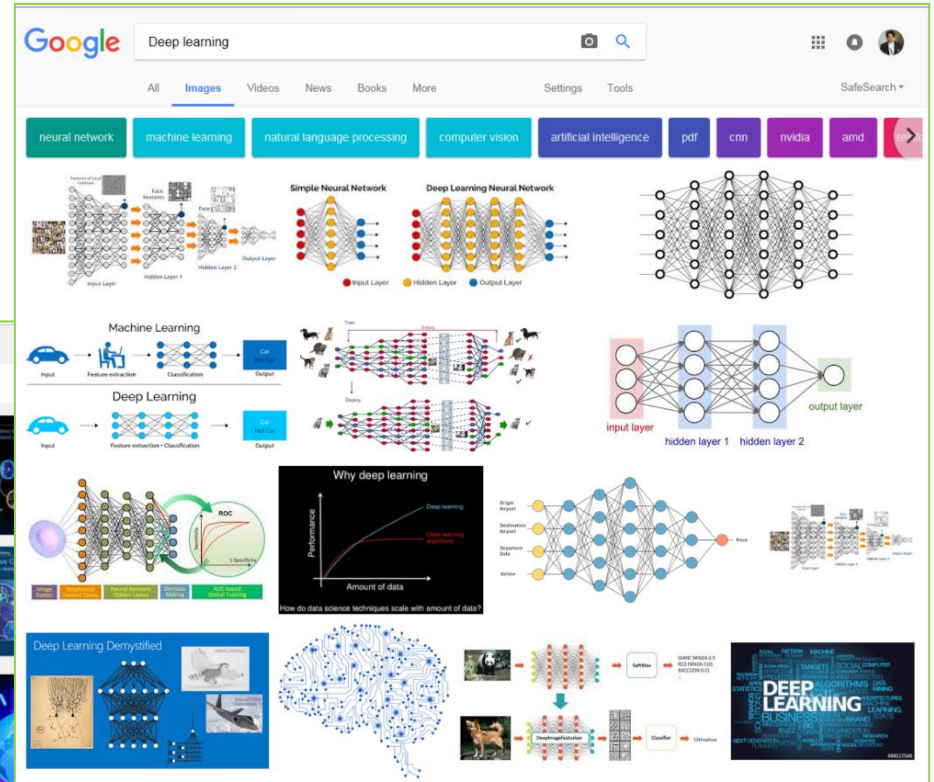
Lecture 11: Intro to Reinforcement Learning

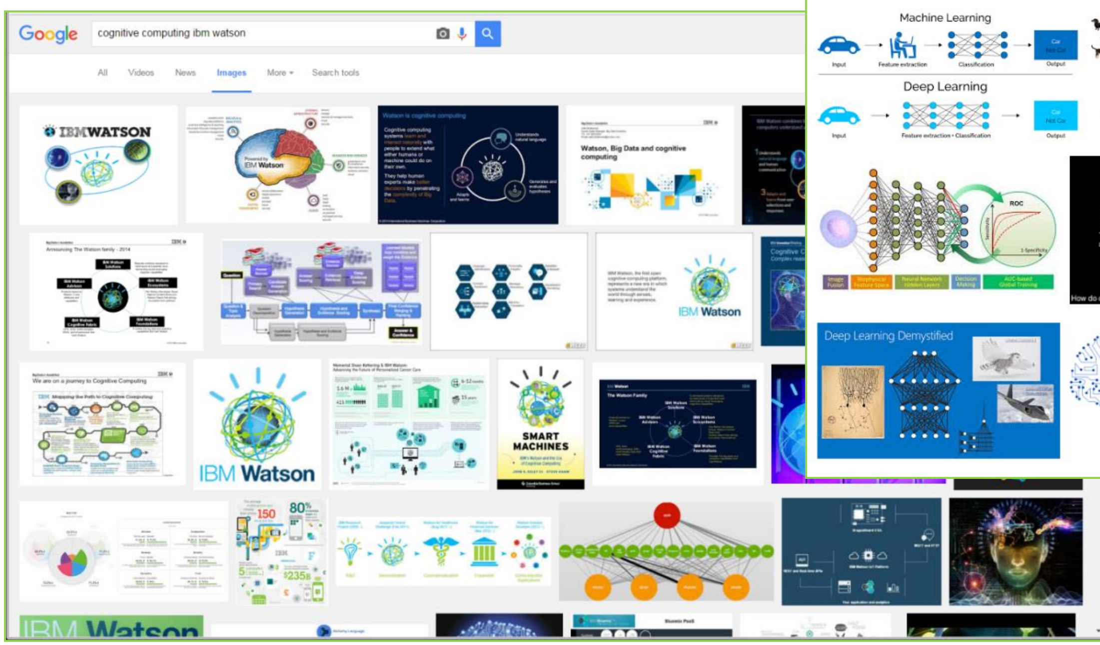
TIES4911 Deep-Learning for Cognitive Computing for Developers
Spring 2025

by:
Dr. Oleksiy Khriyenko
IT Faculty
University of Jyväskylä

Acknowledgement

I am grateful to all the creators/owners of the images that I found from Google and have used in this presentation.



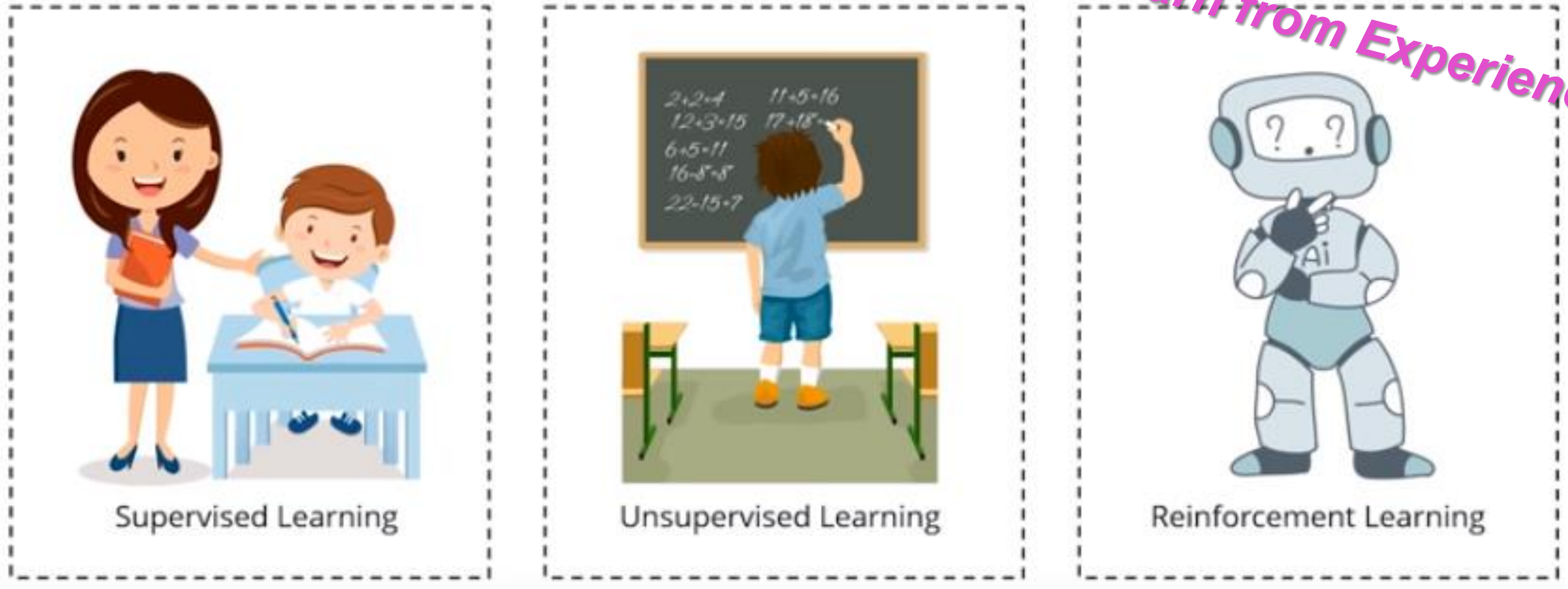



STARCRAFT II

Machine Learning

Reinforcement Learning *is the science of making optimal decisions using experiences…*

learn from Experience



Reinforcement learning has gained significant attention with the relatively recent success of DeepMind's ***AlphaGo*** *system defeating the world champion Go player. The AlphaGo system was trained in part by reinforcement learning on deep neural networks.*

Relevant links:
https://www.youtube.com/watch?v=LzaWrmKL1Z4
https://www.youtube.com/watch?v=JgvyzIkgxF0

Machine Learning

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

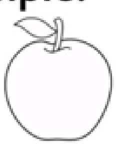
This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs
environment with rewarding

Goal: Maximize future rewards over many time steps

Apple example:

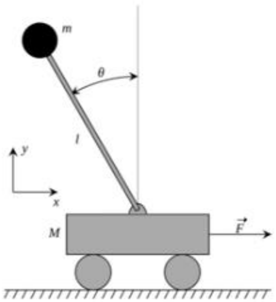


Eat this thing because it will keep you alive.

Relevant links:
https://www.youtube.com/watch?v=nZfaHIxDD5w

Variety of problems

Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

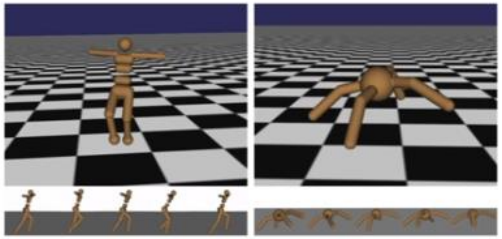
State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

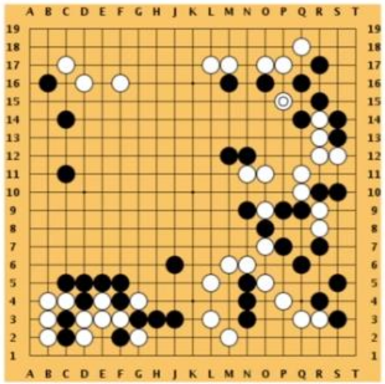
Robot Locomotion



Objective: Make the robot move forward

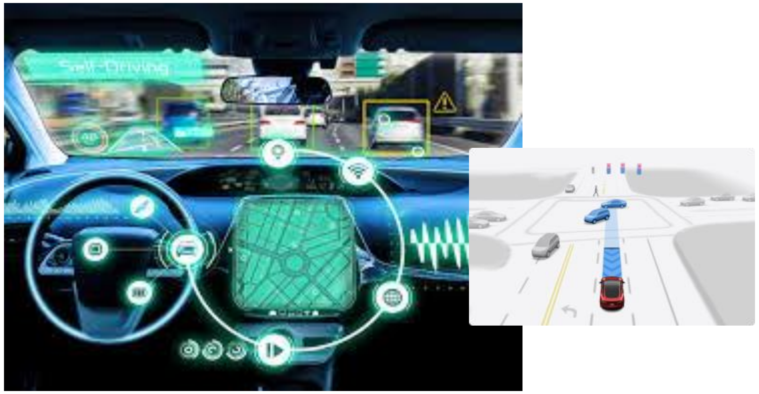
State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise



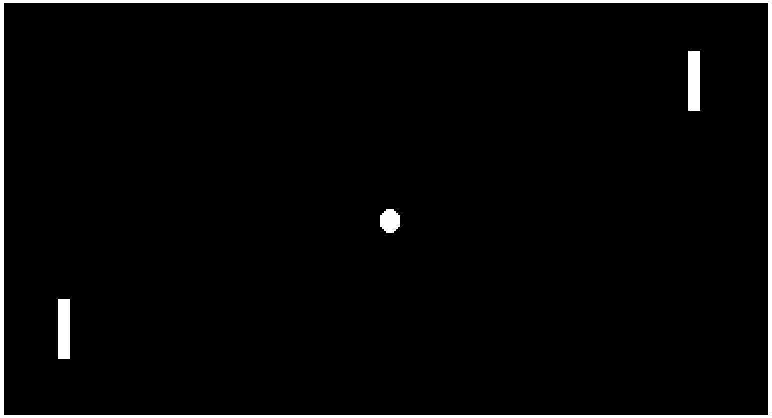
Self-driving…

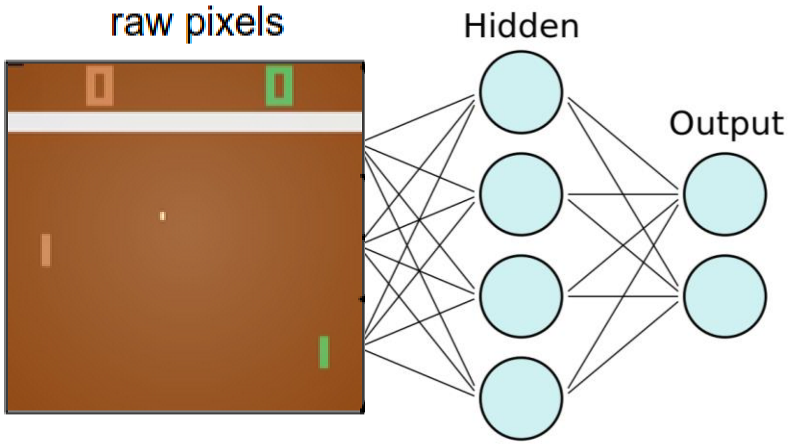
Objective: Reach destination without collisions…
State: sensor based observations (vision, sound, depth, etc.)
Action: steer the wheel, and actuate other actuators…
Reward: 1 if destination is reached without collisions and traffic rules breaking, 0 otherwise

Relevant links:
https://www.youtube.com/watch?v=lvoHnicueoE

?

The game of ***Pong***…





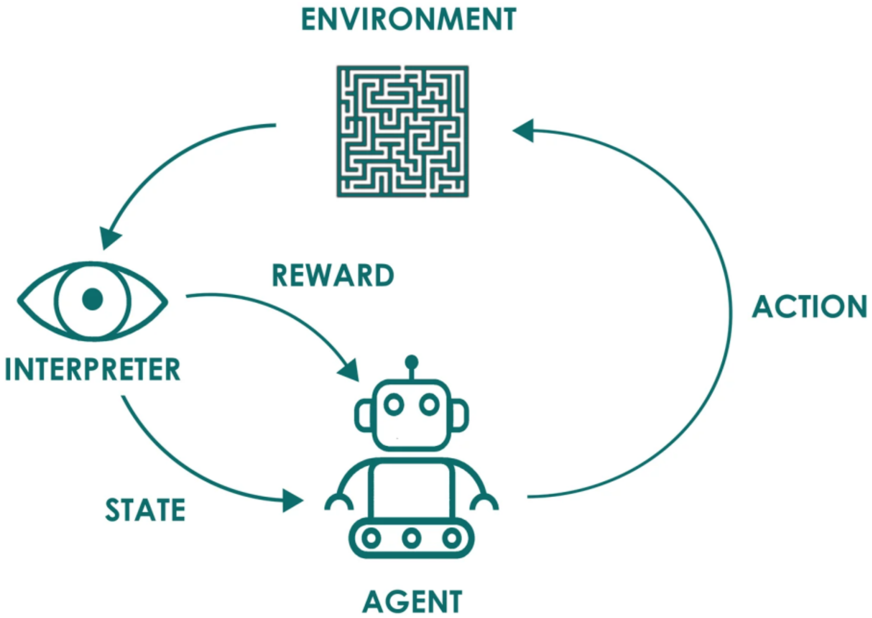
Drawbacks of supervised approach:

- *Not always possible, or very difficult to get labels for all possible situation…*
- *Imitating a behavior (present within a training set), agent will never be better that someone it tries to mimic…*

Relevant links:
http://karpathy.github.io/2016/05/31/rl/

Reinforcement Learning

Reinforcement Learning (RL) is a type of Machine Learning where system learns to behave in the environment based on the results achieved by performing actions… Depending on the results, it gets rewards or punishment and learns to take appropriate actions in order to maximize the reward…

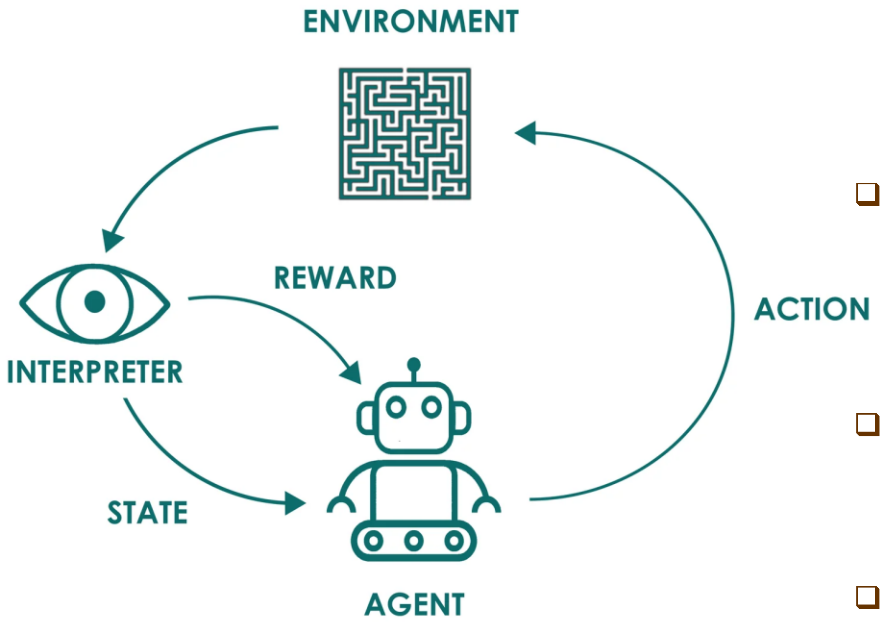


Just follow simple steps:

- *Observation of the environment*
- *Deciding how to act using some strategy*
- *Acting accordingly*
- *Receiving a reward or penalty*
- *Learning from the experiences and refining strategy*
- *Iterate until an optimal strategy is found*

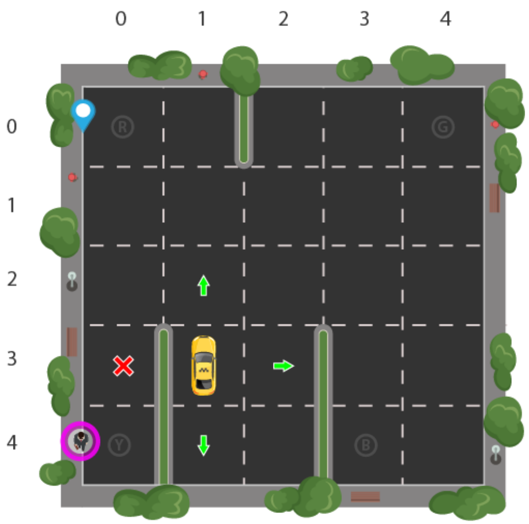
Reinforcement Learning

Reinforcement Learning (RL) is ***not a Supervised Learning*** that we have been studying so far…



- *Stochasticity issue of reinforcement learning. Inputs (states, or rewards) might be noisy - random or incomplete. Unknown model, non-deterministic function of environment transition (change of the state). Reward function is not the same as deterministic loss function in supervised learning. It might be different for different time steps even if state and action are the same.*
- *Credit assignment. Not direct dependence between action and reward. It is kind of long-term dependency when reward depends on the action (or several actions) happened in the past.*
- *Nondifferentiability of the environment. We cannot backpropagate through the world, we do not have a model of world's behaviour.*
- *Nonstationarity. The agent learning process depends on actions it performs. Training data is a kind of function of how agent is doing at the current point of time.*

RL Vocabulary



Agent is an algorithm that learns from trial and error.

Environment is a world in which agent performs actions.

Action (A) is any possible activity (step) that agent can take. ***Action Space*** is the set of all the actions that an agent can take in a given state.

State (S) is a current condition returned by the environment in particular point of time. ***State Space*** is the set of all possible situations (states) an agent could inhabit. The state should contain useful information the agent needs to make the right action.

Reward (R) is the feedback of the environment by which the success or failure of an agent's actions is measured in a given state. Rewards can be immediate or delayed. They effectively evaluate the agent's action. Total Reward is $R_t = \sum_{i=t}^{\infty} r_i$

Policy (π) is the strategy that the agent employs to determine the next action based on the current state. It is agent's behavior function that maps states to actions, the actions that promise the highest reward.

Value (V) is an expected long-term return with discount, as opposed to the short-term reward ***R*** *(*$V\pi(s)$*)*. We discount rewards, or lower their estimated value, the further into the future they occur. ***Discount factor*** is designed to make future rewards *worth less* than immediate rewards; i.e. it enforces a kind of short-term hedonism in the agent. Often expressed with the lower-case Greek letter gamma: γ. If γ is *0.8*, and there's a reward of *10* points after *3* time steps, the present value of that reward is *0.8³ x 10*. So, in this case Discounted Total Reward is $R_t = \sum_{i=t}^{\infty} \gamma^i r_i$.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

Action-value (Q-value) (Q) is similar to ***V***, except, it takes an extra parameter, the current action ***A***. $Q\pi(s, a)$ refers to the long-term return of an action, taking action ***a*** under policy π from the current state s. ***Q*** maps state-action pairs to rewards.

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Relevant links:
https://www.youtube.com/watch?v=LzaWrmKL1Z4
https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
https://pathmind.com/wiki/deep-reinforcement-learning

Reinforcement Learning

Better Sample Efficient ← → Less Sample Efficient

Model-based (100 time steps)	Off-policy Q-learning (1 M time steps)	Actor-critic	On-policy Policy Gradient (10 M time steps)	Evolutionary/ gradient-free (100 M time steps)

Model-based

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

Value-based

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

Policy-based

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

Massachusetts Institute of Technology For the full list of references visit: https://hcai.mit.edu/references [331, 333] https://deeplearning.mit.edu 2019

Relevant links:
https://www.youtube.com/watch?v=zR11FLZ-O9M

Q-Learning

Q learning is a value-based method of supplying information to inform which action an agent should take. An initially intuitive idea of creating values upon which to base actions is to create a table which sums up the rewards of taking action ***a*** in state ***s*** over multiple game plays. This could keep track of which moves are the most advantageous.

Bellman Equation: Q^* satisfies the following recurrence relation:

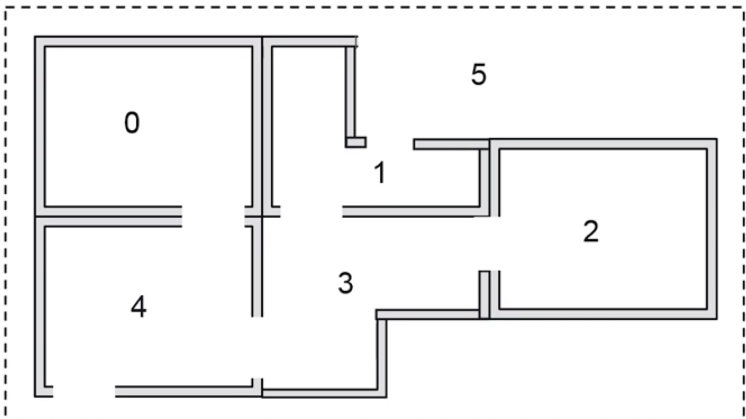
$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

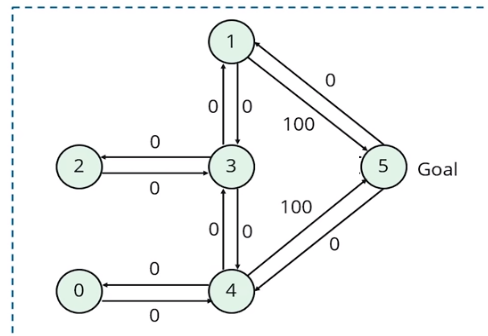
At the heart of Q-learning are things like the *Markov decision process (MDP)* and the *Bellman equation*.

Q function returns an expected total future reward for agent in state ***s*** by executing action ***a***: $Q(s_t, a_t) = \mathrm{E}[R_t | s_t, a_t]$

In a simpler form: Value of an action = Immediate value + sum of all optimal future actions



Place an agent in one of the rooms and reach outside in the shortest way…



Transform the state diagram and the instant reward values into a ***reward table*** *(matrix R)*…

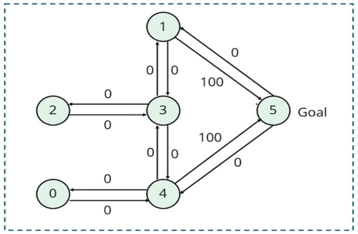
Action
(that leads to another state)

$R =$

State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	-1

Relevant links:
https://blog.valohai.com/reinforcement-learning-tutorial-part-1-q-learning
https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow
https://www.youtube.com/watch?v=LzaWrmKL1Z4
https://www.geeksforgeeks.org/markov-decision-process
https://en.wikipedia.org/wiki/Bellman_equation

Q-Learning



Action (that leads to another state)

$R =$

State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

Build a ***matrix Q*** that represents the agent's memory (things learned through experience) using the following formula (from Bellman Equation):

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma * \max_a [Q(s_{t+1}, a)]$$

, where discount factor $\gamma = [0:1]$

Q learning algorithm:

- set γ parameter and rewards
- initialize matrix Q to zero
- select a random initial state and use it as a current
- select one among all possible actions for the current state
- using this action and corresponding next state, get max Q value based on all possible next actions
- compute update for Q value of the current state and chosen action
- make a considered next state as a current one and repeat further action selection until current state is equal to the goal state.
- do further training by choosing a random initial state.

Let's start with random state (e.g. room 1), and let's move to room 5 ($\boldsymbol{\gamma = 0.8}$)

Q =

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$Q(1,5) = R(1,5) + 0.8 * \max[Q(5,1), Q(5,4)]$
$= 100 + 0.8 * \max[0,0] = 100$

Since room 5 was a goal state, we continue with random state (e.g. room 3), and let's move to room 1 ($\boldsymbol{\gamma = 0.8}$)

Q =

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$Q(3,1) = R(3,1) + 0.8 * \max[Q(1,3), Q(1,5)]$
$= 0 + 0.8 * \max[0,100] = 80$

Since room 1 is not a goal state, the next state is room 1, and let's move to room 5 ($\boldsymbol{\gamma = 0.8}$)

Q =

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$Q(1,5) = R(1,5) + 0.8 * \max[Q(5,1), Q(5,4)]$
$= 100 + 0.8 * \max[0,0] = 100$

Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

- Current Q-table value we are updating
- Learning rate
- Reward
- Discount
- Estimated reward from our next action

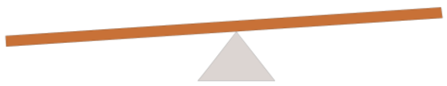
Finally, agent needs a ***Policy*** $\pi(s)$ to infer the best action to be taken at the state ***s***:

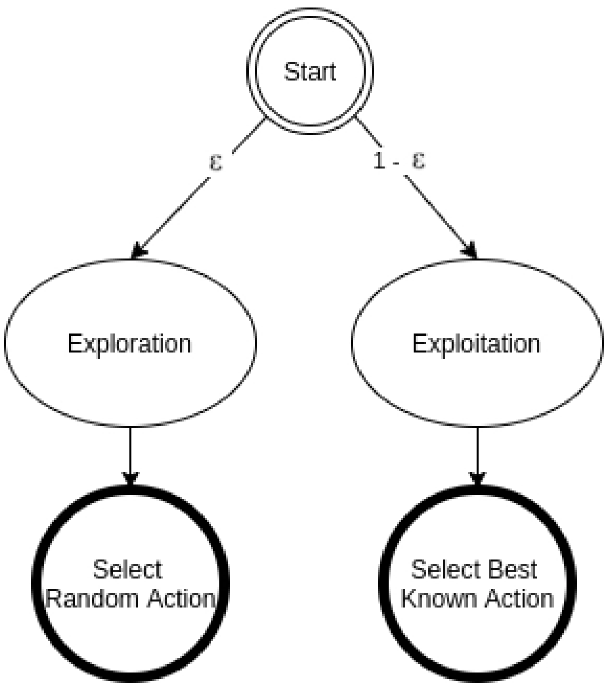
$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Relevant links:
https://blog.valohai.com/reinforcement-learning-tutorial-part-1-q-learning

Q-Learning

Exploration vs. Exploitation: Epsilon Greedy Action Selection





$$\text{Action at time}(t) \begin{cases} \max Q_t(a) & \text{with probability } 1-\epsilon \\ \text{any action } (a) & \text{with probability } \epsilon \end{cases}$$

Epsilon (ε) parameter is related to the ***epsilon-greedy action*** selection procedure in the Q-learning algorithm. In the action selection step, we select the specific action based on the Q-values we already have. The epsilon parameter introduces randomness into the algorithm, forcing us to try different actions. This helps not getting stuck in a local optimum.

If epsilon is set to 0, we never explore but always exploit the knowledge we already have. On the contrary, having the epsilon set to 1 force the algorithm to always take random actions and never use past knowledge. Usually, epsilon is selected as a small number close to 0.

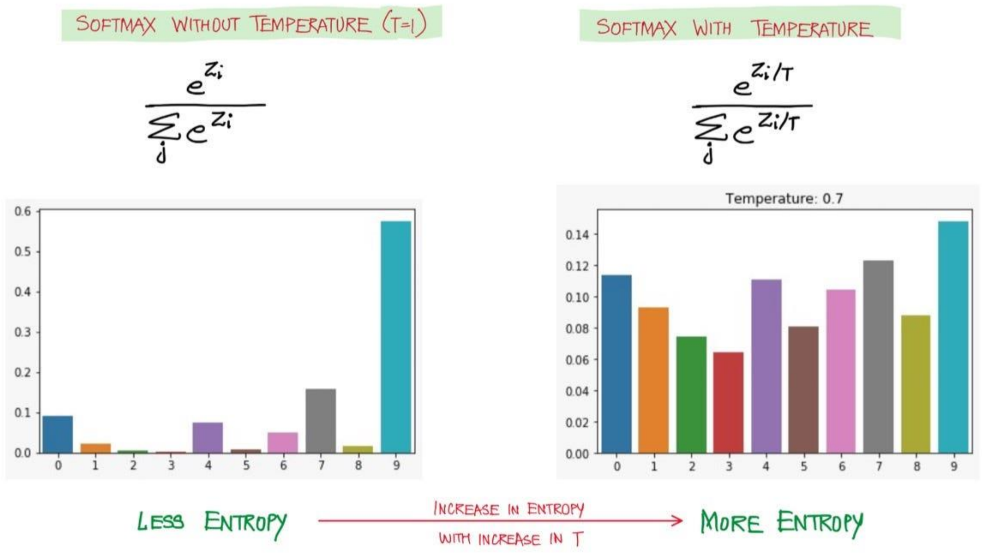
Relevant links:
https://www.baeldung.com/cs/epsilon-greedy-q-learning#:~:text=The%20epsilon%2Dgreedy%20approach%20selects,what%20we%20have%20already%20learned.

Q-Learning

Exploration vs. Exploitation: Softmax

Tau (τ) is the temperature factor. The parameter τ controls the magnitude of the weight assigned to the action-value estimates. When τ is high, more actions are more likely to be selected by the Agent and as such increases the likelihood of selecting an undesirable action. Conversely, when τ is low, actions with higher action values estimates and consequently higher weights are more likely to be selected by the Agent. It turns out that when $\tau \rightarrow 0$, the softmax action-selection method acts like a purely ***greedy action-selection method***.

$$\pi^*(s) = Softmax(\frac{Q(s,a)}{\tau})$$



Relevant links:
https://ekababisong.org/evaluating-actions/
https://medium.com/@harshit158/softmax-temperature-5492e4007f71

Q-Learning

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

Breakout Atari game

- Image size: **84 × 84** (resized)
- Consecutive **4** images
- Grayscale with **256** gray levels

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

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

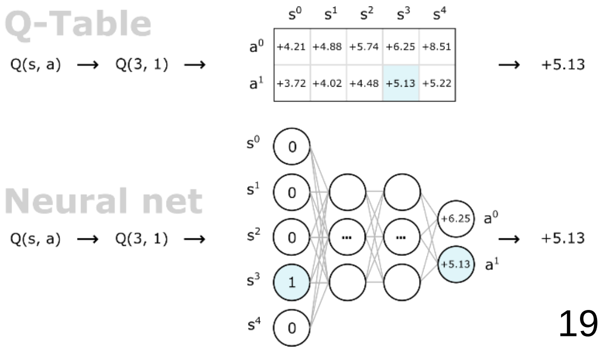
So, size of the Q table is ***intractable***…

*Let's go **Deeper** !!!*

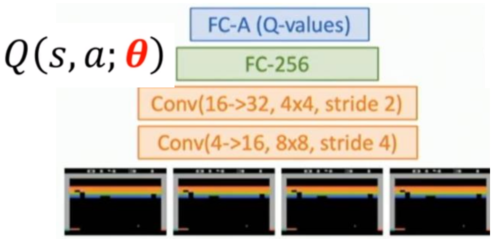
Deep RL **=** ***RL*** **+** ***Neural Networks***

Let's learn compress representation of Q table with Neural Network, learning an approximator for the Q-function with use of Bellman Equation as loss…

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

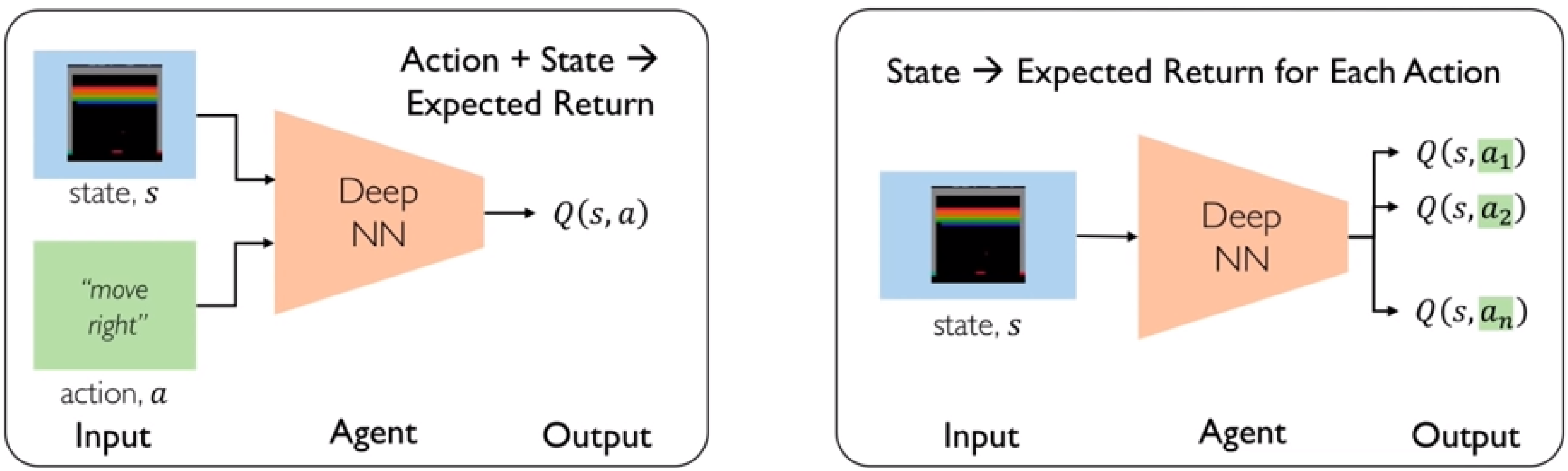


Deep Q-Learning



Q values for corresponding 4 actions:
$Q(s_t,a_1)$, $Q(s_t,a_2)$, $Q(s_t,a_3)$ and $Q(s_t,a_4)$

State s_t: 4*84*84 stack of last 4 frames
(grayscaled, downsampled and cropped)



$$\mathcal{L} = \mathbb{E}\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s',a')\right)}^{\text{target}} - \overbrace{Q(s,a)}^{\text{predicted}} \right\|^2\right]$$

Q-Loss

Relevant links:
https://www.youtube.com/watch?v=AhyznRSDjw8
https://www.youtube.com/watch?v=V1eYniJ0Rnk
https://www.youtube.com/watch?v=SgC6AZss478

- Deep Q-Network (DQN): uses the same network for both Q
- Double DQN: separate networks for each Q to reduce bias caused by the inaccuracies of Q network at the beginning of training

Deep Q-Learning improvements

- Experience Replay $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$
  - 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.

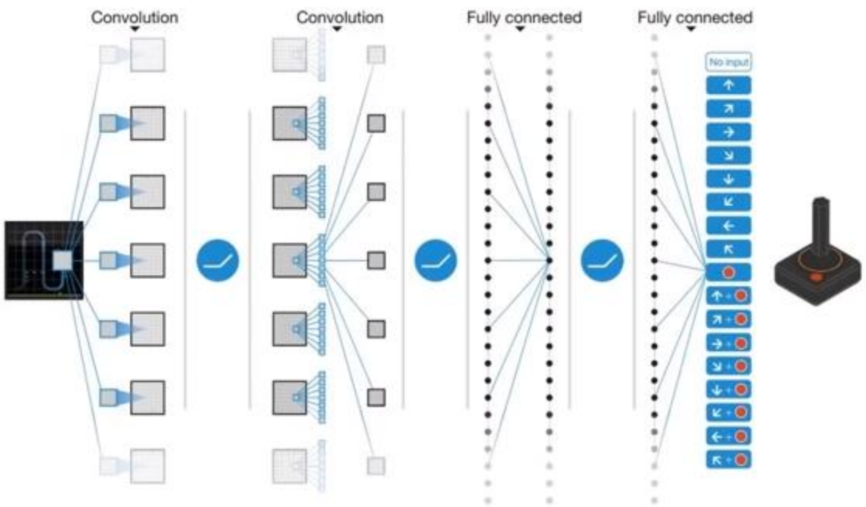
$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

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

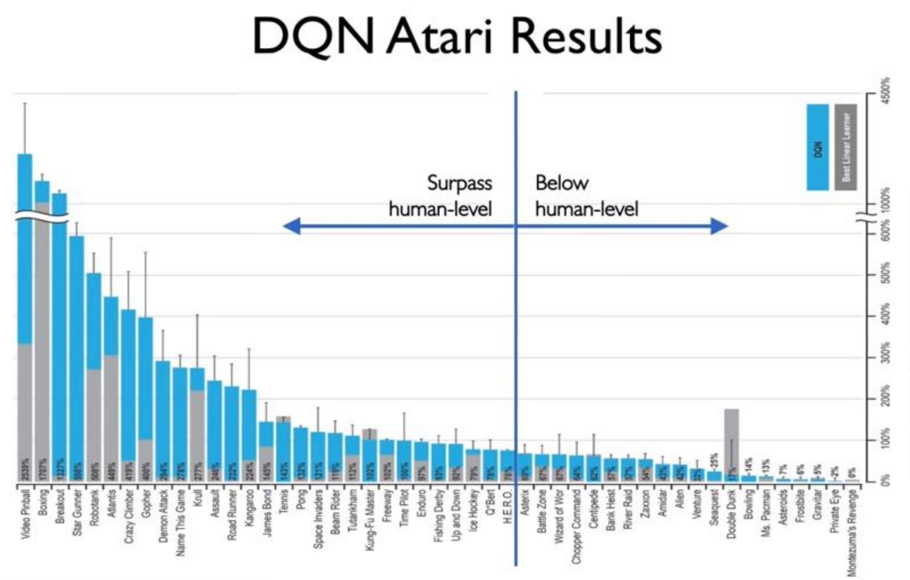
Relevant links:
https://www.youtube.com/watch?v=zR11FLZ-O9M

Deep Q-Learning



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



Relevant links:
https://arxiv.org/abs/1312.5602
https://www.ncbi.nlm.nih.gov/pubmed/25719670
https://www.youtube.com/watch?v=V1eYniJ0Rnk

Deep Q-Learning

Approximate Q and infer optimal policy

Value Learning

Find $Q(s,a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

Directly optimize policy space

Policy Learning

Find $\pi(s)$

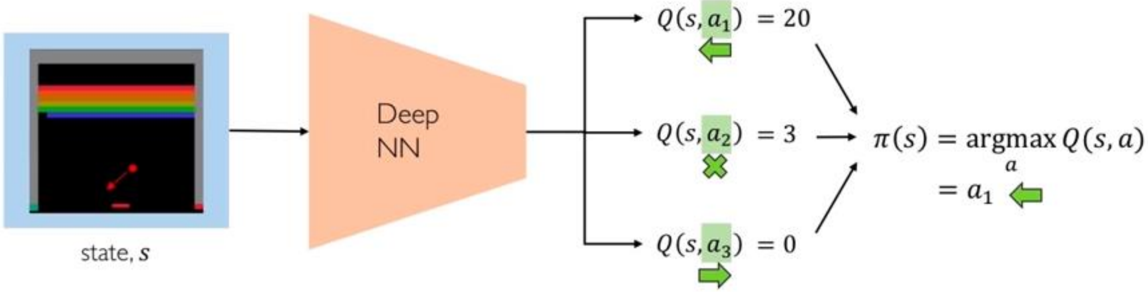
Sample $a \sim \pi(s)$

Downsides of Q-learning:

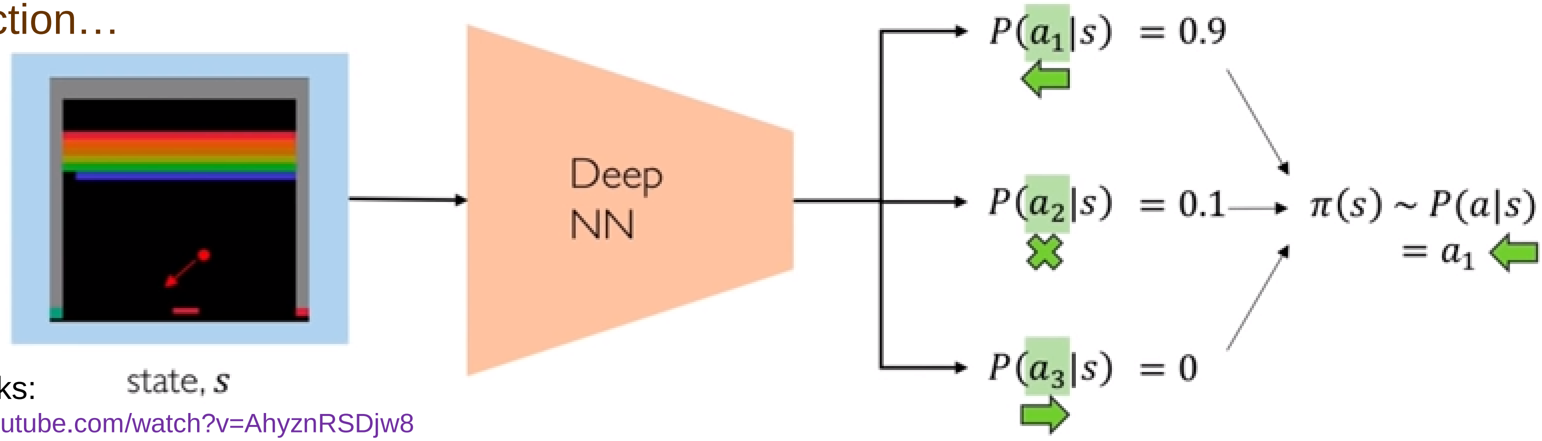
- *It models only scenarios with discrete and small action space, and cannot handle continuous action space*
- *It cannot learn stochastic policies since policy is deterministically computed from Q-function maximizing the reward*

Policy Gradients

Instead of Q-function approximation and inferencing the optimal policy $\pi(s)$



Let's optimize the policy $\pi(s)$ directly, and output probability distribution over the space of all actions given that state $P(a|s)$ (probability to resolve the highest Q value taking action **a** in the state ***s***). Now, simply sample from the learned probability distribution to get the next action…



Relevant links:
https://www.youtube.com/watch?v=AhyznRSDjw8
https://www.youtube.com/watch?v=SgC6AZss478
https://www.youtube.com/watch?v=TjHH_--7l8g
https://www.youtube.com/watch?v=VnpRp7ZglfA

Policy Gradients

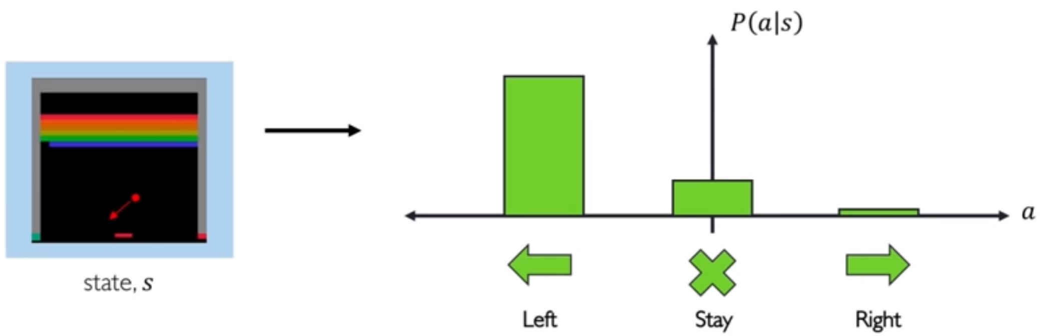
Now, dealing with probability distribution we may work with ***Continuous Action Space***

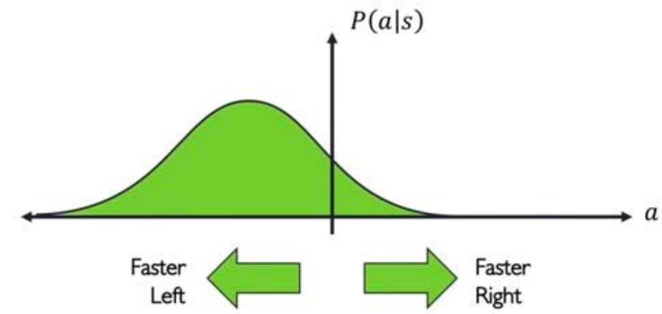
Discreet Action Space

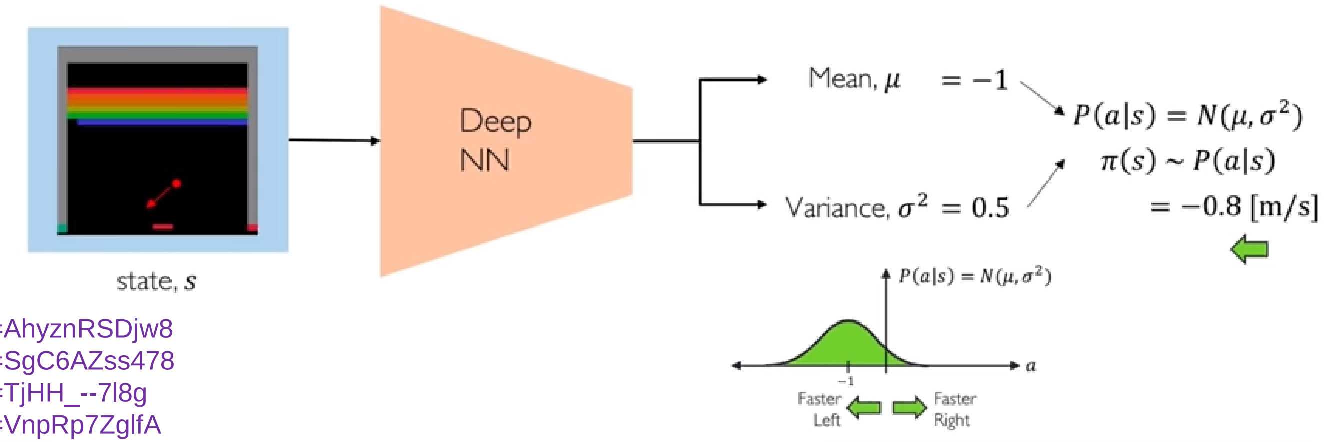
Which direction …?

Continuous Action Space

How fast …?



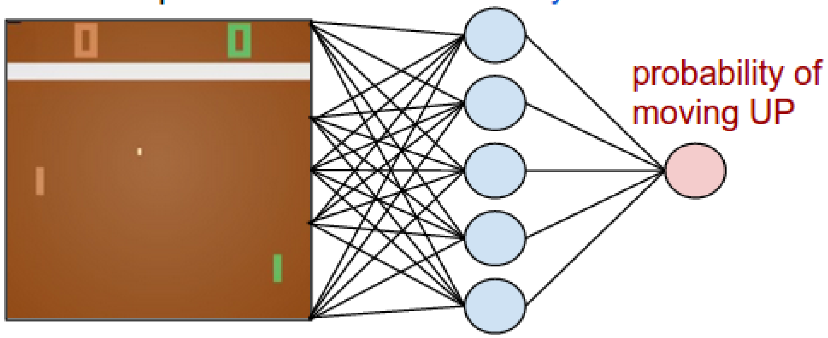


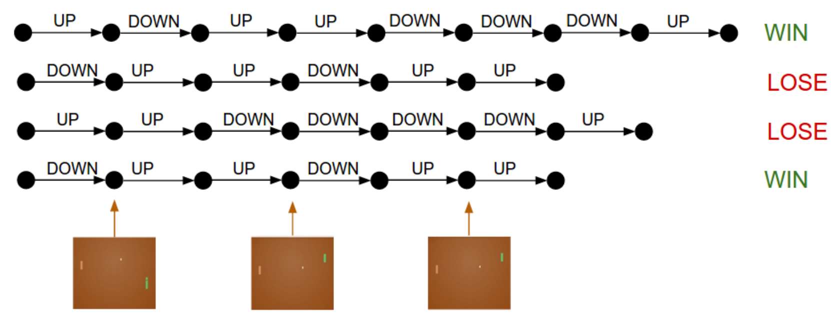


Relevant links:
https://www.youtube.com/watch?v=AhyznRSDjw8
https://www.youtube.com/watch?v=SgC6AZss478
https://www.youtube.com/watch?v=TjHH_--7l8g
https://www.youtube.com/watch?v=VnpRp7ZglfA

Policy Gradients

Define ***Policy Network*** …

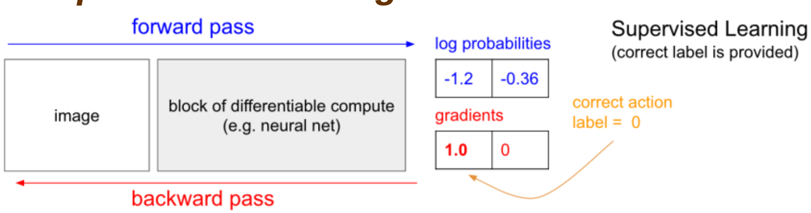




- Run a policy for a while
- See what actions led to high rewards and increase their probability, slightly encouraging every single action we made
- See what actions led to low rewards and decrease their probability, slightly discouraging every single action we made

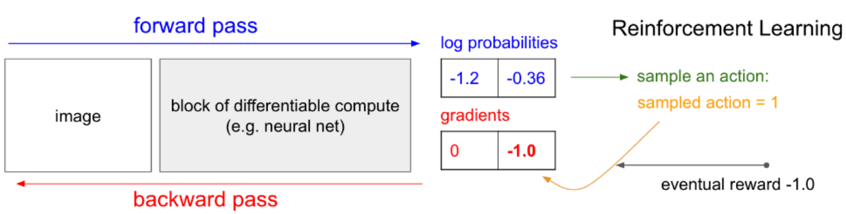
In particular, anything with frequent reward signals that requires precise play, fast reflexes, and not too much long-term planning would be ideal, as these short-term correlations between rewards and actions can be easily "noticed" by the approach.

Supervised Learning…



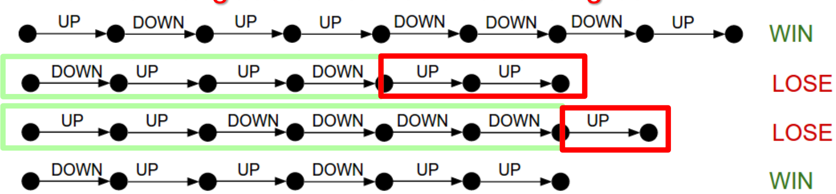
Policy Gradients…

Just ***sample from the distribution*** to get actual action, giving a chance for agent to explore an environment…



Drawbacks of Policy Gradients:

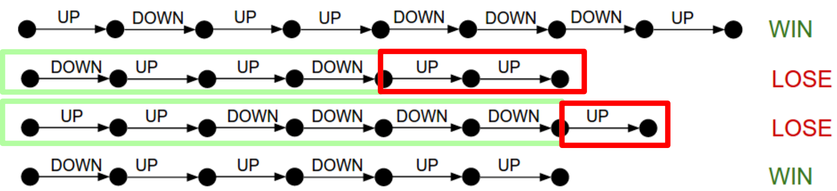
- Punishment of good actions and rewarding bad…



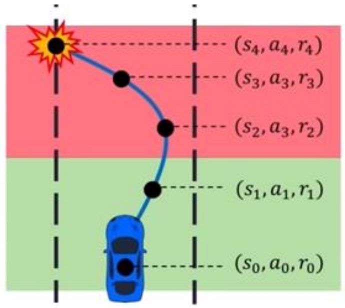
- In complex cases, random behavior of the agent might not lead to "win" result at all, and Policy Gradients is never going to see a single positive reward…
- Needs more data, and less stable during training…

Relevant links:
http://karpathy.github.io/2016/05/31/rl/
https://www.youtube.com/watch?v=JgvyzIkgxF0

Policy Gradients



Use ***Sparse Rewards*** giving penalty to the latest actions that lead to the termination.



Another solution for ***Sparse Rewards*** is Reward Shaping – process of manually designing of a reward function that needs to guide the policy to the desired behavior. Basically, it could be seen as a set of some "local rewards" given for positive completion of sub-goals.

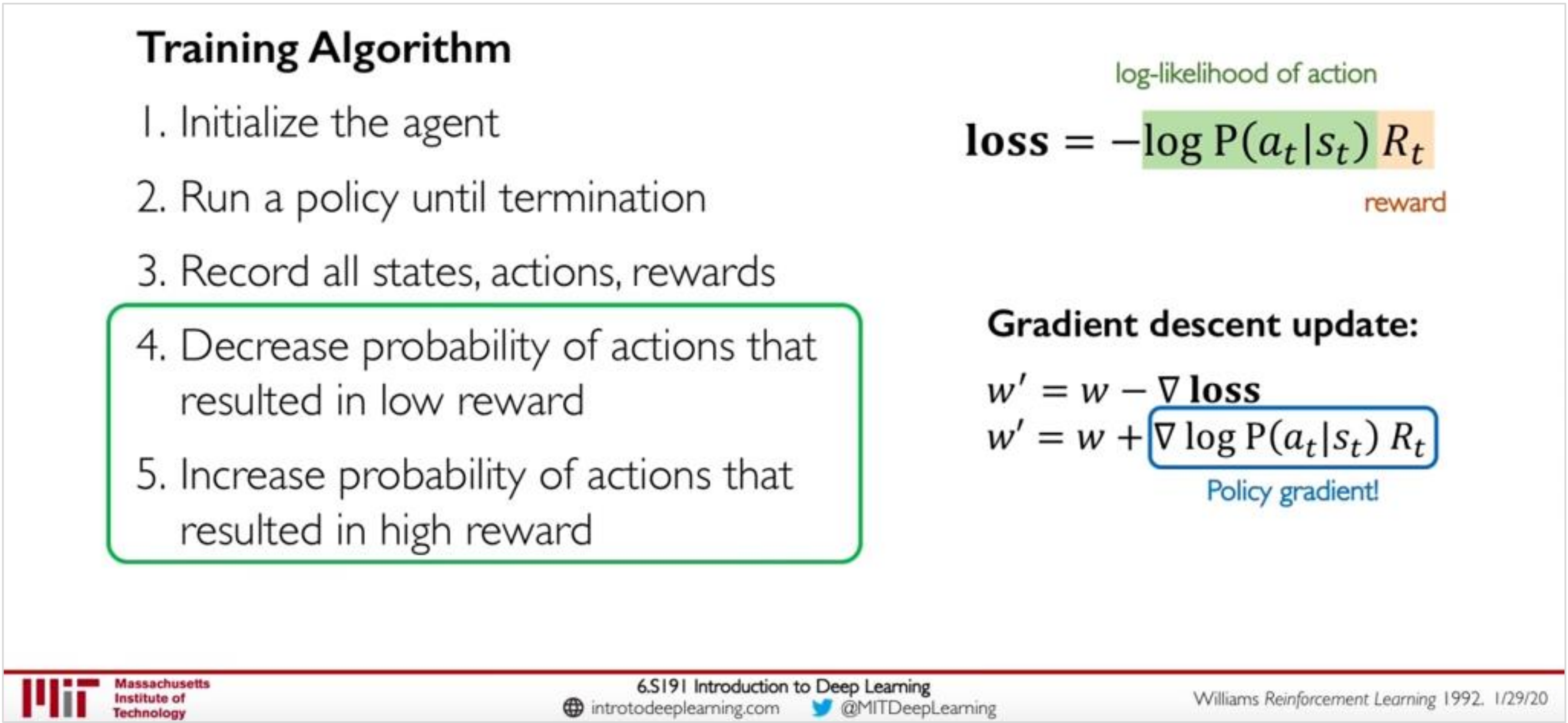
Downsides of ***Reward Shaping:***

- *It is a custom process that needs to be redone for any new environment…*

- *It suffers from the Alignment Problem. The Policy becomes over fitted to the specific reward function (ensure collection of as many as possible local rewards) and is not generalized to the initially intended behavior…*
- *In complicated case (e.g. AlphaGo), it leads to constraining the Policy to the particular behavior (which might not be optimal)…*

Relevant links:
http://karpathy.github.io/2016/05/31/rl/
https://www.youtube.com/watch?v=JgvyzIkgxF0

Policy Gradient



Relevant links:
https://www.youtube.com/watch?v=AhyznRSDjw8
https://www.youtube.com/watch?v=SgC6AZss478
https://www.youtube.com/watch?v=TjHH_--7l8g
https://www.youtube.com/watch?v=VnpRp7ZglfA

Advantage Actor-Critic (A2C)(A3C)

It combines policy-based and value-based networks…

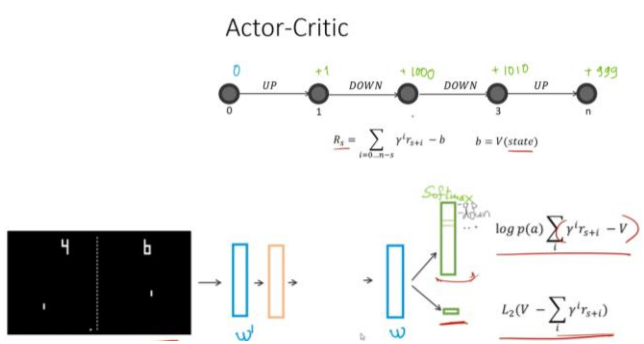
- Combine DQN (value-based) and REINFORCE (policy-based)
- Two neural networks (Actor and Critic):
  - **Actor** is policy-based: Samples the action from a policy
  - **Critic** is value-based: Measures how good the chosen action is

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

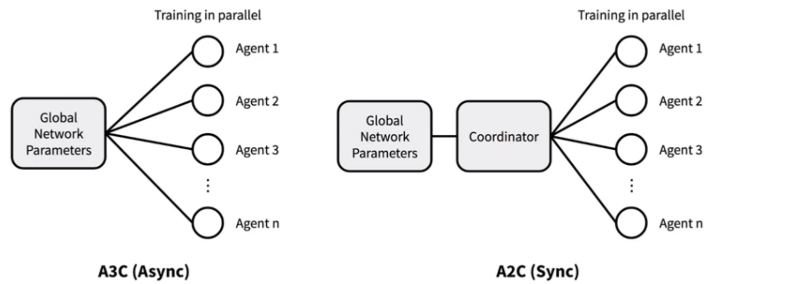
New update: $\Delta\theta = \alpha * \nabla_\theta * (log\ \pi(S_t, A_t, \theta)) * Q(S_t, A_t)$

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

Massachusetts Institute of Technology For the full list of references visit: https://hcai.mit.edu/references [335] https://deeplearning.mit.edu 2019



DeepMind OpenAI



- Both use parallelism in training
- A2C syncs up for global parameter update and then start each iteration with the same policy

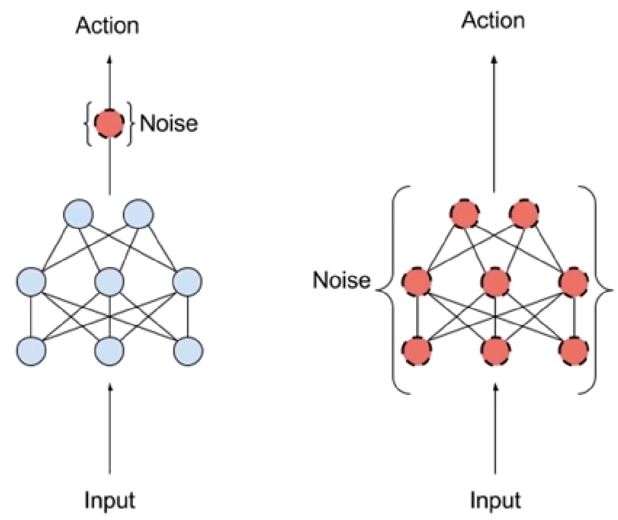
Massachusetts Institute of Technology For the full list of references visit: https://hcai.mit.edu/references [337] https://deeplearning.mit.edu 2019

Relevant links:
https://www.youtube.com/watch?v=zR11FLZ-O9M

Deep Deterministic Policy Gradient (DDPG)

Injects exploration to the system by adding noise: either to the action space (to the output), or to the parameters of the network.

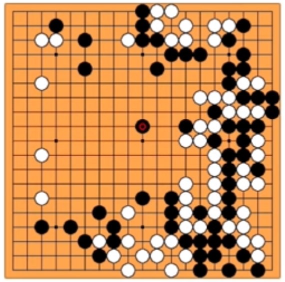
- Actor-Critic framework for learning a deterministic policy
- Can be thought of as: DQN for continuous action spaces
- As with all DQN, following tricks are required:
  - Experience Replay
  - Target network
- Exploration: add noise to actions, reducing scale of the noise as training progresses



Massachusetts Institute of Technology For the full list of references visit: https://hcai.mit.edu/references [341] https://deeplearning.mit.edu 2019

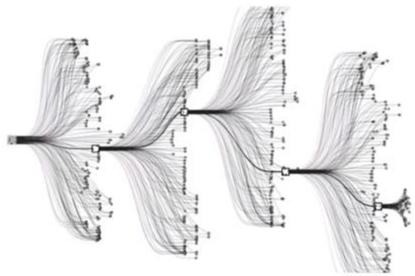
Relevant links:
https://www.youtube.com/watch?v=zR11FLZ-O9M

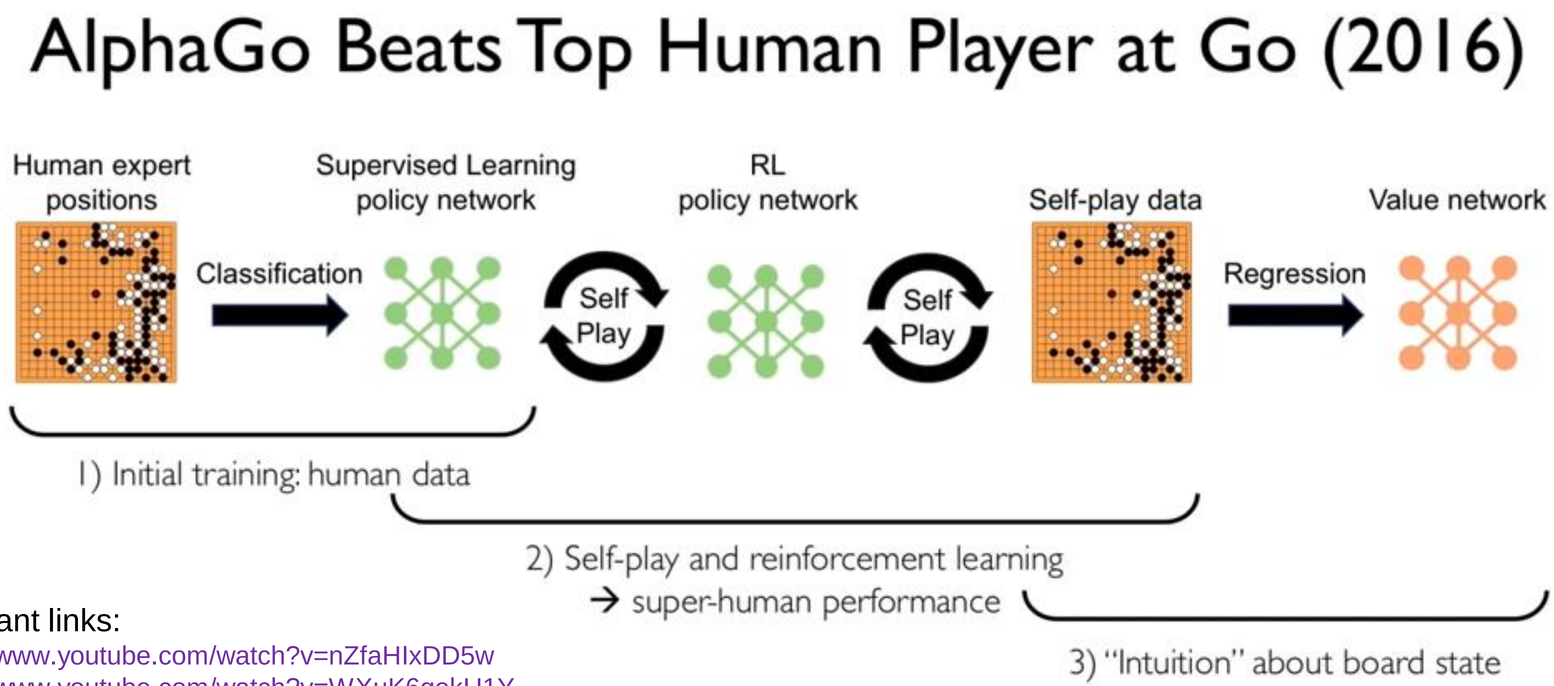
The Game of Go



Board Size n x n	Positions 3^{n^2}	% Legal	Legal Positions
1×1	3	33.33%	1
2×2	81	70.37%	57
3×3	19,683	64.40%	12,675
4×4	43,046,721	56.49%	24,318,165
5×5	847,288,609,443	48.90%	414,295,148,741
9×9	$4.434264882 \times 10^{38}$	23.44%	$1.039191487911 \times 10^{38}$
13×13	$4.300233593 \times 10^{80}$	8.66%	$3.72497923077 \times 10^{79}$
19×19	$1.740896506 \times 10^{172}$	1.20%	$2.08168199382 \times 10^{170}$

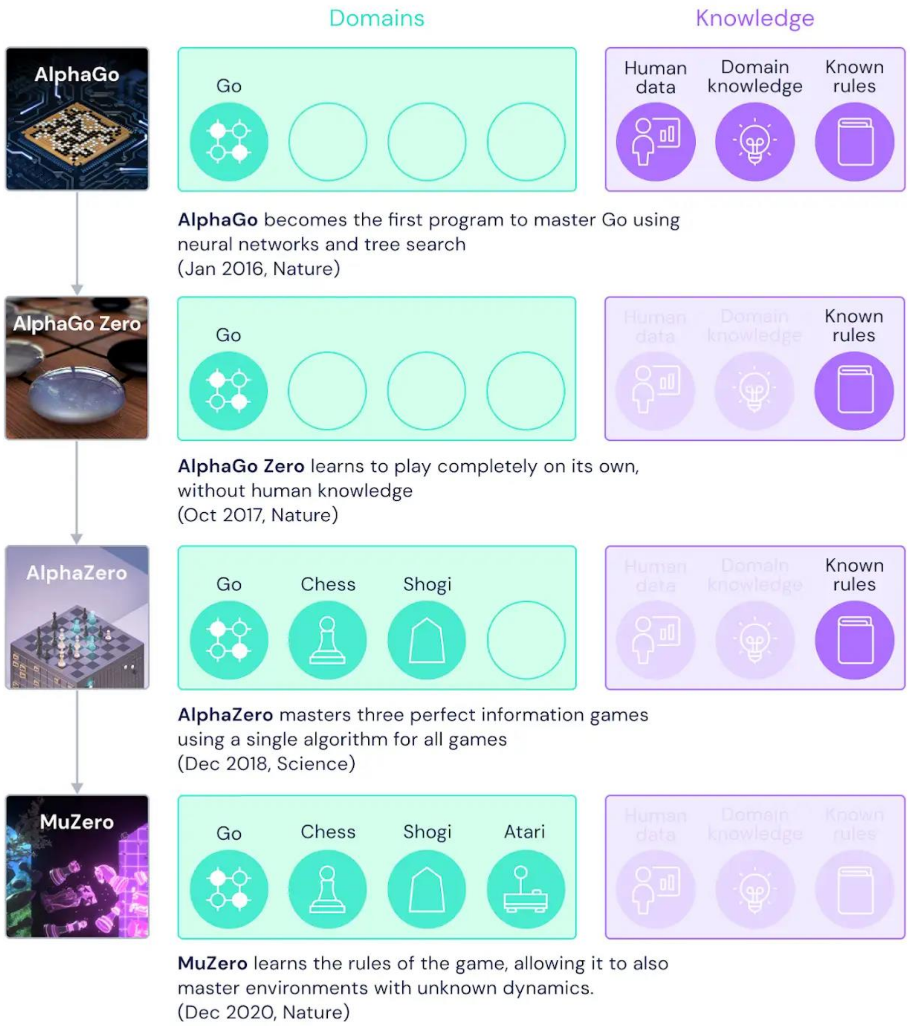
Greater number of legal board positions than atoms in the universe.





Relevant links:
https://www.youtube.com/watch?v=nZfaHIxDD5w
https://www.youtube.com/watch?v=WXuK6gekU1Y
https://www.youtube.com/watch?v=4PyWLgrt7YY

Evolution of –Zero algorithms

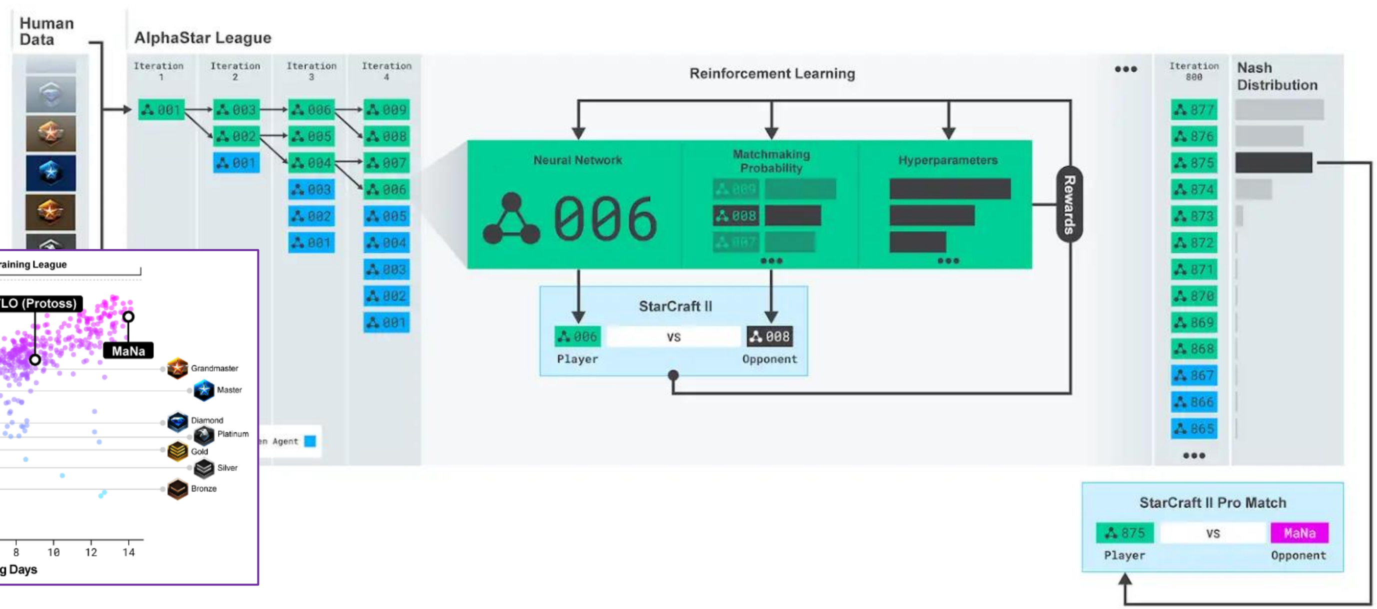


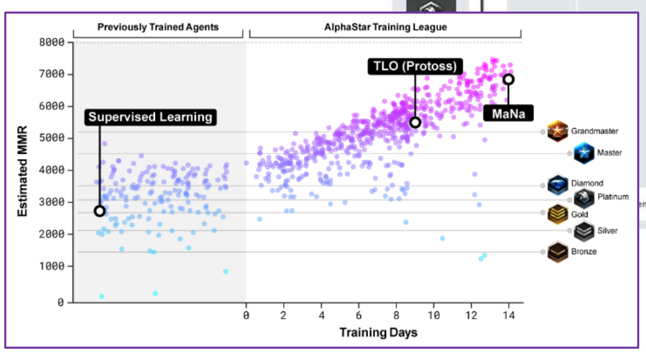
Relevant links:
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

AlphaStar (2019)

Mastering the Real-Time Strategy Game ***StarCraft II***

AlphaStar's behavior is generated by a deep neural network that receives input data from the raw game interface (a list of units and their properties), and outputs a sequence of instructions that constitute an action within the game. More specifically, the neural network architecture applies a transformer torso to the units (similar to relational deep reinforcement learning), combined with a deep LSTM core, an auto-regressive policy head with a pointer network, and a centralized value baseline.

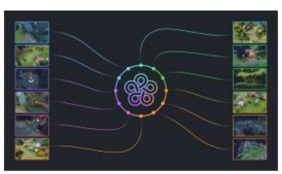




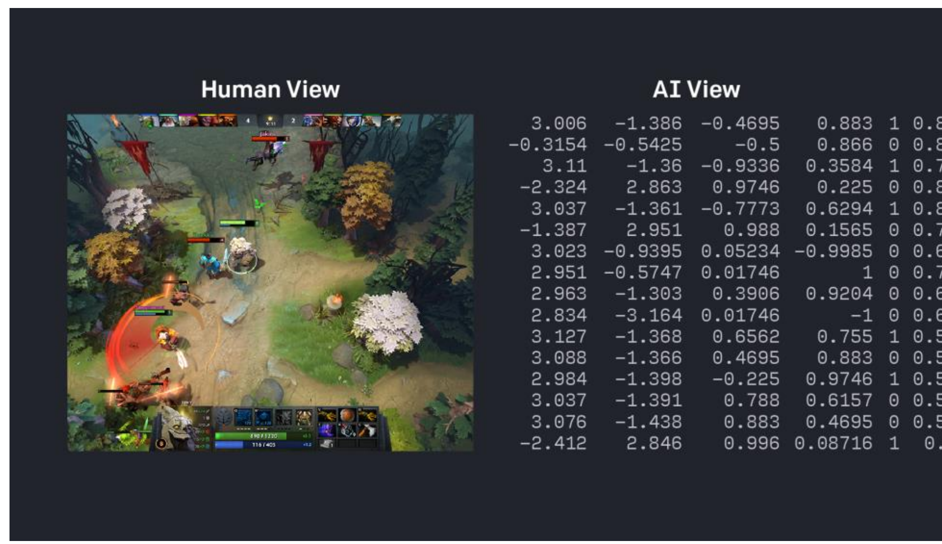
Relevant links:
https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
https://www.youtube.com/watch?v=lPERfjRaZug
https://www.youtube.com/watch?v=UuhECwm31dM

OpenAI Five (2016-2019)

Team of five neural networks, OpenAI Five, has started to defeat amateur human teams at ***Dota 2.***
OpenAI has used the multiplayer video game Dota 2 as a research platform for general-purpose AI systems. Their Dota 2 AI, called OpenAI Five, learned by playing over 10,000 years of games against itself. It demonstrated the ability to achieve expert-level performance, learn human–AI cooperation, and operate at internet scale.



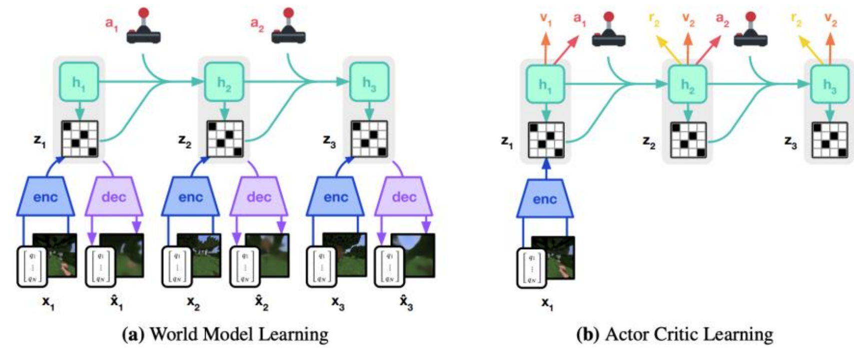
April 13, 2019. OpenAI Five wins back-to-back games versus Dota 2 world champions OG at Finals, becoming the first AI to beat the world champions in an esports game.



Relevant links:
https://openai.com/projects/five/
https://arxiv.org/abs/1912.06680
https://openai.com/blog/openai-five-defeats-dota-2-world-champions/
https://openai.com/blog/openai-five/

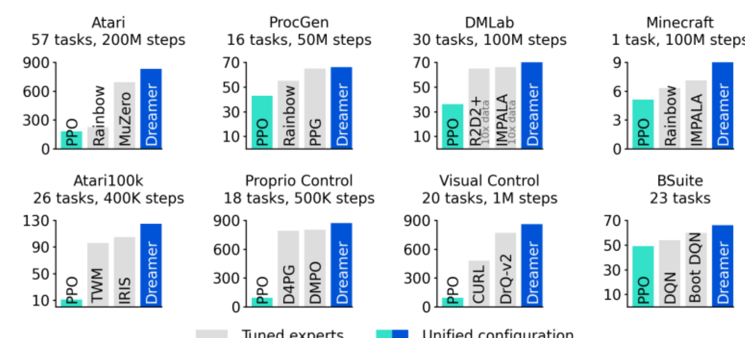
Dreamer v3 (2023)

DreamerV3 *learns a world model from experiences and uses it to train an actor critic policy from imagined trajectories. The world model encodes sensory inputs into categorical representations and predicts future representations and rewards given actions.* https://danijar.com/project/dreamerv3/



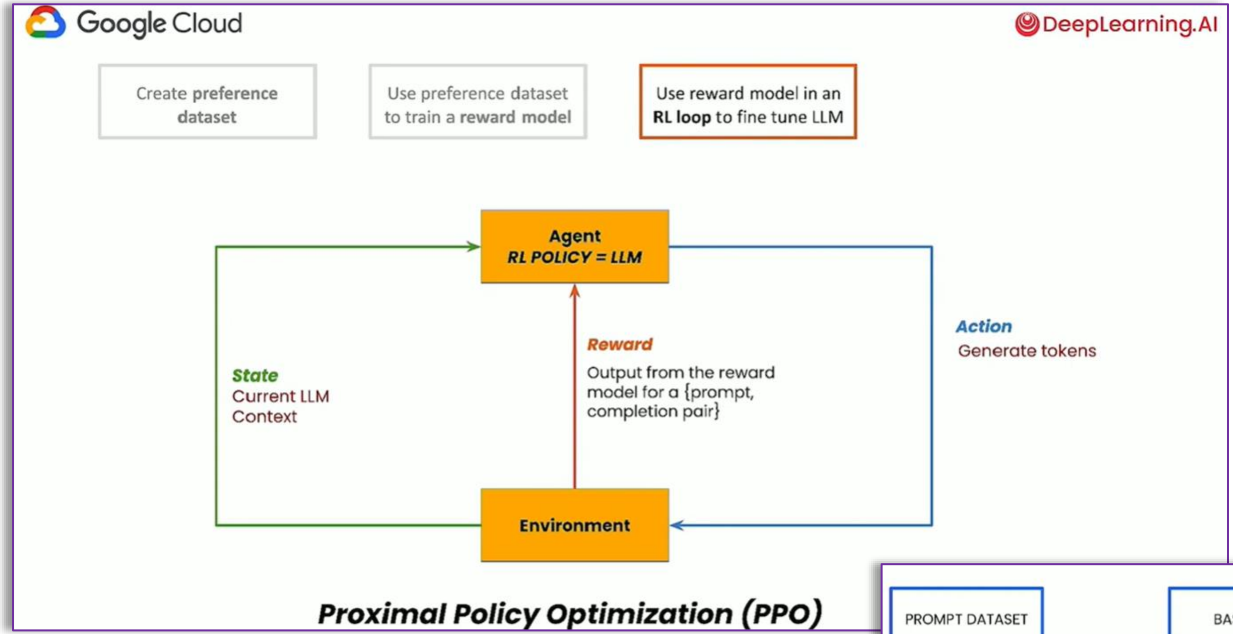
It is a general algorithm that outperforms specialized methods across over 150 diverse tasks, with a single configuration. Dreamer learns a model of the environment and improves its behaviour by imagining future scenarios. Robustness techniques based on normalization, balancing and transformations enable stable learning across domains. Applied out of the box, Dreamer is the first algorithm to collect diamonds in ***Minecraft*** from scratch.

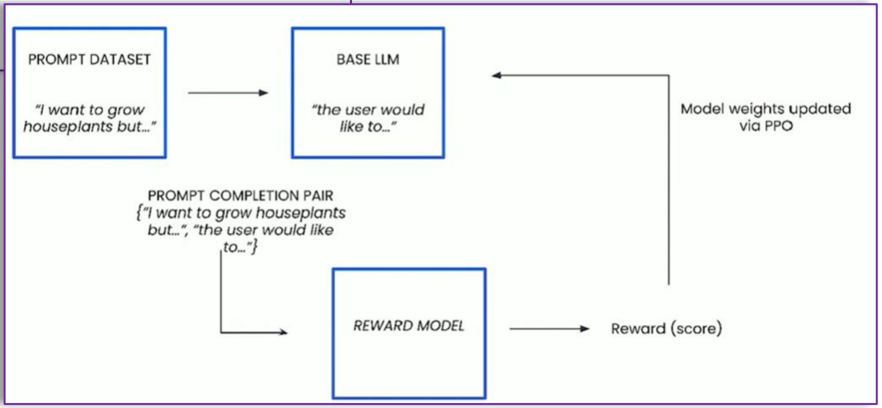
DreamerV3 masters a wide range of domains with a fixed set of hyperparameters, outperforming specialized methods. Removing the need for tuning reduces the amount of expert knowledge and computational resources needed to apply reinforcement learning.



Relevant links:
https://arxiv.org/pdf/2301.04104, https://github.com/danijar/dreamerv3
https://www.youtube.com/watch?v=vfpZu0R1s1Y
https://arxiv.org/abs/2503.02279

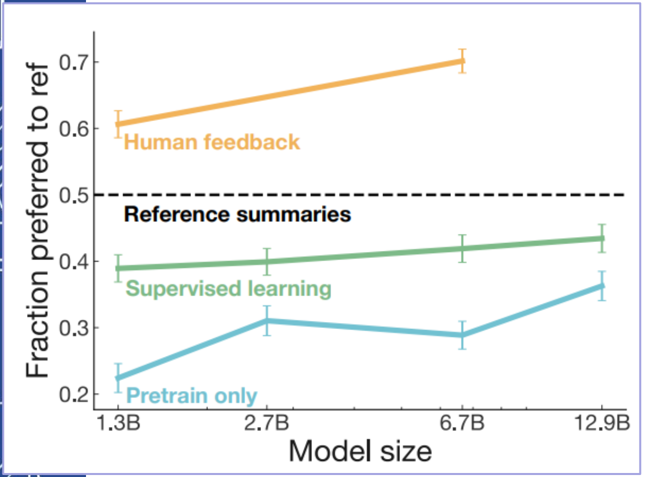
Reinforcement Learning From Human Feedback





Relevant links:
https://www.youtube.com/watch?v=MrIUl6TUV6k&list=PLu-JywtxEqEwh3tbOCi4tqP-WdVTxJswl&index=13
https://www.youtube.com/watch?v=WMmGzx-jWvs , https://www.youtube.com/watch?v=SXpJ9EmG3s4
https://www.youtube.com/watch?v=Mu_-FWIuhDA , https://www.youtube.com/watch?v=Z_JUqJBpVOk
https://www.youtube.com/watch?v=qGyFrqc34yc

Reinforcement Learning From Human Feedback



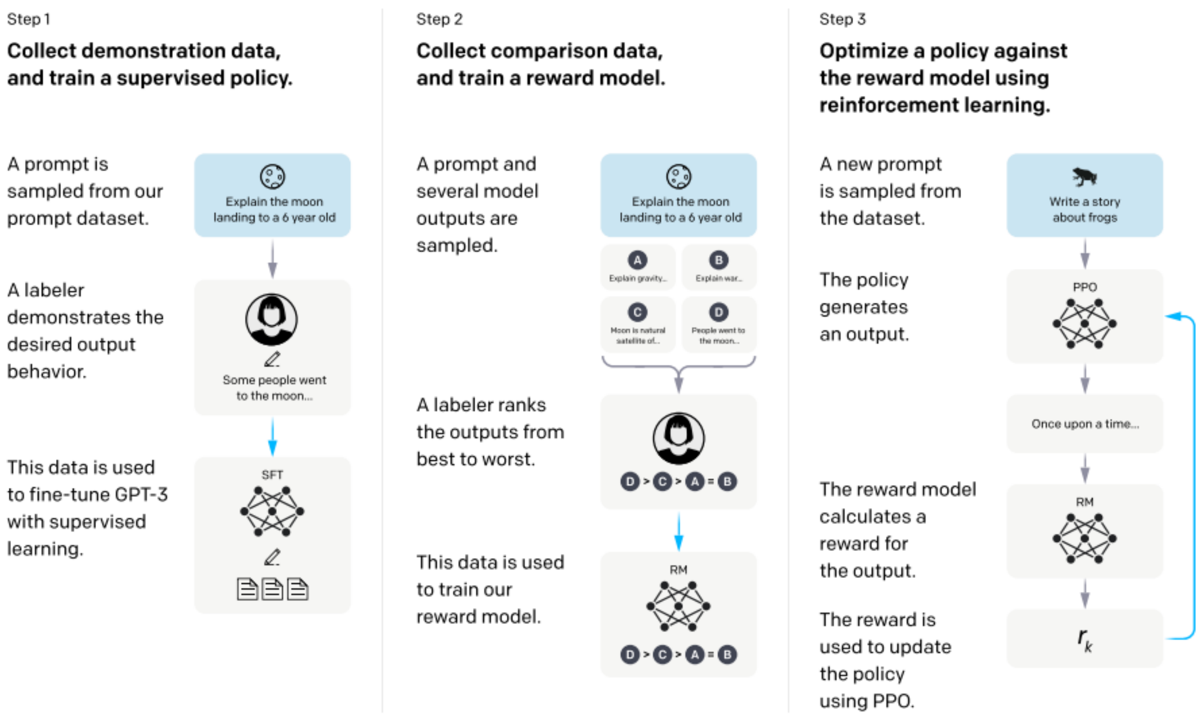


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

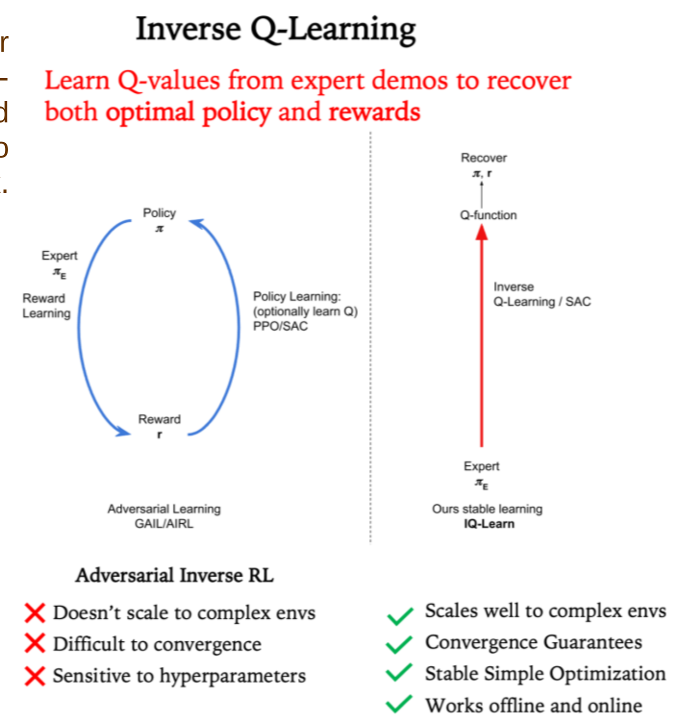
Relevant links:
https://arxiv.org/abs/2203.02155
https://arxiv.org/abs/2009.01325
https://www.youtube.com/watch?v=SXpJ9EmG3s4

even more approaches…

Imitation Learning (IL) *allows agents to learn from demonstrations or examples, while RL relies on trial-and-error to discover optimal strategies.*

Inverse reinforcement learning (IRL) *is the problem of inferring the reward function of an agent, given its policy or observed behavior. Analogous to RL, IRL is perceived both as a problem and as a class of methods*

Inverse Q-Learning (IQ-Learn) is a simple, stable & data-efficient framework for Imitation Learning (IL), that directly learns soft Q-functions from expert data. IQ-Learn enables non-adverserial imitation learning, working on both offline and online IL settings. It is performant even with very sparse expert data, and scales to complex image-based environments, surpassing prior methods by more than 3x. https://arxiv.org/abs/2106.12142



Relevant links:
https://arxiv.org/abs/1806.06877
https://arxiv.org/abs/2401.03857
https://medium.com/@hassaanidrees7/reinforcement-learning-vs-0744a860ffa7
https://bair.berkeley.edu/blog/2022/04/25/rl-or-bc/
https://arxiv.org/abs/2108.04763
https://danieltakeshi.github.io/2019/04/30/il-and-rl/

Challenges for RL in Real World Applications

"*Run a policy until termination*" among training steps…

Open Challenges:

- ***Real world observation + one-shot trial & error***

  Improve the ability of algorithms to form policies transformable across multiple of domains (including the real world)…

- ***Realistic simulation + transfer learning***

  Improve simulation environment to be realistically similar to the real world so that things learned in simulation could be directly transformed to the real world…

Relevant links:
https://www.youtube.com/watch?v=AhyznRSDjw8

RL Tools

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball. The team that has been maintaining Gym since 2021 has moved all future development to ***Gymnasium***

Links: https://github.com/openai/gym , https://www.gymlibrary.dev/
https://gymnasium.farama.org/ , https://github.com/Farama-Foundation/Gymnasium

TensorFlow Agents (TF-Agents) is a reliable, scalable and easy to use Reinforcement Learning library for TensorFlow.

Links: https://github.com/tensorflow/agents
https://www.tensorflow.org/agents/overview

RLCard is a toolkit for Reinforcement Learning in Card Game. It supports multiple card environments with easy-to-use interfaces for implementing various reinforcement learning and searching algorithms. The goal of RLCard is to bridge reinforcement learning and imperfect information games.

Links: https://rlcard.org/
https://github.com/datamllab/rlcard

REINFORCEjs is a Reinforcement Learning library that implements several common RL algorithms supported with fun web demos, and is currently maintained by *@karpathy*. In particular, the library currently includes: *Dynamic Programming*, *Tabular Temporal Difference Learning*, *Deep Q Learning*, *Policy Gradients*.

Links: https://cs.stanford.edu/people/karpathy/reinforcejs/
https://github.com/karpathy/reinforcejs
http://karpathy.github.io/2016/05/31/rl/

DeepTraffic playground

Links: https://deeplearning.mit.edu/deeptraffic/

Relevant Sources

Intro to Reinforcement Learning:
https://www.youtube.com/watch?v=AhyznRSDjw8
https://www.youtube.com/watch?v=lvoHnicueoE
https://www.youtube.com/watch?v=JgvyzIkgxF0
https://www.youtube.com/watch?v=zR11FLZ-O9M
https://www.youtube.com/watch?v=LzaWrmKL1Z4

RL Courses:
https://www.youtube.com/playlist?list=PLoROMvodv4rOSOPzutgyCTapiGIY2Nd8u
https://www.youtube.com/watch?v=2pWv7GOvuf0&list=RDQMWhPupz0YKeA&start_radio=1
https://www.youtube.com/watch?v=0MNVhXEX9to&list=PLMrJAkhIeNNQe1JXNvaFvURxGY4gE9k74
https://www.youtube.com/watch?v=JHrlF10v2Og&list=PL_iWQOsE6TfX7MaC6C3HcdOf1g337dlC9
https://www.youtube.com/watch?v=nyjbcRQ-uQ8&list=PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv
https://www.youtube.com/playlist?list=PLkoCa1tf0XjCU6GkAfRCkChOOSH6-JC_2

RL Courses (Practical Tutorial)
https://www.youtube.com/watch?v=ELE2_Mftqoc
https://simoninithomas.github.io/deep-rl-course/
https://www.youtube.com/watch?v=K2qjAixgLqk

Deep Reinforcement Learning from AlphaGo to AlphaStar:
https://www.youtube.com/watch?v=x5Q79XCxMVc

OpenAI Spinning Up: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Relevant Sources

RL books:
http://incompleteideas.net/book/the-book-2nd.html

RL tutorials:
https://spinningup.openai.com/en/latest/spinningup/rl_intro.html
https://towardsdatascience.com/reinforcement-learning-tutorial-part-1-q-learning-cadb36998b28
https://towardsdatascience.com/newbies-guide-to-study-reinforcement-learning-8b9002eff643
https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html
https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/
https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
https://towardsdatascience.com/deeptraffic-dqn-tuning-for-traffic-navigation-75-01-mph-solution-23087e2411cf
https://pathmind.com/wiki/deep-reinforcement-learning
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0
https://missinglink.ai/guides/tensorflow/tensorflow-reinforcement-learning-introduction-and-hands-on-tutorial/
https://towardsdatascience.com/introduction-to-tf-agents-a-library-for-reinforcement-learning-in-tensorflow-68ab9add6ad6

RL - Practical Tutorial with Python (Q learning and DQN)
https://www.youtube.com/playlist?list=PLQVvvaa0QuDezJFIOU5wDdfy4e9vdnx-7
https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/
https://www.youtube.com/watch?v=SMZfgeHFFcA
https://www.youtube.com/watch?v=yMk_XtIEzH8&list=RDCMUCfzlCWGWYyIQ0aLC5w48gBQ&start_radio=1
https://www.youtube.com/watch?v=Mut_u40Sqz4