Different than the rest of machine learning

- Markov Decision Processes (MDP)
- The Bellman equation
- Q-networks
- Policy gradients

Temporal Difference (TD) Learning

Learn at each time step

Each network gets a cost

$$\delta = R_t + \gamma V(S_{t+1}) - V(S_t)$$

$\boxed{\delta^2}$

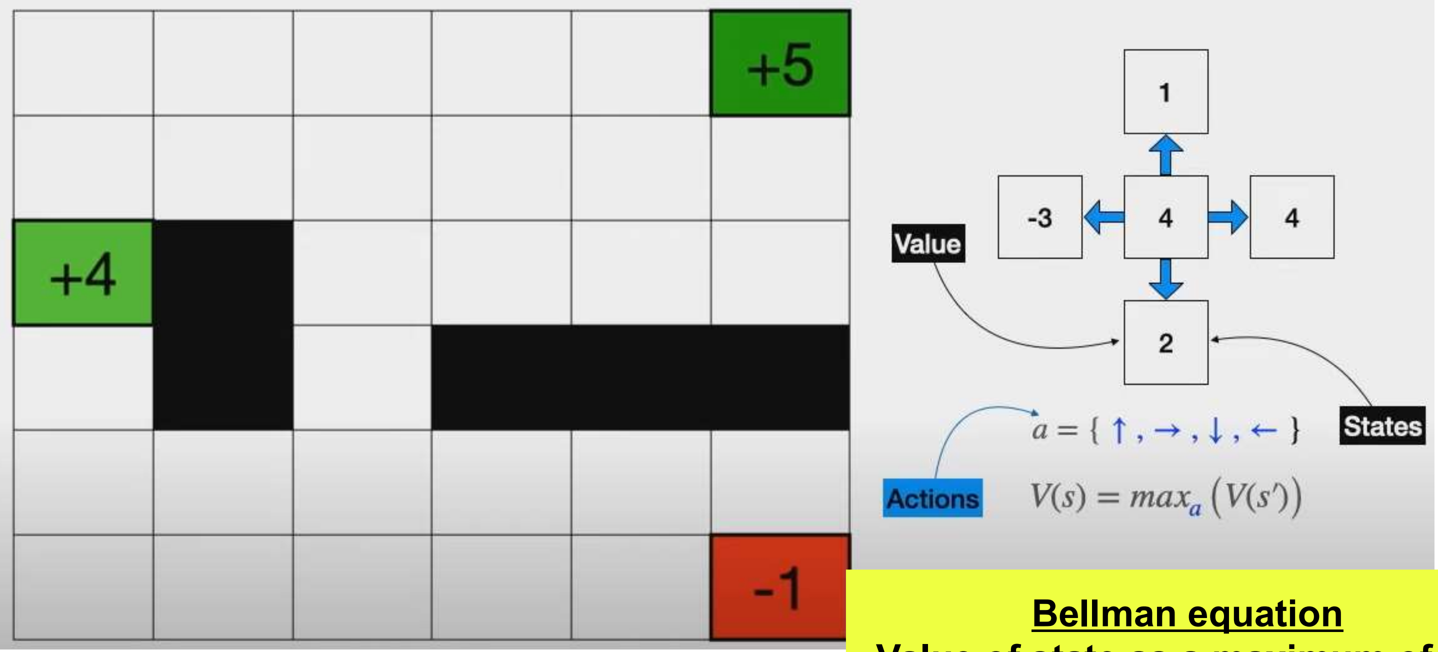
Critic loss

$\boxed{\delta \ln \pi(A_t|S_t)}$

Actor loss

Values, states, actions, and policy

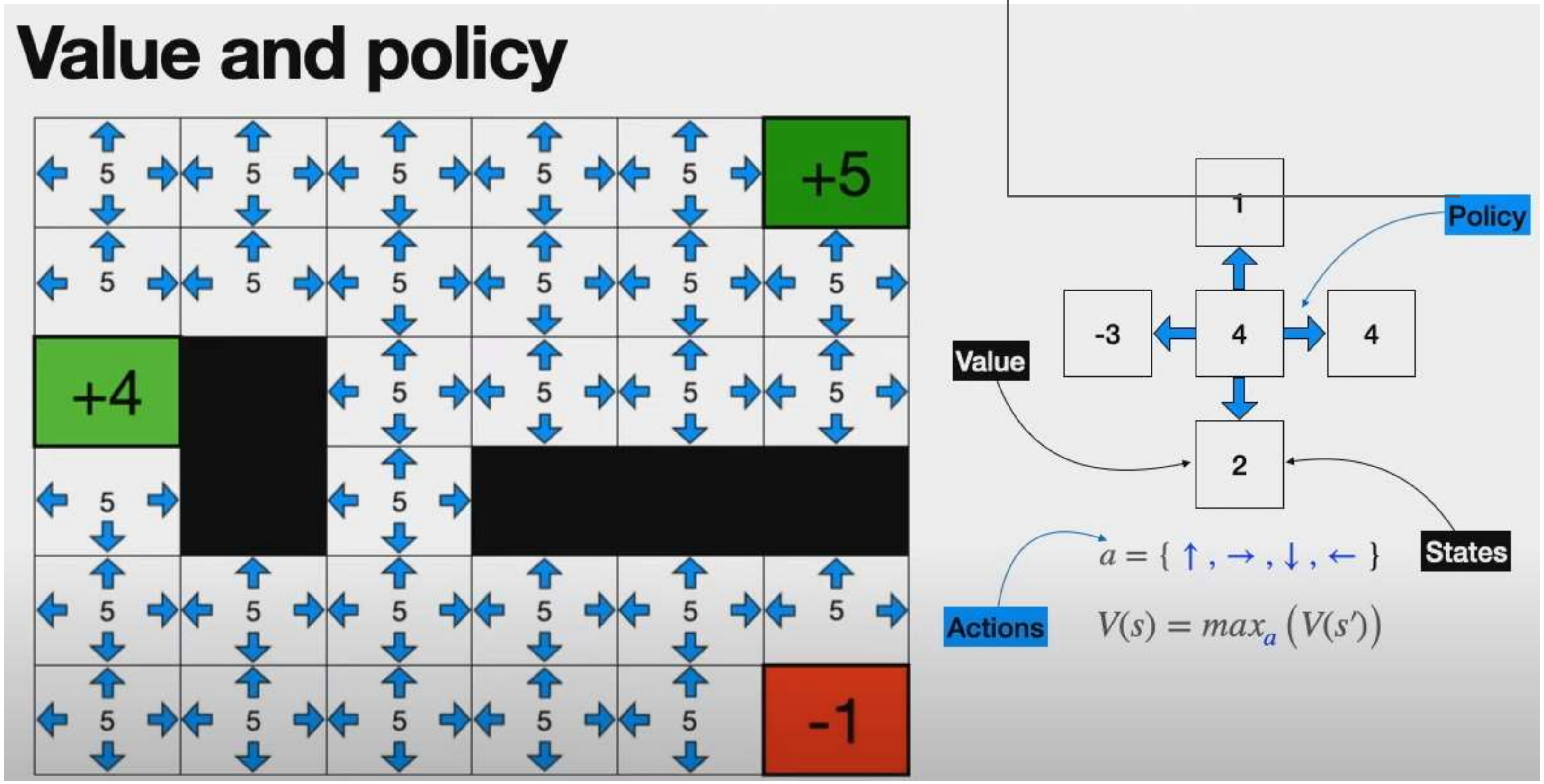
Value and policy



<u>Bellman equation</u>
Value of state as a maximum of the value of its neighbors for the neighbors states obtained by applying any possible action

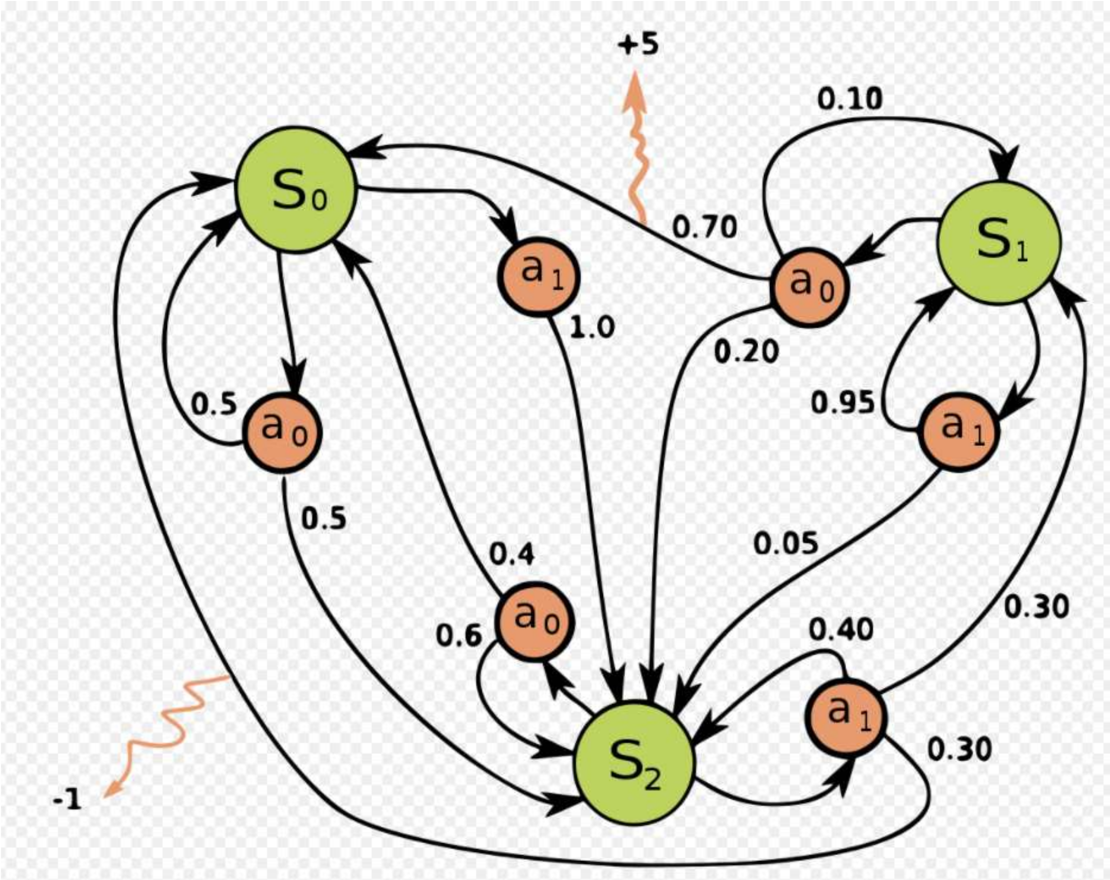
<u>Best possible decision</u>
⇔ POLICY

<u>POLICY (best decision)</u>
-> Set of instructions to always take the agent in the best possible path with respect to gaining points

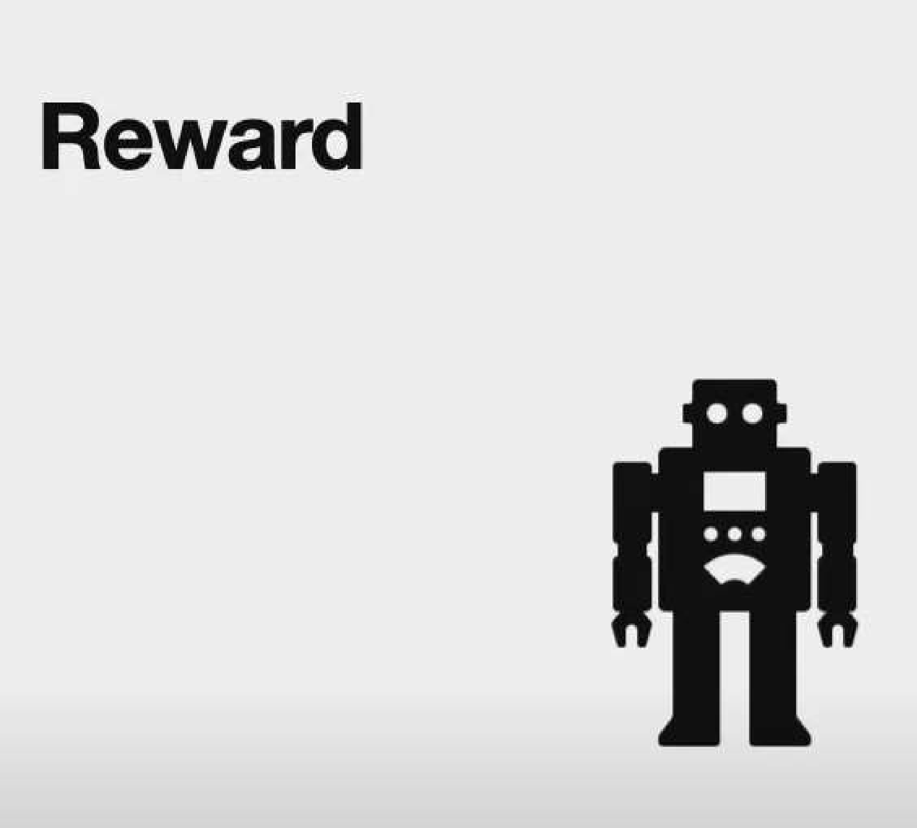


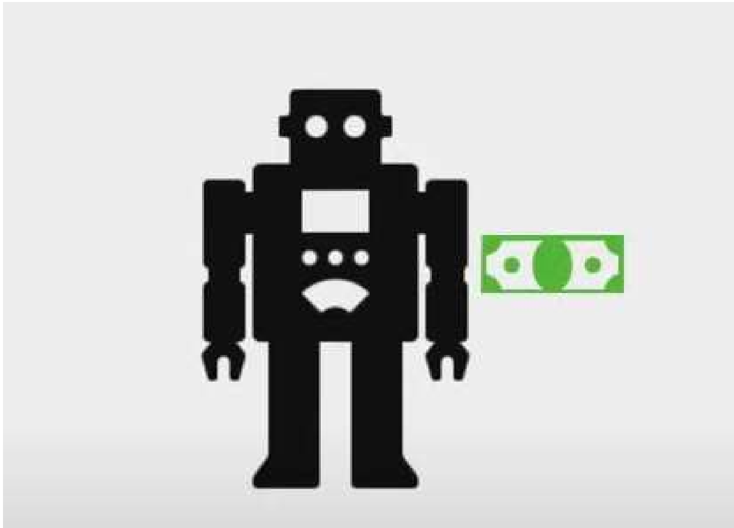
Markov decision processes (MDP)

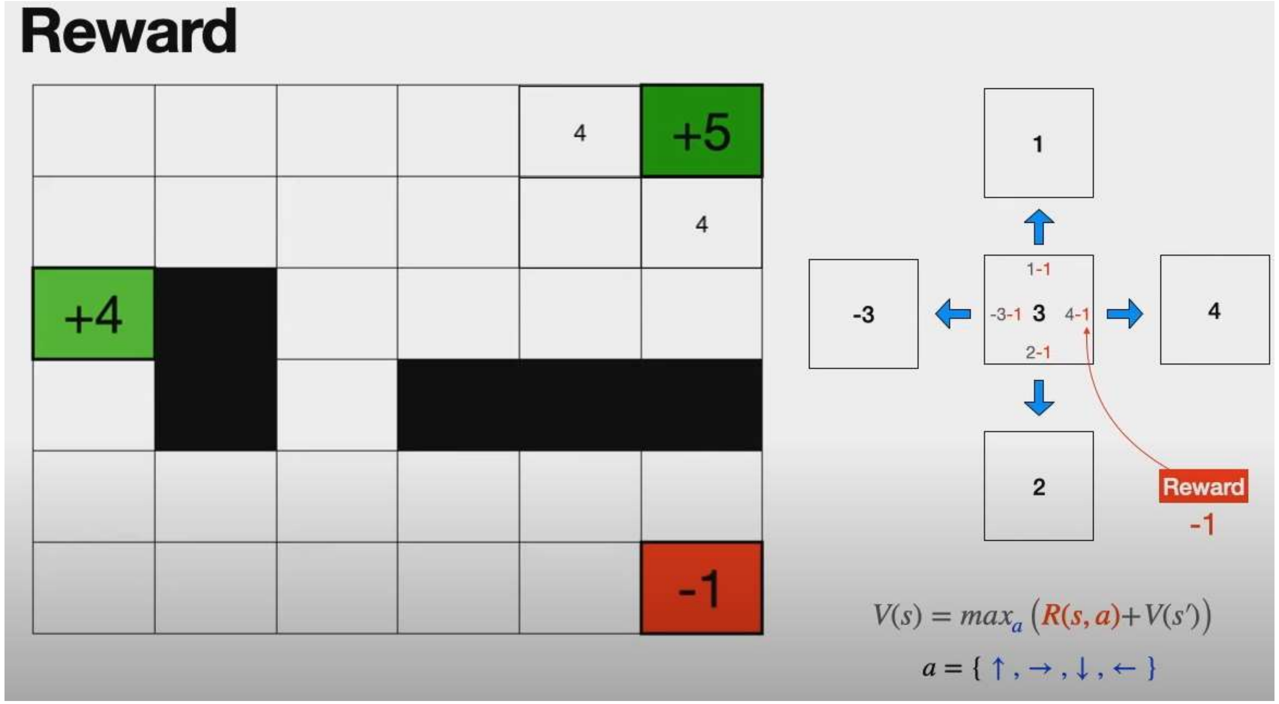
Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).



REWARD
-> Positive
-> Negative



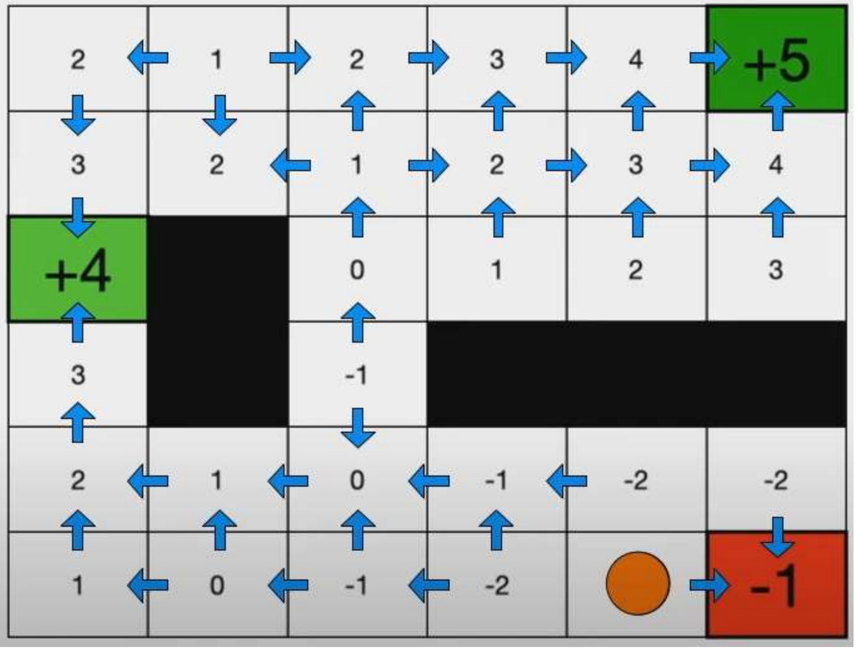


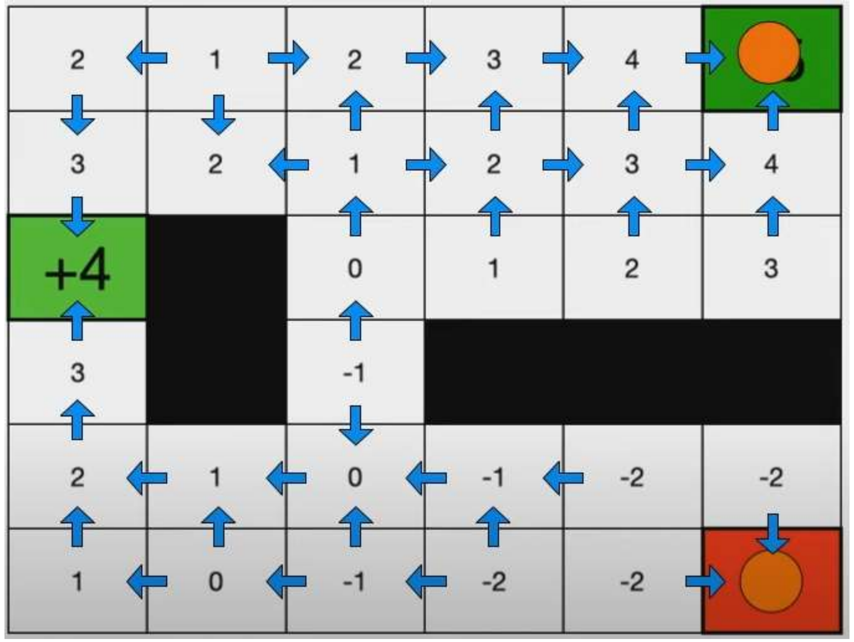


Bellman equation 1st change

each value of state is the maximum of points you can obtain if you walk in the best possible way, & that best possible way is given by policy (point to the neighbor that has the highest value notice)

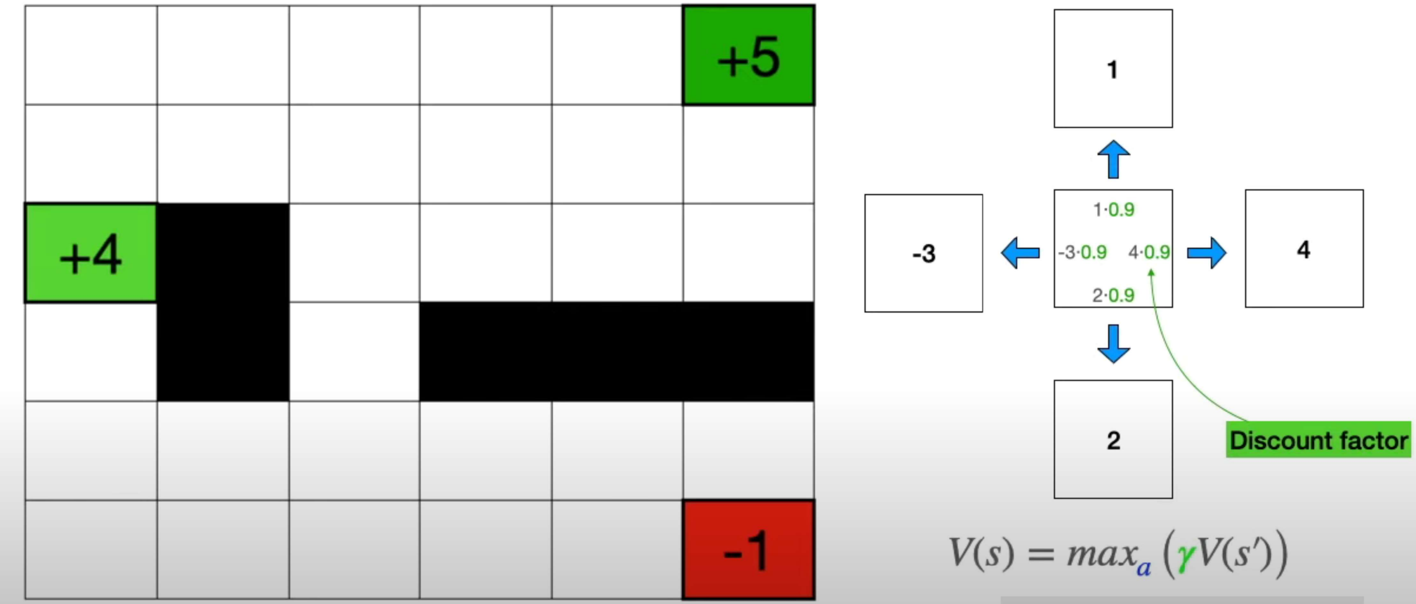
Reward


2 1 2 3 4 +5
3 2 1 2 3 4
+4 0 1 2 3
3 -1
2 1 0 -1 -2 -2
1 0 -1 -2 -1


2 1 2 3 4
3 2 1 2 3 4
+4 0 1 2 3
3 -1
2 1 0 -1 -2 -2
1 0 -1 -2 -2

Discount factor

Discount factor



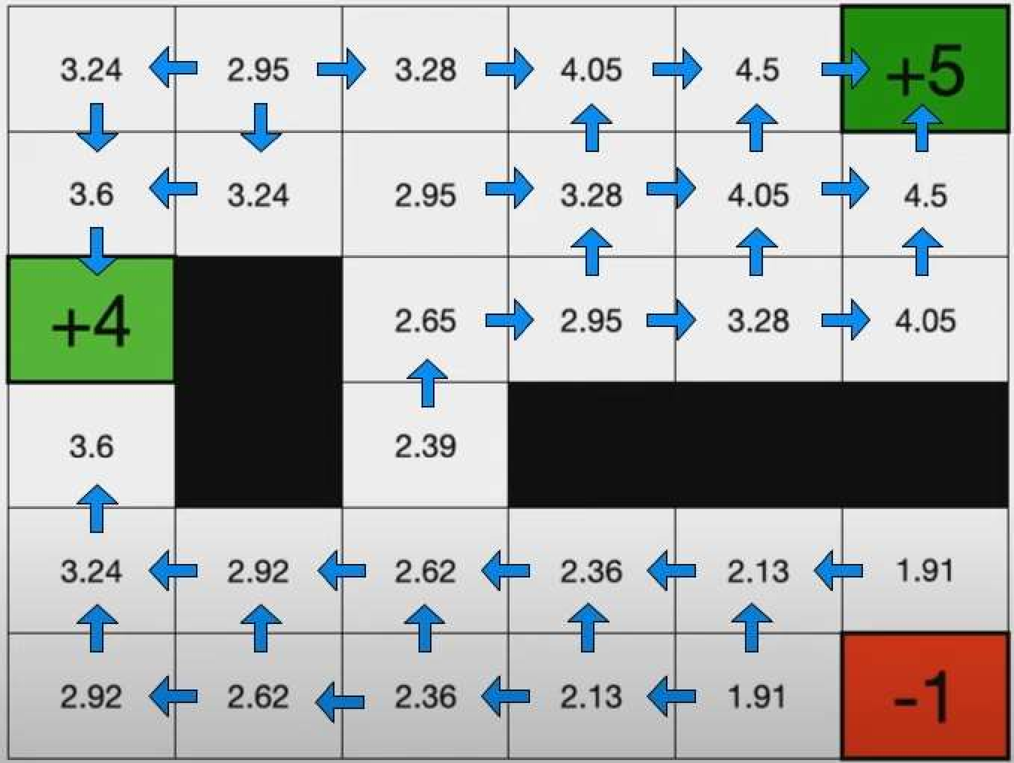
$$V(s) = max_a \left(\gamma V(s')\right)$$

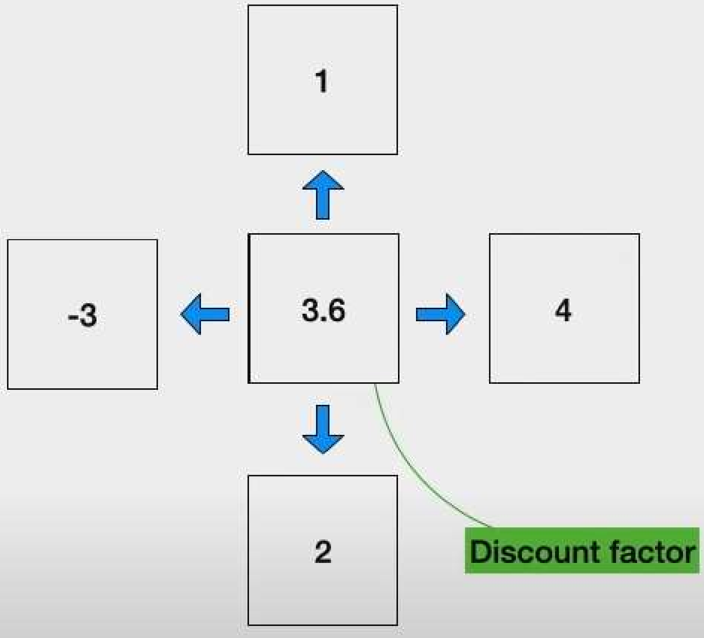
$$a = \{ \uparrow, \rightarrow, \downarrow, \leftarrow \}$$

<u>Bellman equation 2nd change</u>

Value of state as a maximum over all its neighbors where all the states i can obtain by applying the actions of a DISCOUNT FACTOR gamma times the value of that new state

Discount factor





$$V(s) = max_a \left(\gamma V(s')\right)$$

$$a = \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$$

Bellman equation

Bellman equation

$$V(s) = max_a \left(R(s,a) + V(s')\right) \qquad V(s) = max_a \left(\gamma V(s')\right)$$

Value

States

$$V(s) = max_a \left(R(s,a) + \gamma V(s')\right)$$

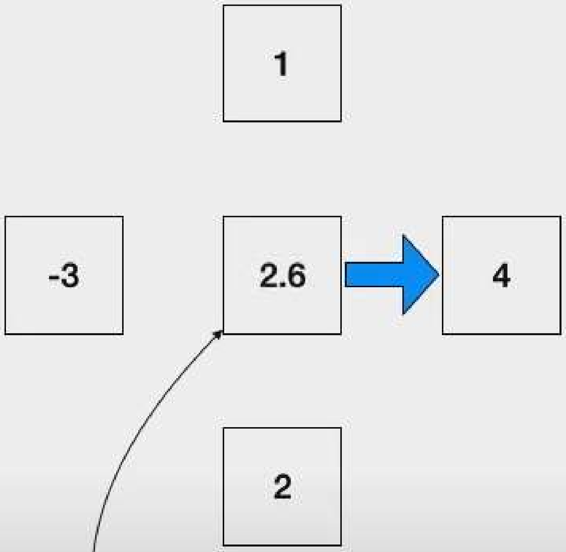
Actions

Reward

-1

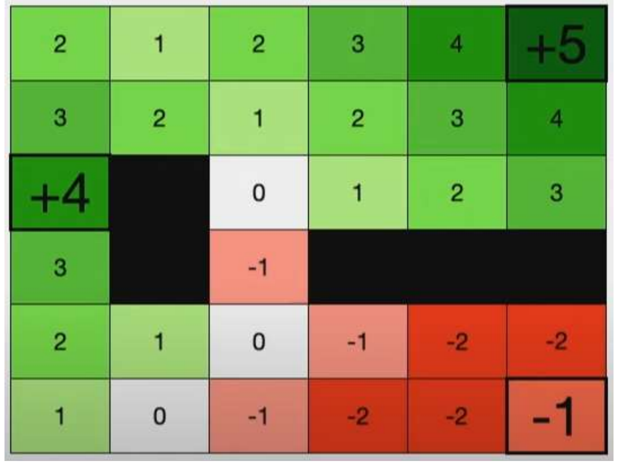
Discount factor

0.9

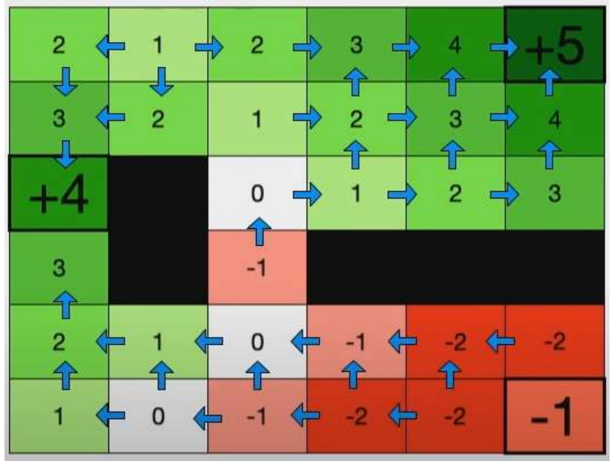


$4 \cdot 0.9 - 1 = 2.6$

Solving the Bellman equation



Bellman equation allways satisfied

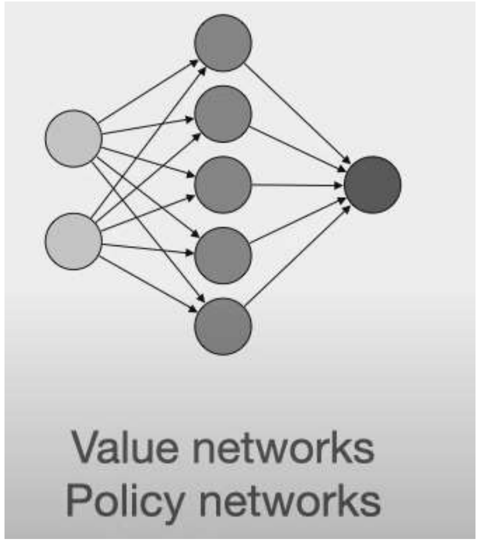


Best possible decision
⇔ **POLICY**

If we have a much bigger example : the algo says : you have to visit all the boxes, not only once but several times until the values start converging & that's impossible to do when you have a very very large universe with millions of states & actions… : it is very expensive to go over all of them, so what can we do ?

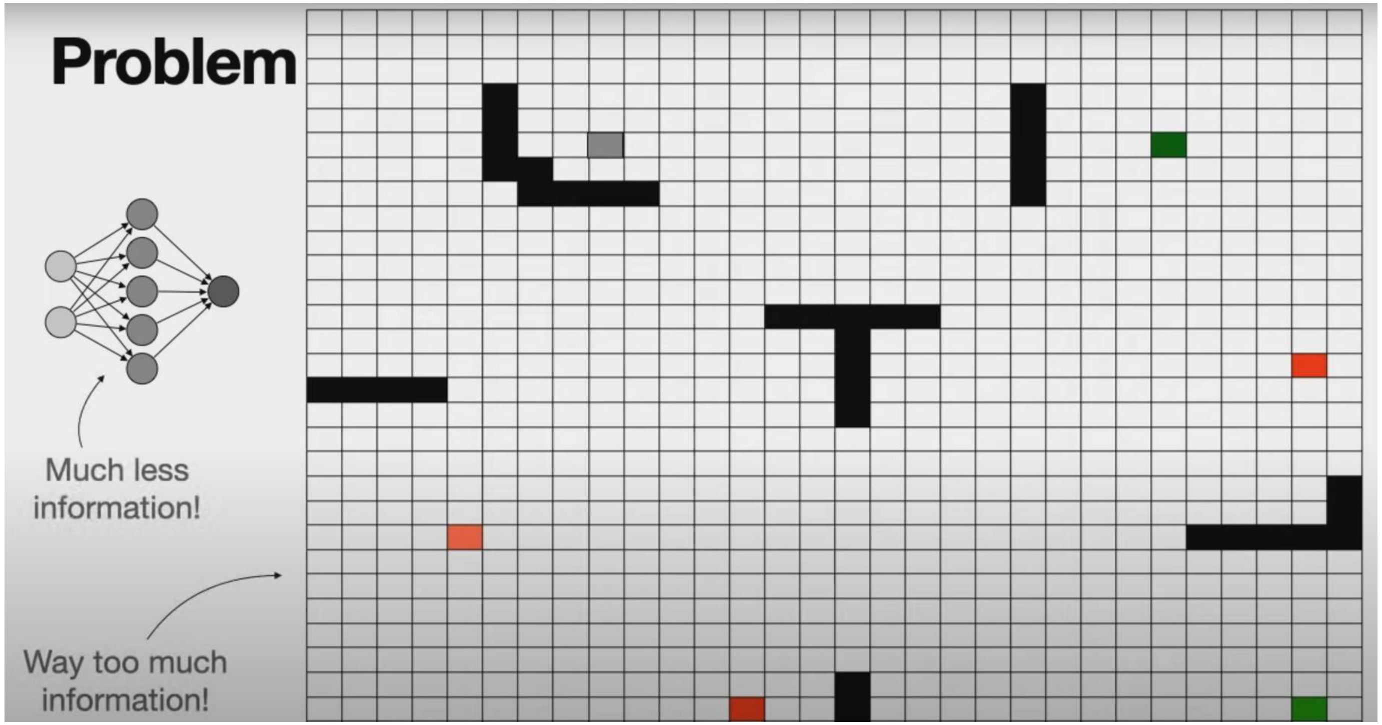
THIS IS WHERE <u>NEURAL NETWORK</u> COME TO OUR RESCUE

NN will come in 2 ways : one of them for calculating the value which is called a <u>value network (Q NETWORK</u>) & the other one for calculating the <u>policy</u> (<u>Policy NETWORK</u>)

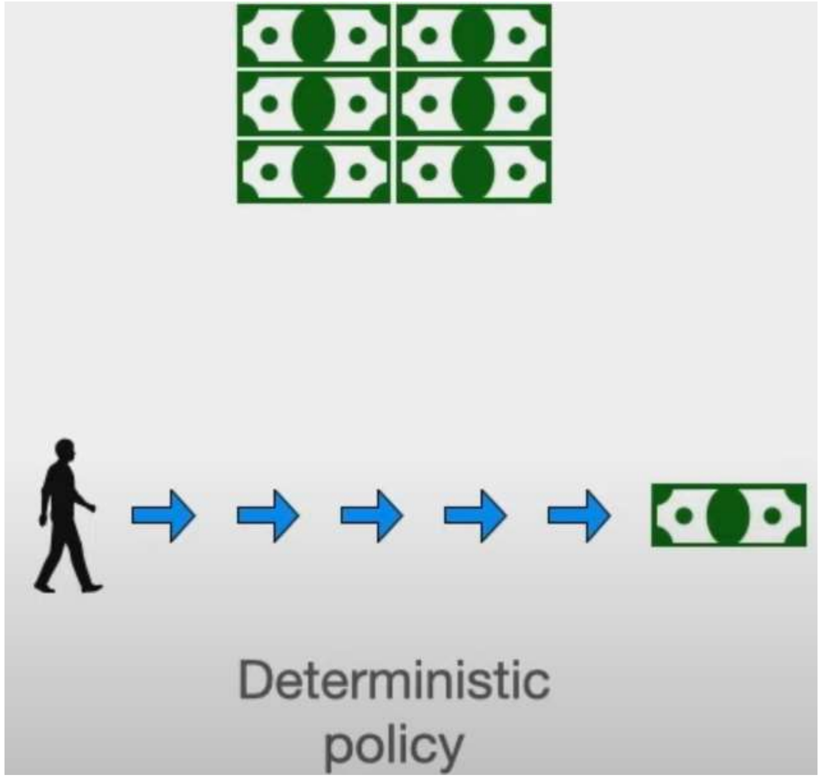


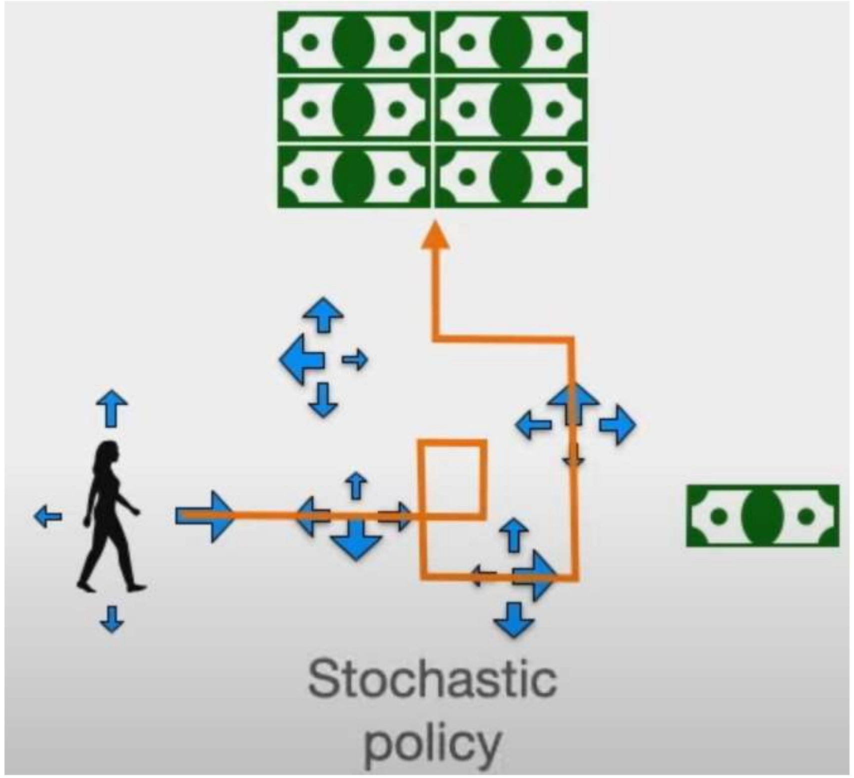
In this way we won't have to go over every state several times to learn the value of the policy we simply have to let our agent wander and the NN will be smart enough to pick up information from the places that this agent manages to visit & propagate them to the entire grid.

This not only good for saving time but also for saving space



Deterministic vs stochastic policies

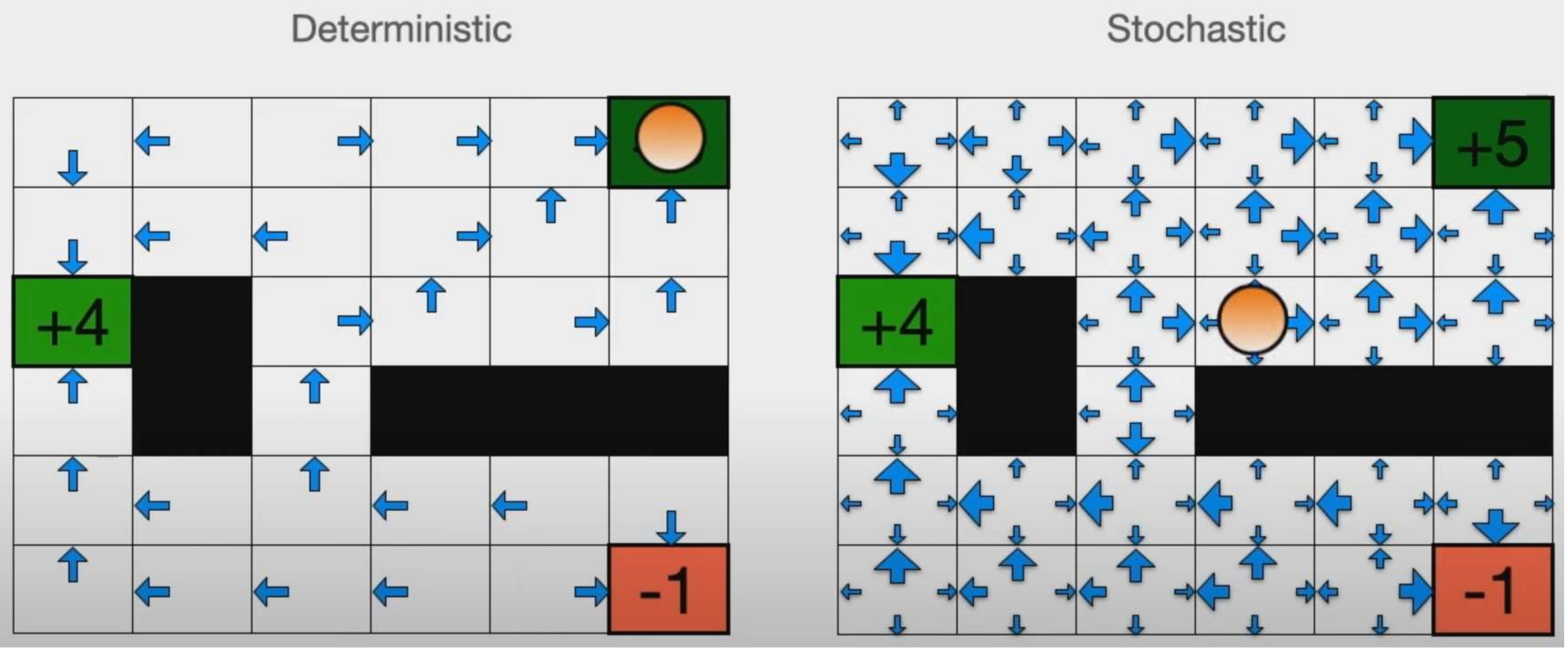




EXPLOITATION

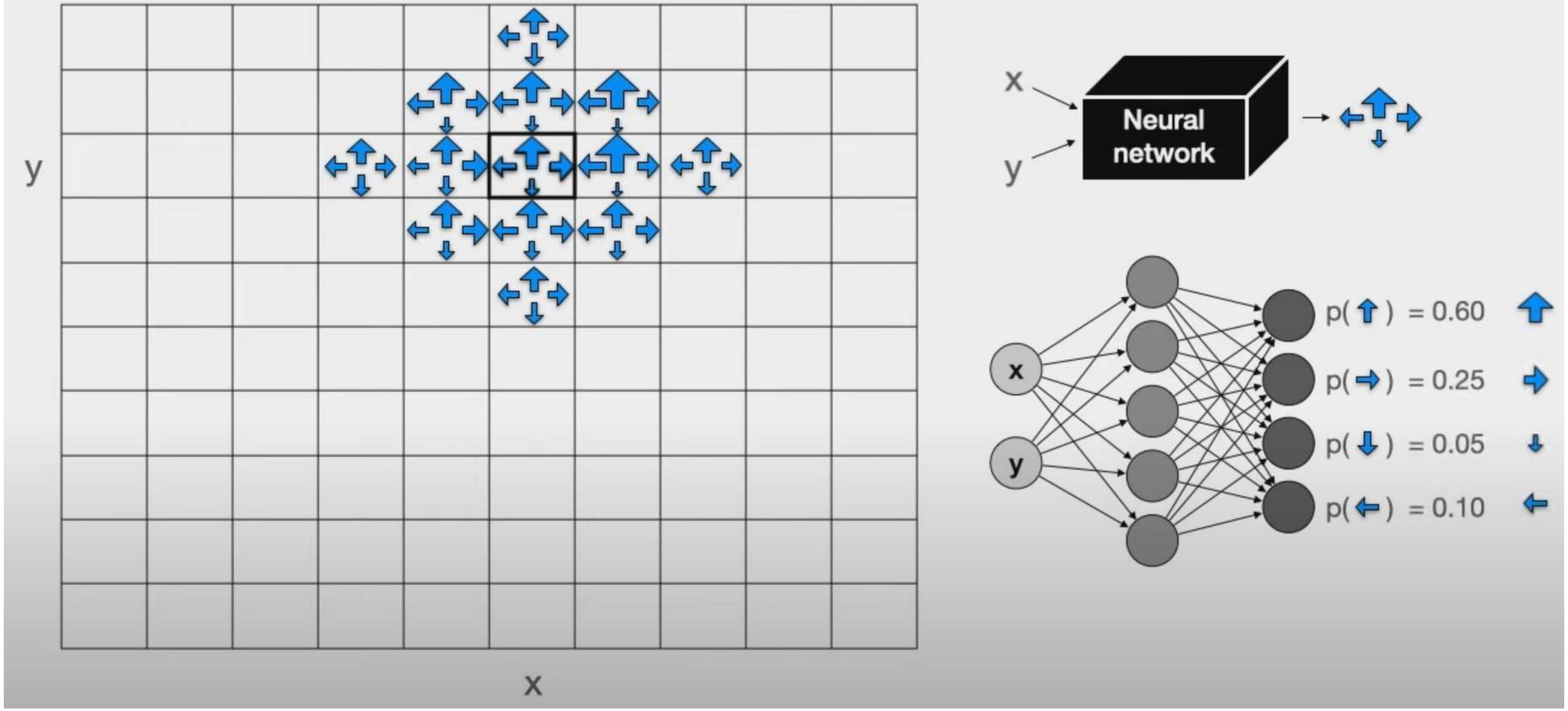
EXPLORATION

Deterministic vs stochastic



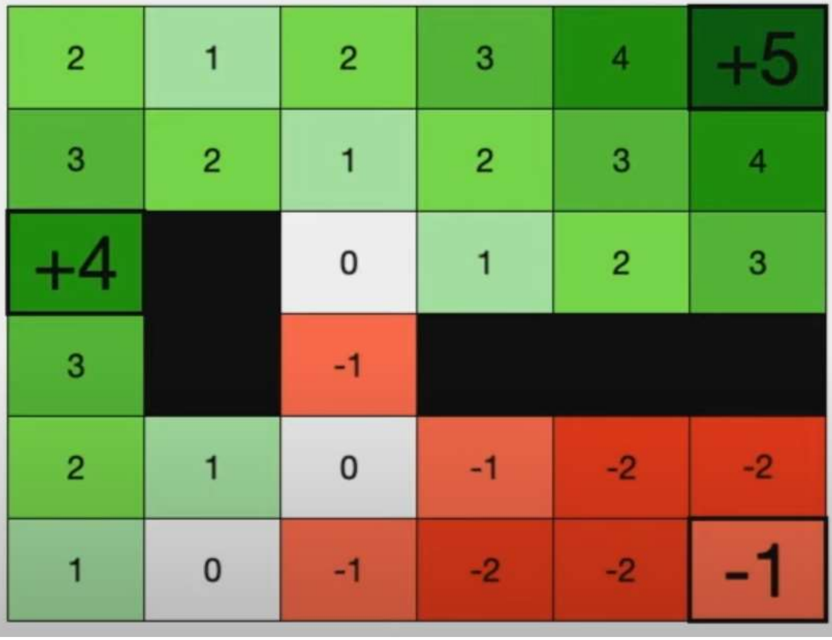

Neural networks

Policy network

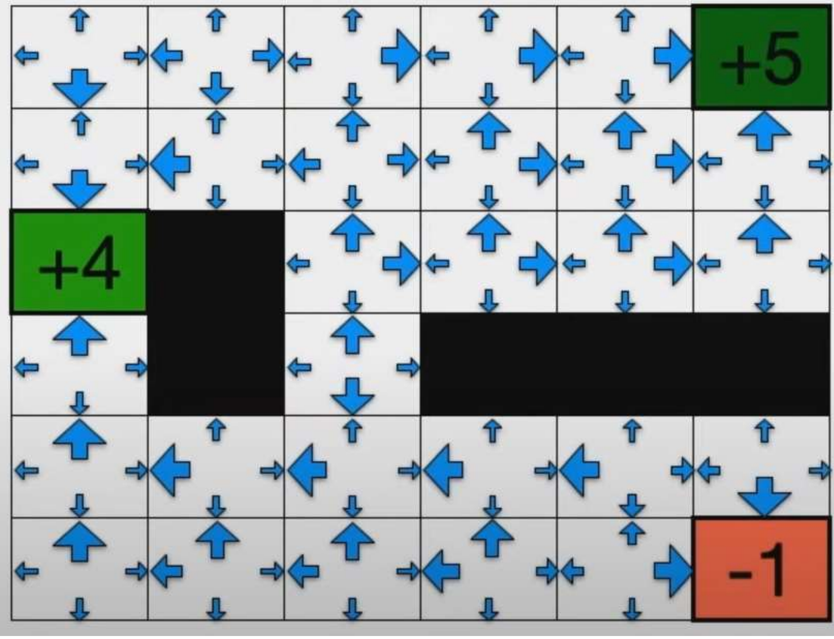


Two neural networks

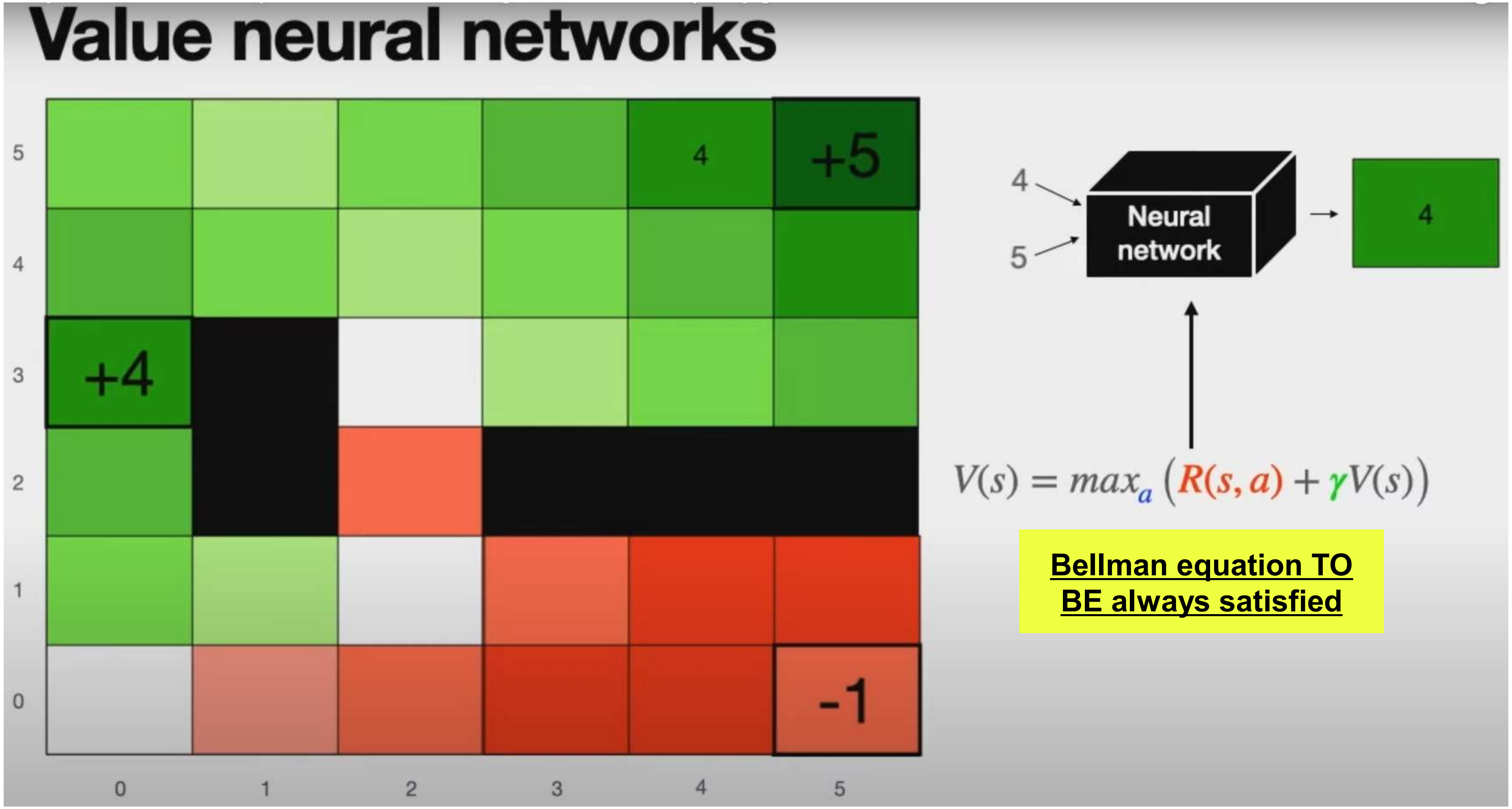
Value neural network

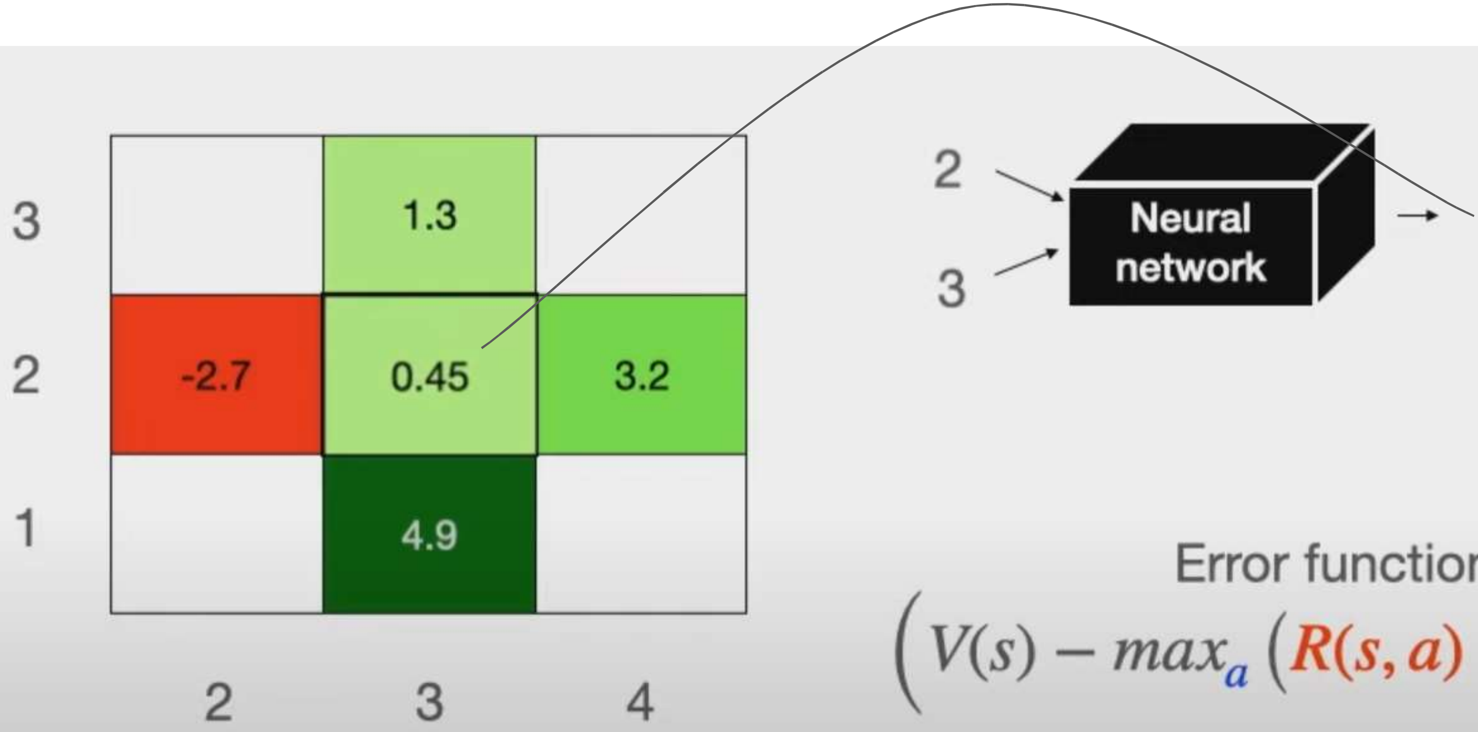


Policy neural network



Value neural networks


Value neural networks
5
4
3
2
1
0
0
1
2
3
4
5
4
+5
+4
-1
4
5
Neural network
4
$V(s) = max_a \left(R(s,a) + \gamma V(s)\right)$
Bellman equation TO BE always satisfied



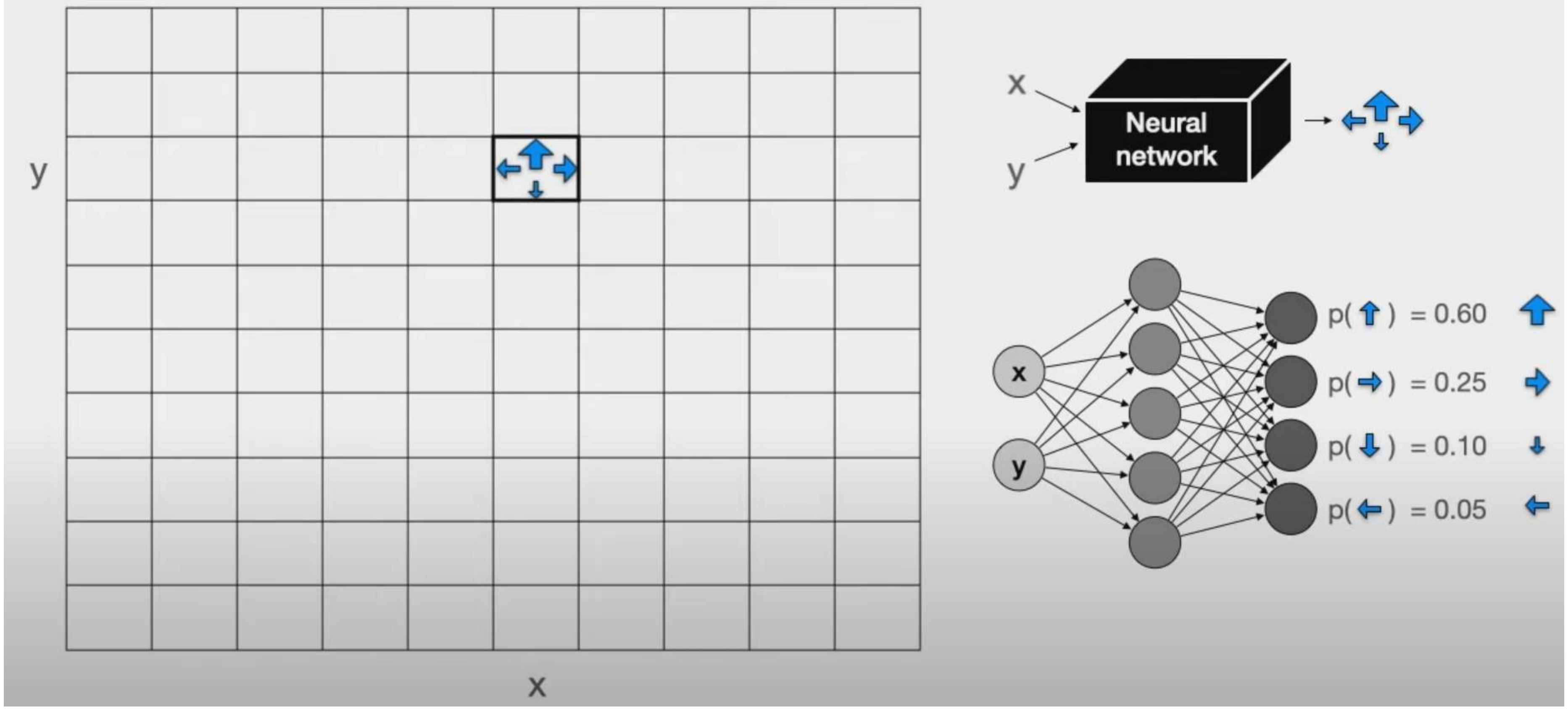
Error function

$$\left(V(s) - max_a \left(R(s,a) + \gamma V(s)\right)\right)^2$$

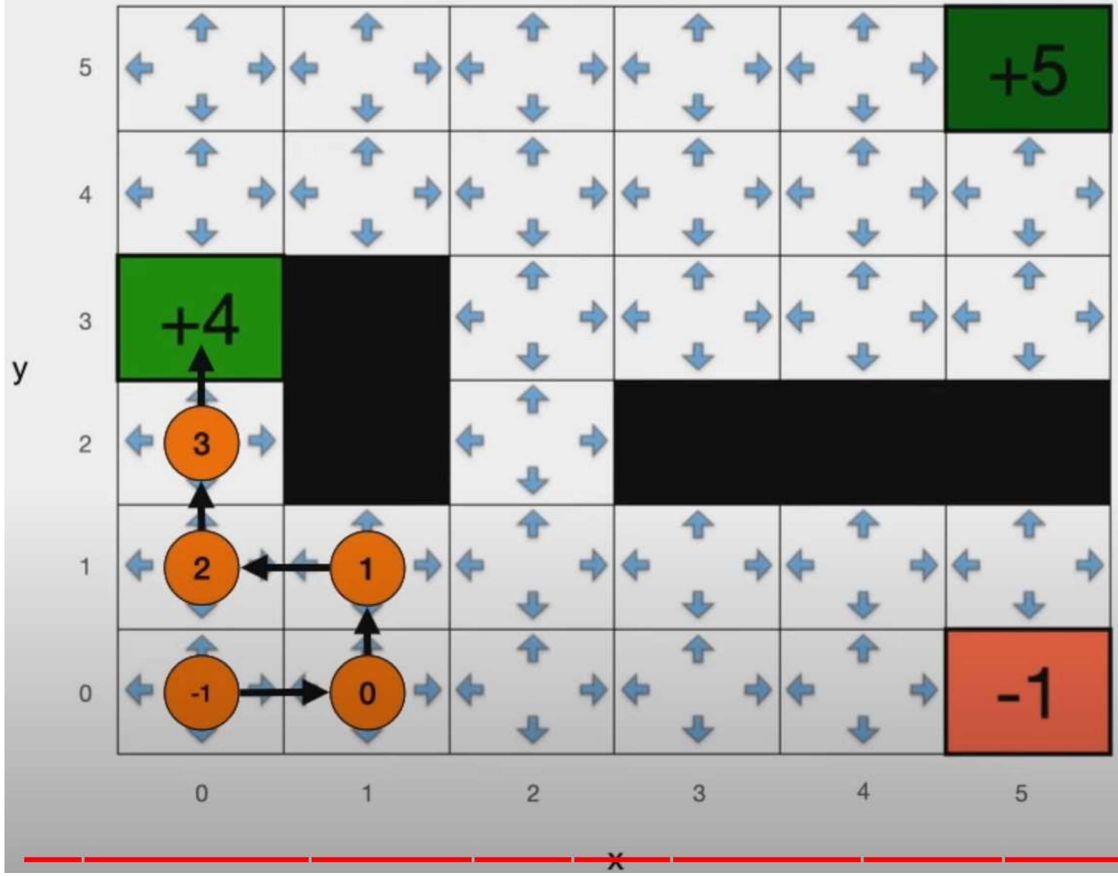
$$(0.2 - \max\{2.2, 0.3, -3.7, 3.9\})^2$$

Policy neural networks

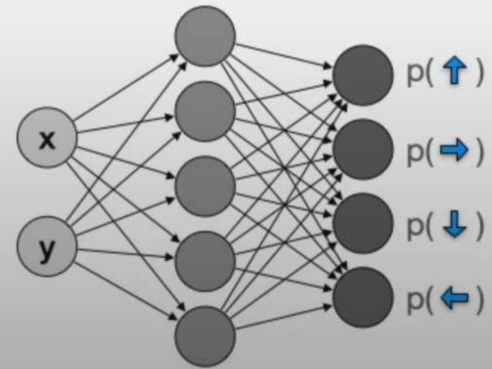
Policy neural network

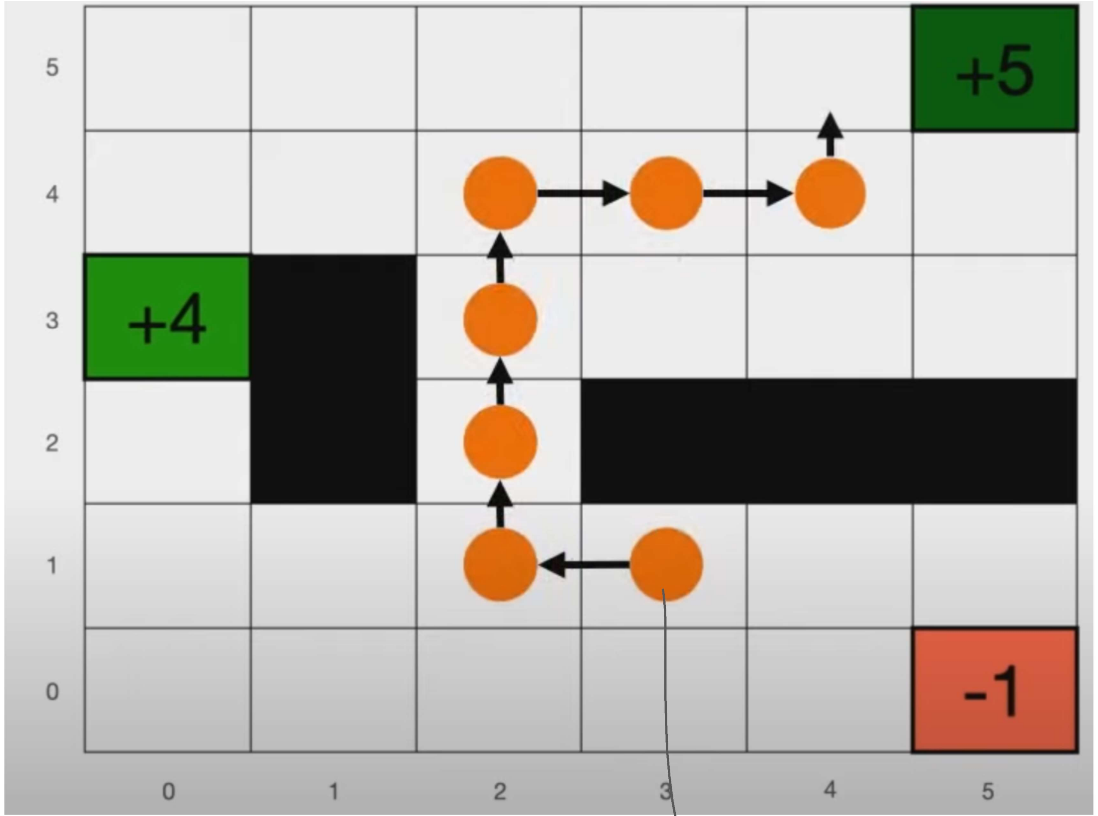


How to train it?

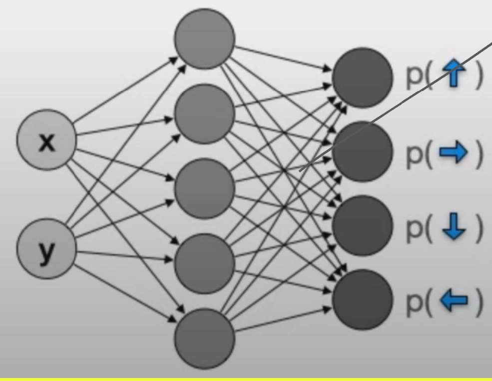


Gain	x	y	Direction	Change
3	0	2	↑	increase
2	0	1	↑	increase
1	1	1	←	increase
0	1	0	↑	stay
-1	0	0	→	decrease





Gain	x	y	Direction	Change
3	0	2	↑	increase
2	0	1	↑	increase
1	1	1	←	increase
0	1	0	↑	stay
-1	0	0	→	decrease



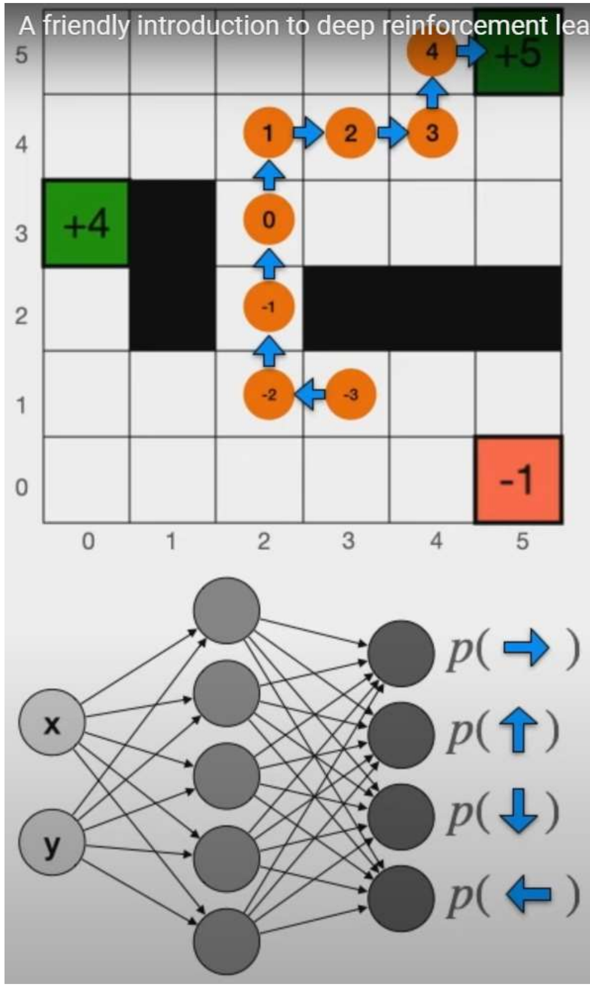
we take another walk

That is the table we give to NN

Training the policy neural network

A friendly introduction to deep reinforcement learning, Q-networks and policy gradients

Training the policy network



Gain	x	y	action	Probability	Increase log loss	change
4	4	5	→	$p(\rightarrow) = 0.3$	4 $\ln(p(\rightarrow))$	increase
3	4	4	↑	$p(\uparrow) = 0.9$	3 $\ln(p(\uparrow))$	increase
2	3	4	→	$p(\rightarrow) = 0.1$	2 $\ln(p(\rightarrow))$	increase
1	2	4	→	$p(\rightarrow) = 0.2$	1 $\ln(p(\rightarrow))$	increase
0	2	3	↑	$p(\uparrow) = 0.5$	0 $\ln(p(\uparrow))$	stay
-1	2	2	↑	$p(\uparrow) = 0.4$	-1 $\ln(p(\uparrow))$	decrease
-2	2	1	↑	$p(\uparrow) = 0.3$	-2 $\ln(p(\uparrow))$	decrease
-3	3	1	←	$p(\leftarrow) = 0.7$	-3 $\ln(p(\leftarrow))$	decrease