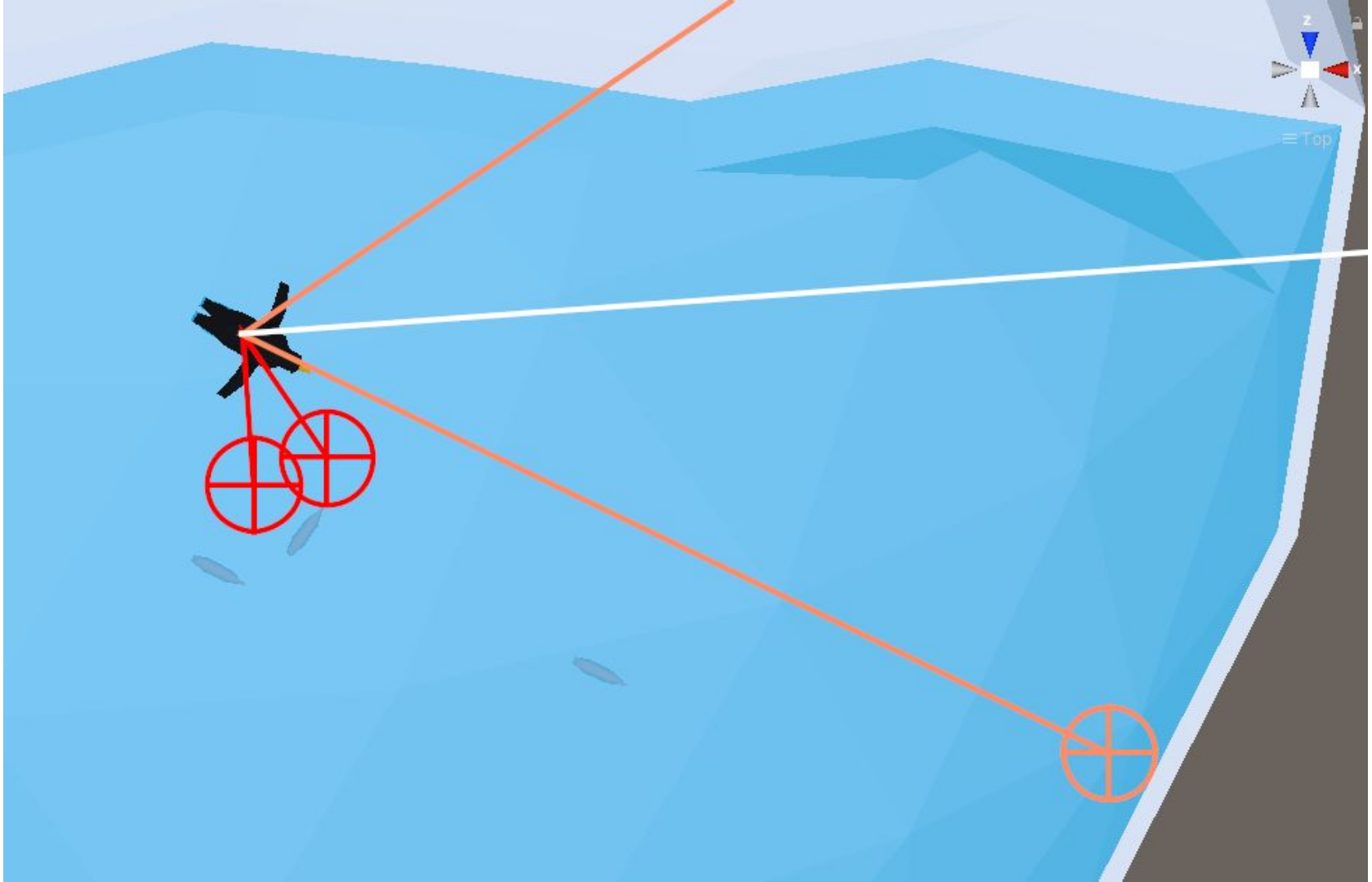
Mesh Renderer

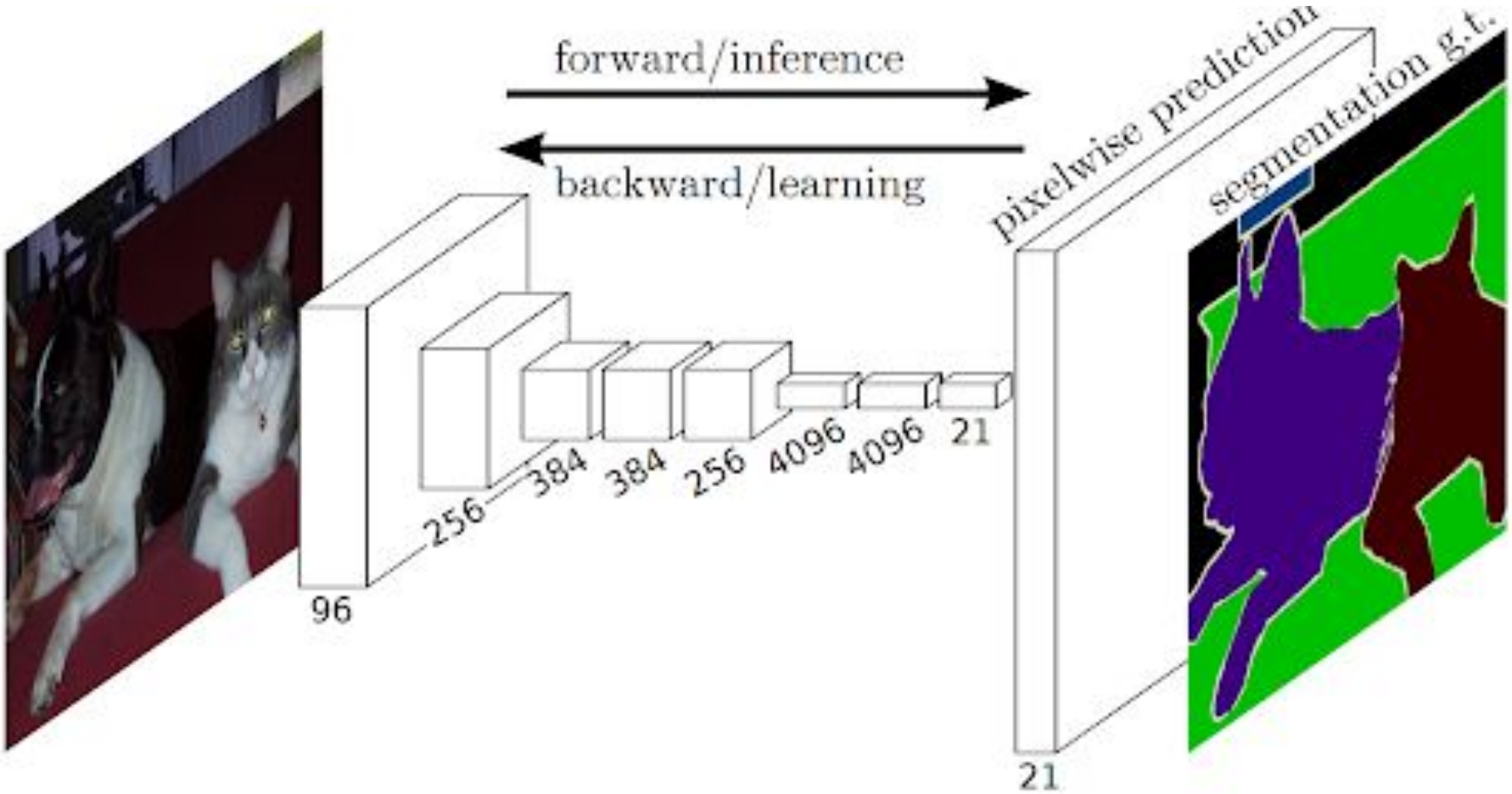
Rigidbody

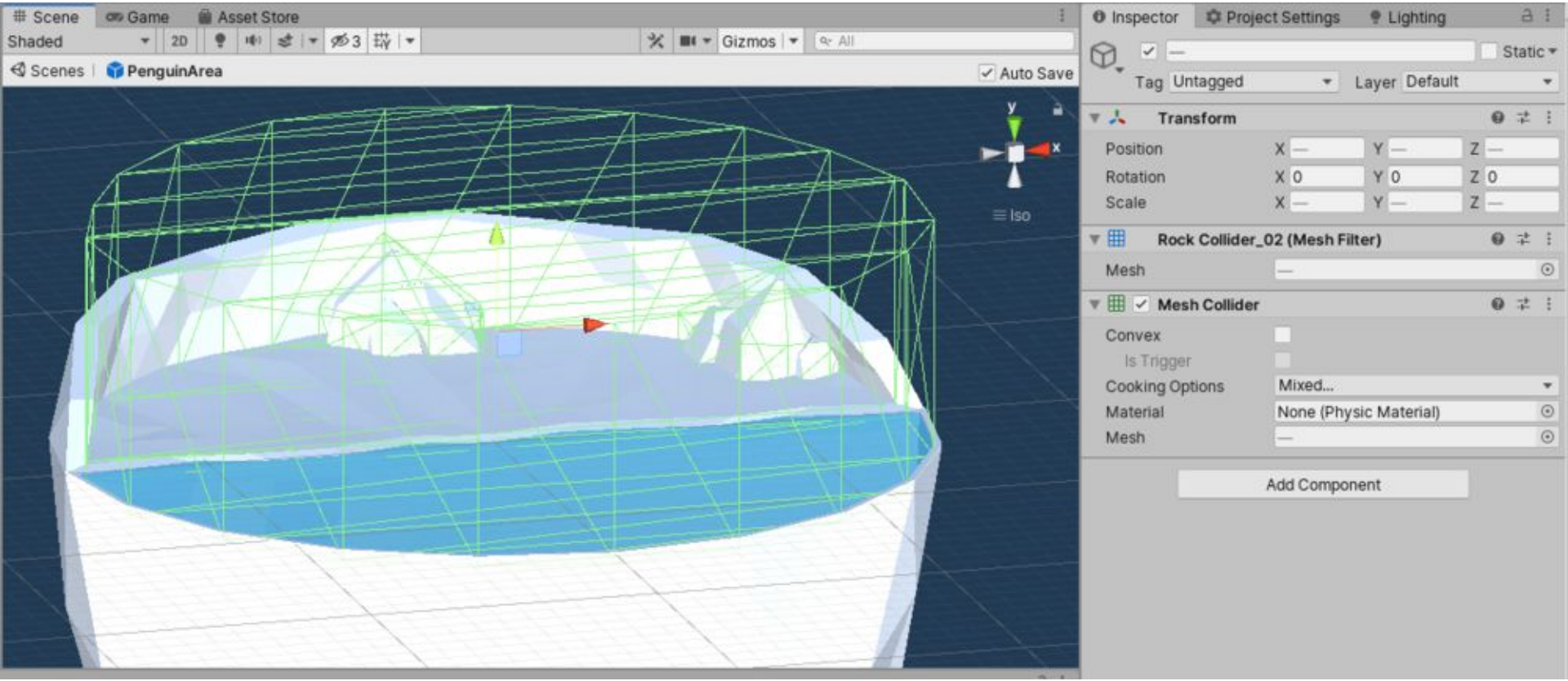
Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ☐ Y ☐ Z
Freeze Rotation	✓ X ☐ Y ✓ Z
Info	

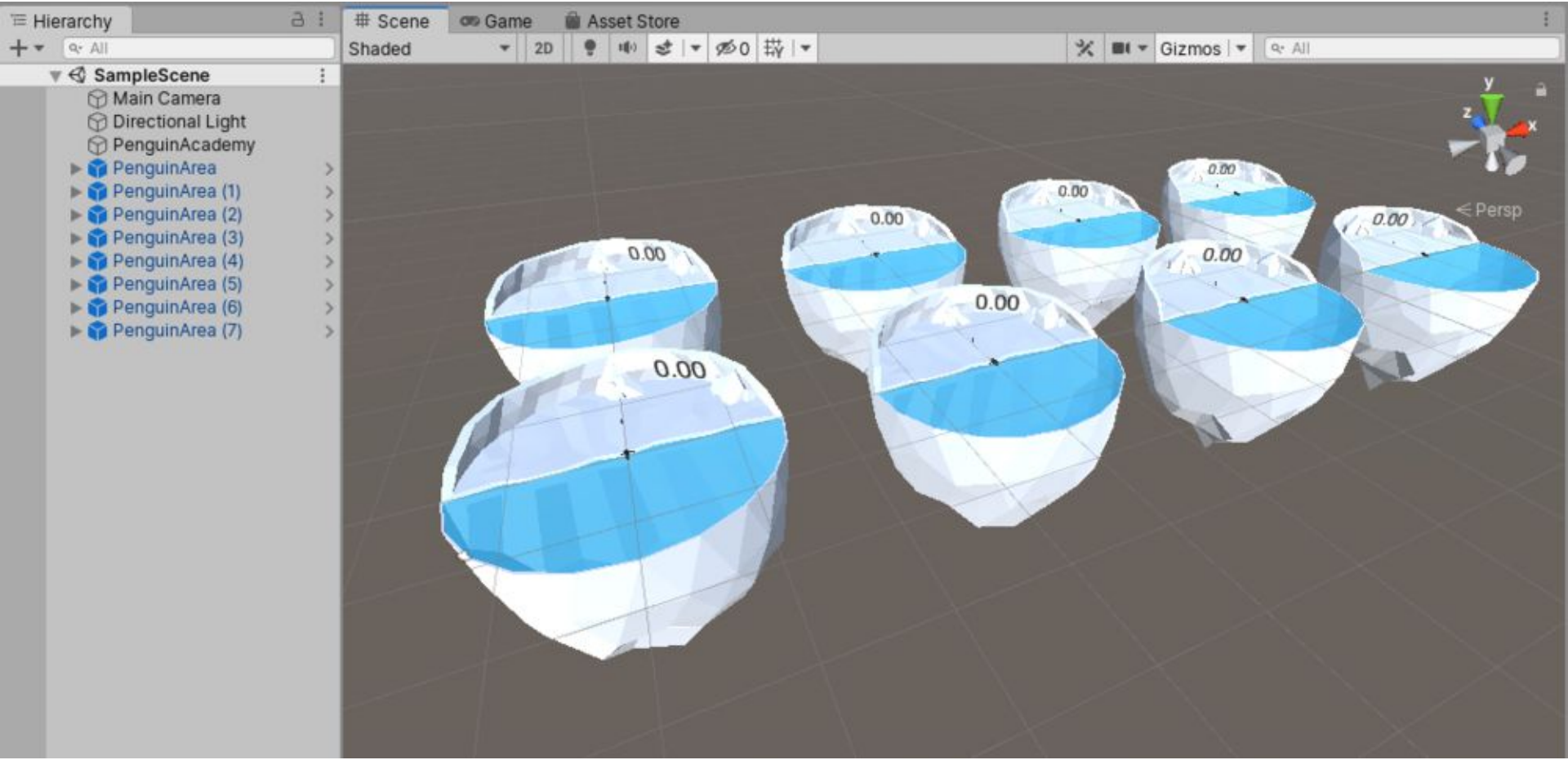
Capsule Collider

Edit Collider	
Is Trigger	☐
Material	None (Physic Material)
Center	X 0 Y 0.22 Z 0.13
Radius	0.24
Height	1.41
Direction	Z-Axis


z
x
Top


forward/inference
backward/learning
pixelwise prediction
segmentation g.t.
96
256
384
384
256
4096
4096
21
21

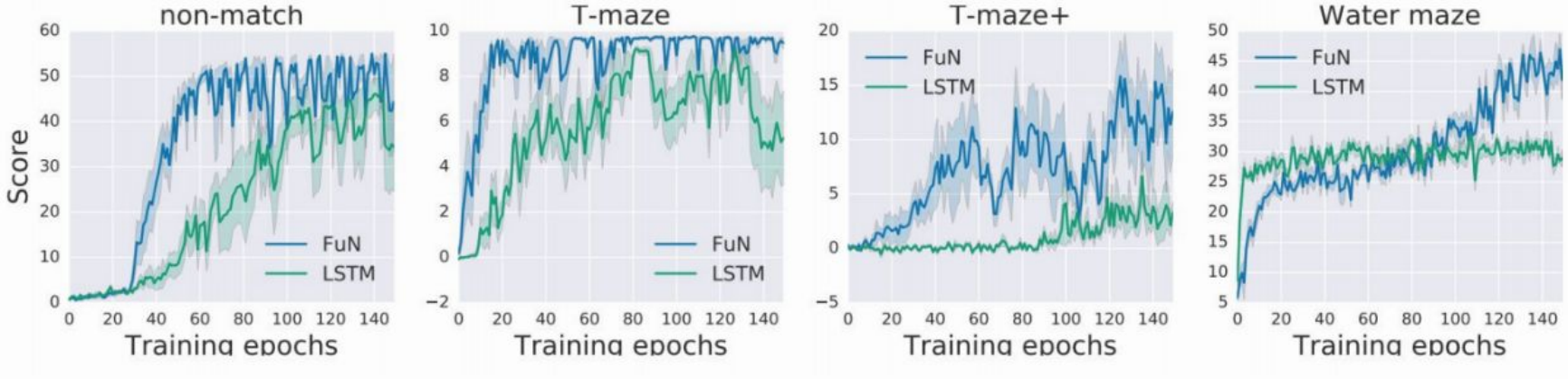

Scene
Game
Asset Store
Shaded
2D
Gizmos
Scenes
PenguinArea
Auto Save
Iso
Inspector
Project Settings
Lighting
Static
Tag Untagged
Layer Default
Transform
Position X Y Z
Rotation X 0 Y 0 Z 0
Scale X Y Z
Rock Collider_02 (Mesh Filter)
Mesh
Mesh Collider
Convex
Is Trigger
Cooking Options Mixed...
Material None (Physic Material)
Mesh
Add Component

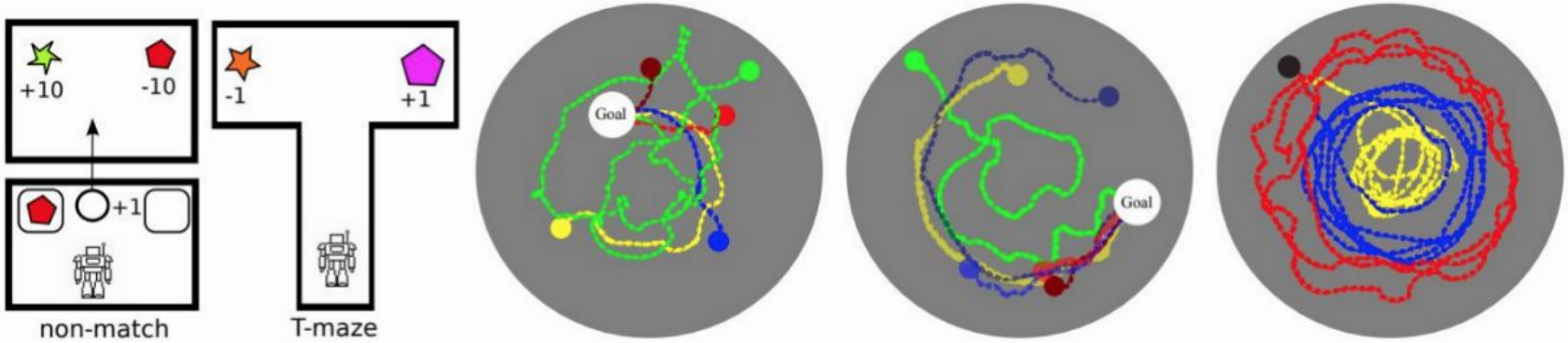

Hierarchy
Scene
Game
Asset Store
Shaded
2D
Gizmos
SampleScene
Main Camera
Directional Light
PenguinAcademy
PenguinArea
PenguinArea (1)
PenguinArea (2)
PenguinArea (3)
PenguinArea (4)
PenguinArea (5)
PenguinArea (6)
PenguinArea (7)
0.00
Persp

Anaconda Prompt (Anaconda3)

```
 Version information:
  ml-agents: 0.14.0,
  ml-agents-envs: 0.14.0,
  Communicator API: API-14,
  TensorFlow: 2.0.1
WARNING:tensorflow:From C:\Users\ak12\Anaconda3\envs\ml-agents-0.14\lib\site-packages\tensorflow_core\python\compat\v2_c
ompat.py:65: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in
 a future version.
Instructions for updating:
non-resource variables are not supported in the long term
INFO:mlagents_envs:Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
```

FuN Results





(a) (b)

"Feudal Networks for Hierarchical Reinforcement Learning", Vezhnevets et al. (2017)

Acknowledgements and Special Thanks

Rubin and Cindy Gruber

MPCR Lab Students

FAU College of Science

FAU Mathematical Sciences

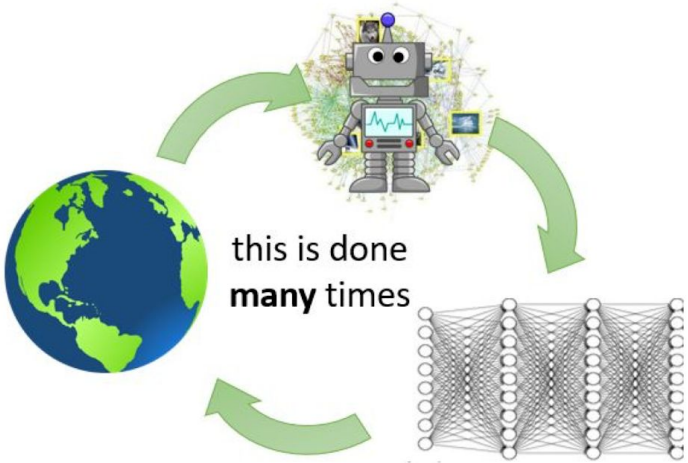
NVIDIA Corporation

Max Planck Florida Institute

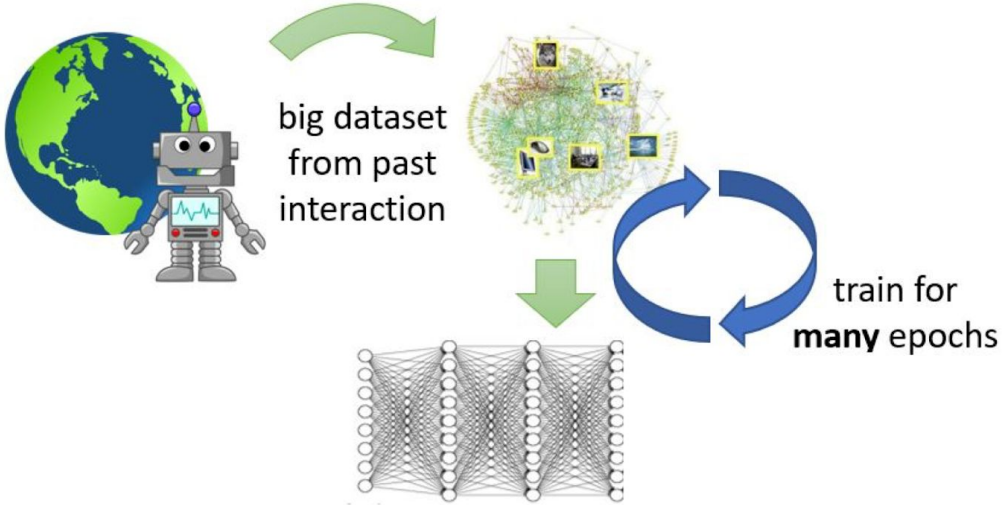
FAU Brain Institute

Center for Complex Systems

reinforcement learning

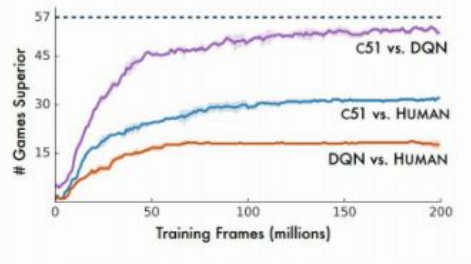


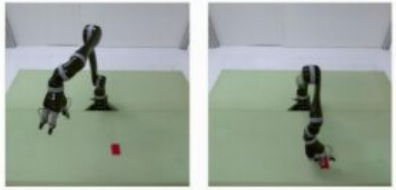
fully off-policy/offline reinforcement learning



Two Problems

Data/Sample Efficiency





Exploration

What's the problem?

this is easy (mostly)

Why?

this is impossible

Montezuma's revenge

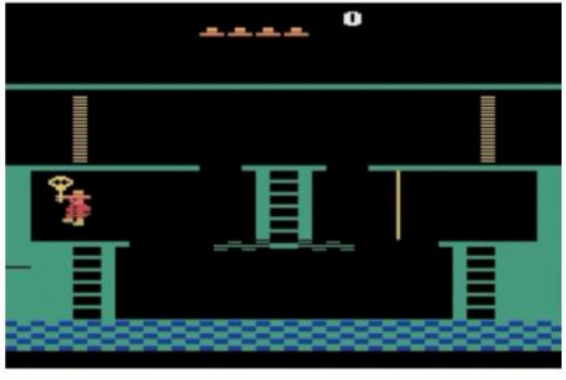
- Getting key = reward
- Opening door = reward
- Getting killed by skull = nothing (is it good? bad?)
- Finishing the game only weakly correlates with rewarding events
- We know what to do because we **understand** what these sprites mean!

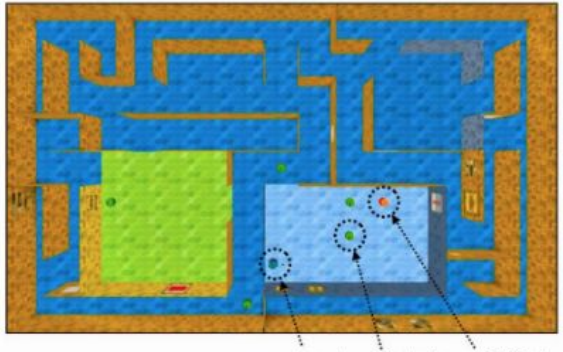
Exploration and exploitation examples

- Restaurant selection
  - Exploitation: go to your favorite restaurant
  - Exploration: try a new restaurant
- Online ad placement
  - Exploitation: show the most successful advertisement
  - Exploration: show a different random advertisement
- Oil drilling
  - Exploitation: drill at the best known location
  - Exploration: drill at a new location

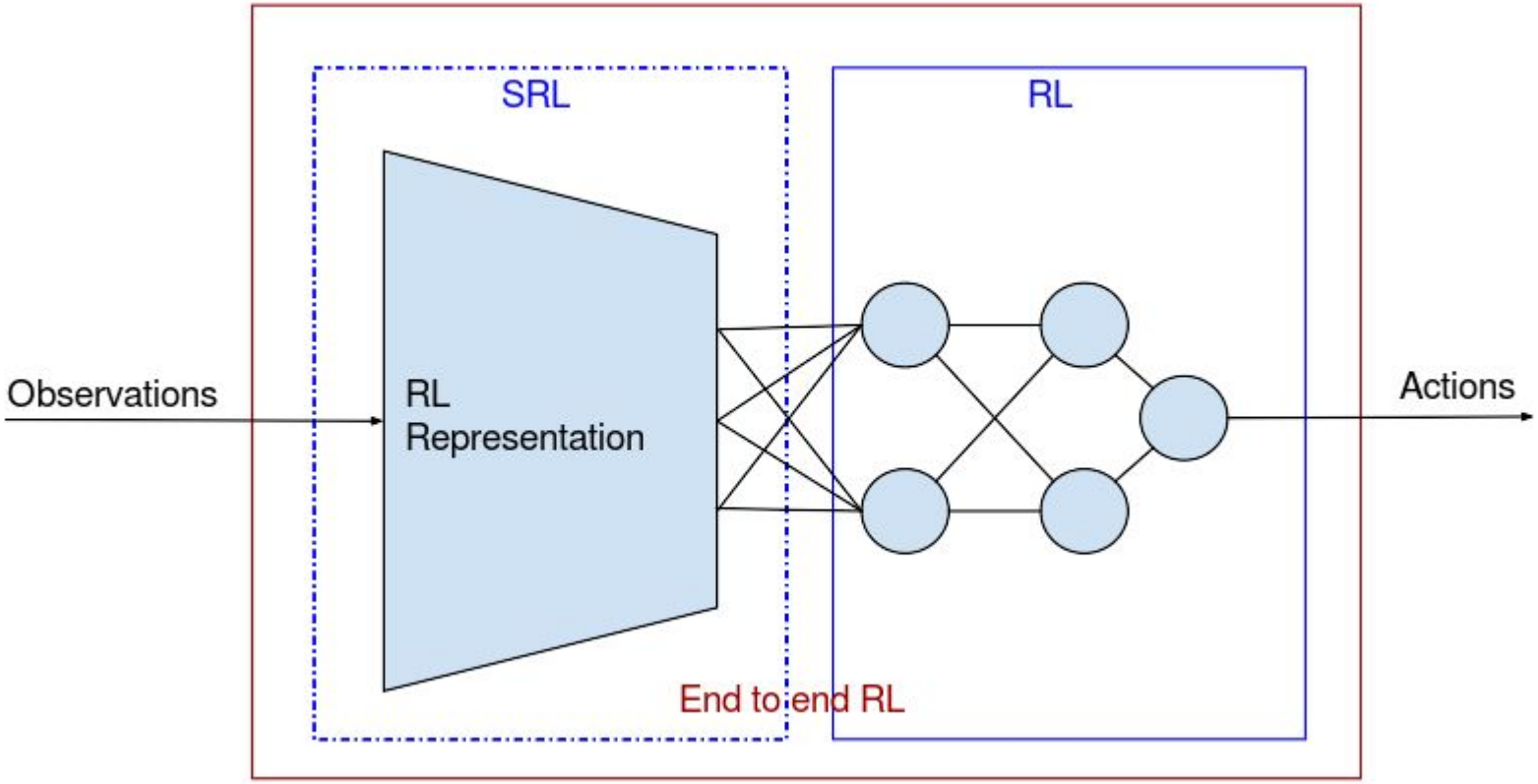
Unsupervised Reinforcement Learning

- The best deep RL methods are still very data hungry. Especially with **sparse rewards**.
- Obvious solution - Learn about the environment.
- We can augment an RL agent with **auxiliary prediction and control tasks** to improve data efficiency.
- The UNREAL agent - UNsupervised REinforcement and Auxiliary Learning.
  - "Reinforcement Learning with Unsupervised Auxiliary Tasks", Jaderberg et al. (2017)





"Reinforcement Learning with Unsupervised Auxiliary Tasks", Jaderberg, Mnih, Czarnecki et al. (2017)


SRL
RL
Observations
RL
Representation
Actions
End to end RL

How Much Information Does the Machine Need to Predict?

Y LeCun

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- Supervised Learning (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- Unsupervised/Predictive Learning (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

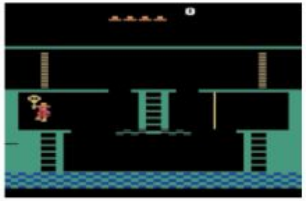
Deep RL Frontiers

- What are the exciting areas in Deep RL research?
  - HRL?
  - Deep RL for robotics?
  - Model-based RL?
- **Everything in deep RL is still exciting.**
- **Nothing is solved!**
  - How to learn a value function. How to learn a policy. How to train a deep network. Which nonlinearity to use ...

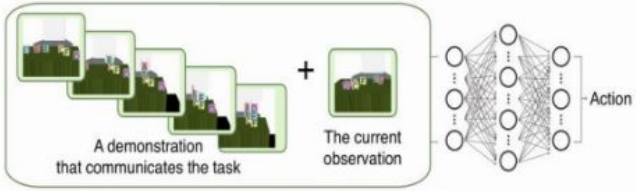
"Sim to Real Robot Learning from Pixels with Progressive Nets
Rusu et al. (2016)

"Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World", Tobin et al. (2017)

"Imagination-Augmented Agents for Deep RL", Weber, Racanière, Reichert et al. (2017)



Feudal Networks, Vezhnevets et al. (2017)

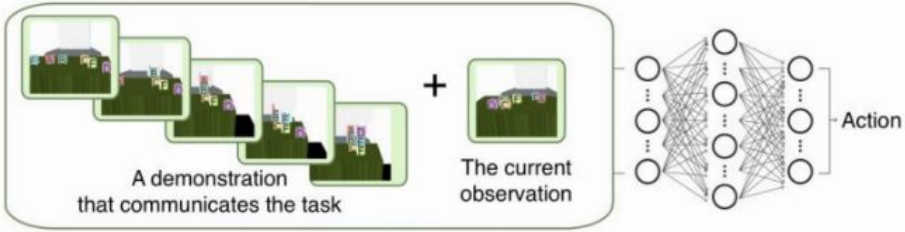


"One-Shot Imitation Learning", Duan et al. (2017)

Deep RL for Real Robots

- Already a lot of exciting work from Levine, Abbeel, Finn, and many others.
- Feels like a breakthrough is coming.
- But how can slow, data inefficient deep RL help with robots?

Imitation Learning



"One-Shot Imitation Learning", Duan et al. (2017)

Sim-to-Real Learning

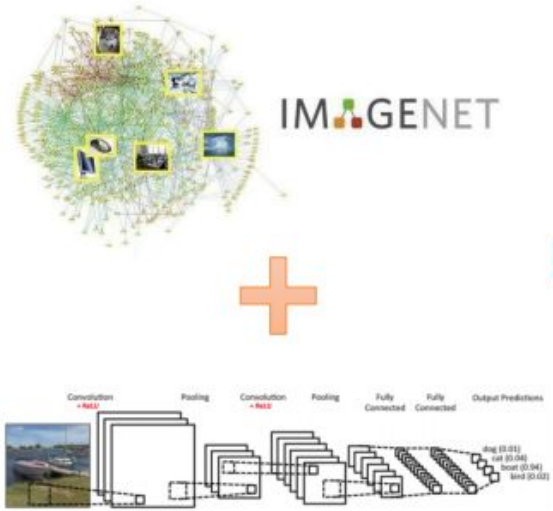
"Sim to Real Robot Learning from Pixels with Progressive Nets"
Rusu et al. (2016)

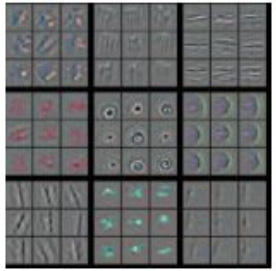
"Domain Randomization for Transferring Deep Neural Networks
from Simulation to the Real World", Tobin et al. (2017)

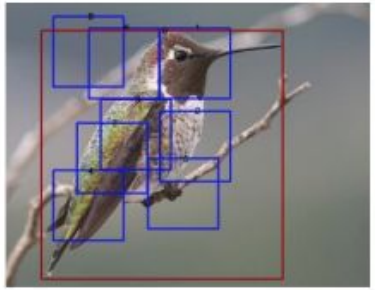
Reframing Control as an Inference Problem

Finetuning

The most popular transfer learning method in (supervised) deep learning!



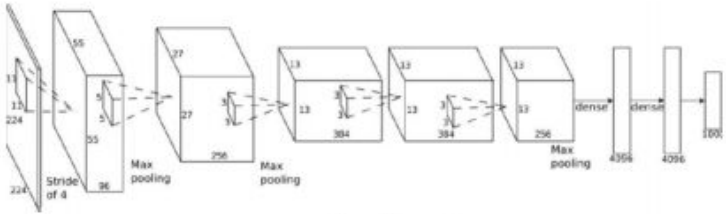




Where are the “ImageNet” features of RL?

Imitation Learning

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

$\mathbf{o}_t$

$\mathbf{a}_t$

training data

supervised learning

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

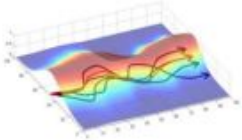
behavioral cloning

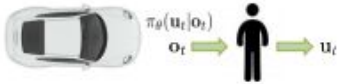
Imitation learning: recap

$\mathbf{o}_t$
$\mathbf{a}_t$

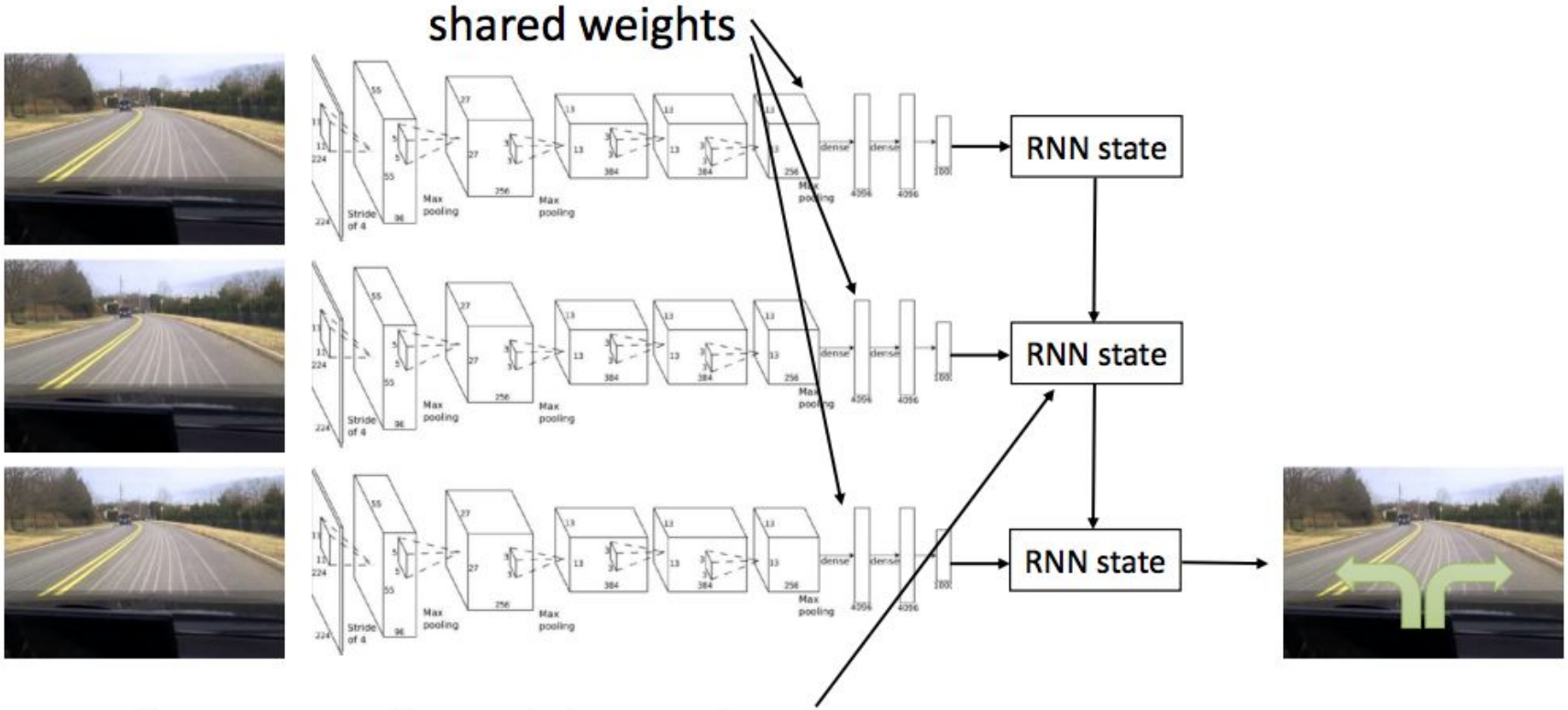
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

- Often (but not always) insufficient by itself
  - Distribution mismatch problem
- Sometimes works well
  - Hacks (e.g. left/right images)
  - Samples from a stable trajectory distribution
  - Add more **on-policy** data, e.g. using Dagger
  - Better models that fit more accurately





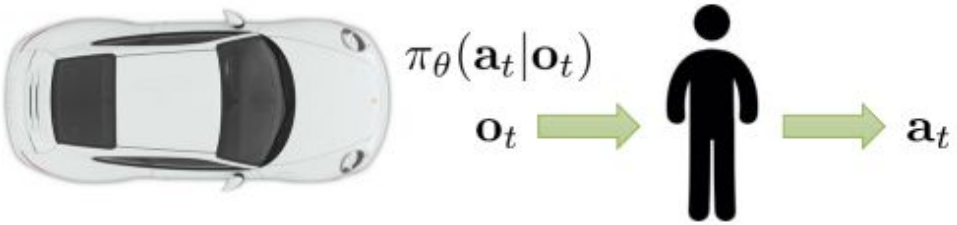
How can we use the whole history?



Typically, LSTM cells work better here

What's the problem?

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



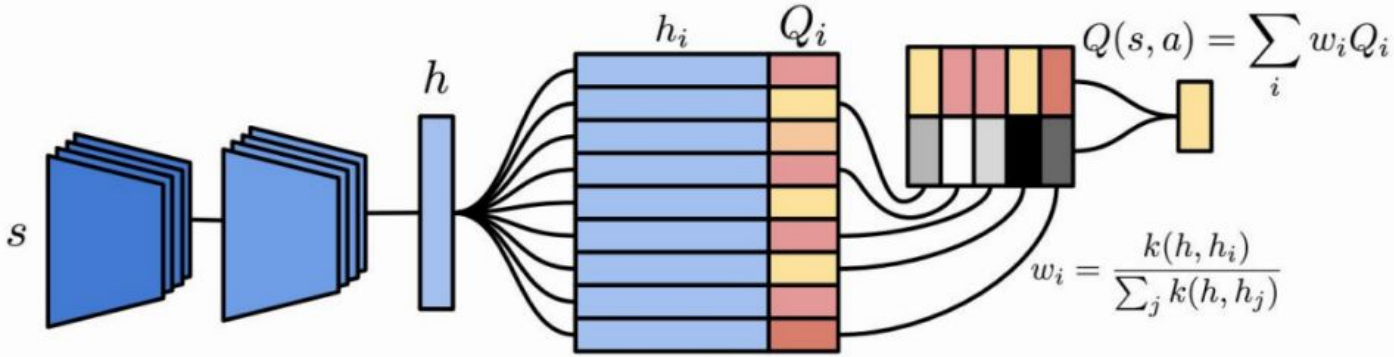
Feudal Reinforcement Learning

- Agent with a two level hierarchy: **manager** and **worker**.
- Manager:
  - Does not act in the environment directly.
  - Sets goals for the worker.
  - Gets **rewarded for setting good goals** with the true reward.
- Worker:
  - Acts in the environment.
  - Gets **rewarded for achieving goals** set by the manager.
  - This is potentially a much richer learning signal.
- Key problems: how to represent goals and determine when they've been achieved.

"Feudal Reinforcement Learning", Dayan and Hinton (1993)

Neural Episodic Control

- Purely parametric approaches to deep RL are very data inefficient.
- A hybrid parametric/non-parametric method can be much more efficient.
- NEC - Represent the action value function as a table:
  - Slowly changing learned keys.
  - Rapidly changing Q-value estimates.

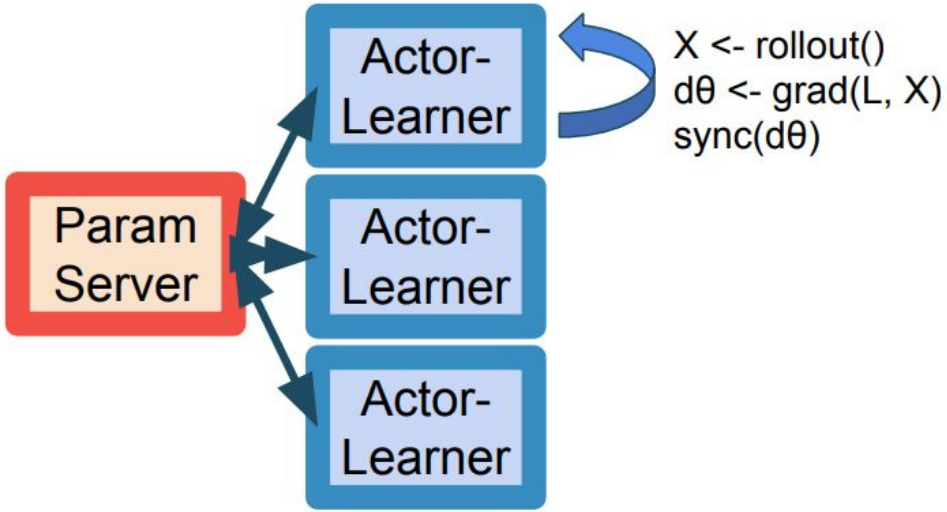


History of large scale distributed RL

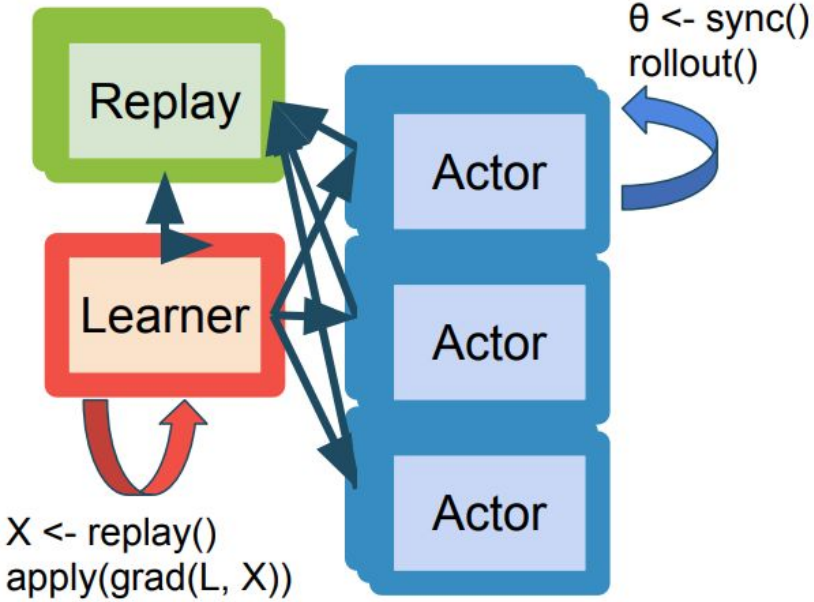
2013	2015	2016	2018	2018	2019
DQN	**GORILA**	**A3C**	**IMPALA**	**Ape-X**	**R2D3**
Playing Atari with Deep Reinforcement Learning (Mnih 2013)	Massively Parallel Methods for Deep Reinforcement Learning (Nair 2015)	Asynchronous Methods for Deep Reinforcement Learning (Mnih 2016)	IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures (Espeholt 2018)	Distributed Prioritized Experience Replay (Horgan 2018)	Making Efficient Use of Demonstrations to Solve Hard Exploration Problems (Le Paine 2019)

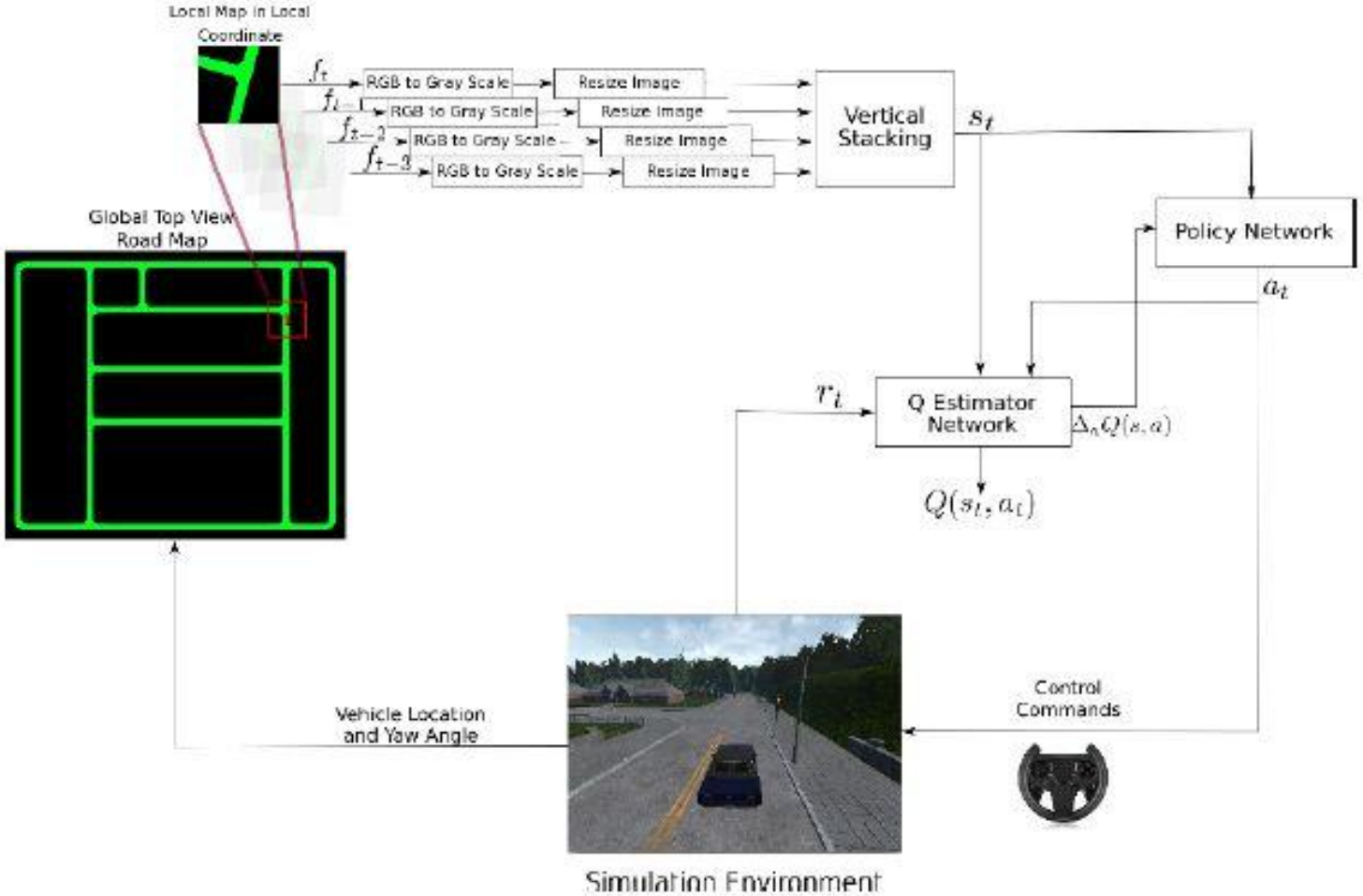
Many RL loop decompositions

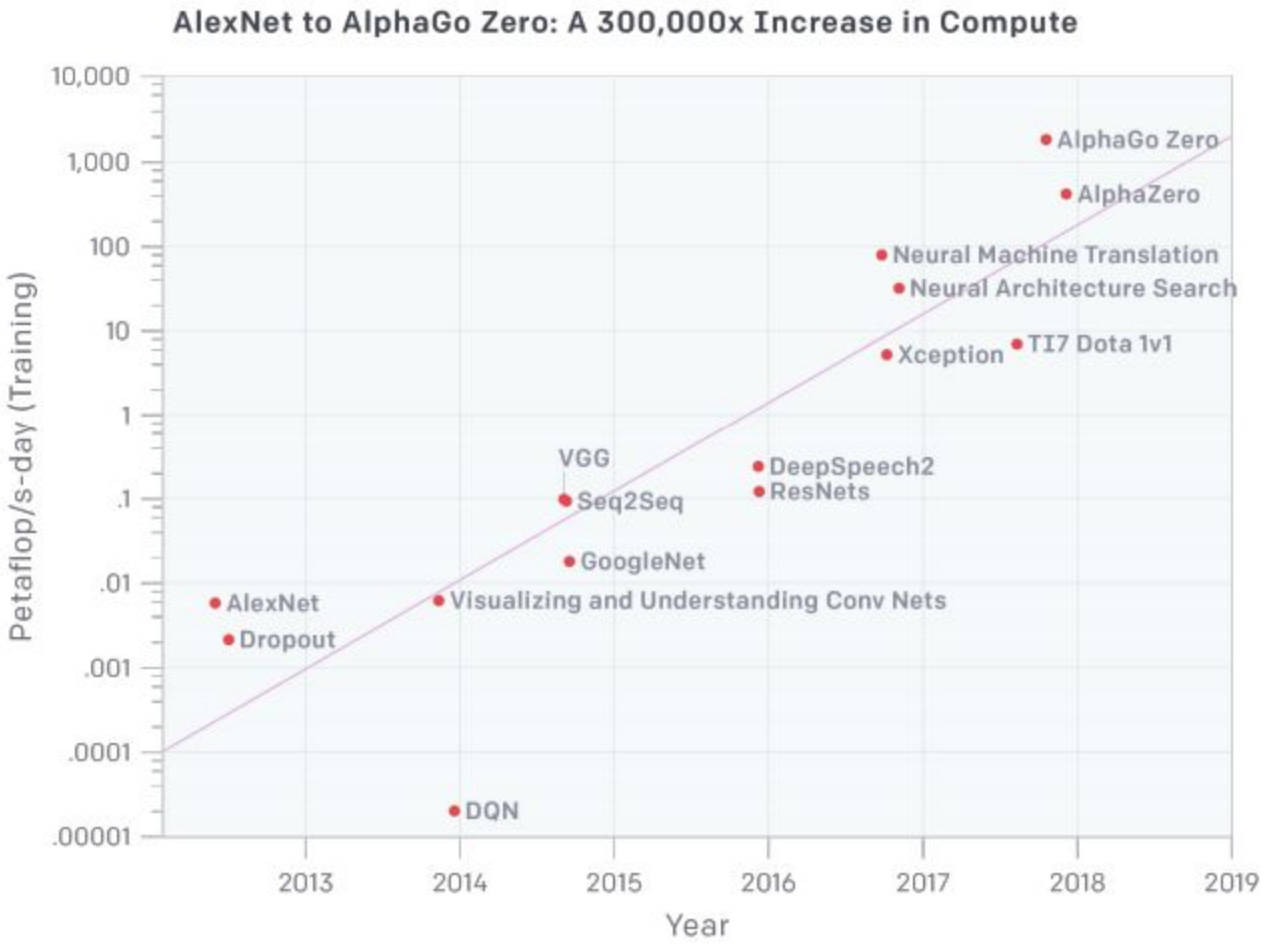
Async DQN (Mnih et al; 2016)



Ape-X DQN (Horgan et al; 2018)




Local Map in Local Coordinate
f_t
f_{t-1}
f_{t-2}
f_{t-3}
RGB to Gray Scale
RGB to Gray Scale
RGB to Gray Scale
RGB to Gray Scale
Resize Image
Resize Image
Resize Image
Resize Image
Vertical Stacking
s_t
Policy Network
a_t
Global Top View Road Map
r_t
Q Estimator Network
$\Delta_n Q(s, a)$
$Q(s_t, a_t)$
Vehicle Location and Yaw Angle
Control Commands
Simulation Environment


AlexNet to AlphaGo Zero: A 300,000x Increase in Compute
Petaflop/s-day (Training)
10,000
1,000
100
10
1
.1
.01
.001
.0001
.00001
2013
2014
2015
2016
2017
2018
2019
Year
AlphaGo Zero
AlphaZero
Neural Machine Translation
Neural Architecture Search
TI7 Dota 1v1
Xception
VGG
DeepSpeech2
ResNets
Seq2Seq
GoogleNet
AlexNet
Visualizing and Understanding Conv Nets
Dropout
DQN

Population Based Training of Neural Networks

Max Jaderberg **Valentin Dalibard** **Simon Osindero** **Wojciech M. Czarnecki**

Jeff Donahue **Ali Razavi** **Oriol Vinyals** **Tim Green** **Iain Dunning**

Karen Simonyan **Chrisantha Fernando** **Koray Kavukcuoglu**

DeepMind, London, UK

Abstract

Neural networks dominate the modern machine learning landscape, but their training and success still suffer from sensitivity to empirical choices of hyperparameters such as model architecture, loss function, and optimisation algorithm. In this work we present *Population Based Training (PBT)*, a simple asynchronous optimisation algorithm which effectively utilises a fixed computational budget to jointly optimise a population of models and their hyperparameters to maximise performance. Importantly, PBT discovers a schedule of hyperparameter settings rather than following the generally sub-optimal strategy of trying to find a single fixed set to use for the whole course of training. With just a small modification to a typical distributed hyperparameter training framework, our method allows robust and reliable training of models. We demonstrate the effectiveness of PBT on deep reinforcement learning problems, showing faster wall-clock convergence and higher final performance of agents by optimising over a suite of hyperparameters. In addition, we show the same method can be applied to supervised learning for machine translation, where PBT is used to maximise the BLEU score directly, and also to training of Generative Adversarial Networks to maximise the Inception score of generated images. In all cases PBT results in the automatic discovery of hyperparameter schedules and model selection which results in stable training and better final performance.

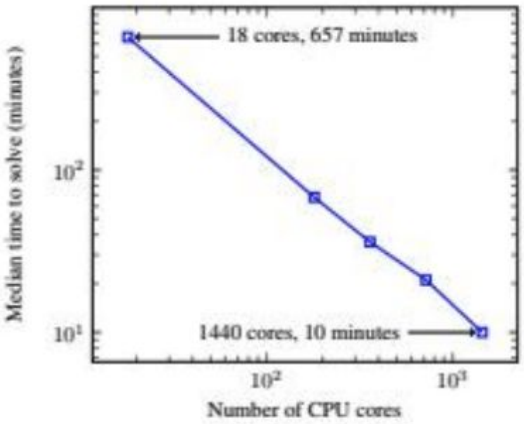
Evolution Strategies

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans Jonathan Ho Xi Chen Szymon Sidor Ilya Sutskever
OpenAI

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for



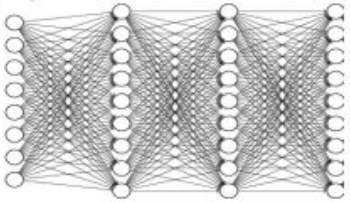
Fitted value iteration

how do we represent $V(\mathbf{s})$?

big table, one entry for each discrete $\mathbf{s}$

neural net function $V : \mathcal{S} \to \mathbb{R}$

$\mathbf{s}$



$V(\mathbf{s})$

parameters ϕ

$$\mathbf{s} = 0 : V(\mathbf{s}) = 0.2$$
$$\mathbf{s} = 1 : V(\mathbf{s}) = 0.3$$
$$\mathbf{s} = 2 : V(\mathbf{s}) = 0.5$$

curse of dimensionality

$$|\mathcal{S}| = (255^3)^{200 \times 200}$$

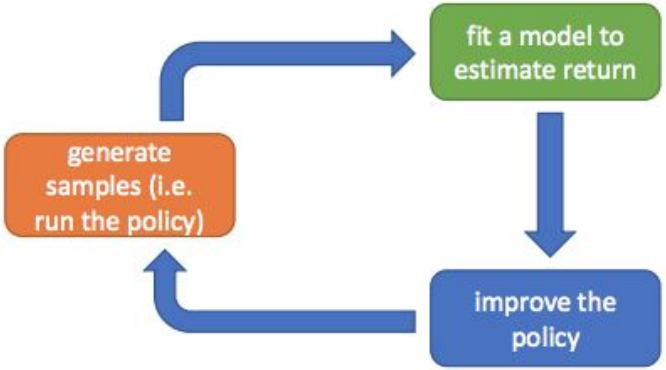
(more than atoms in the universe)

$$Q^\pi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]$$

$$\mathcal{L}(\phi) = \frac{1}{2} \left\| V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \right\|^2$$

fitted value iteration algorithm:

1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)])$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$



$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$

TD-Gammon, Gerald Tesauro 1992

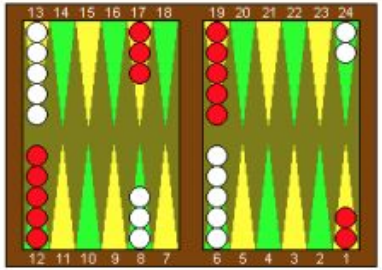


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

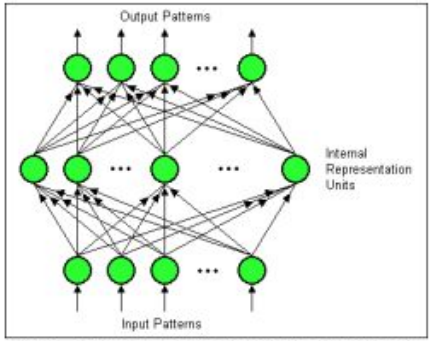
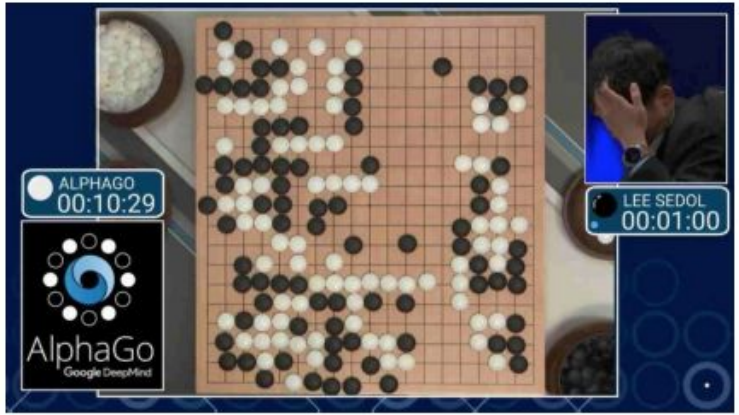


Figure 1. An illustration of the multilayer perception architecture used in TD-Gammon's neural network. This architecture is also used in the popular backpropagation learning procedure. Figure reproduced from [9].

reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:
expected outcome given board state

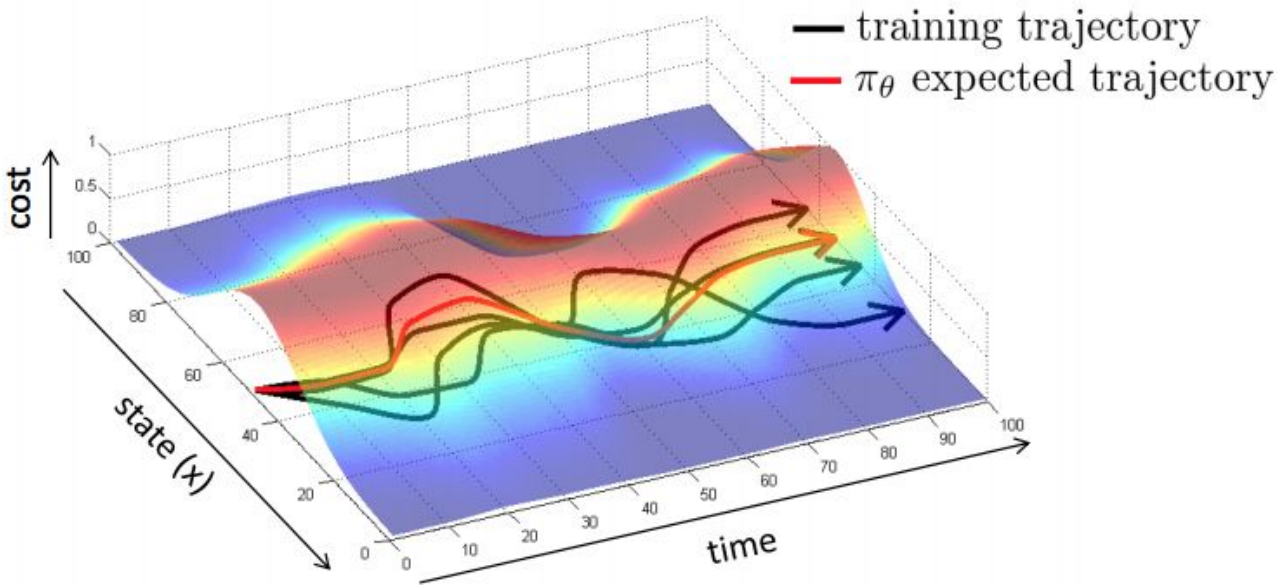
AlphaGo, Silver et al. 2016



reward: game outcome

value function $\hat{V}_\phi^\pi(\mathbf{s}_t)$:
expected outcome given board state

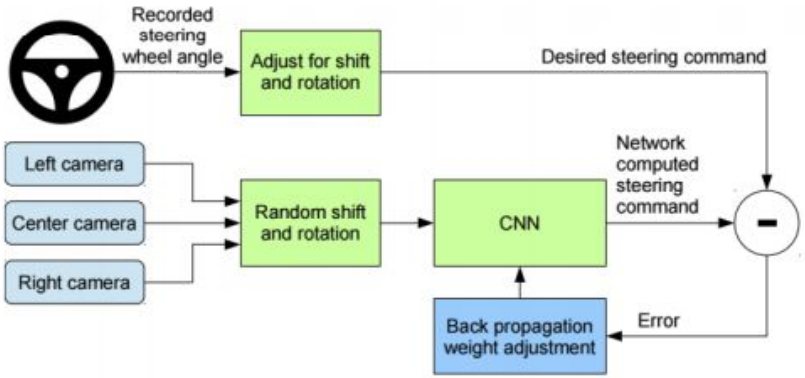
Can we make it work more often?

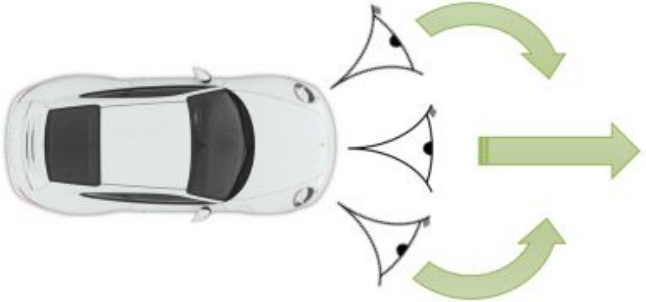


stability

(more on this later)

Why did that work?

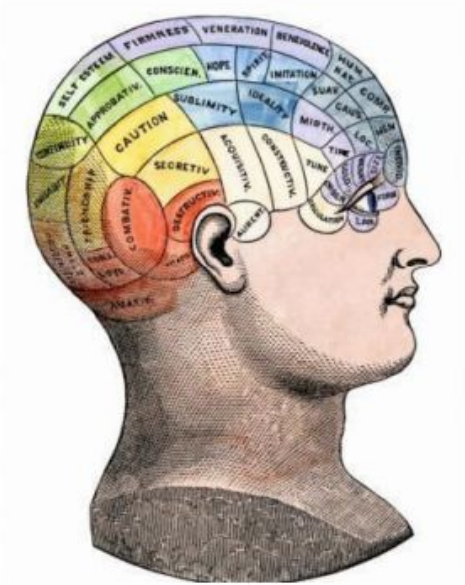




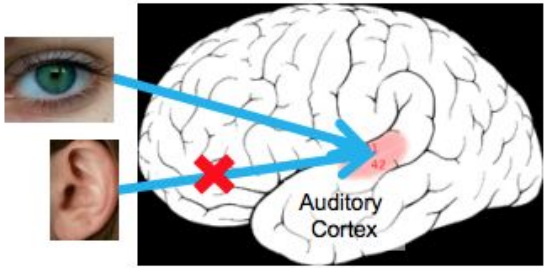
Bojarski et al. '16, NVIDIA

A single algorithm?

- An algorithm for each “module”?
- Or a single flexible algorithm?



Seeing with your tongue



[BrainPort; Martinez et al; Roe et al.]

adapted from A. Ng

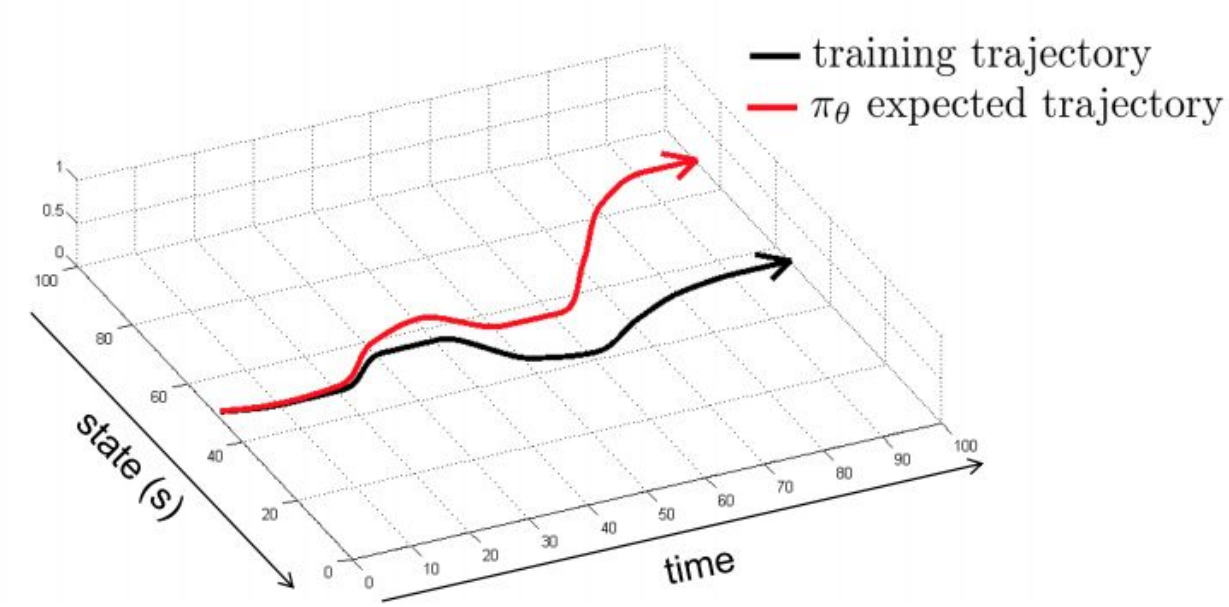
What must that single algorithm do?

- Interpret rich sensory inputs

- Choose complex actions

Does it work?

No!



Are there other forms of supervision?

- Learning from demonstrations
  - Directly copying observed behavior
  - Inferring rewards from observed behavior (inverse reinforcement learning)
- Learning from observing the world
  - Learning to predict
  - Unsupervised learning
- Learning from other tasks
  - Transfer learning
  - Meta-learning: learning to learn

In the real world…

Kalashnikov et al. ‘18

Exploration vs. Exploitation Dilemma

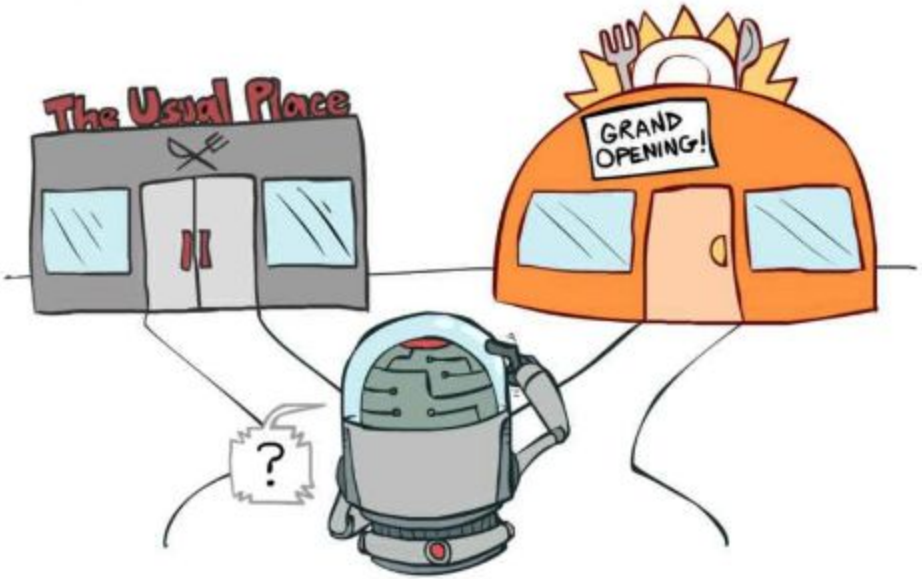


Fig 8. A real-life example of the exploration vs exploitation dilemma: where to eat? Source: UC Berkeley AI course, lecture 11.

Algorithm 1 Distributed DQN Algorithm

Initialise replay memory D to size P.
Initialise the training network for the action-value function $Q(s, a; \theta)$ with weights θ and target network $Q(s, a; \theta^-)$ with weights $\theta^- = \theta$.
for $episode = 1$ **to** M **do**
  Initialise the start state to s_1.
  Update θ from parameters θ^+ of the parameter server.
  for $t = 1$ **to** T **do**
    With probability ϵ take a random action a_t or else $a_t = \operatorname{argmax}_a Q(s, a; \theta)$.
    Execute the action in the environment and observe the reward r_t and the next state s_{t+1}. Store (s_t, a_t, r_t, s_{t+1}) in D.
    Update θ from parameters θ^+ of the parameter server.
    Sample random mini-batch from D. And for each tuple (s_i, a_i, r_i, s_{i+1}) set target y_t as
    if s_{i+1} is $terminal$ **then**
      $y_t = r_i$
    else
      $y_t = r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta^-)$
    end if
    Calculate the loss $L_t = (y_t - Q(s_i, a_i; \theta)^2)$.
    Compute gradients with respect to the network parameters θ using equation 2.
    Send gradients to the parameter server.
    Every global N steps sync θ^- with parameters θ^+ from the parameter server.
  end for
end for

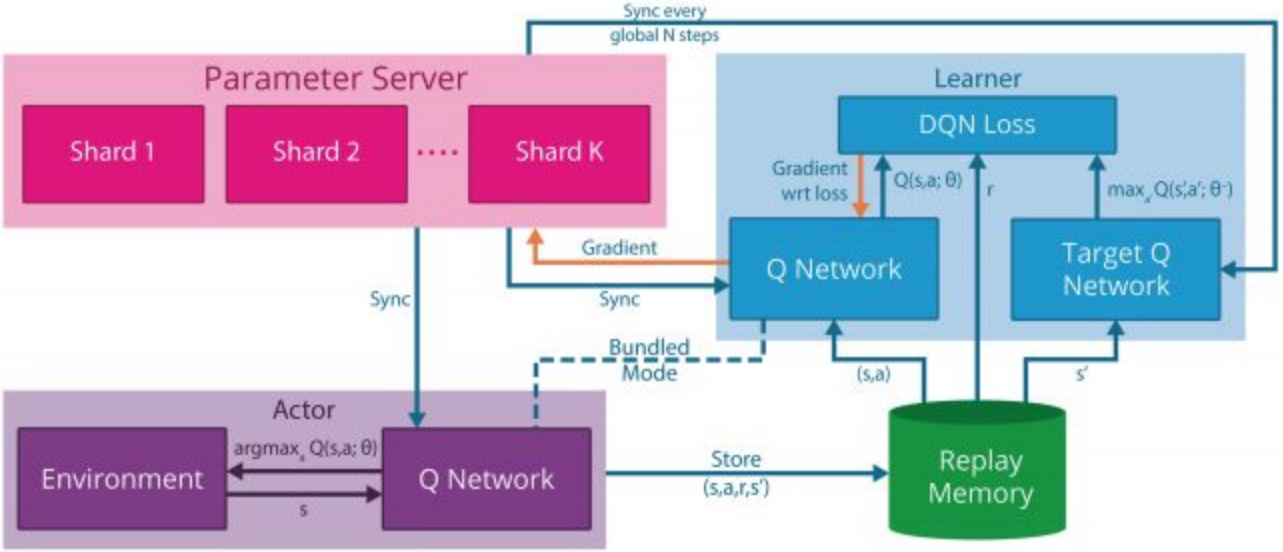


Figure 2. The Gorila agent parallelises the training procedure by separating out learners, actors and parameter server. In a single experiment, several learner processes exist and they continuously send the gradients to parameter server and receive updated parameters. At the same time, independent actors can also in parallel accumulate experience and update their Q-networks from the parameter server.

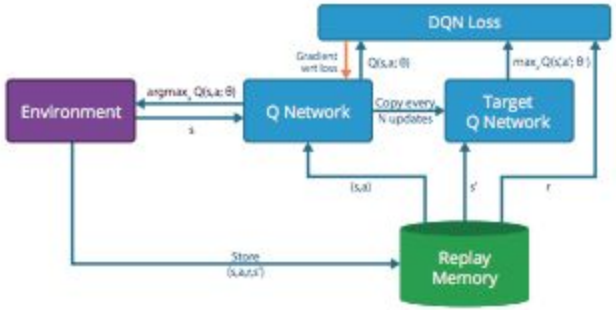


Figure 1. The DQN algorithm is composed of three main components, the Q-network ($Q(s, a; \theta)$) that defines the behavior policy, the target Q-network ($Q(s, a; \theta^-)$) that is used to generate target Q values for the DQN loss term and the replay memory that the agent uses to sample random transitions for training the Q-network.

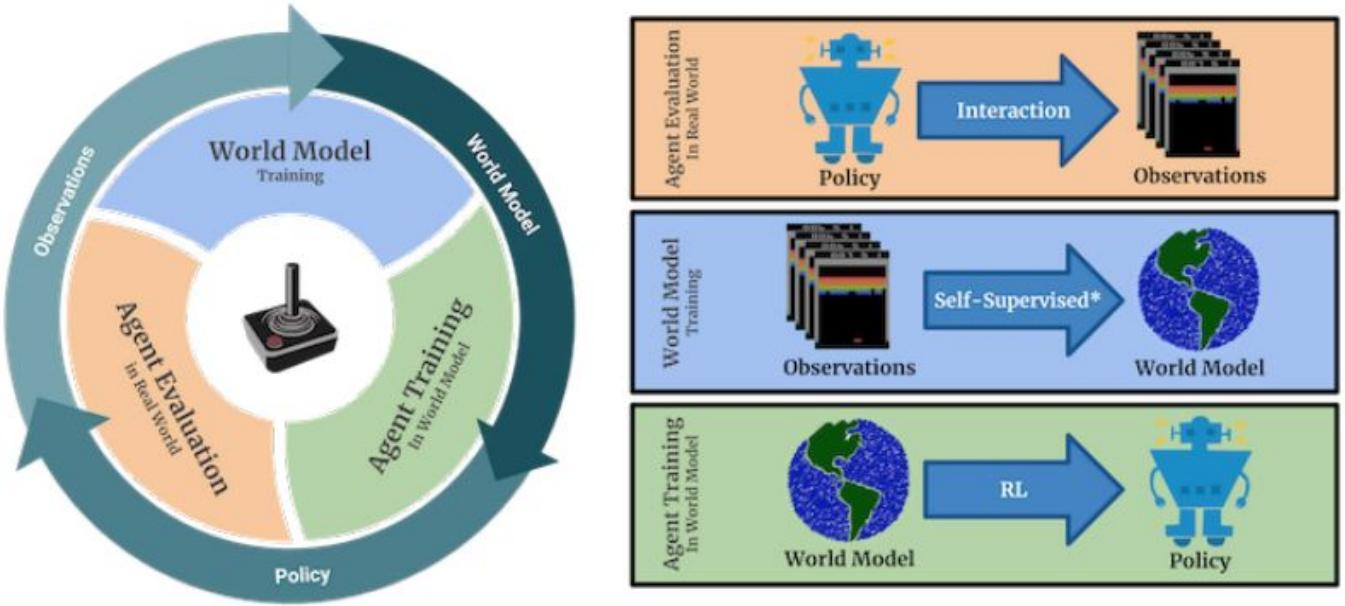
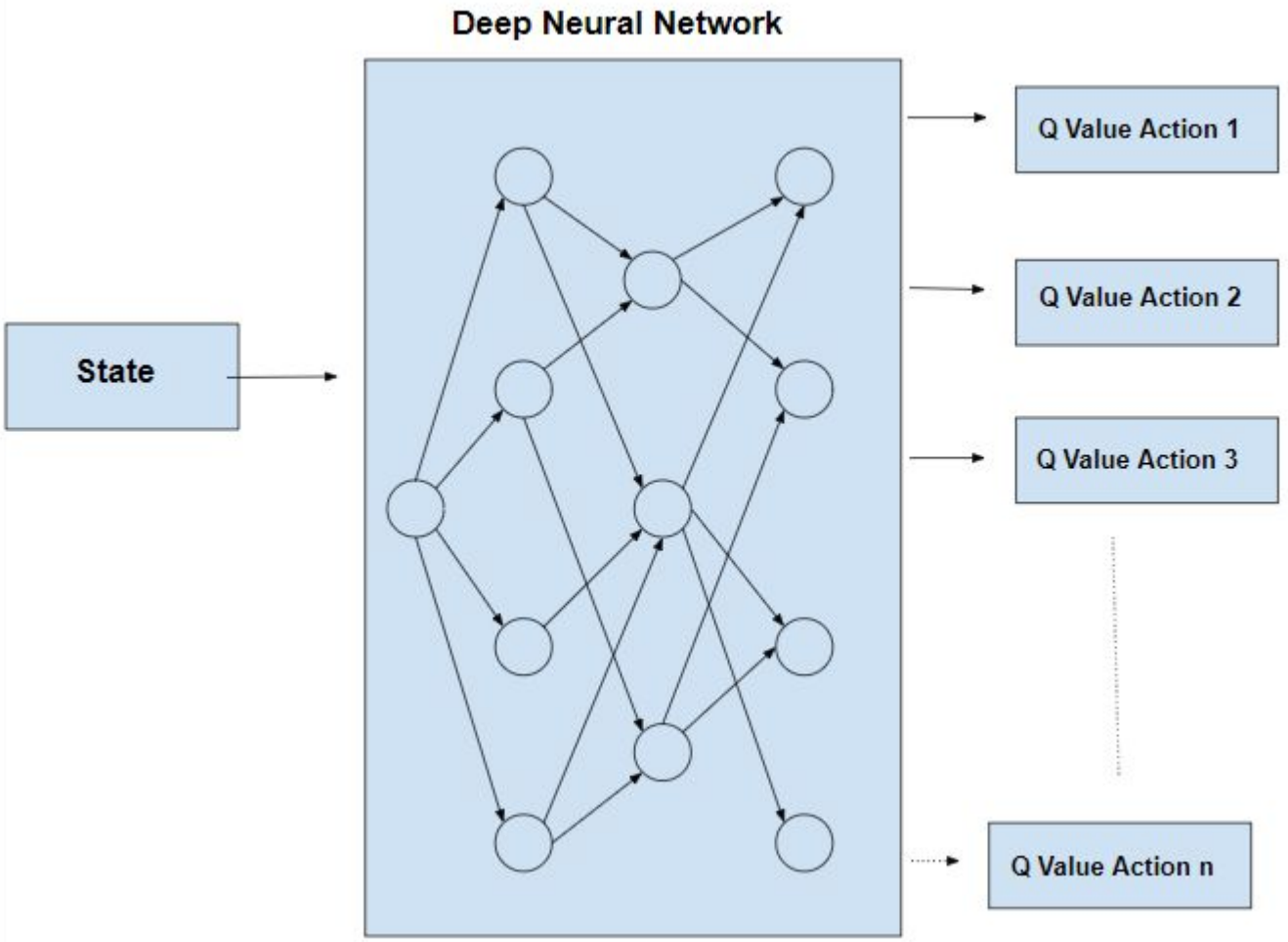
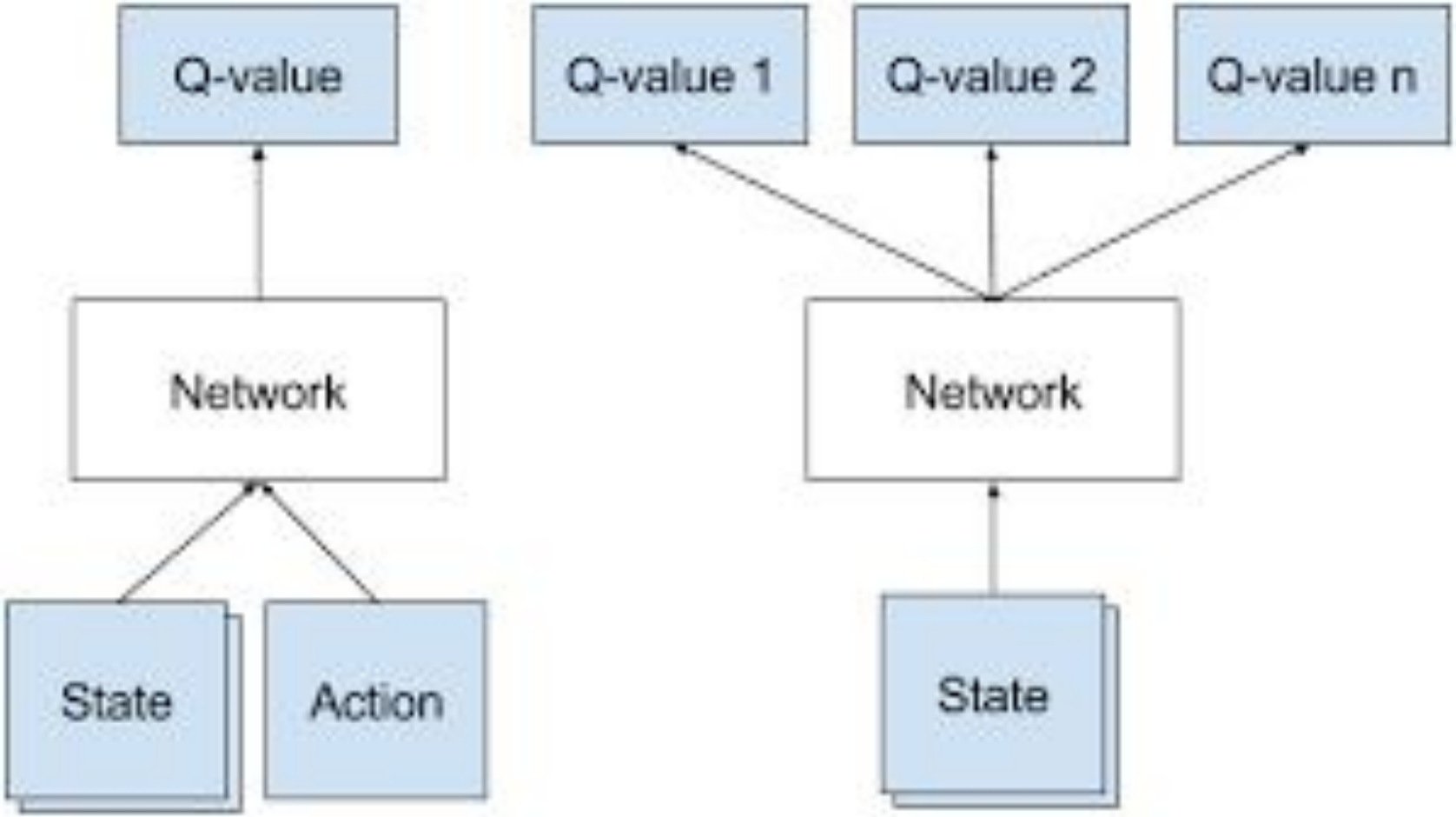


Figure 1: Main loop of SimPLe. 1) the agent starts interacting with the real environment following the latest policy (initialized to random). 2) the collected observations will be used to train (update) the current world model. 3) the agent updates the policy by acting inside the world model. The new policy will be evaluated to measure the performance of the agent as well as collecting more data (back to 1). Note that world model training is self-supervised for the observed states and supervised for the reward.

Algorithm 1: Pseudocode for SimPLe

Initialize policy $\boldsymbol{\pi}$
Initialize model parameters $\boldsymbol{\theta}$ of env'
Initialize empty set $\mathbf{D}$
while not done **do**
  ▷ collect observations from real env.
  $\mathbf{D} \leftarrow \mathbf{D} \cup \text{COLLECT}(env, \pi)$
  ▷ update model using collected data.
  $\boldsymbol{\theta} \leftarrow \text{TRAIN_SUPERVISED}(env', \mathbf{D})$
  ▷ update policy using world model.
  $\boldsymbol{\pi} \leftarrow \text{TRAIN_RL}(\boldsymbol{\pi}, env')$
end while


Deep Neural Network
State
Q Value Action 1
Q Value Action 2
Q Value Action 3
Q Value Action n


Q-value
Network
State
Action
Q-value 1
Q-value 2
Q-value n
Network
State

www.huntleyarchives.com
Film 97453
18:59:00