Introduction to Deep Reinforcement Learning

Module ONE of two

Markus Buchholz

Machine Learning.

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

Goal: Maximize future rewards over many time steps

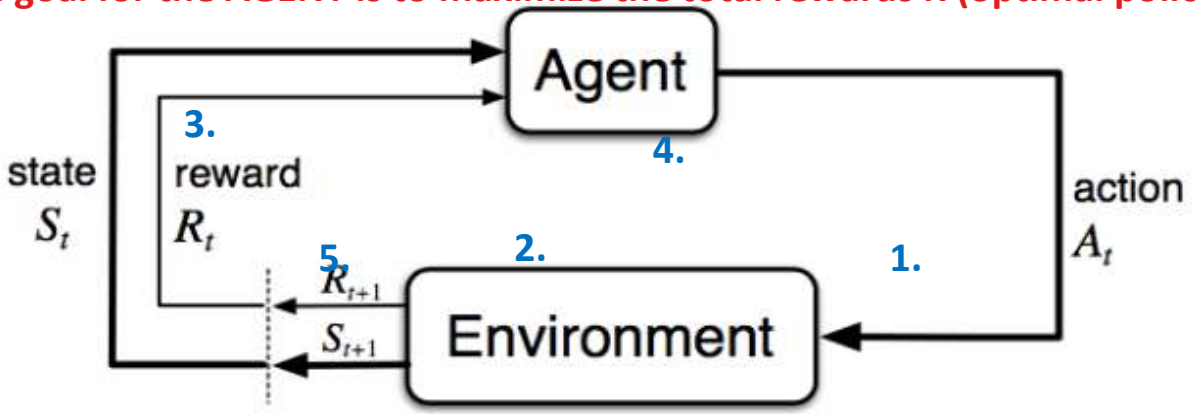
Apple example:

Eat this thing because it will keep you alive.

https://www.youtube.com/watch?v=W_gxLKSsSIE

What is Reinforcement Learning?

The goal for the AGENT is to maximize the total rewards R (optimal policy)



RL algorithm

The steps involved in typical RL algorithm are as follows:

1. First, the agent interacts with the environment by performing an action
2. The agent performs an action and moves from one state to another
3. And then the agent will receive a reward based on the action it performed
4. Based on the reward, the agent will understand whether the action was good or bad
5. If the action was good, that is, if the agent received a positive reward, then the agent will prefer performing that action or else the agent will try performing an other action which results in a positive reward. So it is basically a trial and error learning process

Key Concepts (type of environments)

Deterministic environment

Stochastic environment

Fully observable environment

Partially observable environment

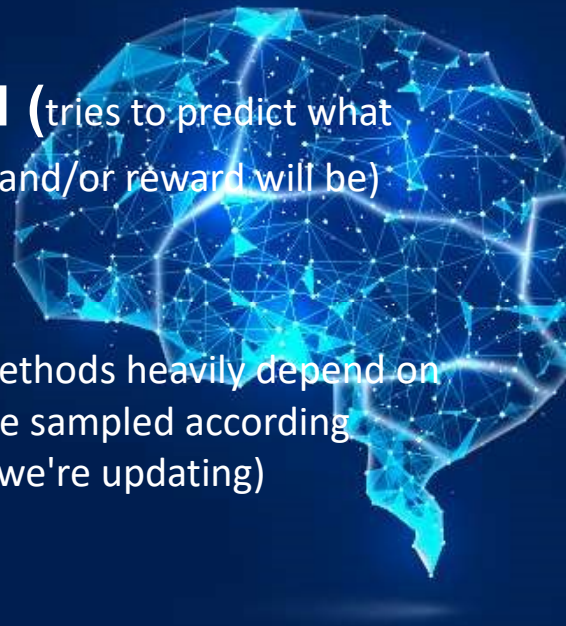
Discrete environment

Continuous environment

Episodic and non-episodic environment

Single and multi-agent environment

Model – free (mainly used;
the agent takes current observations
performs some computations on them,
and the result is the action that it should take.

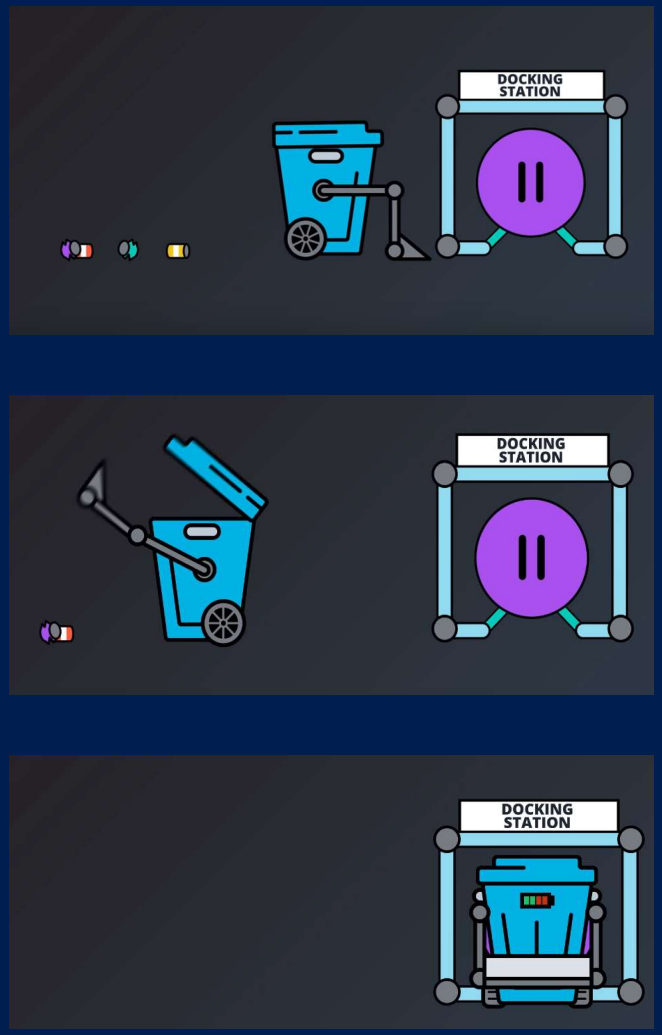


Model – based (tries to predict what
the next observation and/or reward will be)

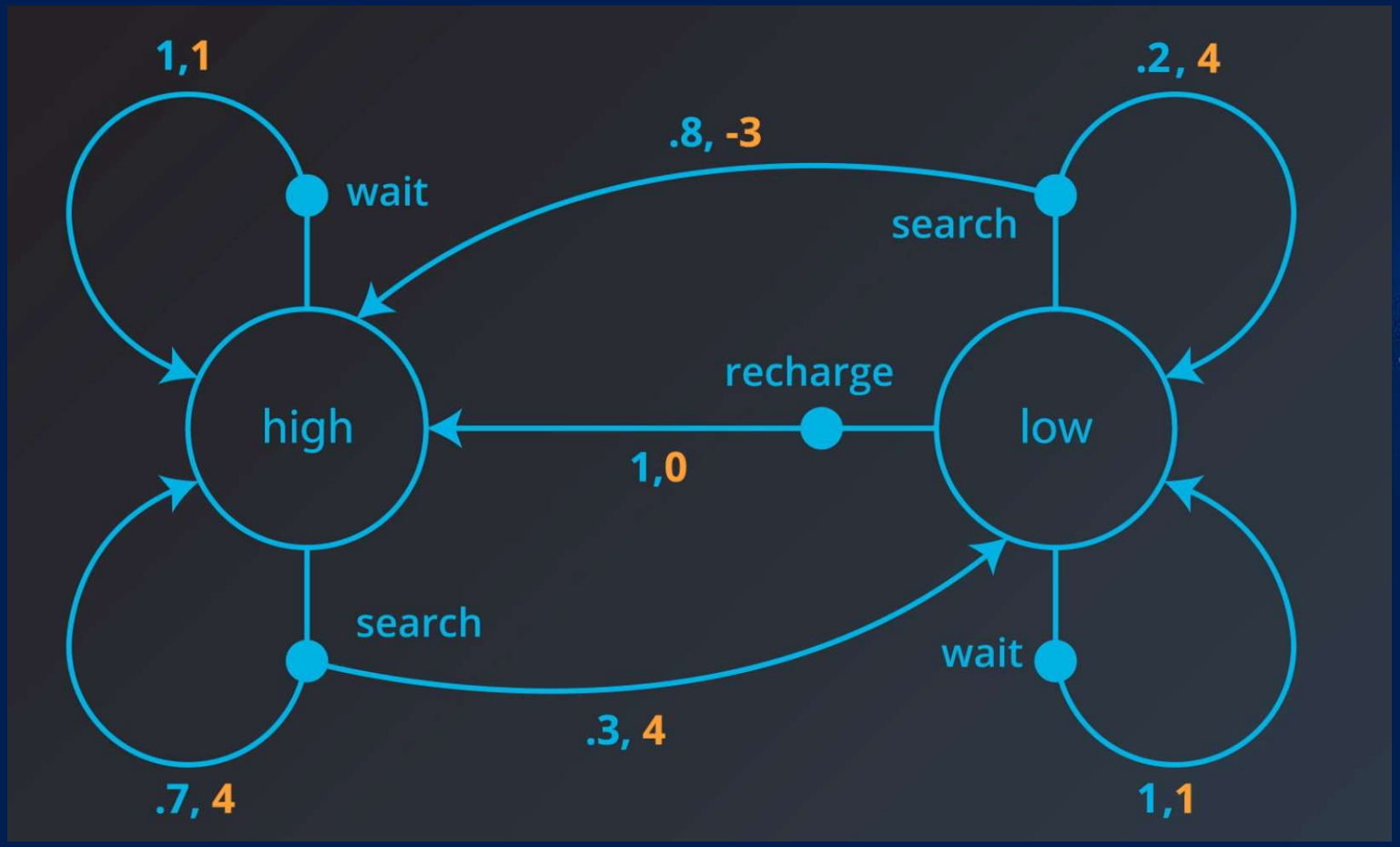
On – policy (methods heavily depend on
the training data to be sampled according
to the current policy we're updating)

Off – policy (learns on historical data)
=> SDC

Markov Decision Process (MDP). Framework for RL.



Markov Decision Process



Markov Decision Process

A (finite) Markov Decision Process (MDP) is defined by:

- a (finite) set of states $\mathcal{S}$
- a (finite) set of actions $\mathcal{A}$
- a (finite) set of rewards $\mathcal{R}$
- the one-step dynamics of the environment

$$p(s', r \mid s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

- a discount rate $\gamma \in [0, 1]$ for all s, s', a and r

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots$$

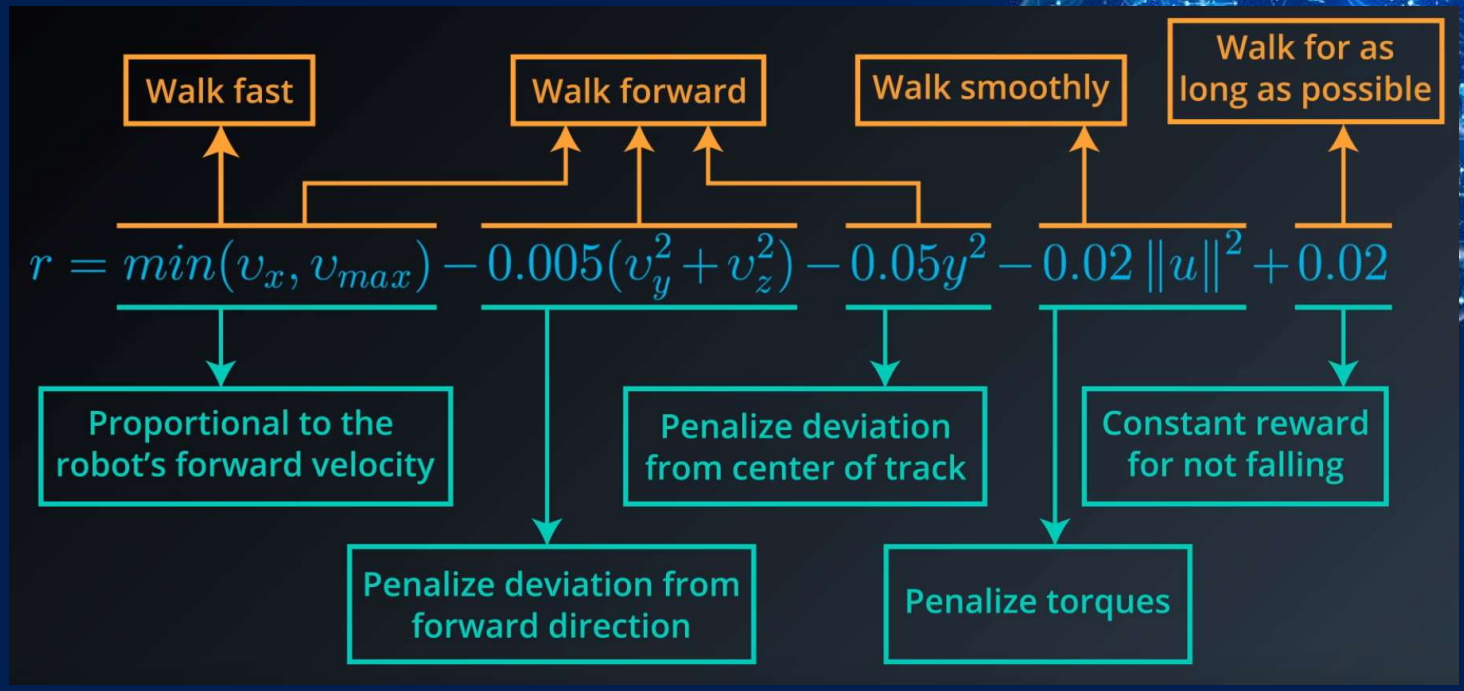
Most important MDP feature: every next state S (t+1) the Agent transits to depends only on actual state S(t) and action A(t). Historical independent.

RL main challenge => Definition of Reward

https://arxiv.org/pdf/1707.02286.pdf

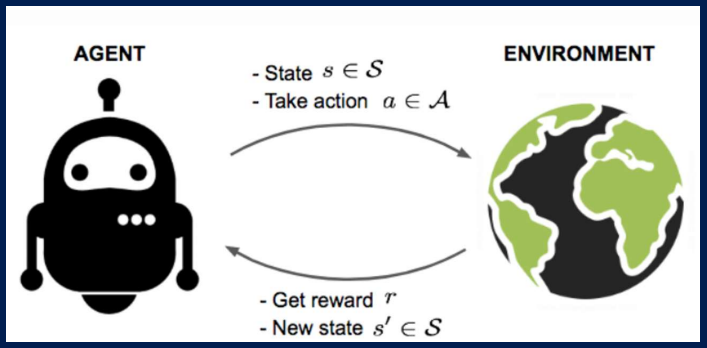
https://openai.com/blog/competitive-self-play/

https://www.youtube.com/watch?v=hx_bgoTF7bs



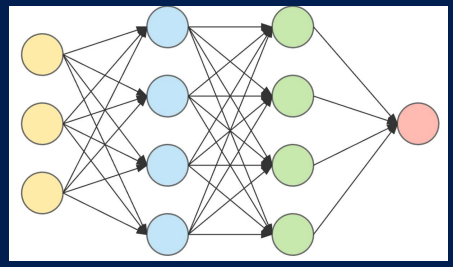
(Deep) Reinforcement Learning process

The Agent explores or exploits the environment



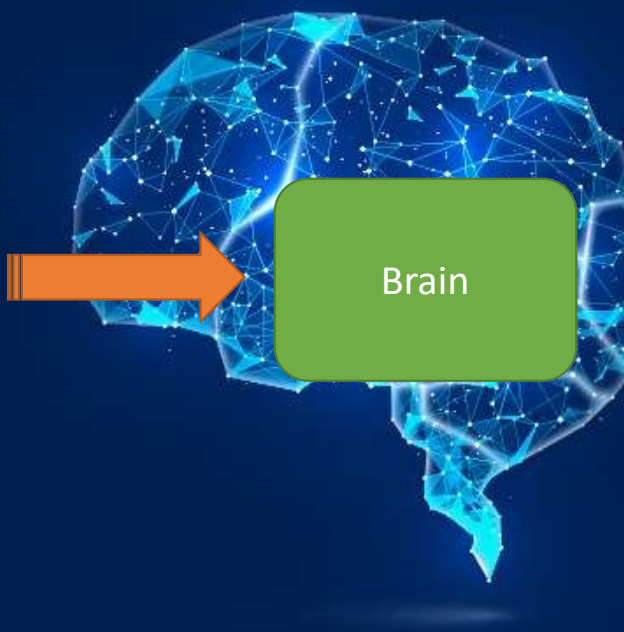
Learning

Update of weights in Deep Neural Network

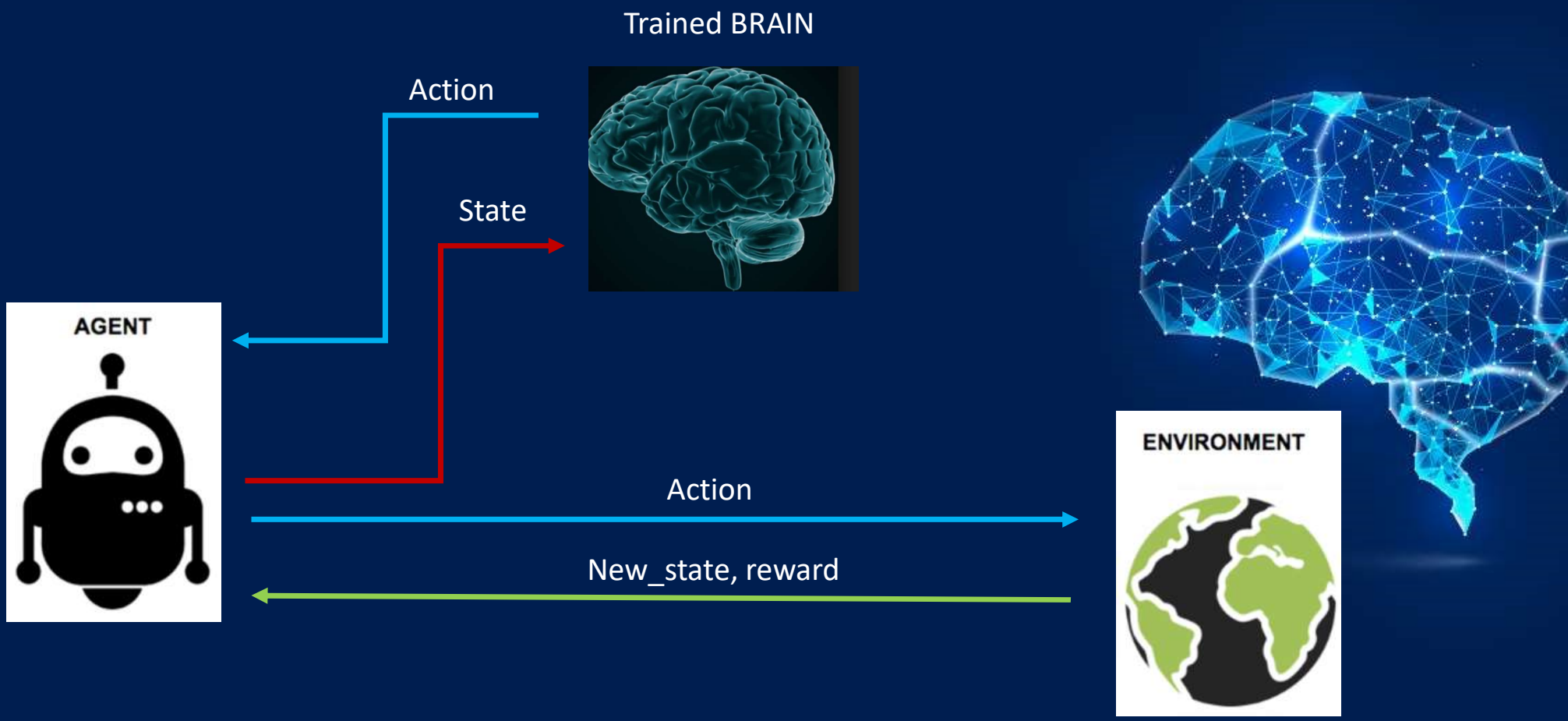


Update Q value

STATES	ACTIONS			
	UP	DOWN	LEFT	RIGHT
...	...	...	...	...
.... .	0.43	0.25	0.12	**0.8**
. ....	-0.5	-0.3	**0.38**	-0.15
:..:	**0.6**	0.18	0	0.53
:.::	-0.32	**0.75**	0.23	0.49
...	...	...	...	...



(Deep) Reinforcement Learning process. Interaction (only concept)



The Agent decisive process (how the Agent decides?) (general concept)

exploration – exploitation => **epsilon – greedy policy**

```
def epsilon_greedy_policy(state, epsilon):
        if random.uniform(0,1) < epsilon:
                return env.action_space.sample() # Random Action
        else:
                return max (list(range(env.action_space.n)), key = lambda x: q[(state,x)])

                        # Action based on max value in Q
                        table / NN value
```

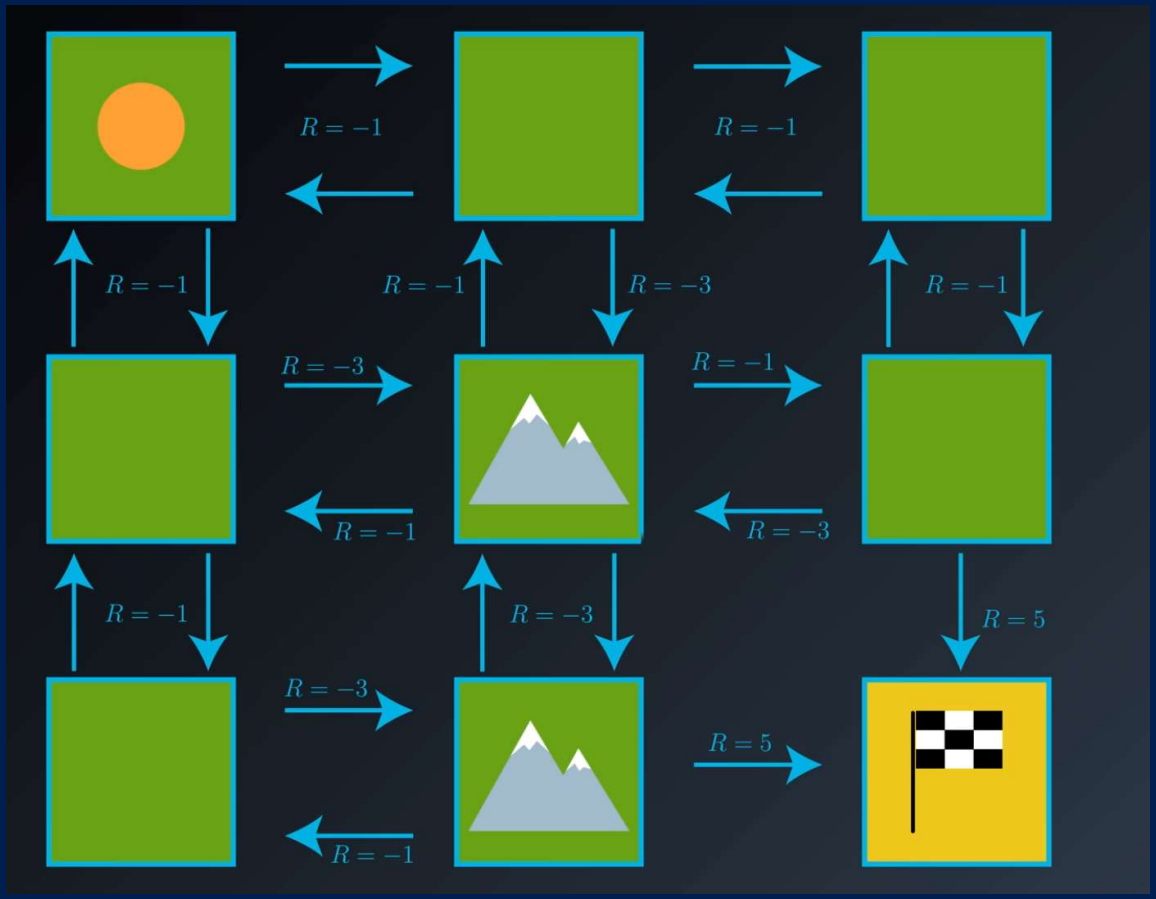
Key Concepts

Policy.

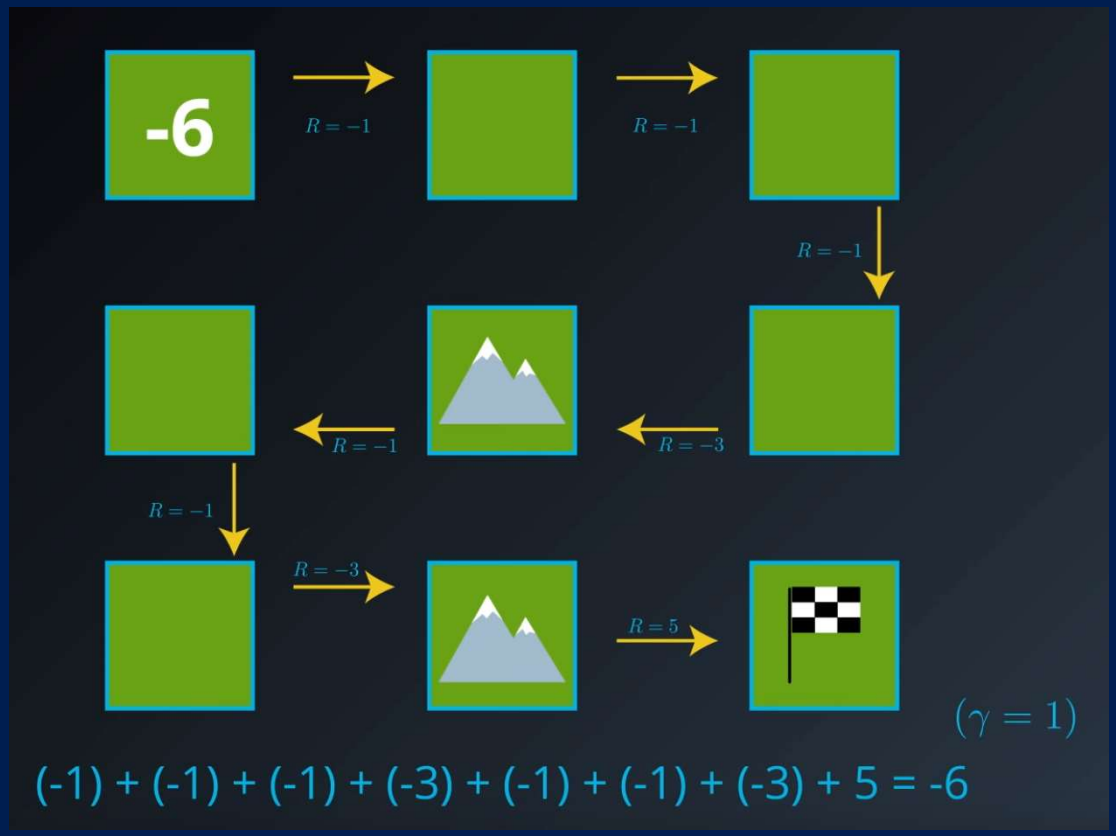
Policy, as the agent's behavior function π, tells us which action to take in state s. It is a mapping from state s to action a and can be either deterministic or stochastic:

- Deterministic: $\pi(s) = a$.
- Stochastic: $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$.

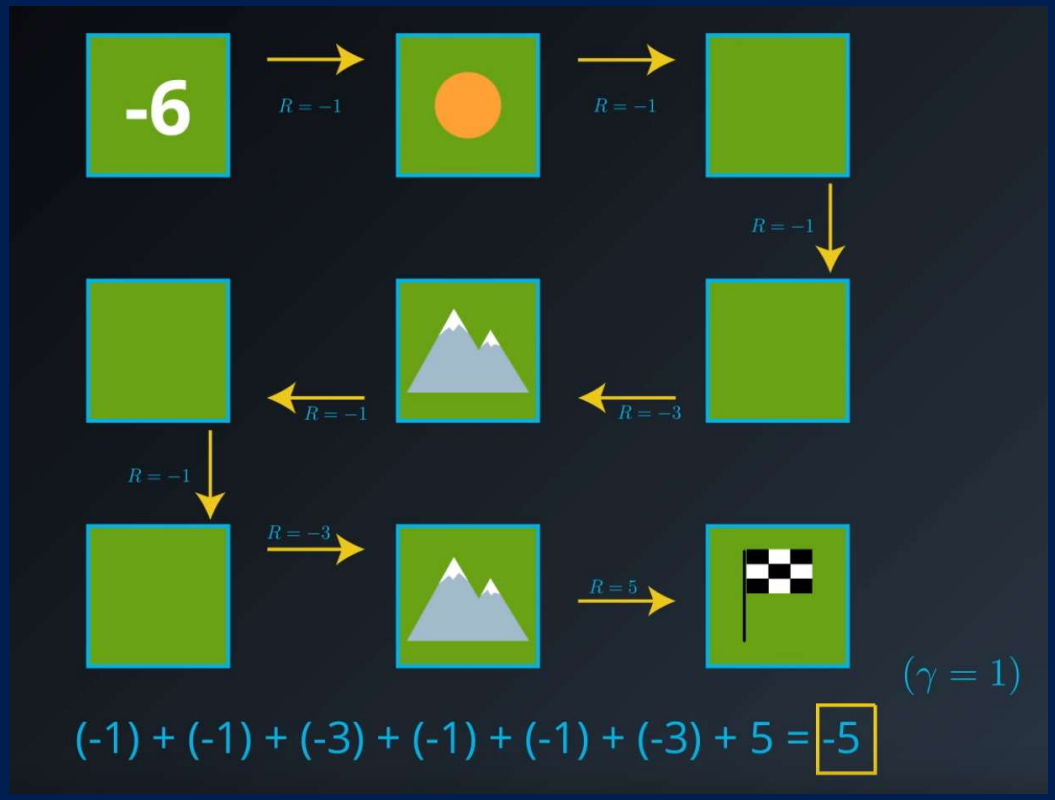
State – Value function – 1



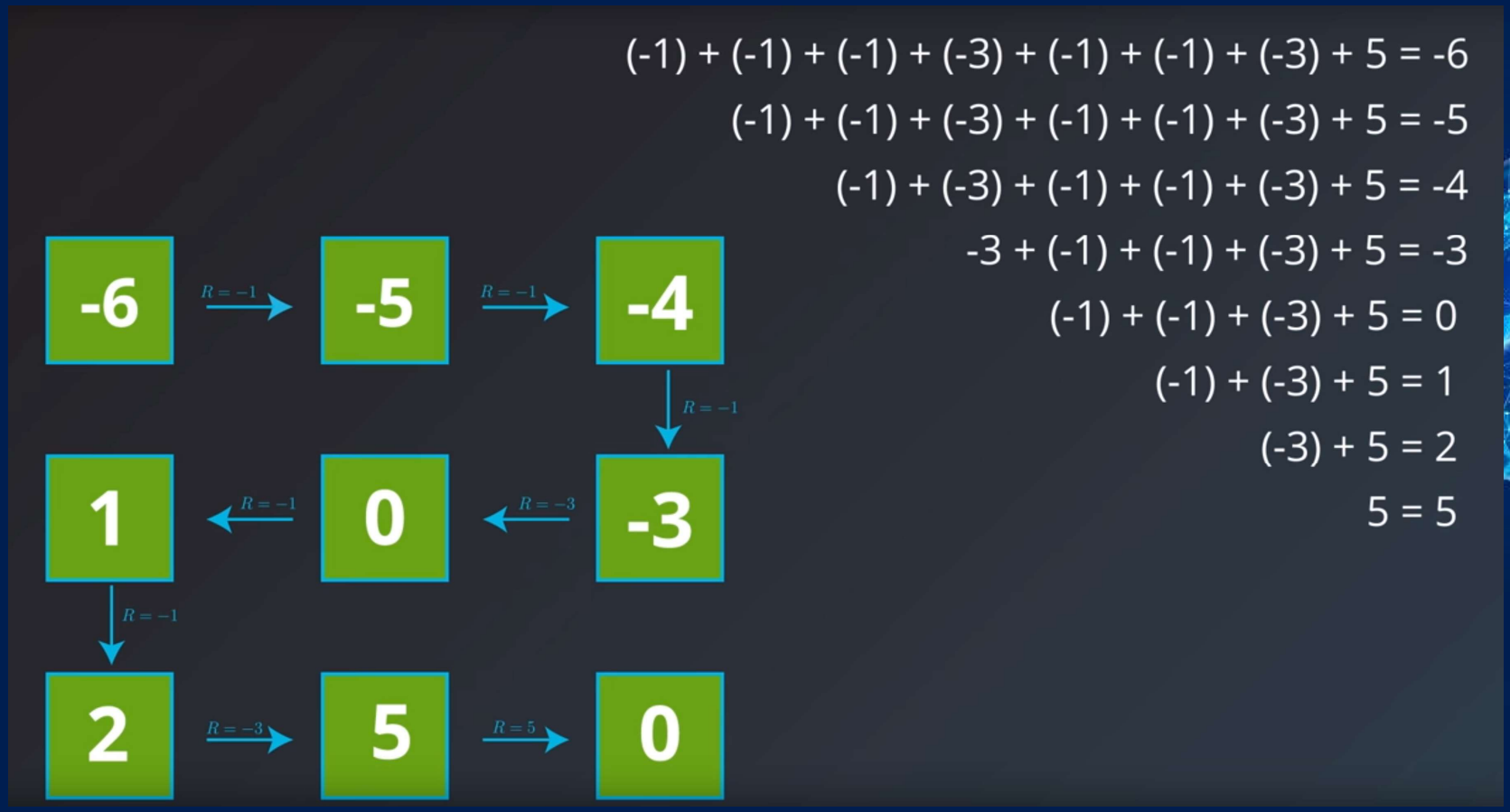
State – Value function – 2 – worst policy



State – Value function – 3 – worst policy



State – Value function – 4 – worst policy



(-1) + (-1) + (-1) + (-3) + (-1) + (-1) + (-3) + 5 = -6

(-1) + (-1) + (-3) + (-1) + (-1) + (-3) + 5 = -5

(-1) + (-3) + (-1) + (-1) + (-3) + 5 = -4

-3 + (-1) + (-1) + (-3) + 5 = -3

(-1) + (-1) + (-3) + 5 = 0

(-1) + (-3) + 5 = 1

(-3) + 5 = 2

5 = 5

State – Value function – 5

-6	-5	-4
1	0	-3
2	5	0

Definition

We call v_π the state-value function for policy π
The value of state s under a policy π is

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

For each **state s**
it yields the **expected return**
if the agent **starts in state s**
and then uses **the policy**
to choose its actions for all time steps

Bellman equation is applied to solve MDP

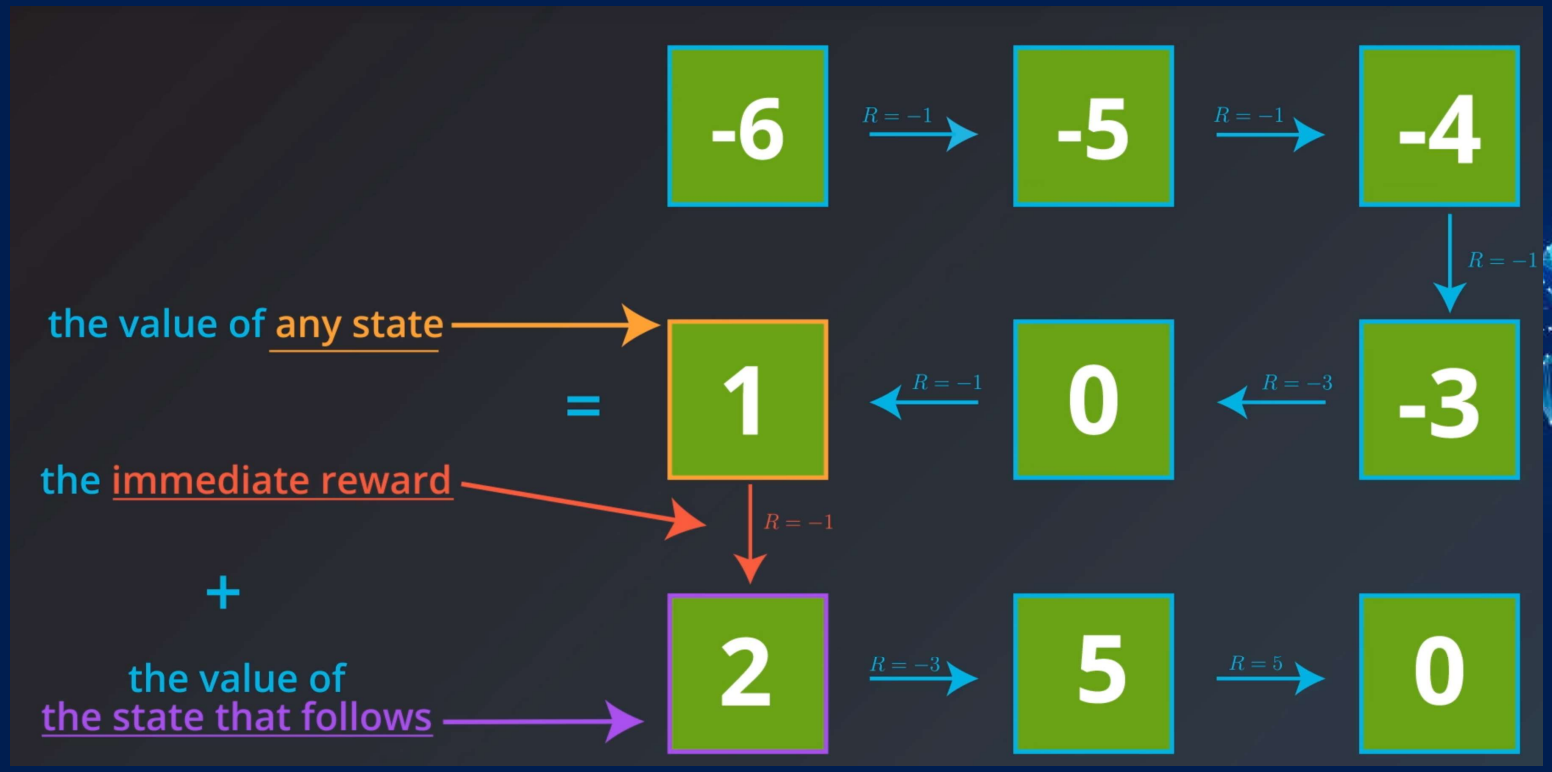
Richard Ernest Bellman (1920 – 1984)

Dynamic Programming (developed by Bellman) is a method for solving a complex problem by breaking it down into a collection of **simpler subproblems**.

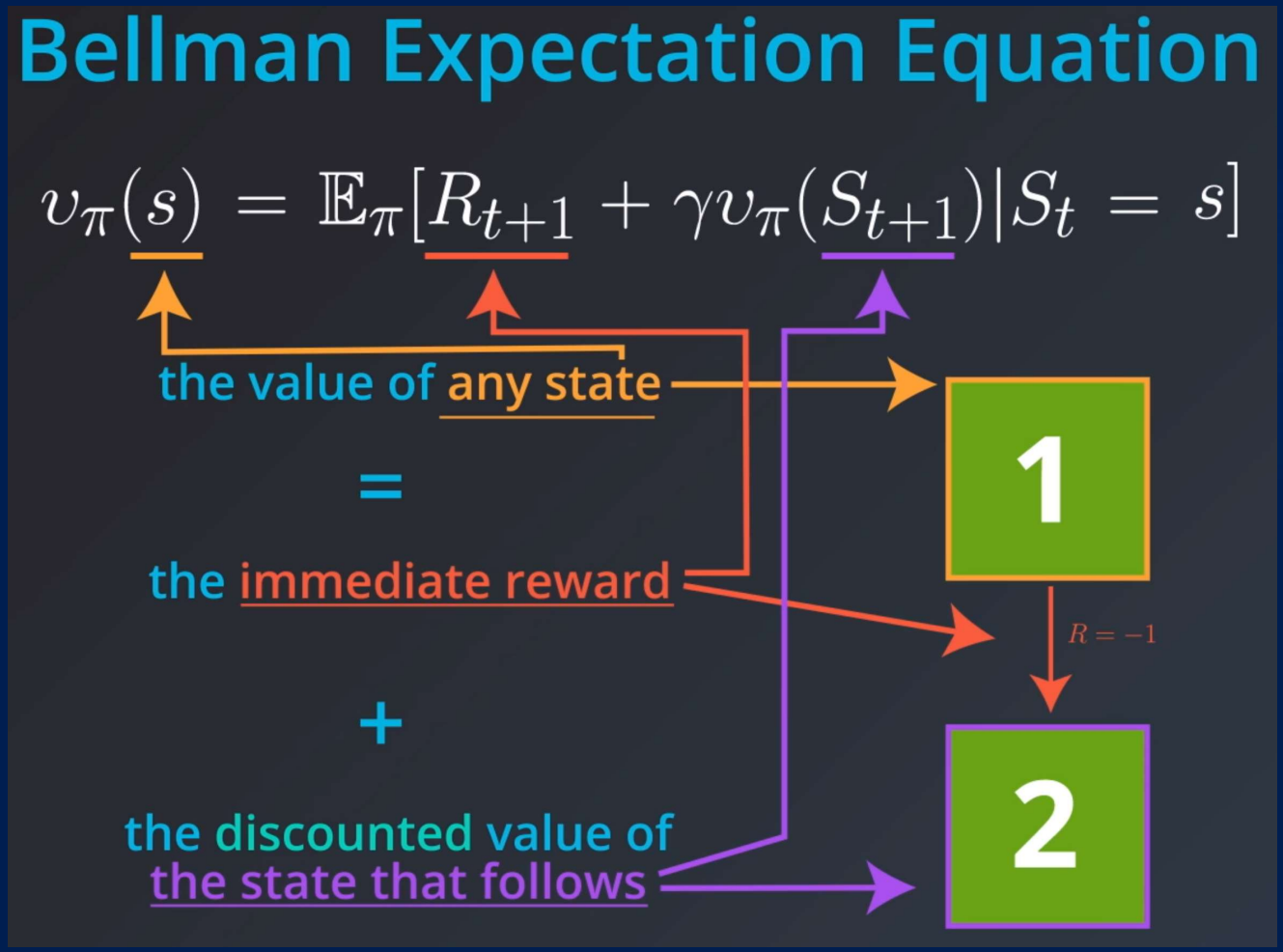
In RL computation of Gt (discounted reward) for whole episode can be very cumbersome therefore we can follow the Bellman concept and consider only **one action**.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots$$

Bellman equation



Bellman equation



Bellman equation

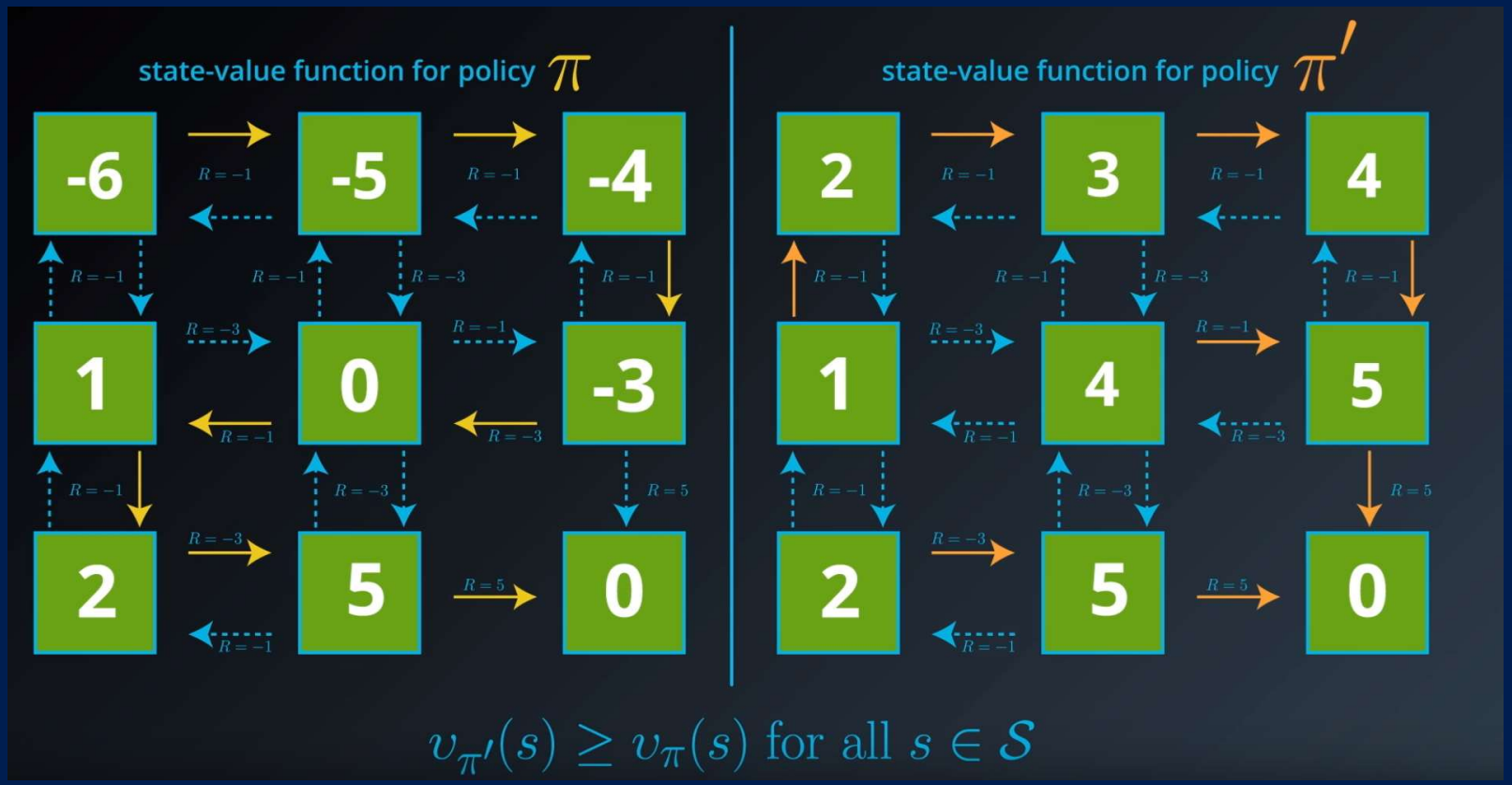
Bellman Expectation Equation

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

the value of any state s = the expected value of (the immediate reward + the discounted value of the state that follows)

if the agent starts in state s and then uses the policy to choose its actions for all time steps

Optimality


state-value function for policy π
-6
-5
-4
1
0
-3
2
5
0
state-value function for policy π'
2
3
4
1
4
5
2
5
0
$v_{\pi'}(s) \geq v_{\pi}(s)$ for all $s \in \mathcal{S}$

Optimality

Definition

$\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_{\pi}(s)$ for all $s \in \mathcal{S}$

(Note: It is often possible to find two policies that cannot be compared.)

Definition

An optimal policy π_* satisfies $\pi_* \geq \pi$ for all π

(Note: An optimal policy is guaranteed to exist, but may not be unique.)

The optimal state-value function is denoted v_*

The optimal action-value function is denoted q_*

What about more complicated MDP?

What about more complicated MDP?

Beside state value function we have to define action value function.

Action Value function

Definition

We call v_π the **state-value function for policy** π.

The value of state s under a policy π is

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

For each state s,
it yields the expected return
if the agent starts in state s
and then uses the policy
to choose its actions for all time steps.

Definition

We call q_π the **action-value function for policy** π.

The value of taking action a in state s under a policy π is

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

For each state s and action a
it yields the expected return
if the agent starts in state s
then chooses action a
and then uses the policy
to choose its actions for all time steps.

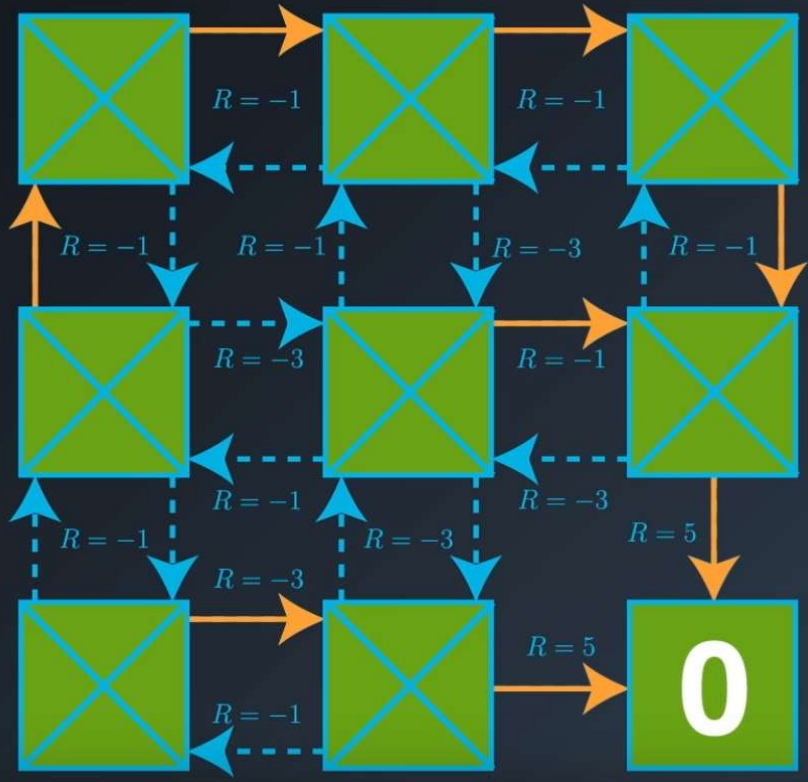
Action – Value function

For each state, the state-value function yields the expected return, if the agent starts in that state, and then follows the policy for all time steps.

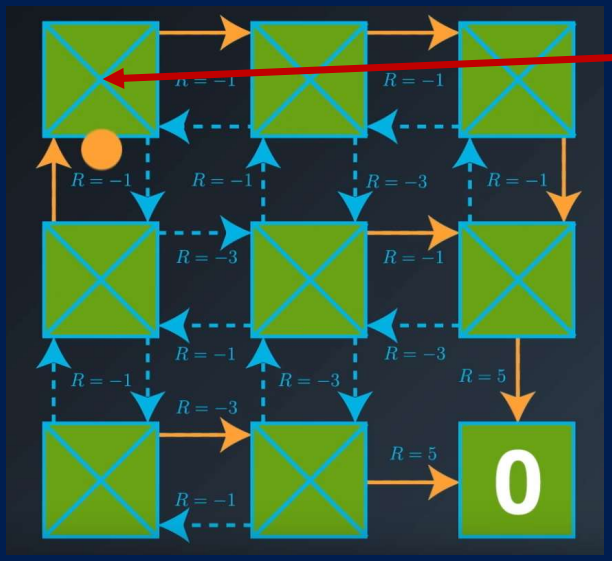
For each state and action, the action-value function yields the expected return, if the agent starts in that state, takes the action, and then follows the policy for all future time steps.

Action – Value function

For each state and action, the action-value function yields the expected return, if the agent starts in that state, takes the action, and then follows the policy for all future time steps.



Action – Value function

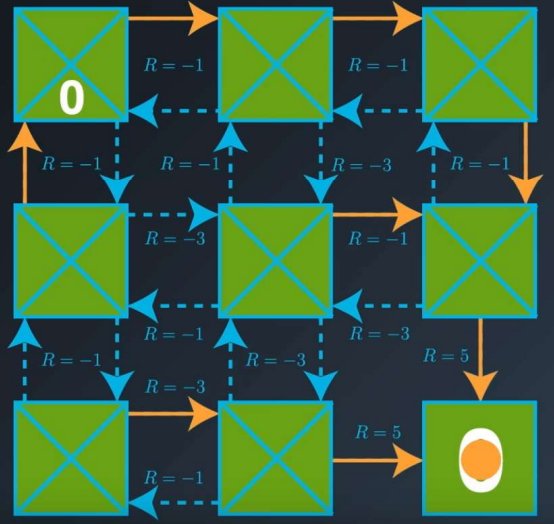


For each state and action, the action-value function yields the expected return, if the agent starts in that state, takes the action, and then follows the policy for all future time steps.

reward from action "down"

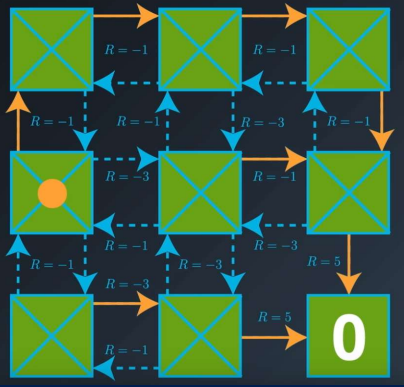
(-1) + (-1) + (-1) + (-1) + (-1) + 5 = 0

reward from following the policy for all future time steps

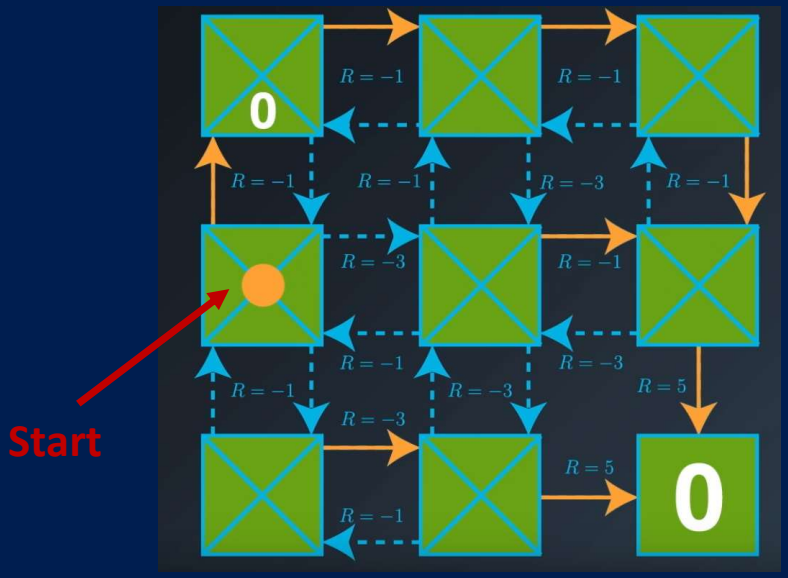


reward from action "down"

(-1)



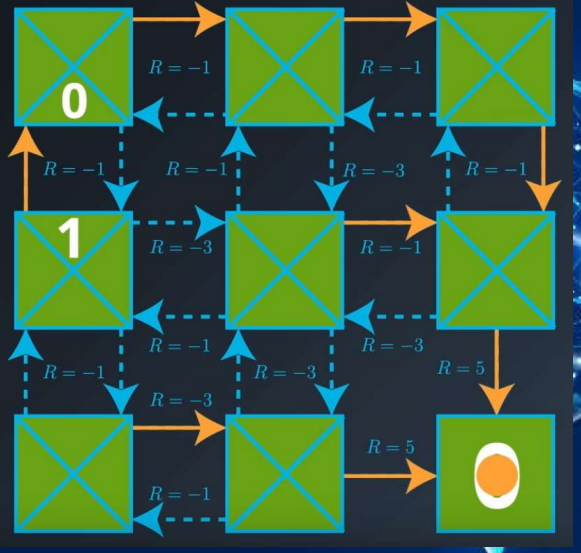
Action – Value function



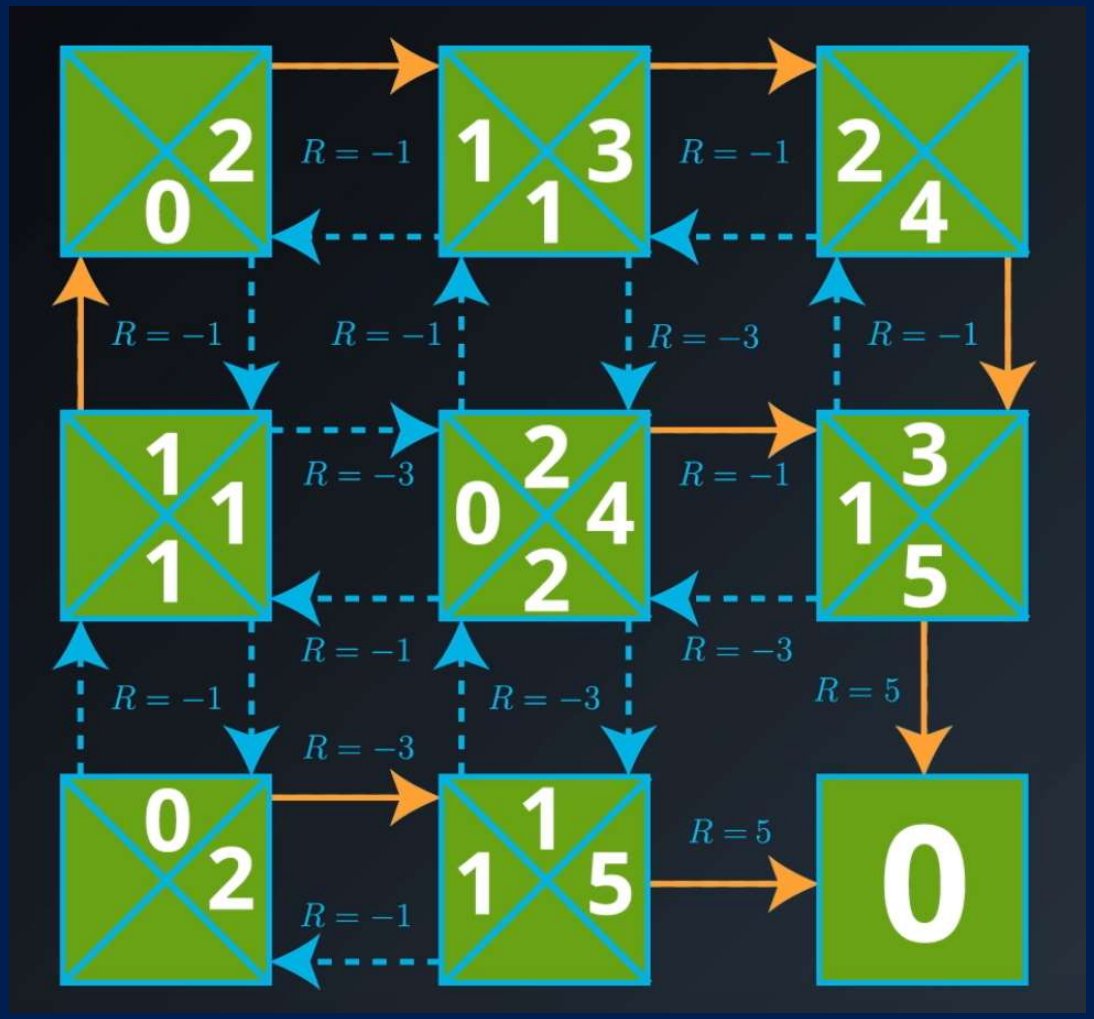
reward from
action “up”

(-1) + (-1) + (-1) + (-1) +5 = 1

reward from following the policy
for all future time steps

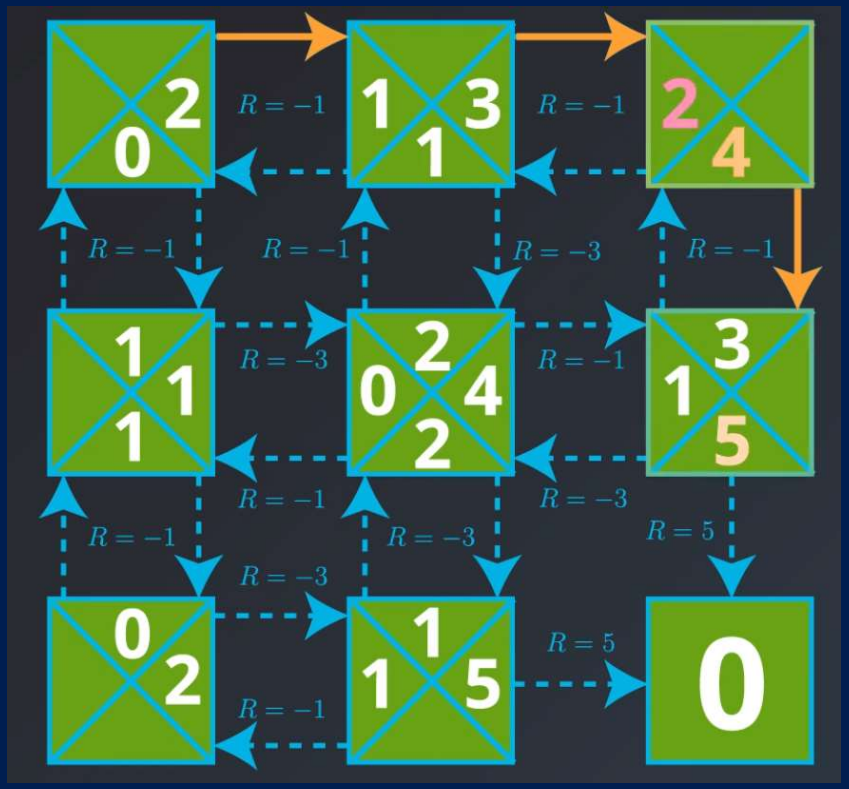


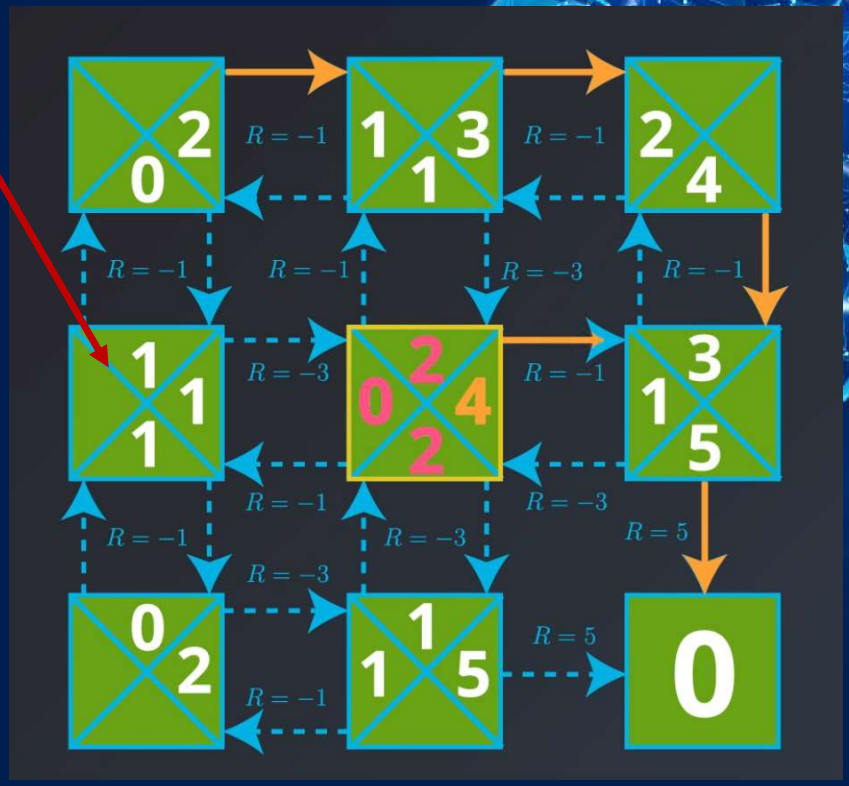
Action – Value function



Optimal policy

The Agent does not know what to do here. Sacrifice state.
All choices yield optimal policy.



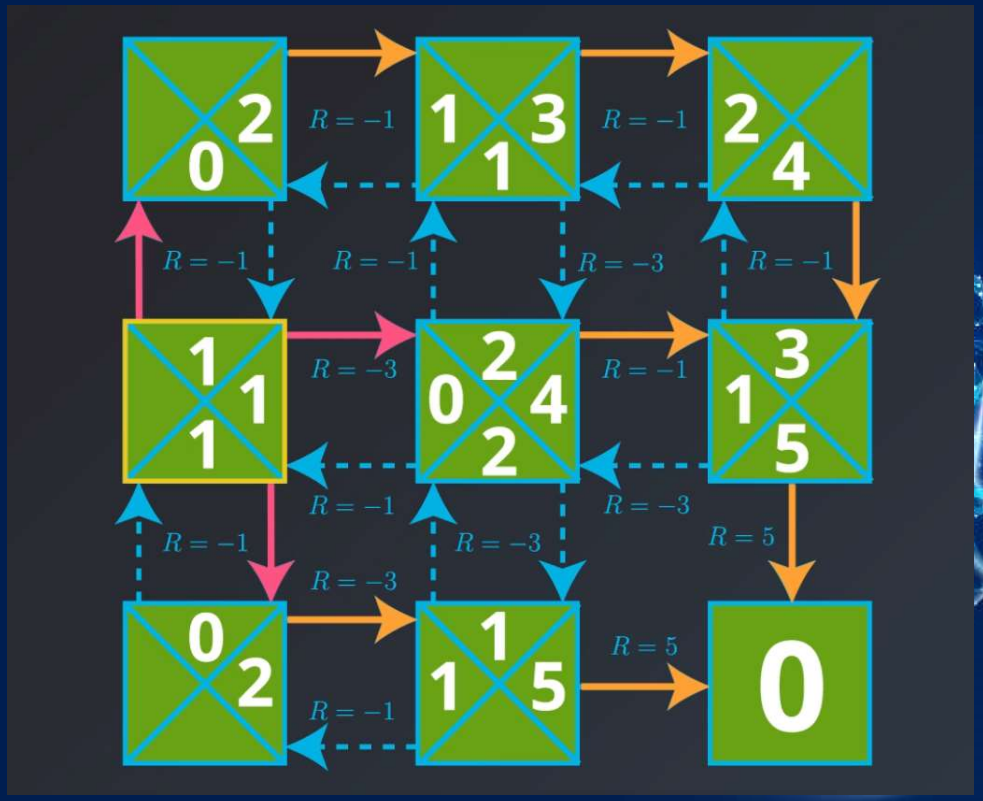


Optimal policy based on optimal action – value function

Agent interacts with environment.

The agent uses optimal action-value function to get optimal policy.

From these interaction the Agent estimates optimal action – value function.



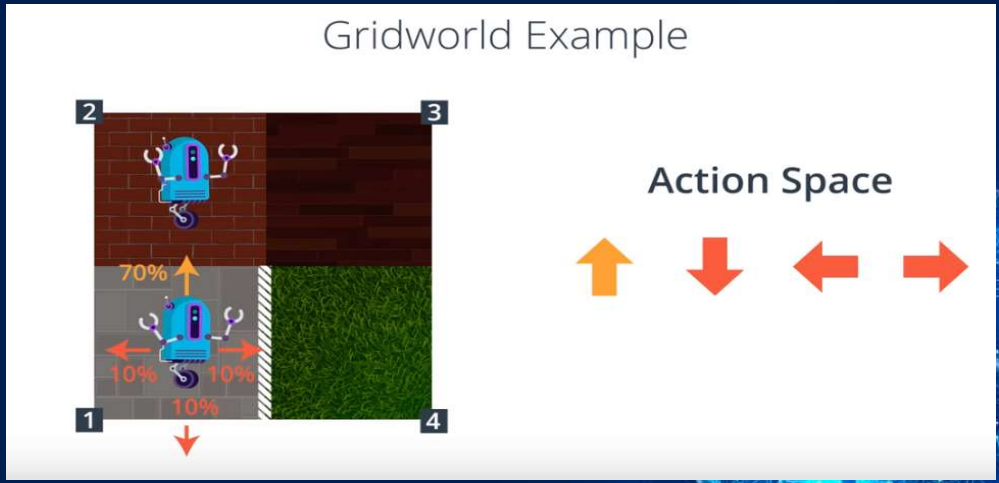
Interaction $\longrightarrow$ q_* $\longrightarrow$ π_*

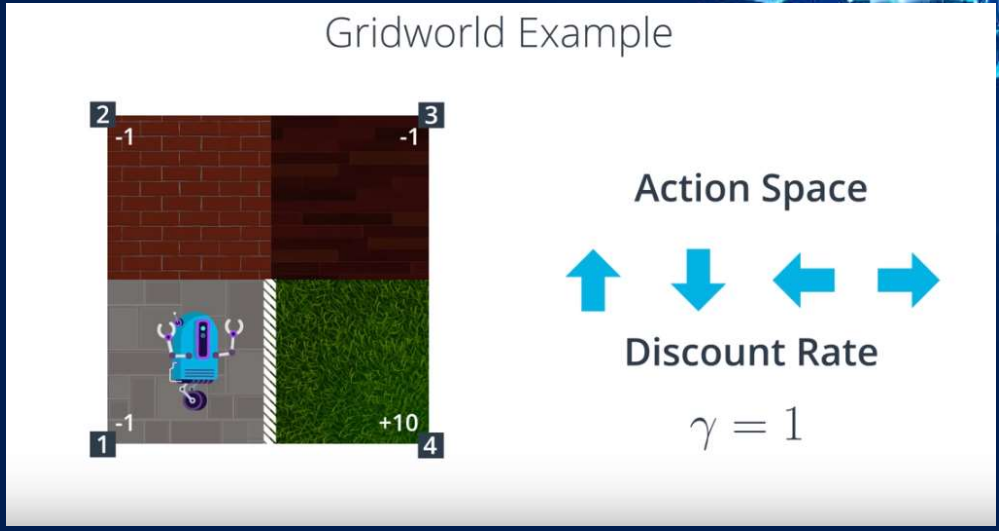
If the Agent has a optimal action-value function, so it can quickly obtain an optimal policy which is the solution to the MDP we are looking for.

Monte Carlo Methods. Q table approach

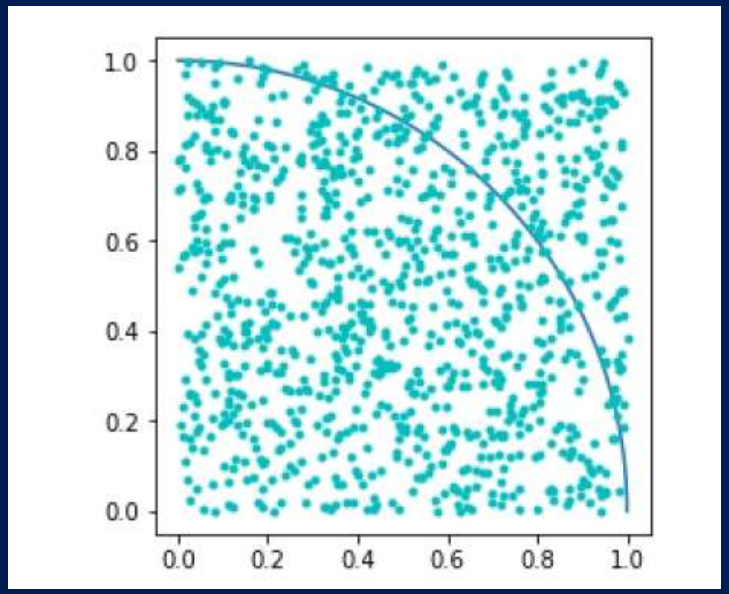
The Monte Carlo method finds approximate solutions through random sampling. (It is a statistical technique to find an approximate answer through sampling).

Drawback: We need to wait to end of episode to compute total reward and compute the action – value. We need in every step estimate if we are going to win or not





Monte Carlo Methods. Q table approach. General concept.

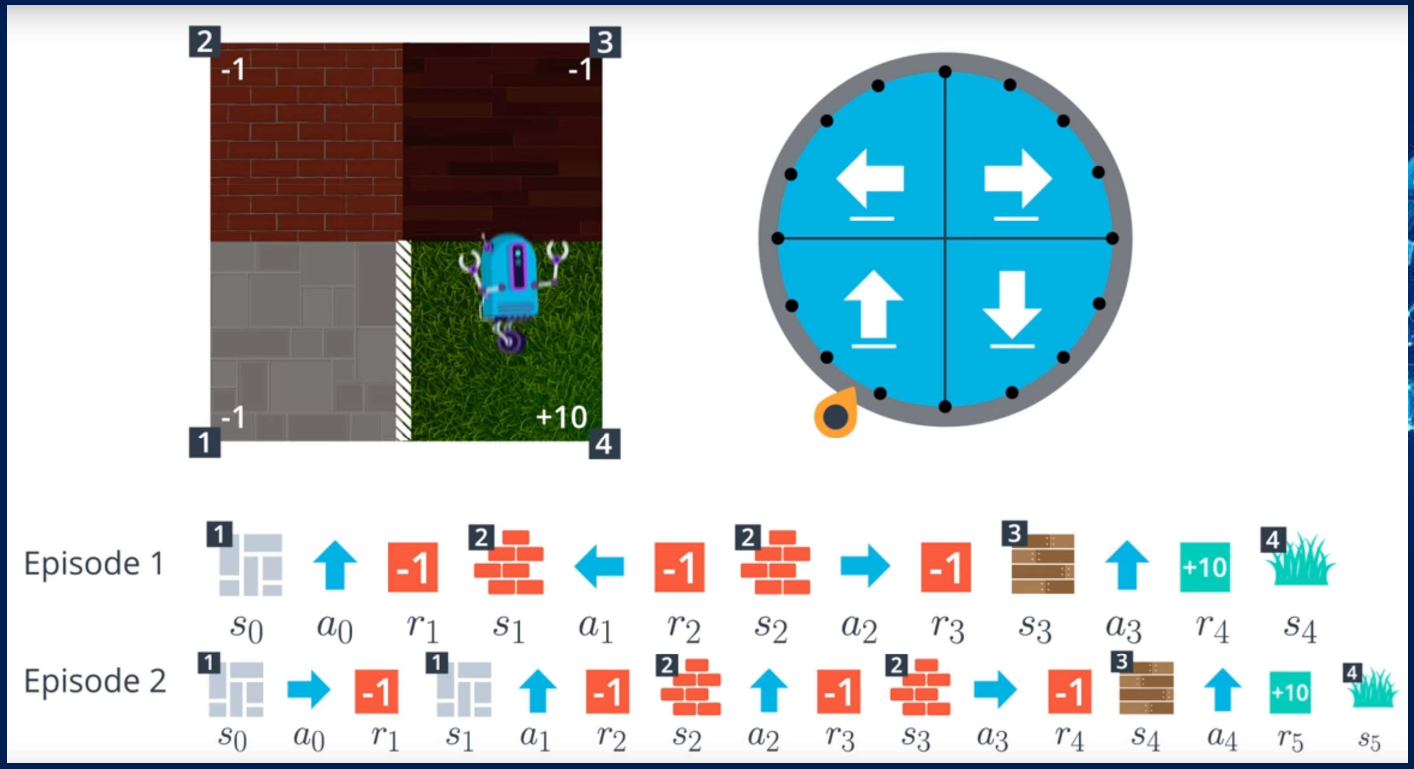


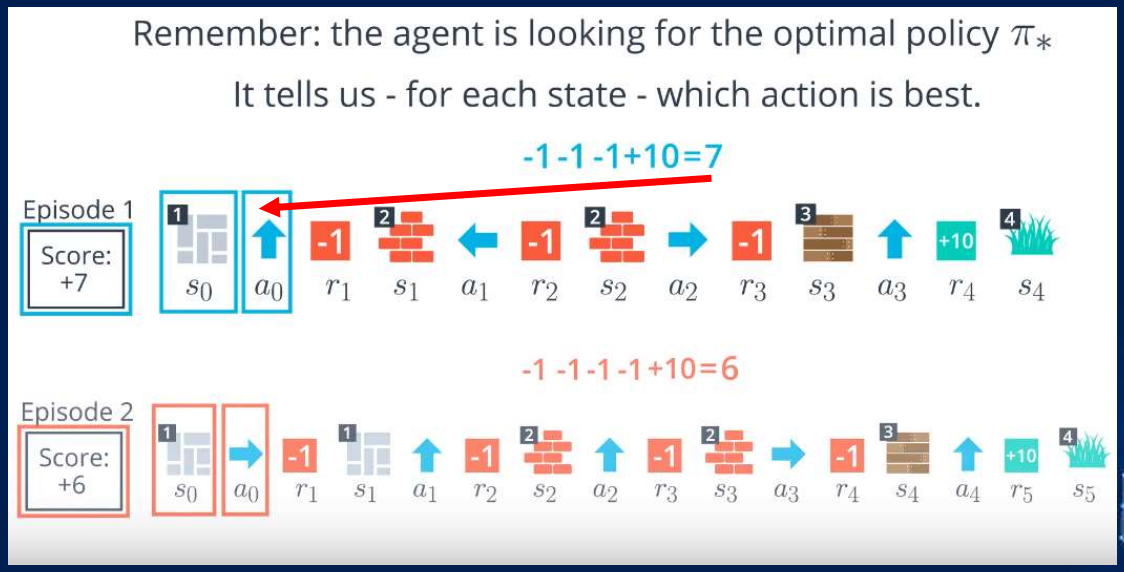
$$\frac{Area\ \ of\ \ a\ \ cirlce}{Area\ \ of\ \ a\ \ square} = \frac{No\ \ of\ \ points\ \ inside\ \ the\ \ circle}{No\ \ of\ \ points\ \ inside\ \ the\ \ square}$$

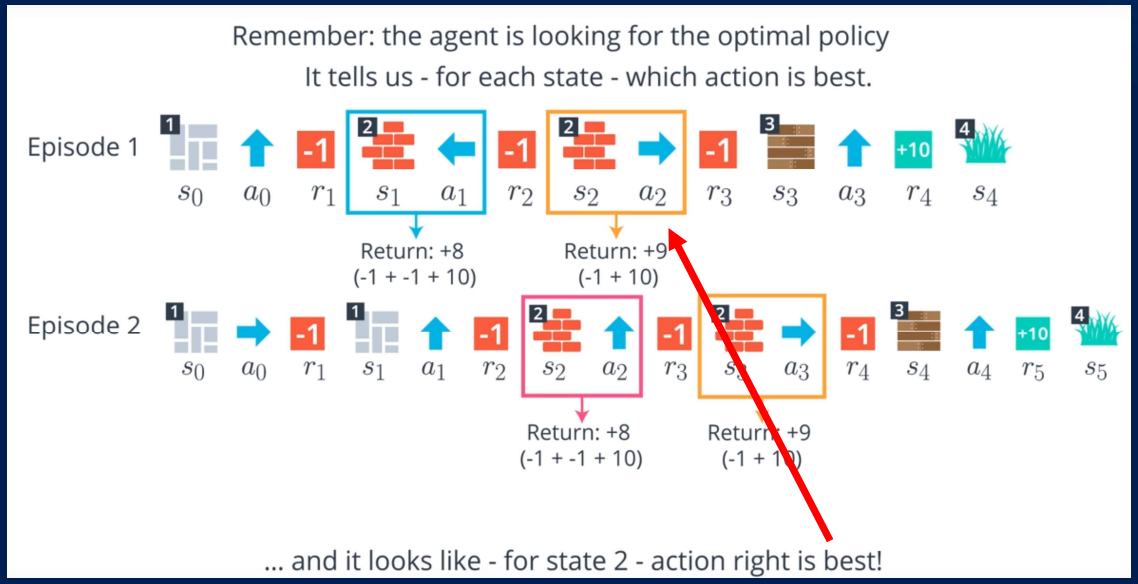
$$\frac{\pi r^2}{a^2} = \frac{No\ \ of\ \ points\ \ inside\ \ the\ \ circle}{No\ \ of\ \ points\ \ inside\ \ the\ \ square}$$

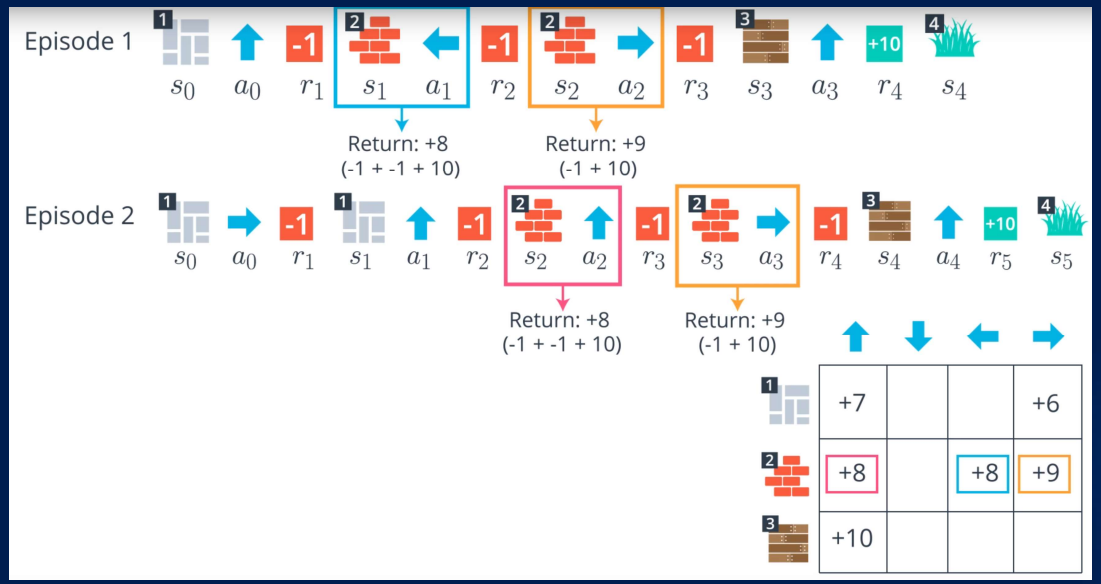
$$\frac{\pi(\frac{1}{2})^2}{1^2} = \frac{No\ \ of\ \ points\ \ inside\ \ the\ \ circle}{No\ \ of\ \ points\ \ inside\ \ the\ \ square}$$

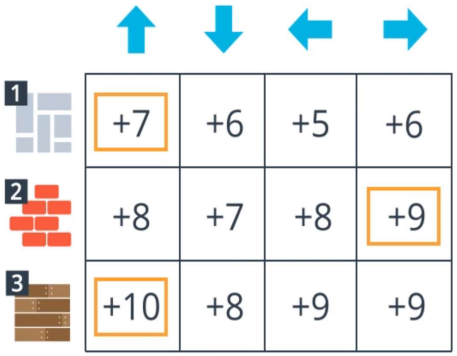
$$\pi = 4 * \frac{No\ \ of\ \ points\ \ inside\ \ the\ \ circle}{No\ \ of\ \ points\ \ inside\ \ the\ \ square}$$


2
-1
3
-1
-1
+10
1
4
Episode 1
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4
Episode 2
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4 a_4 r_5 s_5


Remember: the agent is looking for the optimal policy π_*
It tells us - for each state - which action is best.
-1 -1 -1 +10 = 7
Episode 1
Score: +7
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4
-1 -1 -1 -1 +10 = 6
Episode 2
Score: +6
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4 a_4 r_5 s_5


Remember: the agent is looking for the optimal policy
It tells us - for each state - which action is best.
Episode 1
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4
Return: +8
(-1 + -1 + 10)
Return: +9
(-1 + 10)
Episode 2
s_0 a_0 r_1 s_1 a_1 r_2 s_2 a_2 r_3 s_3 a_3 r_4 s_4 a_4 r_5 s_5
Return: +8
(-1 + -1 + 10)
Return: +9
(-1 + 10)
... and it looks like - for state 2 - action right is best!





For each state - which action is best?

$\pi'($  $) =$ ↑

$\pi'($  $) =$ →

$\pi'($  $) =$ ↑

Bad Policy π Better Policy π' Optimal Policy π_*

$\pi(a|s) = \frac{1}{4}$ for all s, a

π'(1) = ↑ π_*(1) = ↑

π'(2) = → π_*(2) = →

π'(3) = ↑ π_*(3) = ↓

Corresponding to the same state-action pair

Episode Number	N = 1	N = 2	N = 3	N = 4
Return	G = 2	G = 8	G = 11	G = 3
Estimated Action Value	Q = 2	Q = 5	Q = 7	Q = 6

$$Q \leftarrow Q + \frac{1}{N}(G - Q)$$

$$7 + \frac{1}{4}(3 - 7)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \underbrace{(G_t - Q(S_t, A_t))}_{:= \delta_t}$$

If $\delta_t > 0$, increase $Q(S_t, A_t)$.

If $\delta_t < 0$, decrease $Q(S_t, A_t)$.

(by an amount proportional to $\frac{1}{N(S_t, A_t)}$)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{(G_t - Q(S_t, A_t))}_{:= \delta_t}$$

If $\delta_t > 0$, increase $Q(S_t, A_t)$.

If $\delta_t < 0$, decrease $Q(S_t, A_t)$.

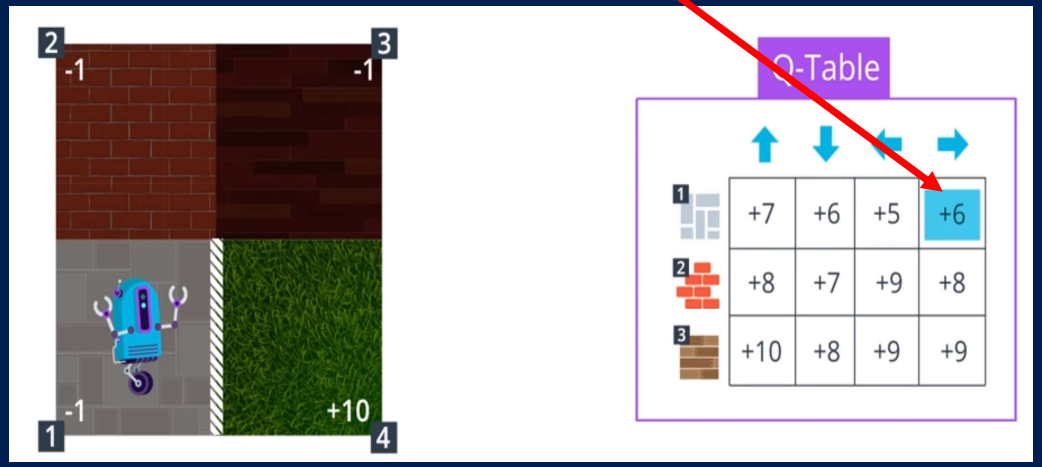
(by an amount proportional to α)

Temporal Difference (TD) methods

will instead update the Q-table after **every time step**.

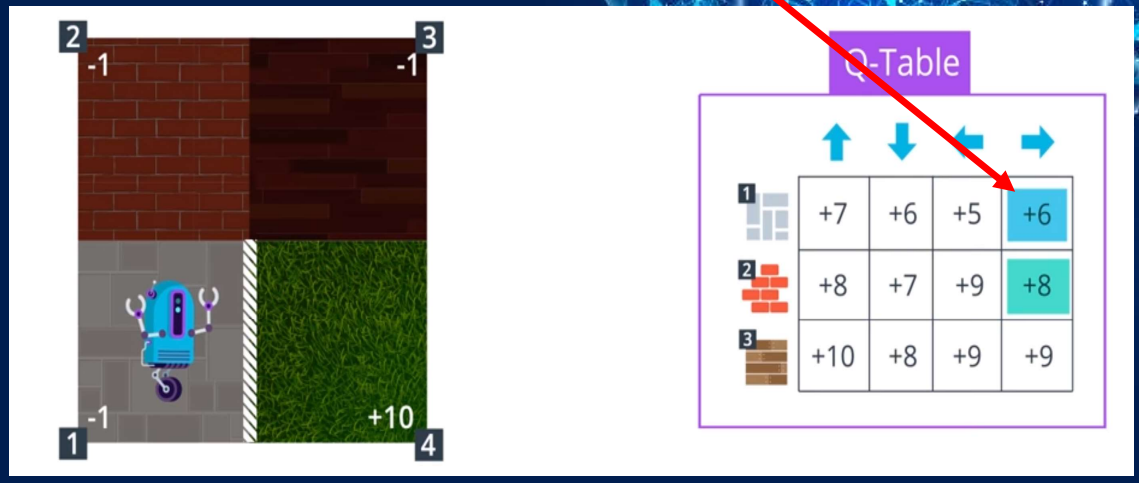
Monte Carlo

Current estimate: +6; state 1; action RIGHT
We update the Q table after the episode is finished = the Agent has reached the goal (state 4)
We calculate total reward 10 -1 – 1 = 8
New Q for state 1 => 6 + 0.1(8 – 6) = 6.2



TD

Current estimate: +6; state 1; action RIGHT
We update the Q table every step-
New Q for state 1 => 6 + 0.1(-1 +8 - 6) = 6.1



(From Monte Carlo Control)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots$$

alternative estimate

current estimate

(From Temporal-Difference Control)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

alternative estimate

current estimate

Sarsamax (aka Q-Learning)

Initialize $Q(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

$\pi \leftarrow \epsilon$-greedy(Q)

A_2

$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1 \quad R_2 \quad S_2$

$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha(R_1 + \gamma \max_{a \in \mathcal{A}} Q(S_1, a) - Q(S_0, A_0))$$

$\pi \leftarrow \epsilon$-greedy(Q)

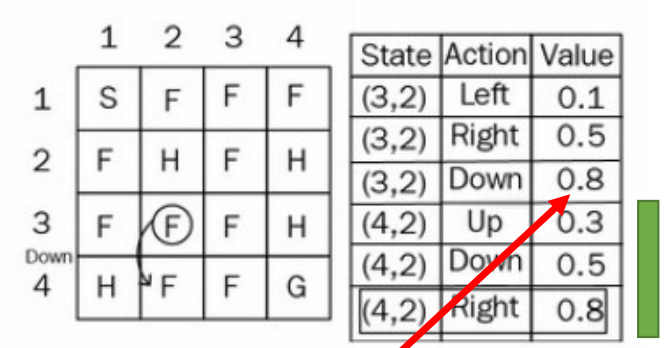
$$Q(S_1, A_1) \leftarrow Q(S_1, A_1) + \alpha(R_2 + \gamma \max_{a \in \mathcal{A}} Q(S_2, a) - Q(S_1, A_1))$$

$\pi \leftarrow \epsilon$-greedy(Q)

1. First, we initialize the Q function to some arbitrary values
2. We take an action from a state using epsilon-greedy policy ($\epsilon > 0$) and move it to the new state
3. We update the Q value of a previous state by following the update rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max Q(s'a) - Q(s,a))$$

4. We repeat the steps 2 and 3 till we reach the terminal state



$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max Q(s'a) - Q(s,a))$$

Q((3,2) down) = Q((3,2), down) + 0.1 (0.3 + 1 max [Q((4,2) action)]- Q((3,2), down)

Q((3,2), down) = 0.8 + 0.1 (0.3 + 1 max [0.3, 0.5, 0.8] - 0.8)

= 0.8 + 0.1 (0.3 + 1 (0.8) - 0.8)

= 0.83

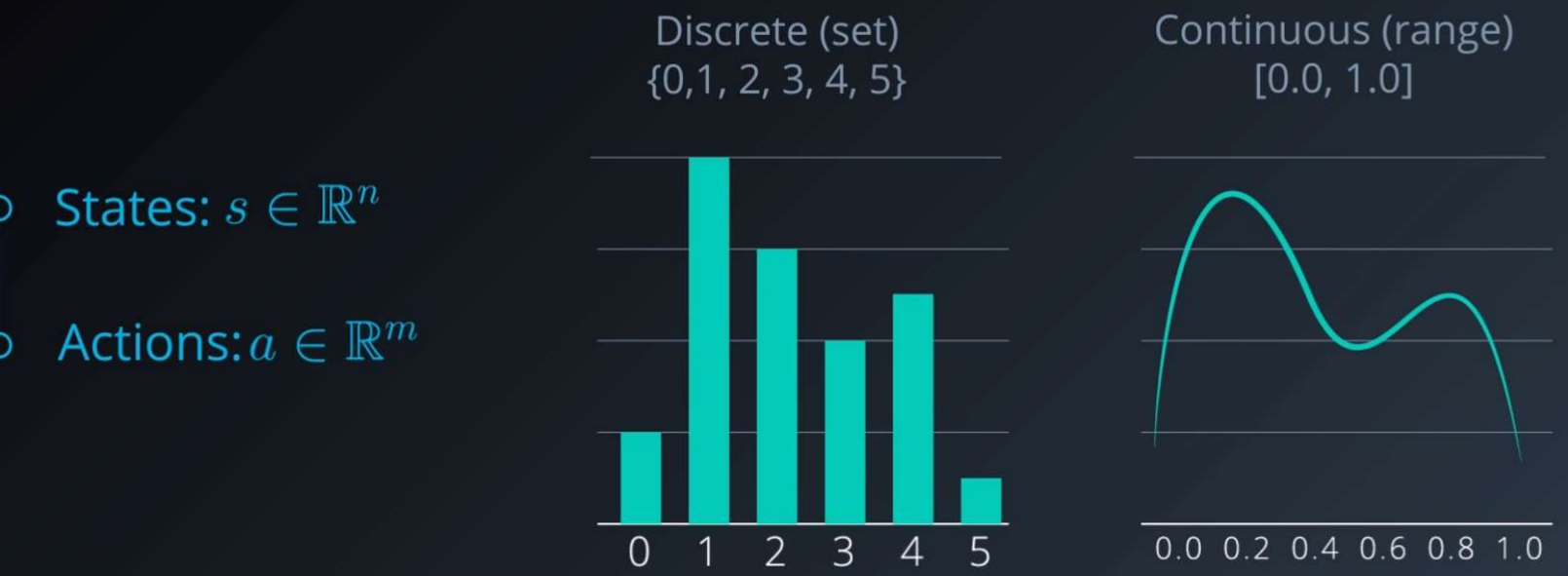
Recap

Reinforcement Learning: Framework

- Markov Decision Process (MDP): $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
  - State Transition and Reward Model: $\mathbb{P}(S_{t+1}, R_{t+1} | S_t, A_t)$
  - State Value Function: $V(S)$
  - Action Value Function: $Q(S,A)$
- Goal: Find optimal policy π^* that maximizes total *expected* reward

Continuous Spaces

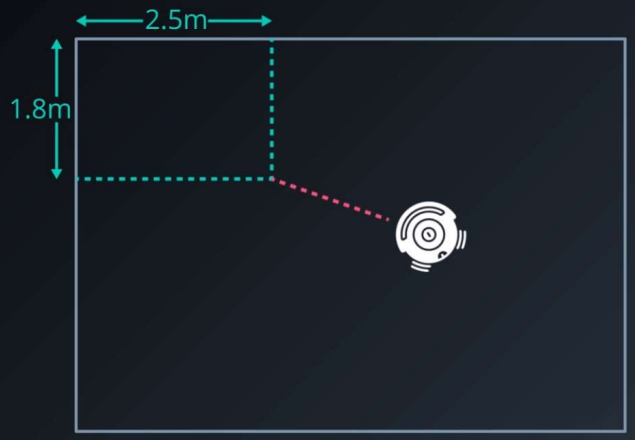
- States: $s \in \mathbb{R}^n$
- Actions: $a \in \mathbb{R}^m$



Why Continuous?

Vacuum cleaner world

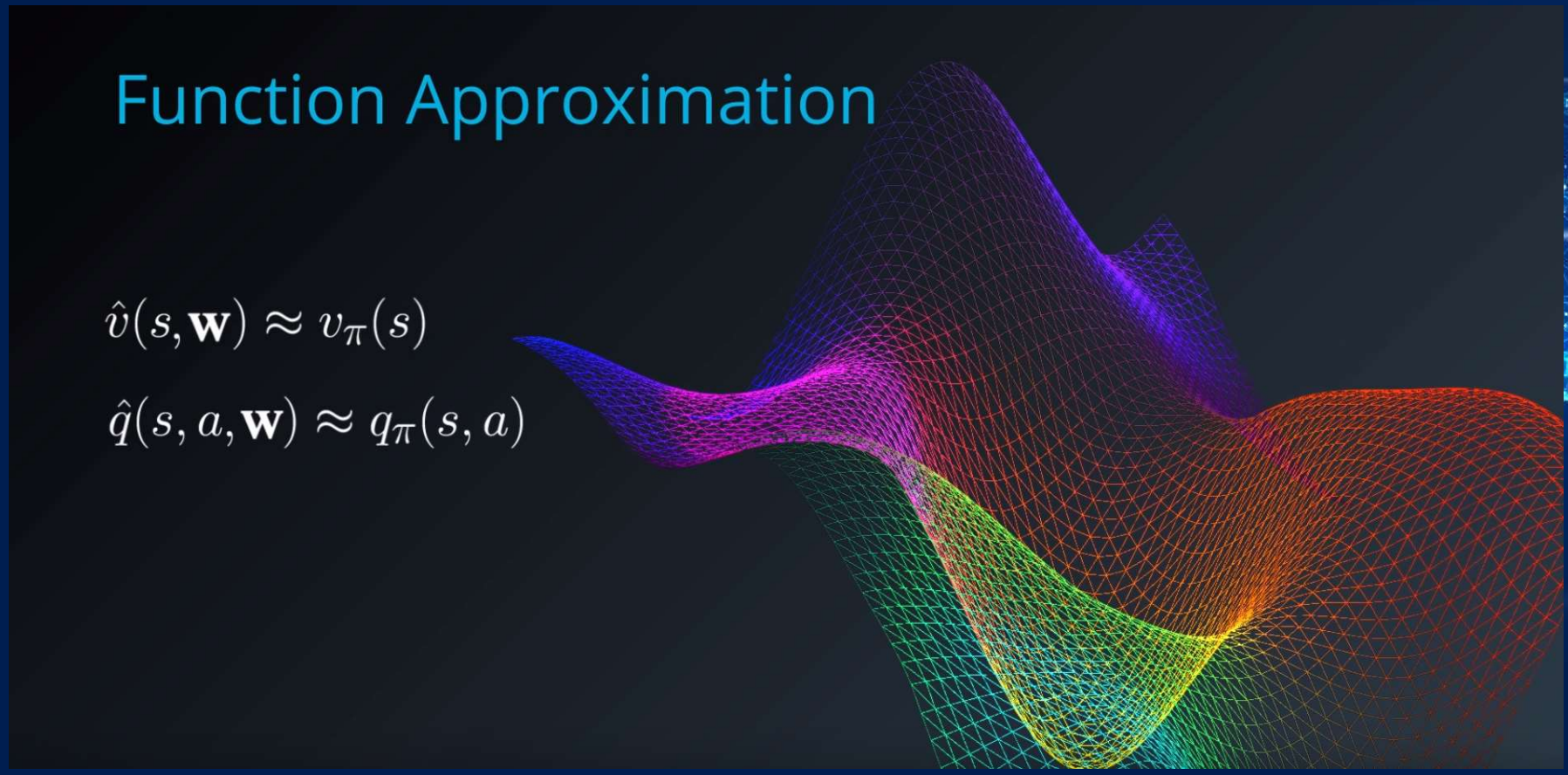
Why Continuous?



Vacuum cleaner world

Continuous Actions

For a continuous spaces, capturing each **state is infeasible**!!!
Therefore we have to apply the function approximation (we still do not the real value).
We introduce the vector **w** to find desired approximation.



Feature Vector

We need a scalar

state

parameters

$$\hat{v}(s,\mathbf{w}) = ?$$

value function

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

feature vector

Dot Product!

$$\hat{v}(s,\mathbf{w}) = \mathbf{x}(s)^\top \cdot \mathbf{w}$$

Dot Product!

$$\hat{v}(s,\mathbf{w}) = \begin{pmatrix} x_1(s) \cdots x_n(s) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

$$= x_1(s) \cdot w_1 + \cdots + x_n(s) \cdot w_n$$

$$= \sum_{j=1}^{n} x_j(s) \cdot w_j$$

Gradient Descent

derivative $w.r.t.$ $\mathbf{w}$

$\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$

- Value Function: $\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \cdot \mathbf{w}$
- Minimize Error: $J(\mathbf{w}) = \mathbb{E}_\pi \left[\left(v_\pi(s) - \mathbf{x}(s)^\top \mathbf{w} \right)^2 \right]$
- Error Gradient: $\nabla_{\mathbf{w}} J(\mathbf{w}) = -2 \left(v_\pi(s) - \mathbf{x}(s)^\top \mathbf{w} \right) \mathbf{x}(s)$
- Update Rule: $\Delta \mathbf{w} = -\alpha \frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w})$

$= \alpha \left(v_\pi(s) - \mathbf{x}(s)^\top \mathbf{w} \right) \mathbf{x}(s)$ Cancel out -2

Intuition

$\Delta \mathbf{w} =$

change weights

α

small step

$\left(v_\pi(s) - \mathbf{x}(s)^\top \mathbf{w} \right)$

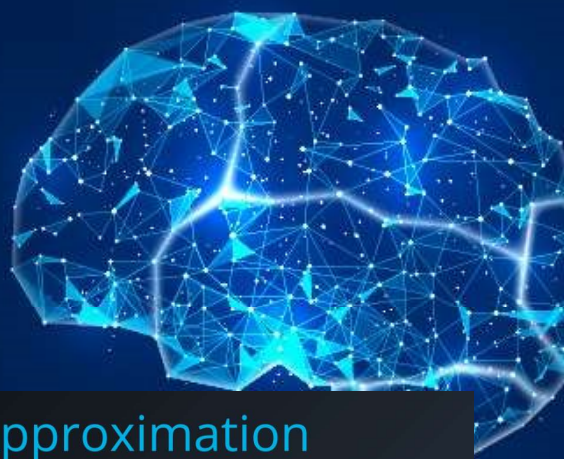
away from error

$\mathbf{x}(s)$

direction

Dot Product!

$$\hat{v}(s,\mathbf{w}) = \mathbf{x}(s)^\top \cdot \mathbf{w}$$



Non-Linear Function Approximation

activation function

$$\hat{v}(s,\mathbf{w}) = f\Big(\mathbf{x}(s)^\top \cdot \mathbf{w}\Big)$$

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

Non-Linear Function Approximation

$$\hat{v}(s,\mathbf{w}) = f\Big(\mathbf{x}(s)^\top \cdot \mathbf{w}\Big)$$

$$\Delta\mathbf{w} = \alpha\Big(v_\pi(s) - \hat{v}(s,\mathbf{w})\Big)\,\nabla_{\mathbf{w}}\,\hat{v}(s,\mathbf{w})$$

gradient descent update rule

Inspirations.

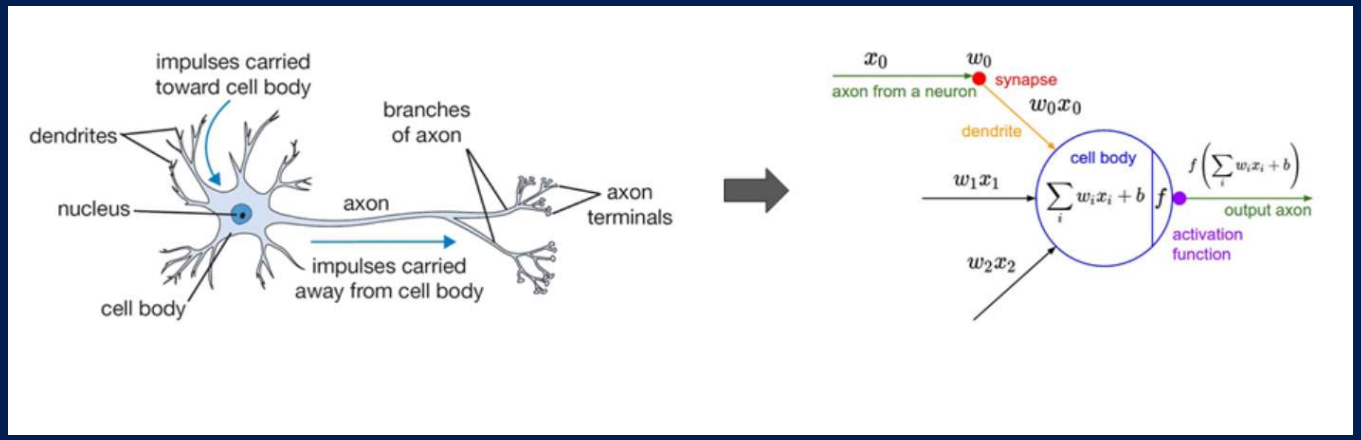
https://www.youtube.com/watch?v=3JQ3hYko51Y

https://www.youtube.com/watch?v=JlTEA0VevSY

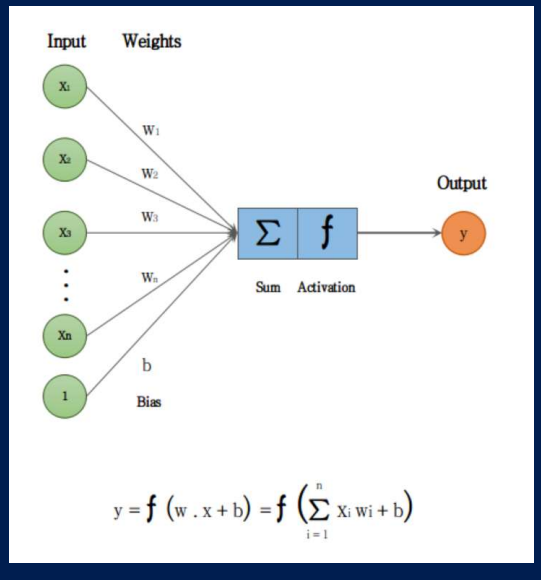
https://www.youtube.com/watch?v=GWL1HNHDSq4

https://www.youtube.com/watch?v=OoC8oH0CLGc

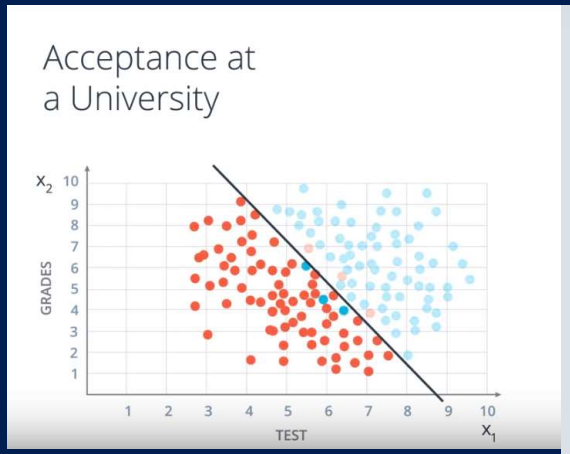
Artificial Brain



Brain neurons have a number of **dendrites** (**inputs**), a **cell nucleus**(**processor**) and an **axon** (**output**).



Why do we need neural network?



BOUNDARY:
A LINE
$2x_1 + x_2 - 18 = 0$

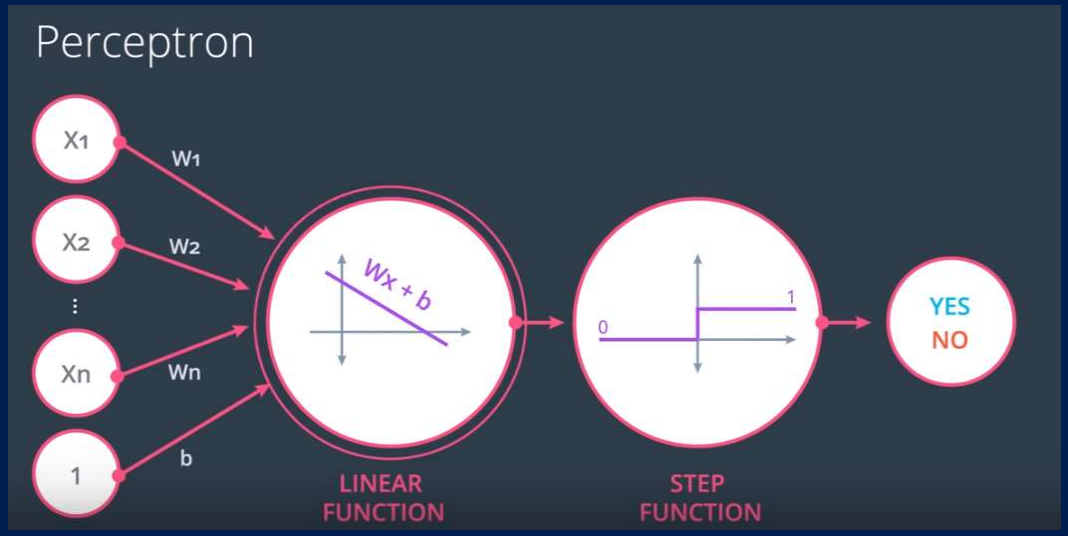
Score =
2*Test + Grades - 18

PREDICTION:
Score > 0: Accept
Score < 0: Reject

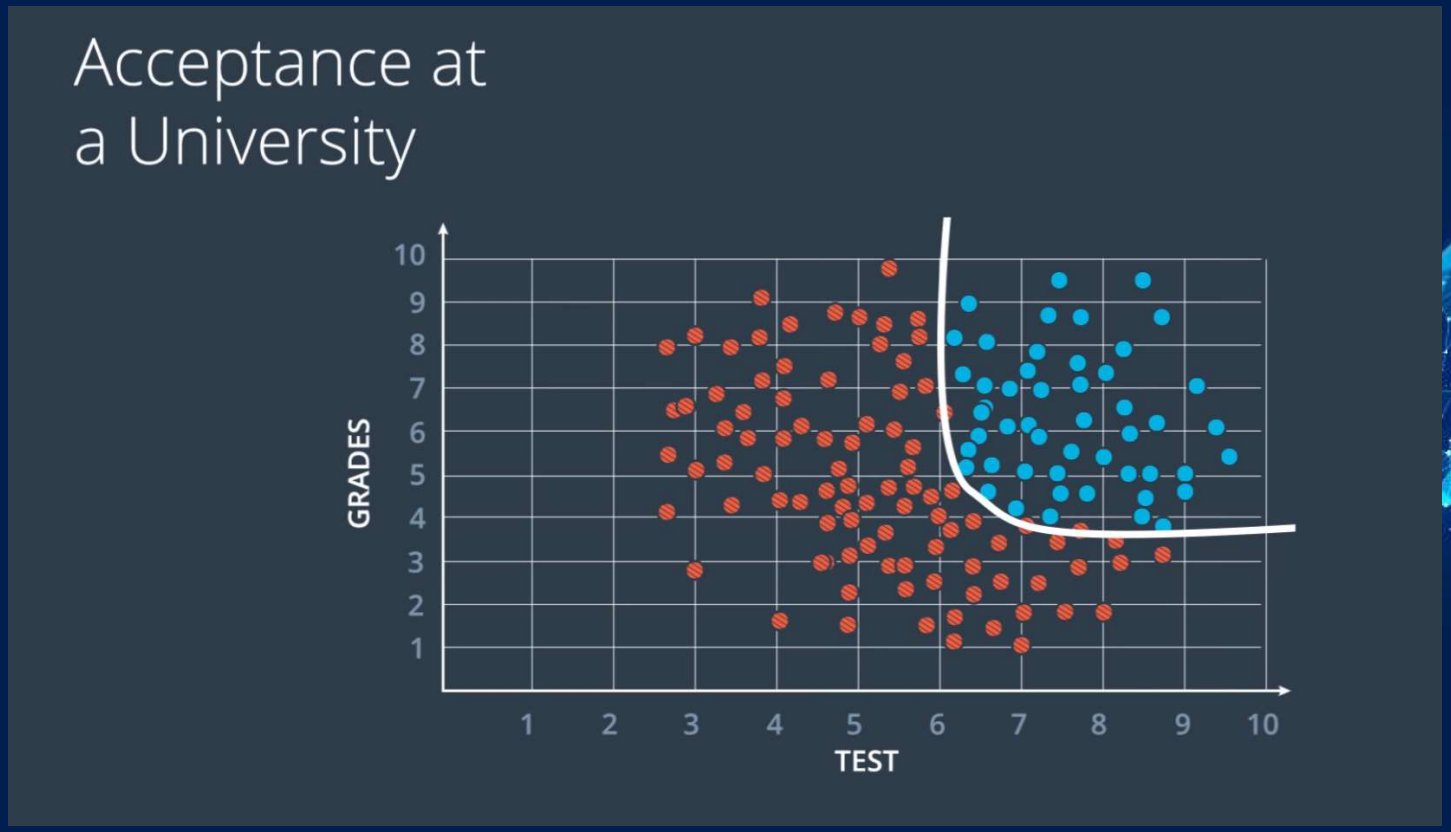
BOUNDARY:
A LINE
$w_1x_1 + w_2x_2 + b = 0$
$Wx + b = 0$
$W = (w_1, w_2)$
$x = (x_1, x_2)$
y = label: 0 or 1

PREDICTION:

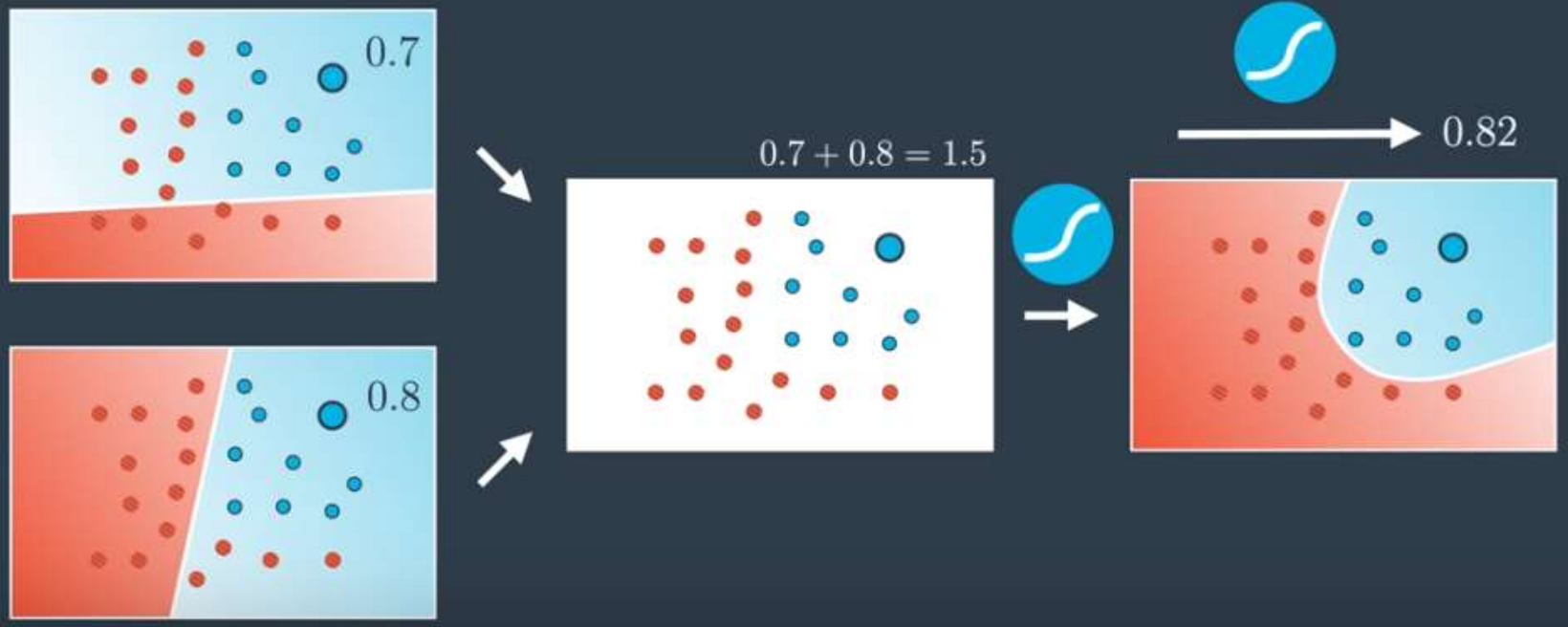
$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$



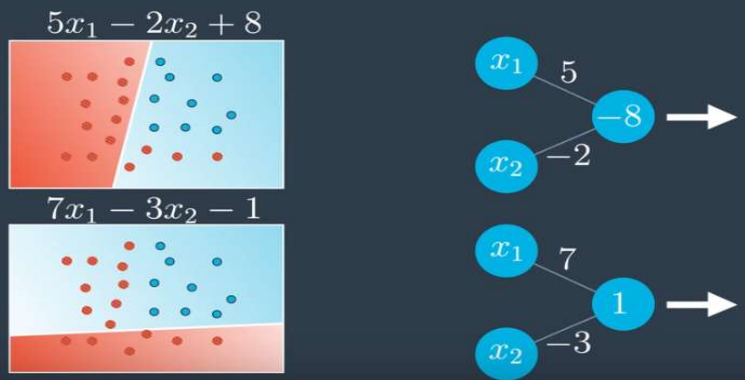
How to divide two classes?



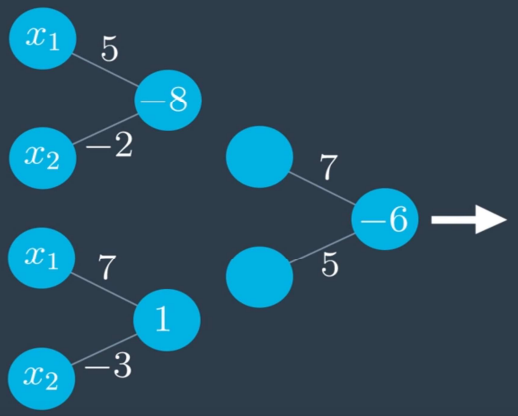
Neural Network

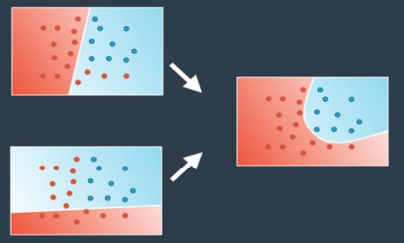


Neural Network

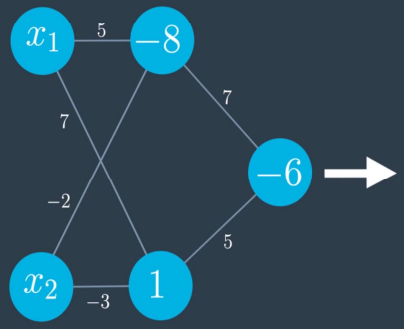


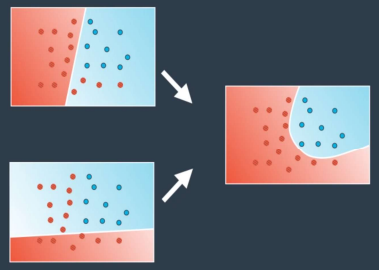
Neural Network



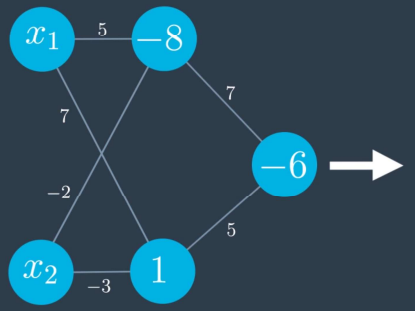


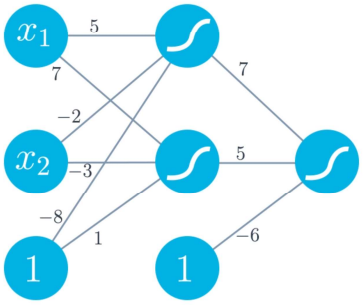
Neural Network

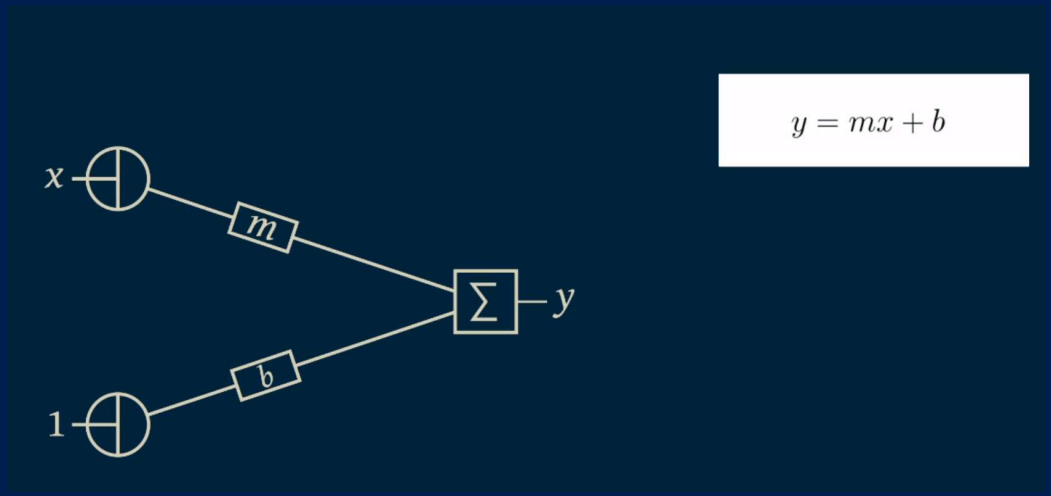




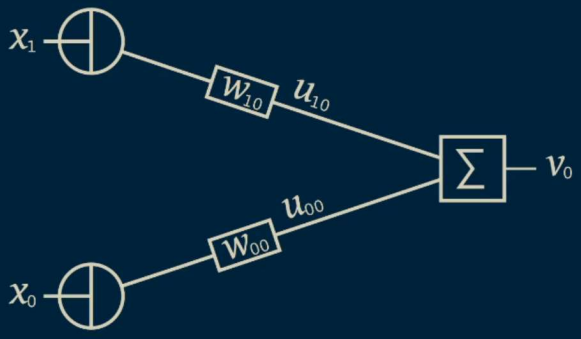
Neural Network







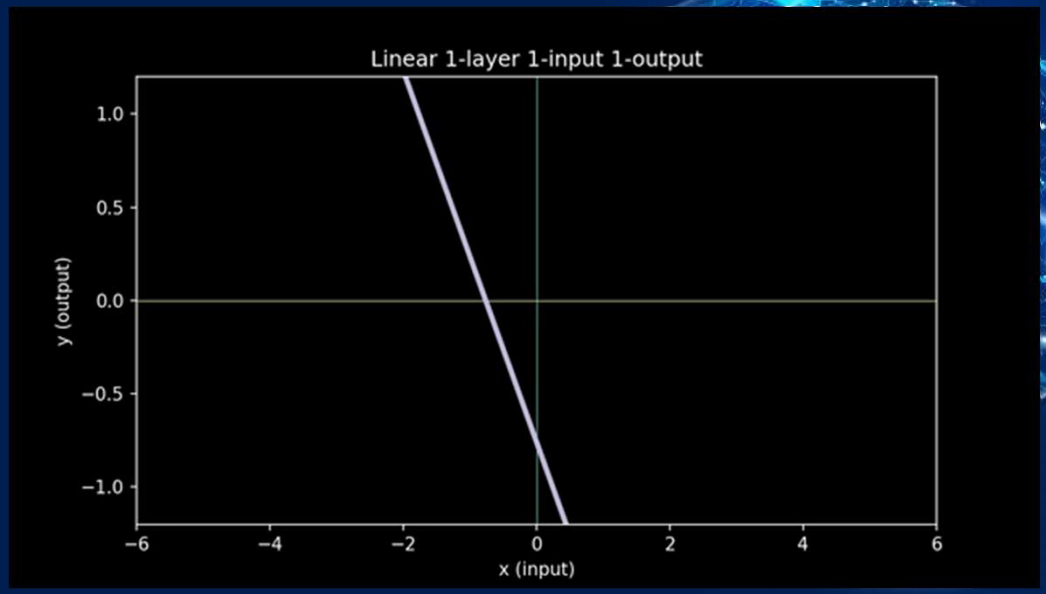
$$y = mx + b$$



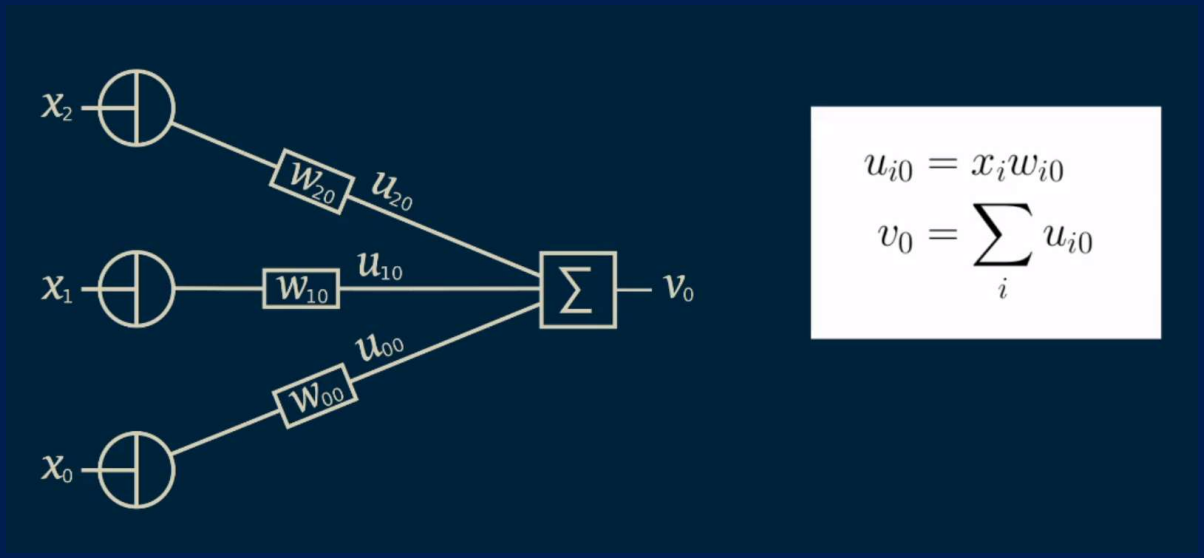
$$y = mx + b$$

$$u_{00} = x_0 w_{00}$$
$$u_{10} = x_1 w_{10}$$
$$v_0 = u_{00} + u_{10}$$

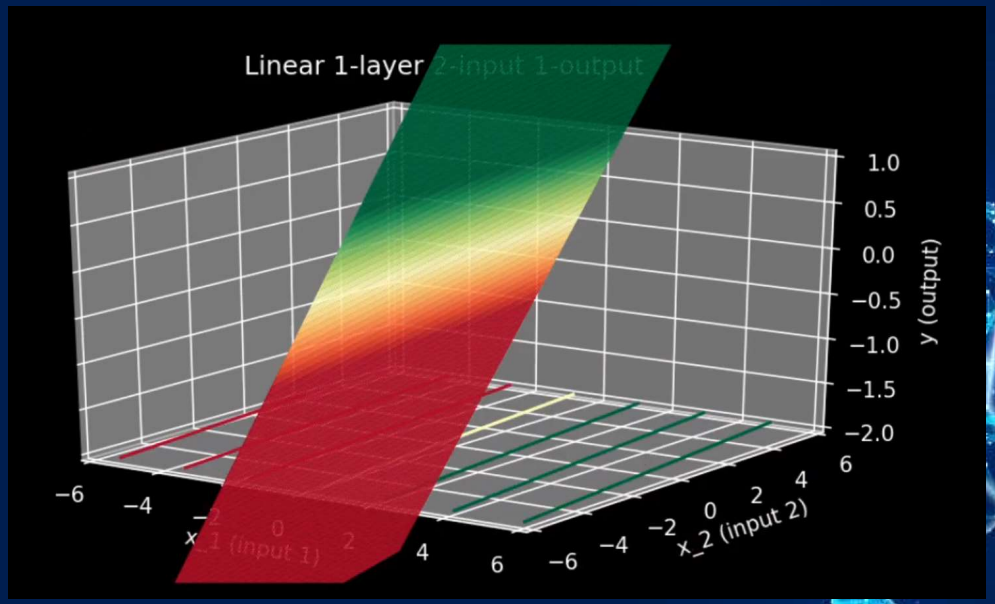
$$x_0 = 1$$
$$x_1 = x$$
$$w_{00} = b$$
$$w_{10} = m$$
$$v_0 = y$$



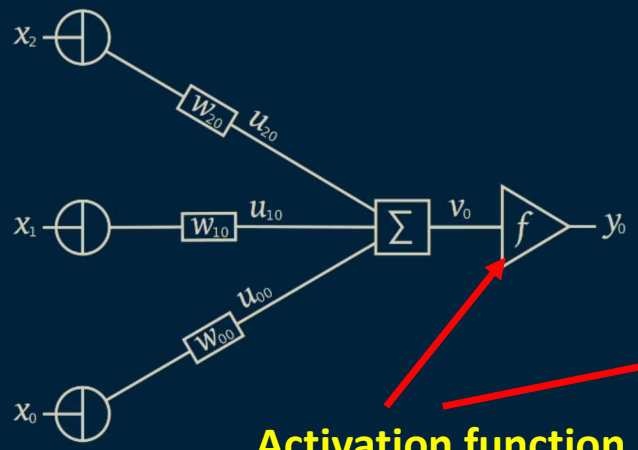
Linear function (classification)



$$u_{i0} = x_i w_{i0}$$
$$v_0 = \sum_i u_{i0}$$

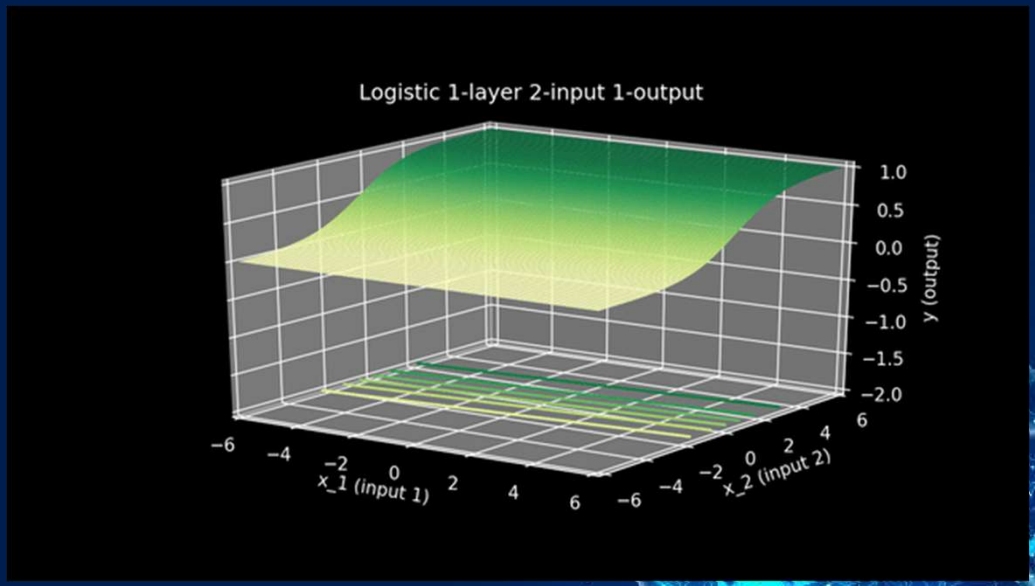


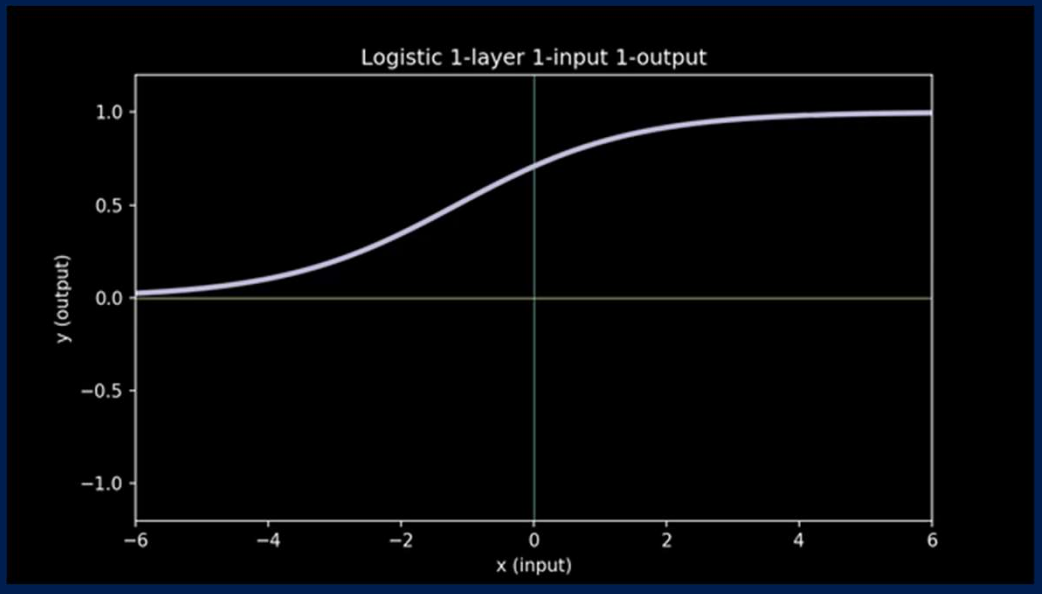
Non – linear function (classification)

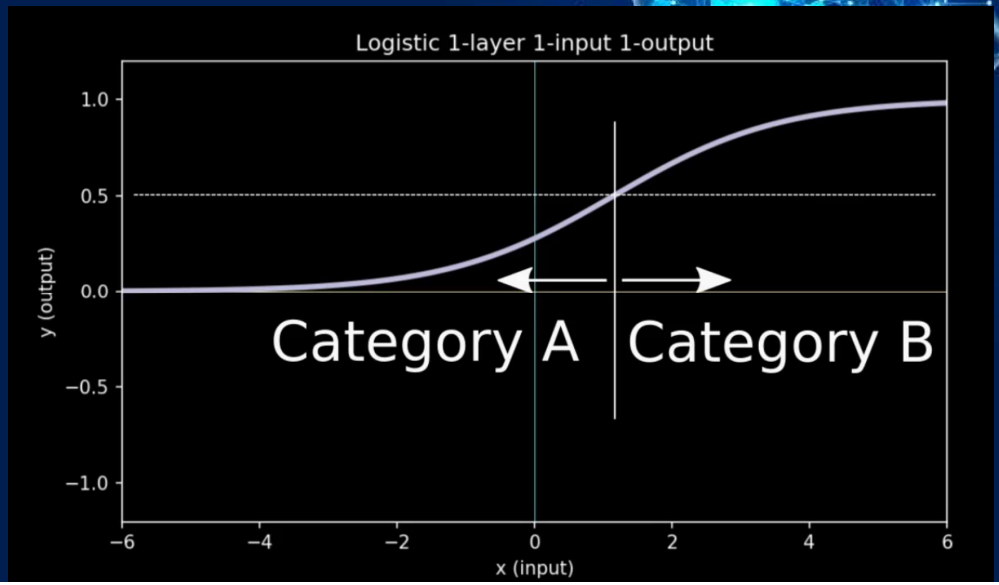


Activation function

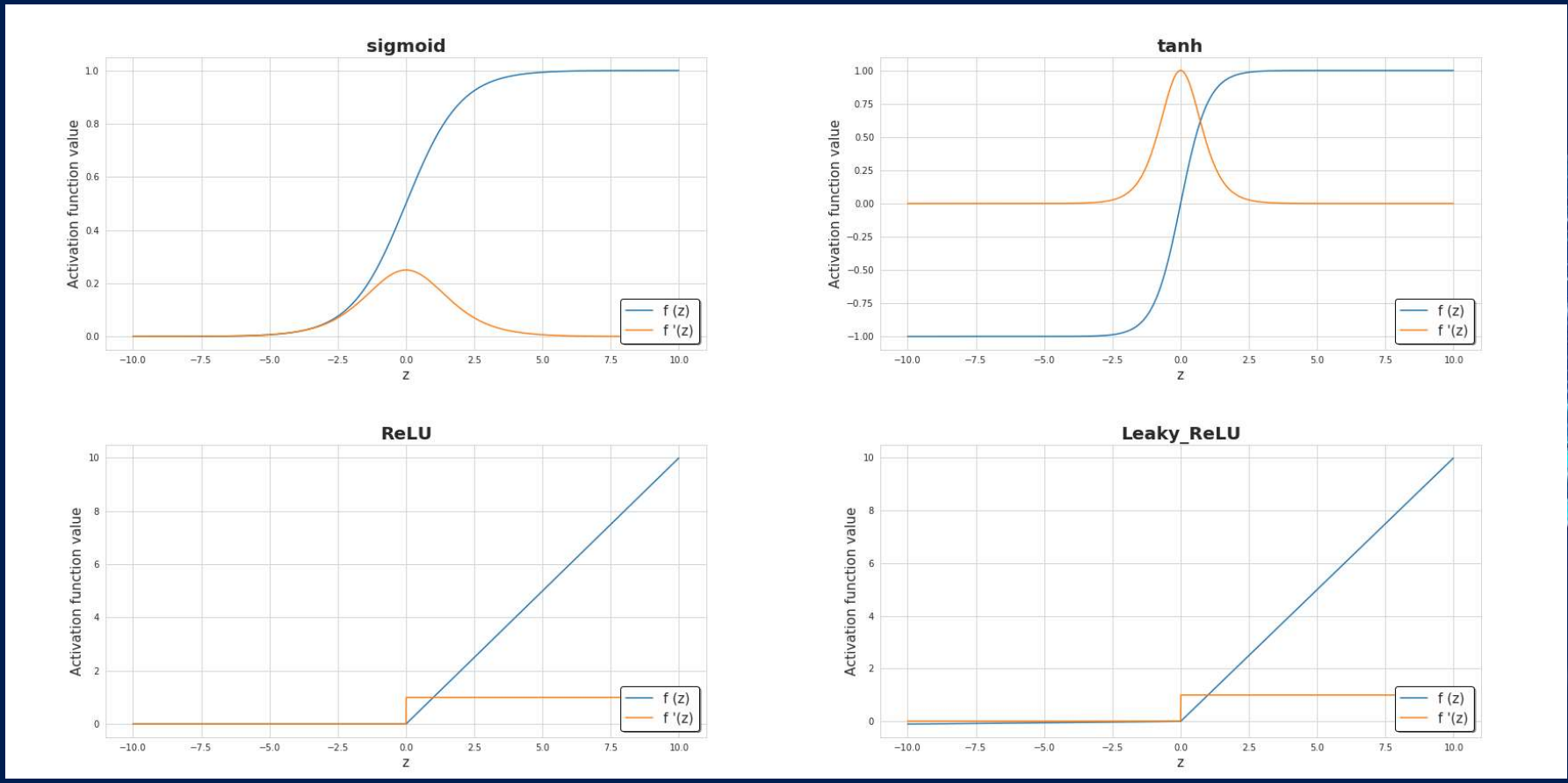
$$u_{i0} = x_i w_{i0}$$
$$v_0 = \sum_i u_{i0}$$
$$y_0 = f(v_0)$$
$$f(a) = \frac{1}{1+e^{-a}}$$



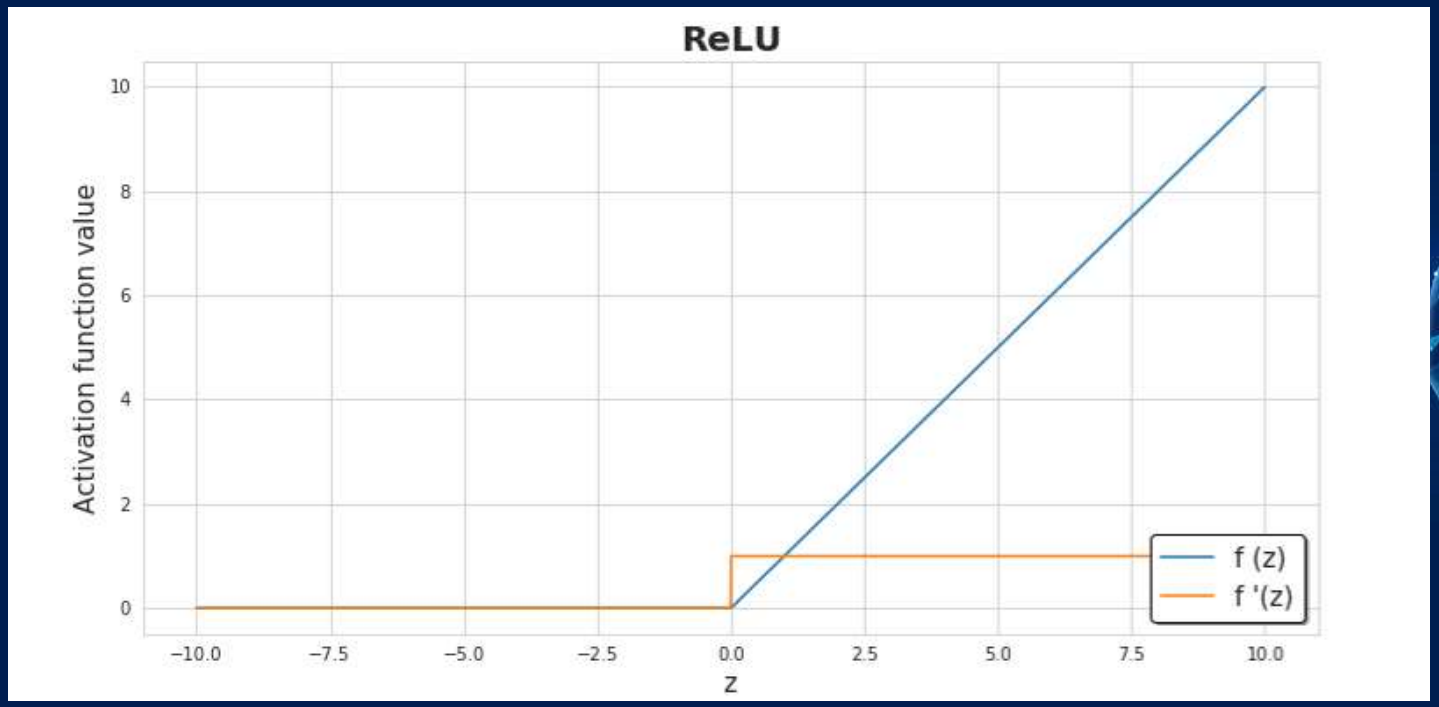




Neuron activation functions



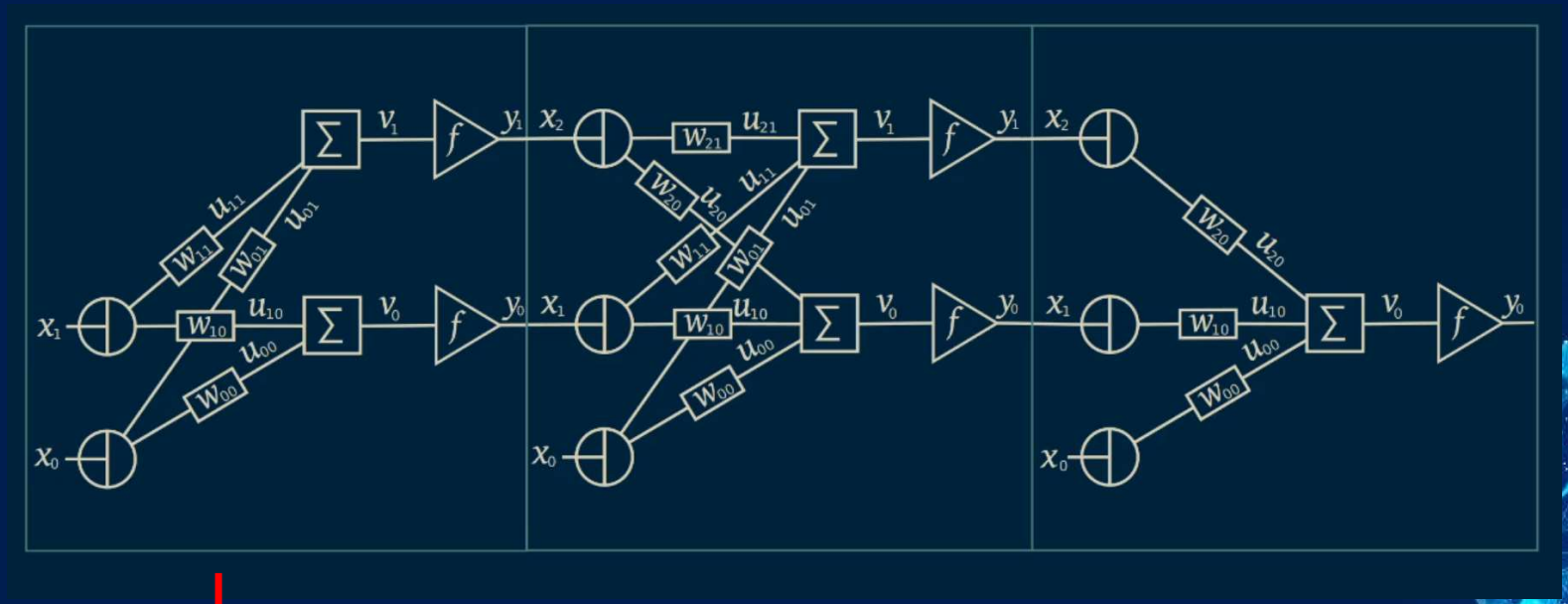
ReLU (rectified linear unit) = > nonlinear function!!!



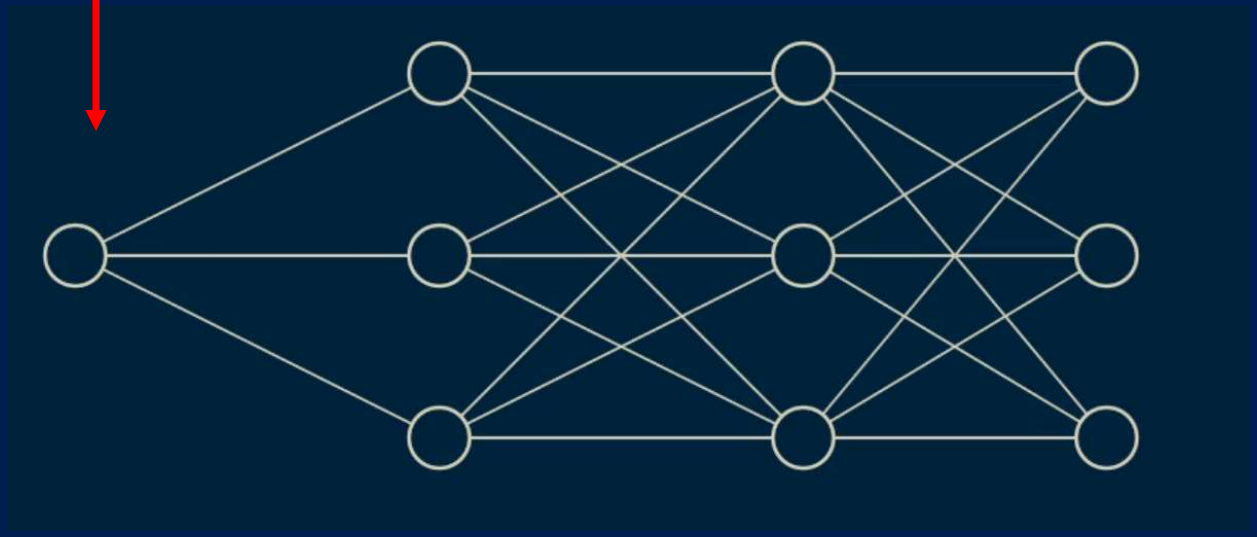
Definition of linear function (main) : f(a*x + b*y) = a*f(x) + b*f(y)
Assuming for simplicity that a = b = 1, let's try x=-5, y=1 with f being the ReLU function:

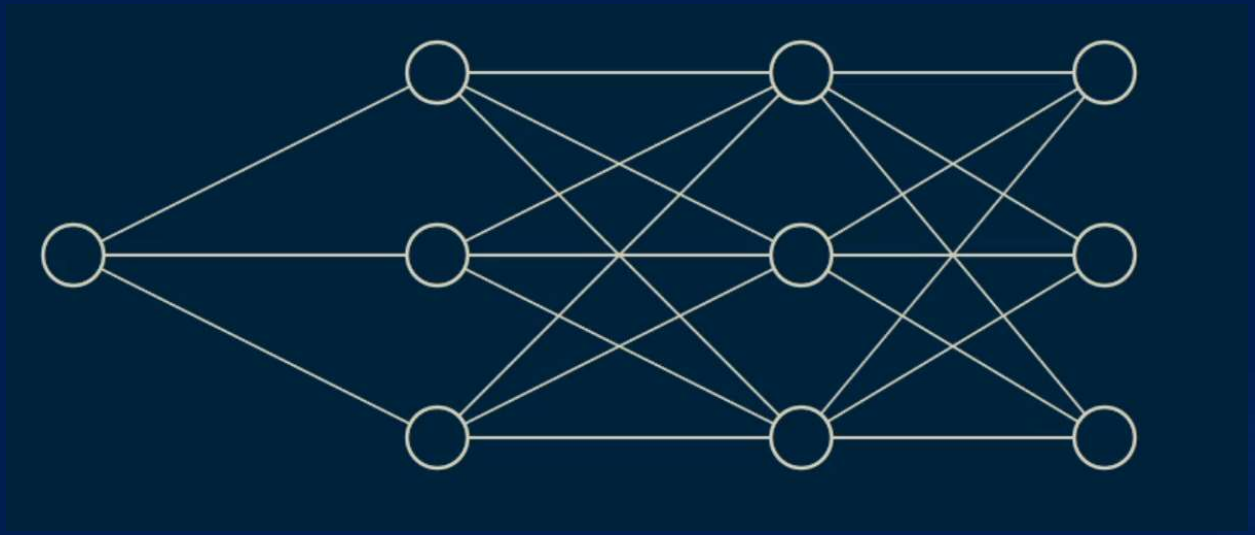
(-5 + 1) = f(-4) = 0

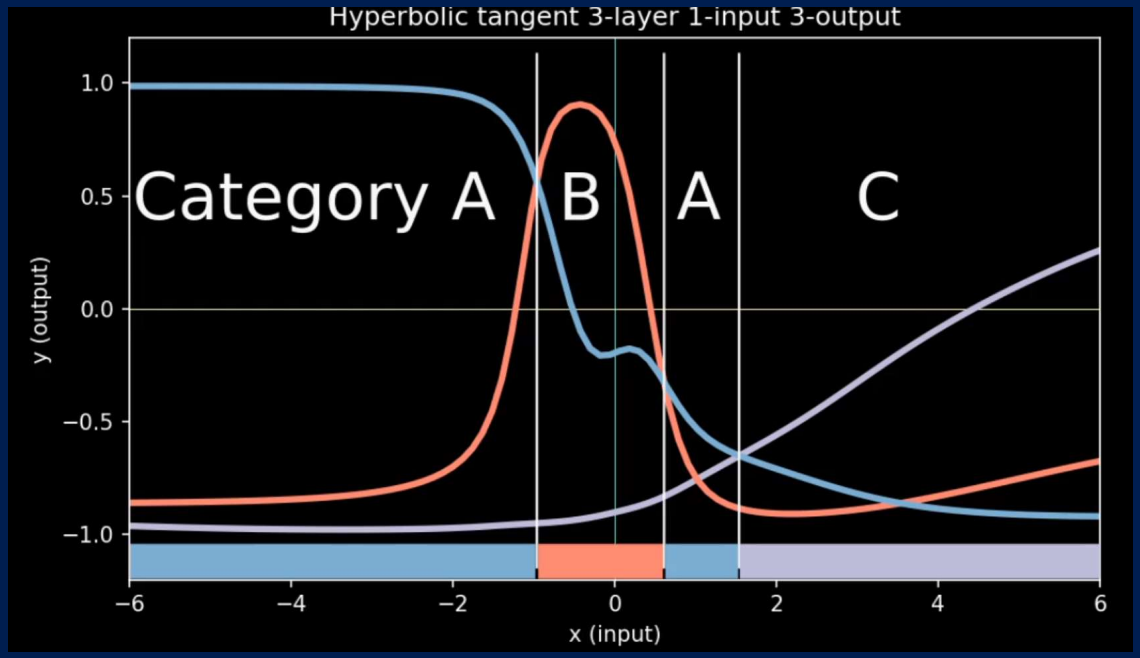
f(-5) + f(1) = 0 + 1 = 1

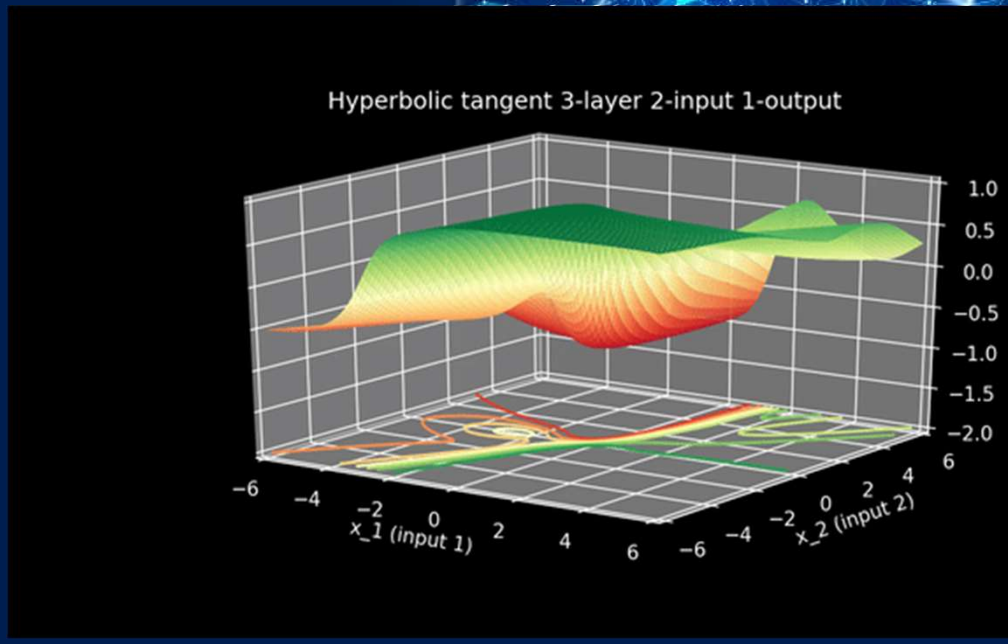


Simplification (schematic)



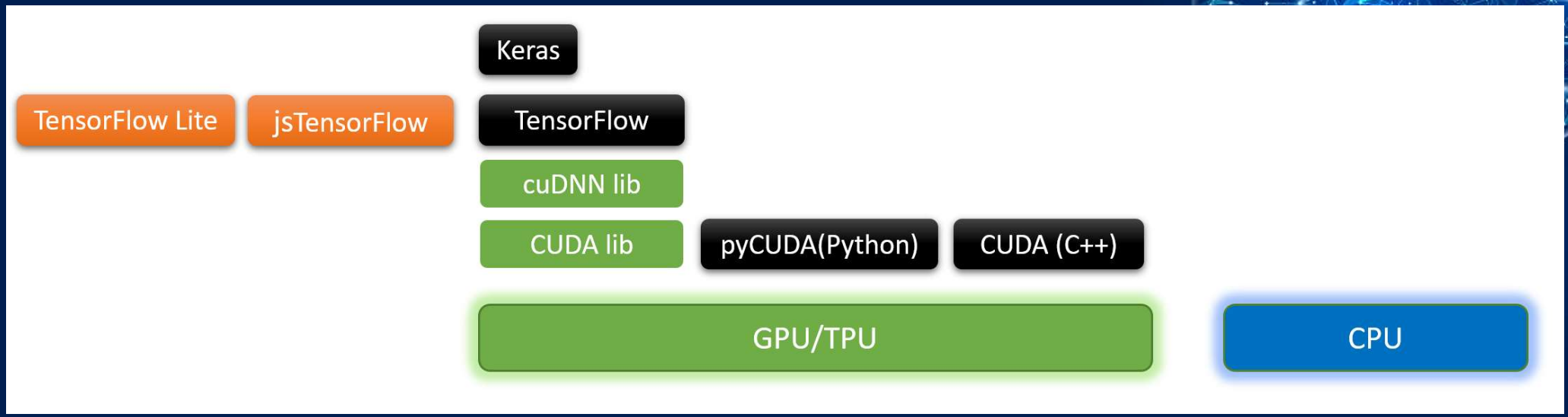



Hyperbolic tangent 3-layer 1-input 3-output
Category A
B
A
C
y (output)
x (input)
1.0
0.5
0.0
−0.5
−1.0
−6
−4
−2
0
2
4
6

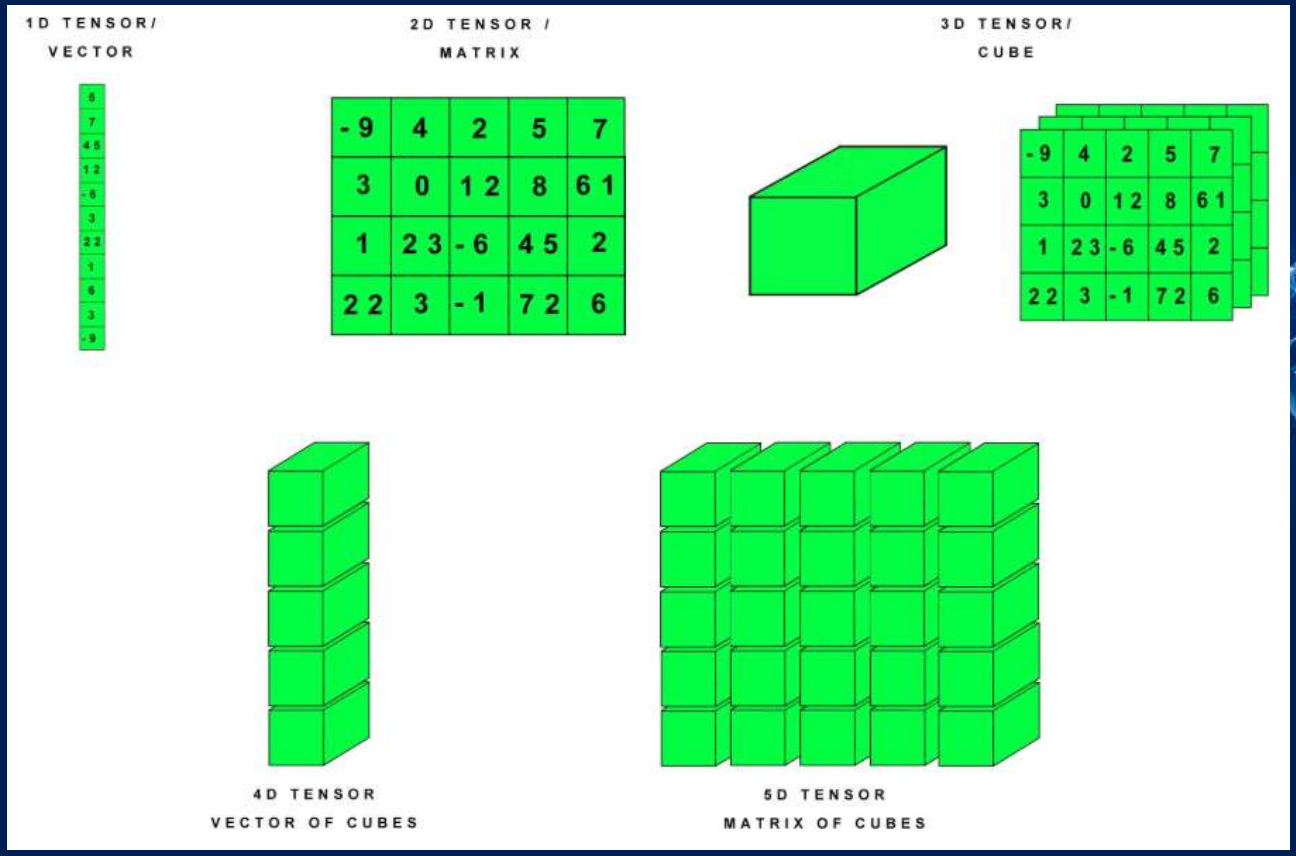

Hyperbolic tangent 3-layer 2-input 1-output
x_1 (input 1)
x_2 (input 2)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0

TensorFlow 1.x. Introduction

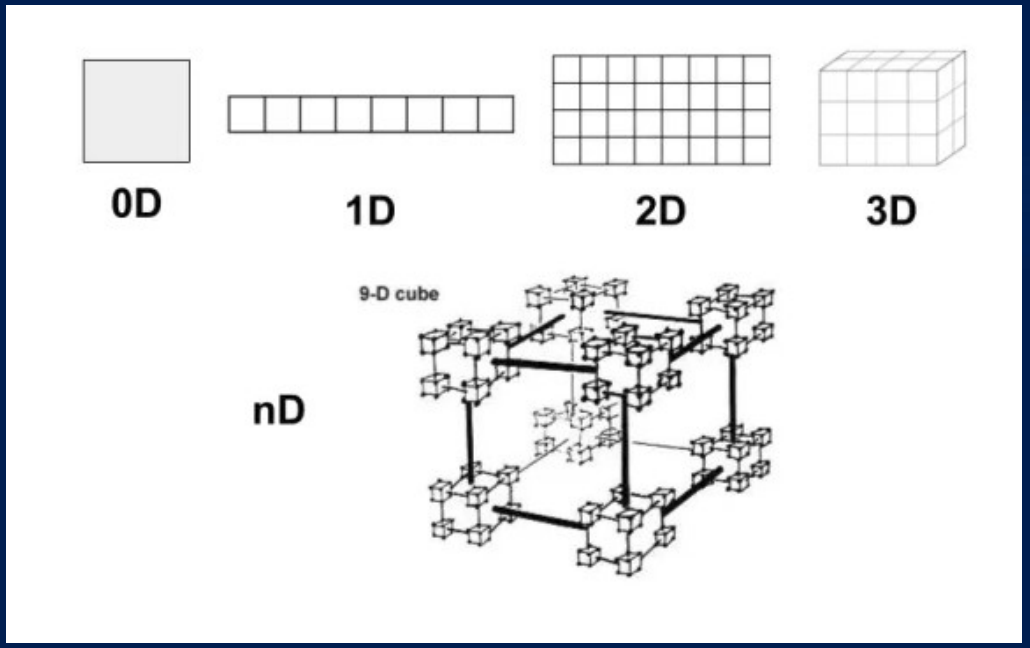
TensorFlow's C++ API provides mechanisms for constructing and executing a data flow graph.



Tensor.


1D TENSOR/
VECTOR
2D TENSOR /
MATRIX
3D TENSOR/
CUBE
4D TENSOR
VECTOR OF CUBES
5D TENSOR
MATRIX OF CUBES

Tensor.



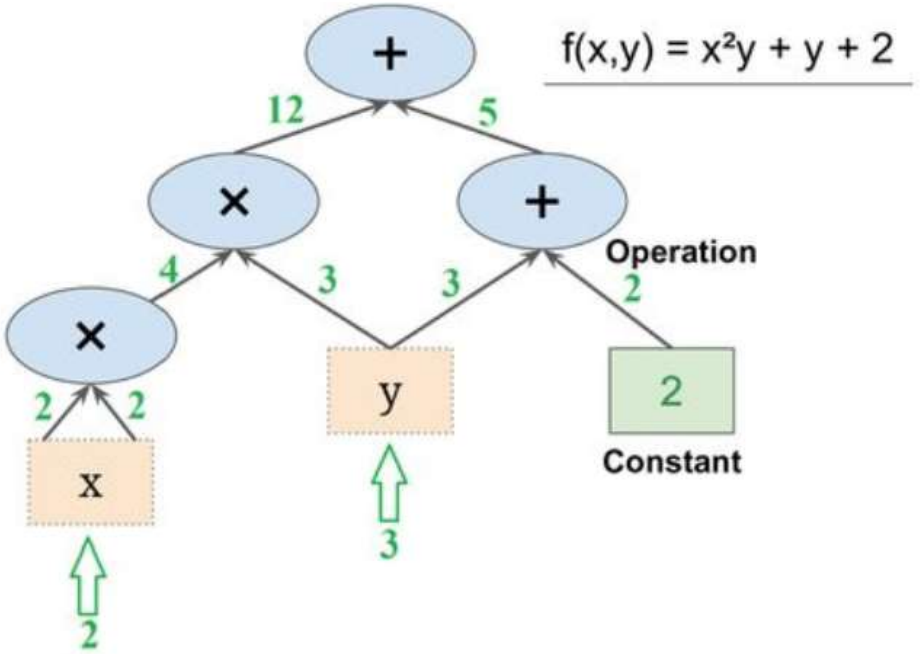
Graph and Session

Graph

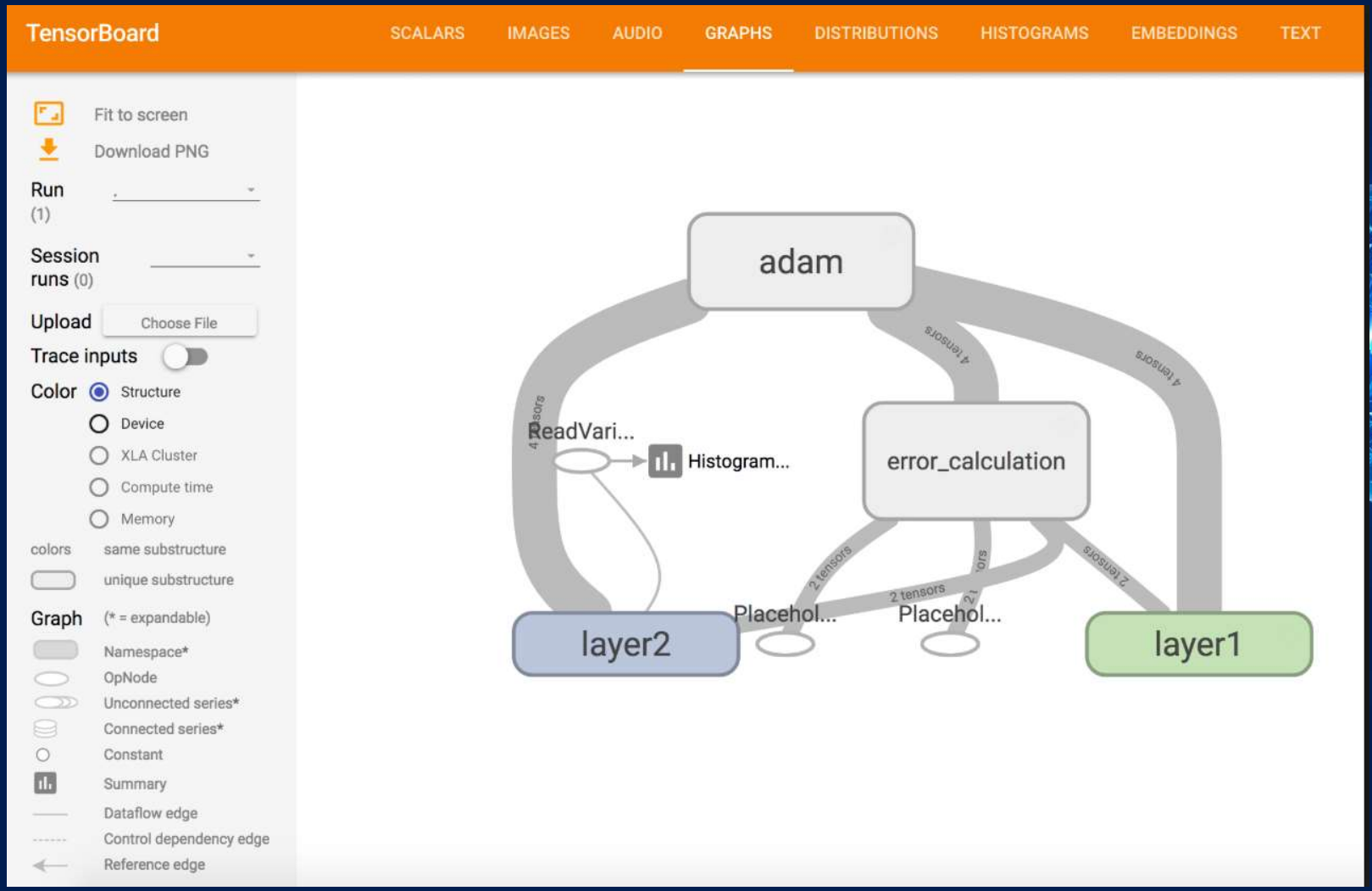
- Nodes = Operations
- Edges = Tensors

Session

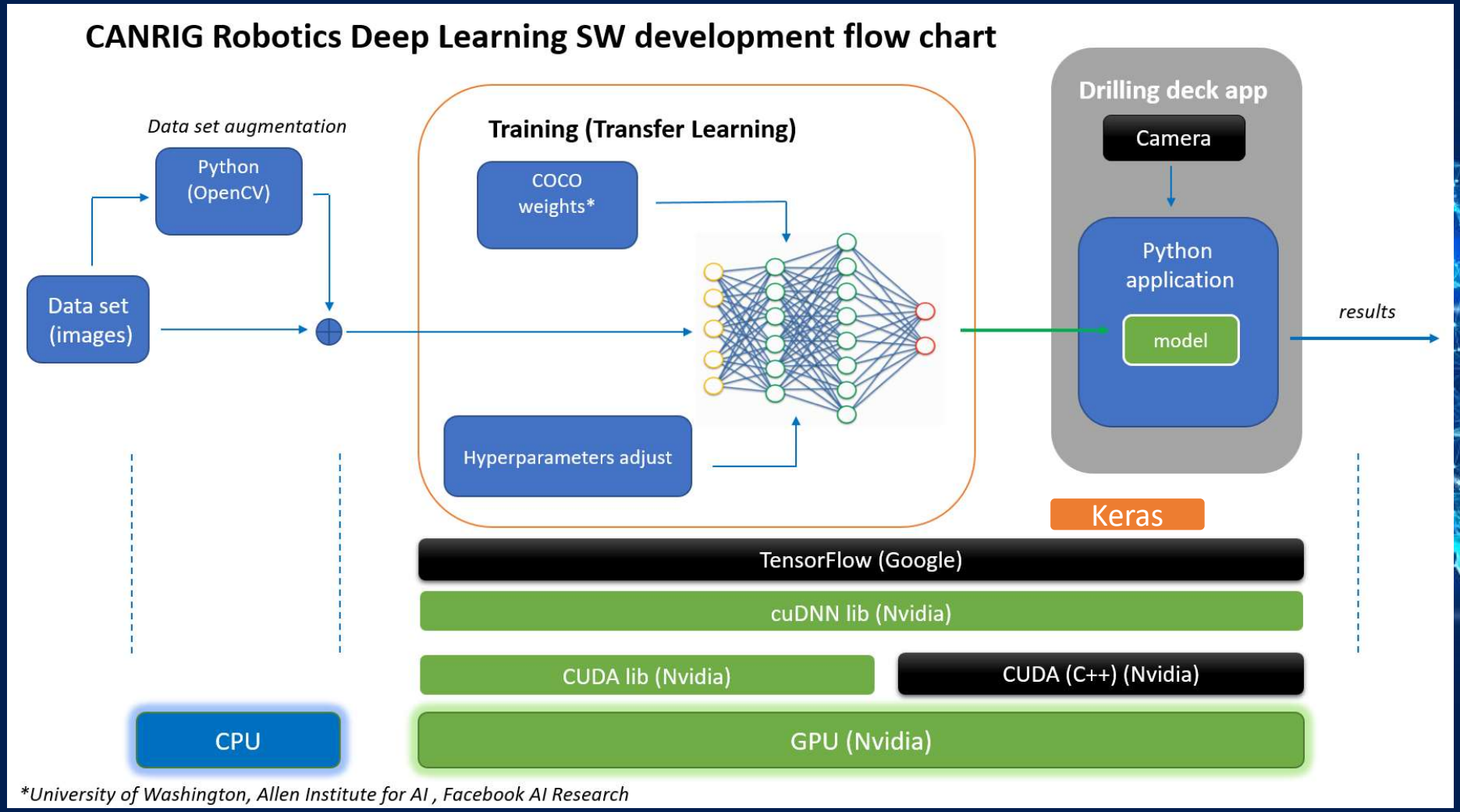
- Tensor = data
- Tensor + flow = data + flow



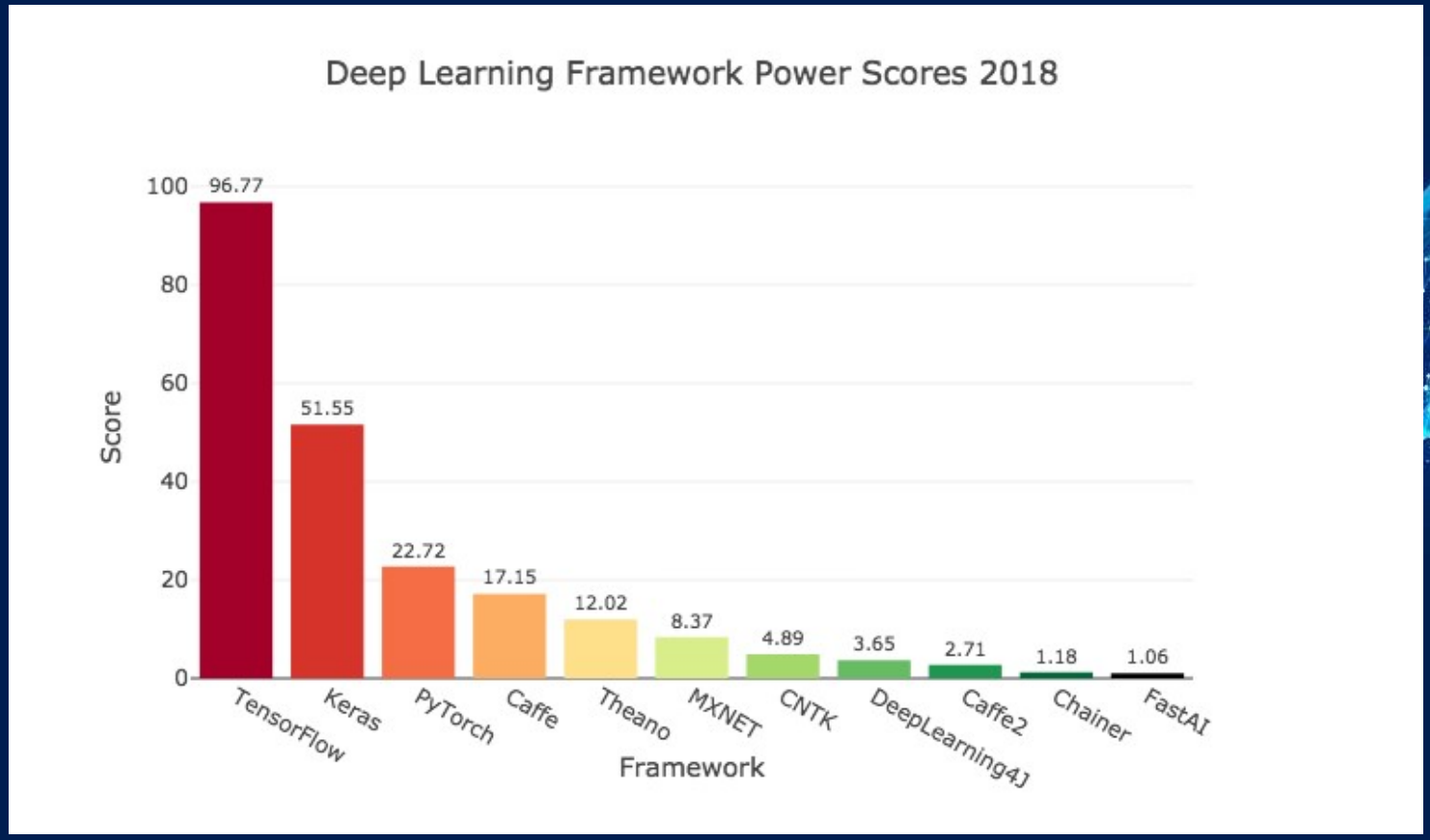
TensorBoard => tensorboard -- logdir = ./path/to/your/log/folder



TensorFlow SW stack



TensorFlow popularity



Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience (by NVIDIA)

https://arxiv.org/pdf/1810.05687.pdf

https://news.developer.nvidia.com/nvidia-research-at-icra-2019/

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf