Deep Reinforcement Learning

- *Crash Course* -

Definition

"Reinforcement learning is learning ***what*** to do,
how to map situations to actions—so as to maximize a numerical reward signal."

- *Richard Sutton & Andrew Barto*

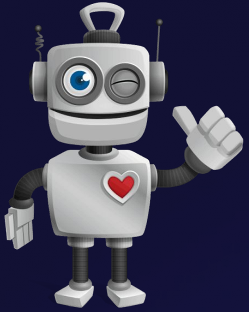
Playing Chess

Driving a Car

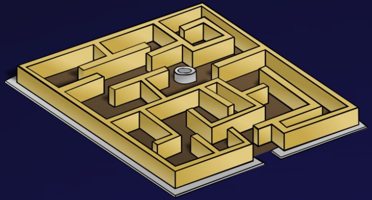
Controlling a Robot

Core Concepts

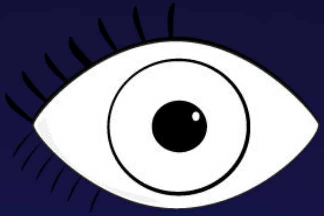
Components that are part of every Reinforcement Learning problem



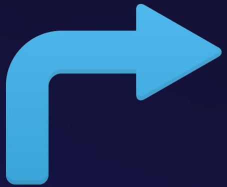
Agent



Environment



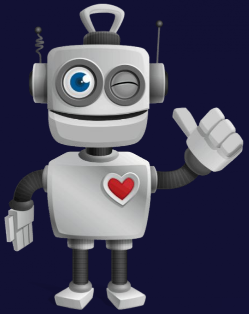
Observations



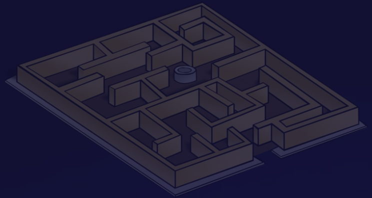
Actions

Rewards

Core Concepts



Agent



Environment

Observations

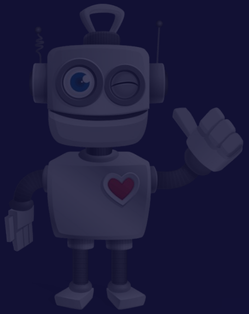
Actions

Rewards

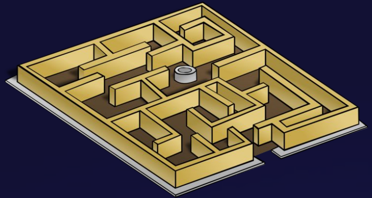
“The learner and decision maker”

A distinct entity that can observe the environment and perform actions

Core Concepts



Agent



Environment

Observations

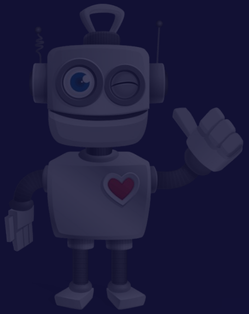
Actions

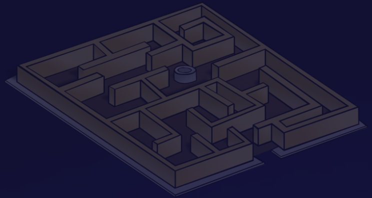
Rewards

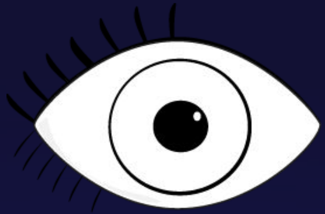
“The system that the agent exists within”

Everything in the system that exists outside of the agent

Core Concepts


Agent


Environment


Observations


Actions


Rewards

“The input to the agent”

The information the agent receives about the environment

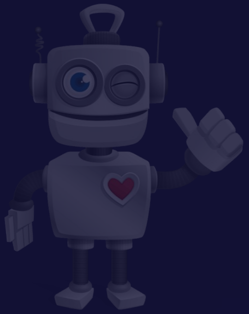
Core Concepts

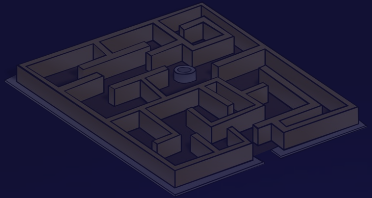
Agent Environment Observations Actions Rewards

“The outputs of the agent”

The tools which the agent can use to interact and impact the environment

Core Concepts


Agent


Environment


Observations


Actions


Rewards

“Numerical values the agent seeks to maximise”

Similar to the loss function, maximising the reward signal should solve the problem of interest

Core Concepts - *Self Driving Car*

Core Concepts - *Self Driving Car*

Agent

The Car

Environment:

Observations:

Actions:

Reward:

Core Concepts - *Self Driving Car*

Agent

The Car

Environment:

The road system, other cars, pedestrians, etc...

Observations:

Actions:

Reward:

Core Concepts - *Self Driving Car*

Agent

The Car

Environment:

The road system, other cars, pedestrians, etc...

Observations:

Camera sensors, Lidar information, gps, etc...

Actions:

Reward:

Core Concepts - *Self Driving Car*

Agent

The Car

Environment:

The road system, other cars, pedestrians, etc...

Observations:

Camera sensors, Lidar information, gps, etc...

Actions:

Turning, Braking, accelerating, etc...

Reward:

Core Concepts - *Self Driving Car*

Agent

The Car

Environment:

The road system, other cars, pedestrians, etc...

Observations:

Camera sensors, Lidar information, gps, etc...

Actions:

Turning, Braking, accelerating, etc...

Reward:

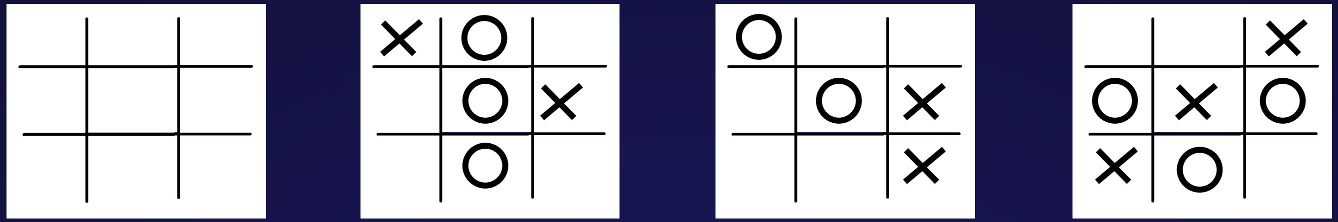
Arriving at target destination, following traffic rules, penalty for crashing, etc...

The Environment

“The system that the agent exists within”

A specific configuration of an environment is called a ***State***

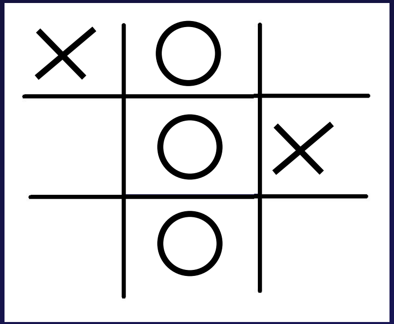
Different states of tic-tac-toe:



The Environment

“The system that the agent exists within”

Certain states might yield a reward



+1

The Policy Function

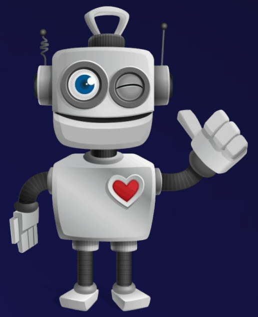
"A policy is a mapping from perceived states of the environment to actions to be taken when in those states."

- *Richard Sutton & Andrew Barto*

The Policy Function

The policy is the crucial component of the Agent.
It can be implement in a multitude of different ways:

- A lookup table
- Tree search algorithm
- Neural Network
- Etc ...



The Reward

A predetermined measure of how well our agent is performing.

The reward defines what behaviours to reinforce and what behaviours to dismiss.

- A numerical value
- Can be given often or rarely
- Can be negative

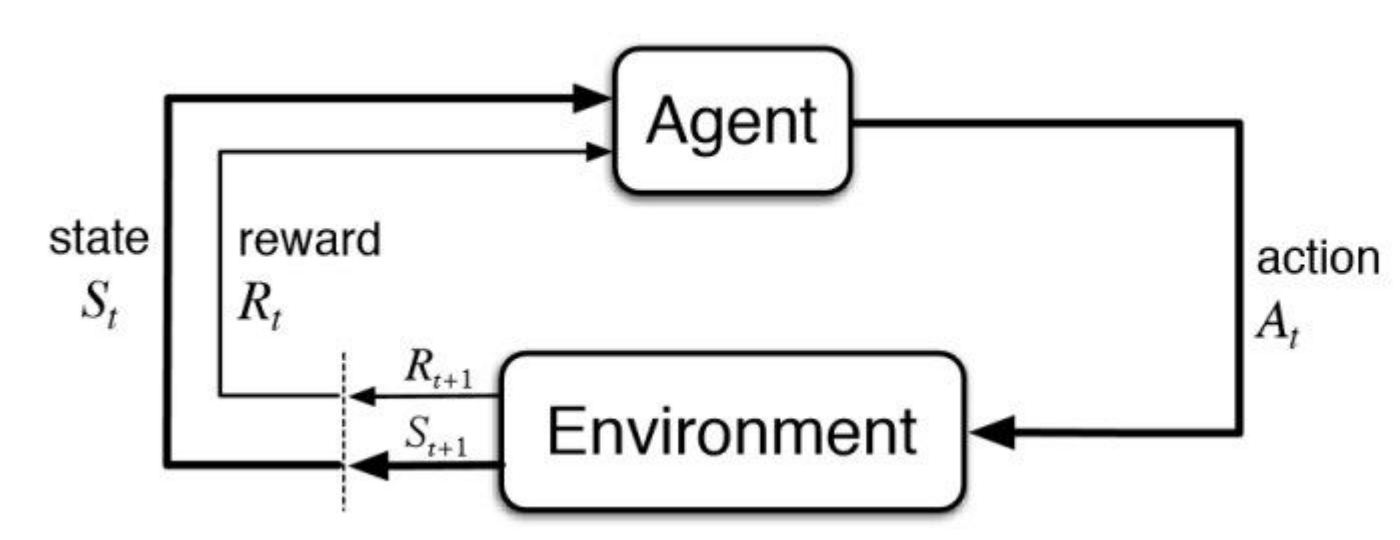
The Reward

Analogous to the loss function in Supervised Learning.

Reward function

Loss Function

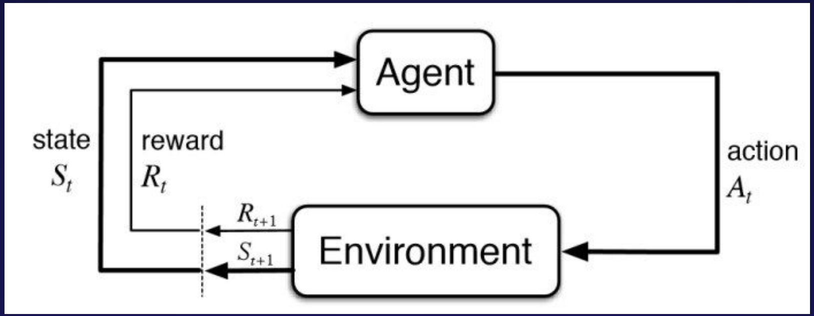
The Framework



The Framework

"Reinforcement learning is learning ***what*** to do,
how to map situations to actions—so as to maximize a numerical reward signal."

"Reinforcement learning is learning a ***Policy***,
that maps ***states*** to ***actions***—so as to maximize the total ***reward***."



Vocabulary

Agent

Action

Environment

Policy

Observation

Reward

State

RL Algorithms

- *Case Study* -

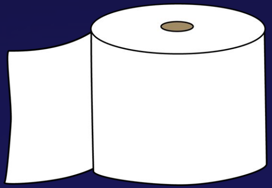
Rock Paper Scissors

Playing the game Rock Paper Scissors against a opponent, we have set the following rewards
Victory -> Reward: ***+1***
Loss -> Reward: ***-1***
Draw -> Reward: ***0***

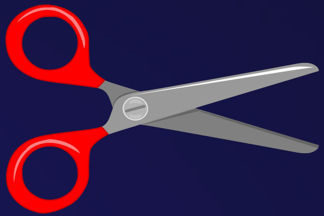
The opponent always plays according to the following probabilistic policy:

p=0.2

p=0.5



p=0.3

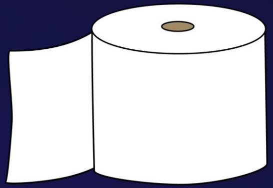
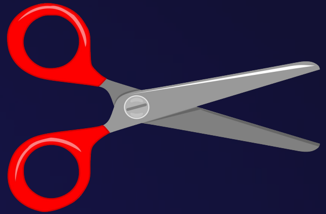


Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

Agent

V=*0* **V=*0*** **V=*0***

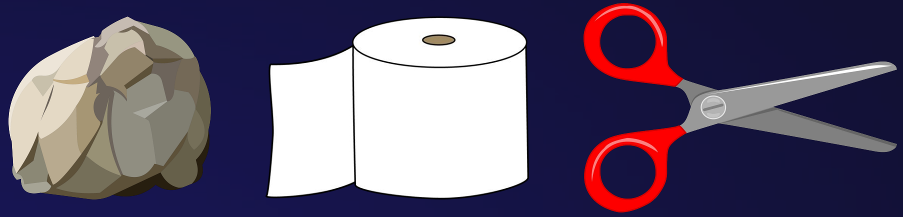
Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*

Agent

V=*-1* **V=*0*** **V=*0***



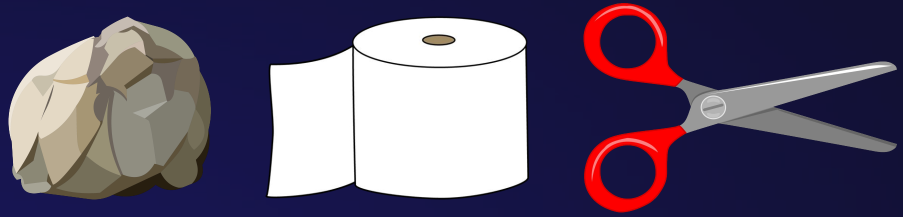
Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*

Agent

V=*-1* **V=*0*** **V=*0***



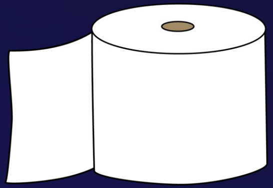
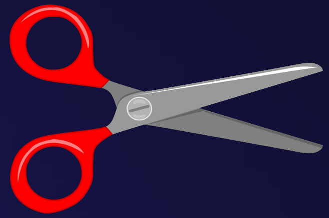
Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*

Agent

V=*-1* **V=*0*** **V=*0***

Rock Paper Scissors - *Greedy Policy*

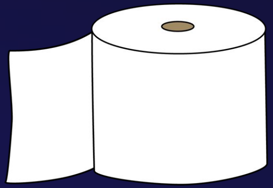
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*

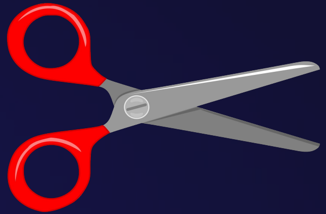
Agent

V=*-1*

V=*1*



V=*0*



Rock Paper Scissors - *Greedy Policy*

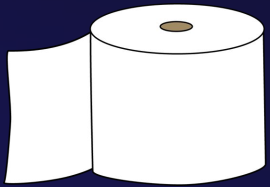
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*

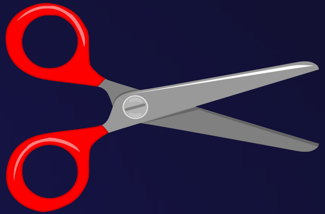
Agent

V=*-1*

V=*0*



V=*0*



Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*
6. Paper: *-1*

Rock Paper Scissors - *Greedy Policy*

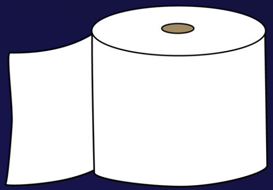
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*
6. Paper: *-1*
7. Scissor: *-1*

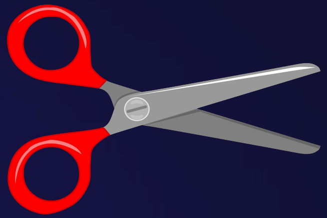
Agent

V=*-1*

V=*-1*



V=*-1*



Rock Paper Scissors - *Greedy Policy*

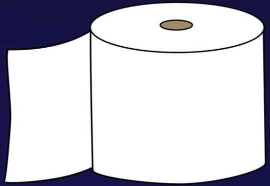
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*
6. Paper: *-1*
7. Scissor: *-1*
8. Rock: *-1*

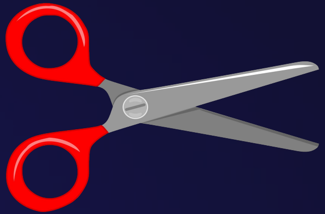
Agent

V=*-2*

V=*-1*



V=*-1*

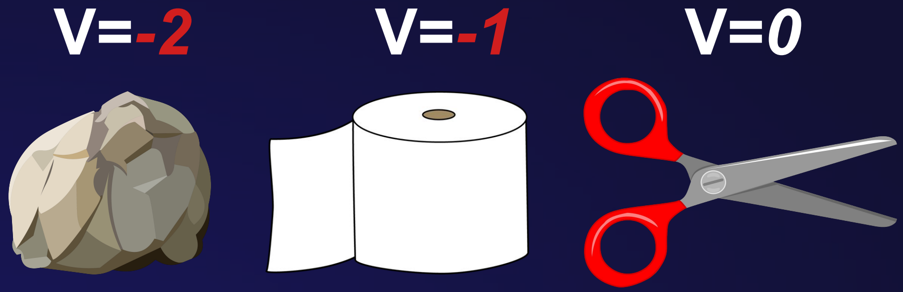


Rock Paper Scissors - *Greedy Policy*

Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*
6. Paper: *-1*
7. Scissor: *-1*
8. Rock: *-1*
9. Scissor: *1*

Agent



Rock Paper Scissors - *Greedy Policy*

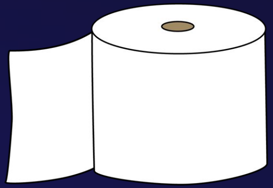
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. Rock: *-1*
2. Paper: *0*
3. Scissor: *0*
4. Paper: *1*
5. Paper: *-1*
6. Paper: *-1*
7. Scissor: *-1*
8. Rock: *-1*
9. Scissor: *1*
10. Scissor: *1*

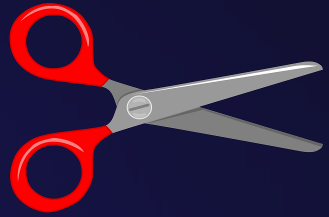
Agent

V=*-2*

V=*-1*



V=*1*



Rock Paper Scissors - *Greedy Policy*

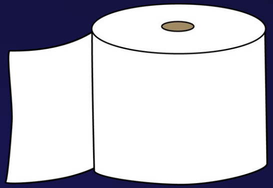
Perform the action that has yielded the highest reward so far.
If two options have been equally good, pick randomly

Agent

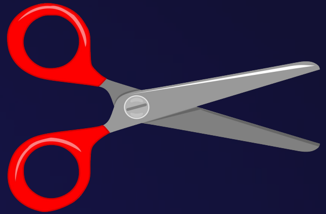
After 10k Games

V=*-2*

V=*-1*



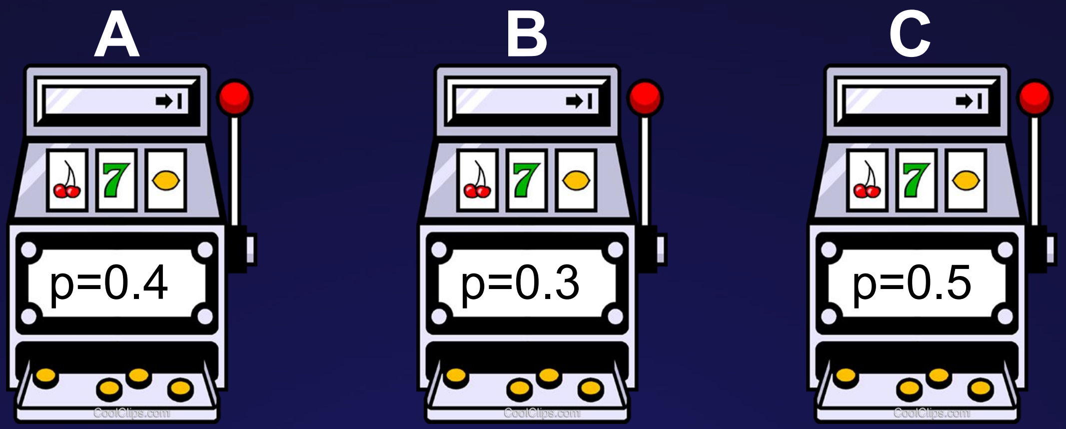
V=*255*



The bandit problem

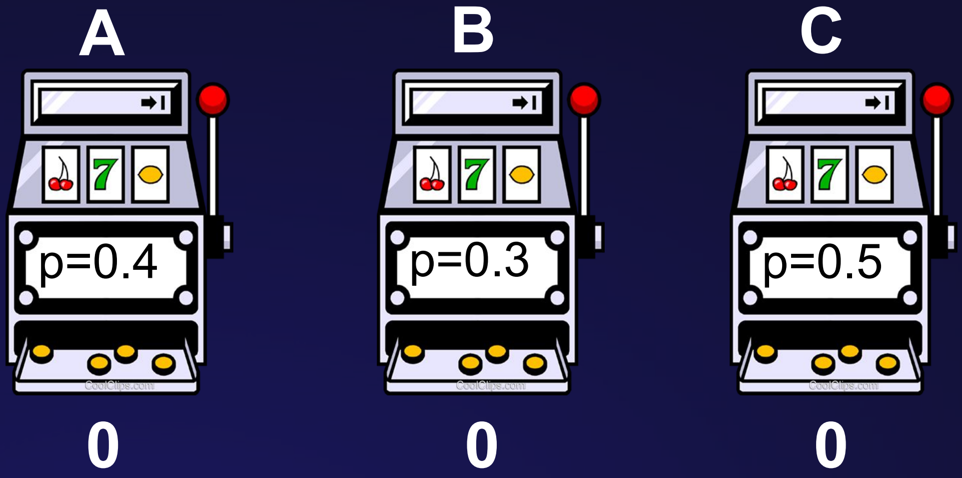
Given 3 different one armed bandits, each with their own, unknown win probability.
Victory -> Reward: ***+1***
Loss -> Reward: ***0***

How should our agent explore and play the slot machines to maximise the reward?



The bandit problem - *Greedy Policy*

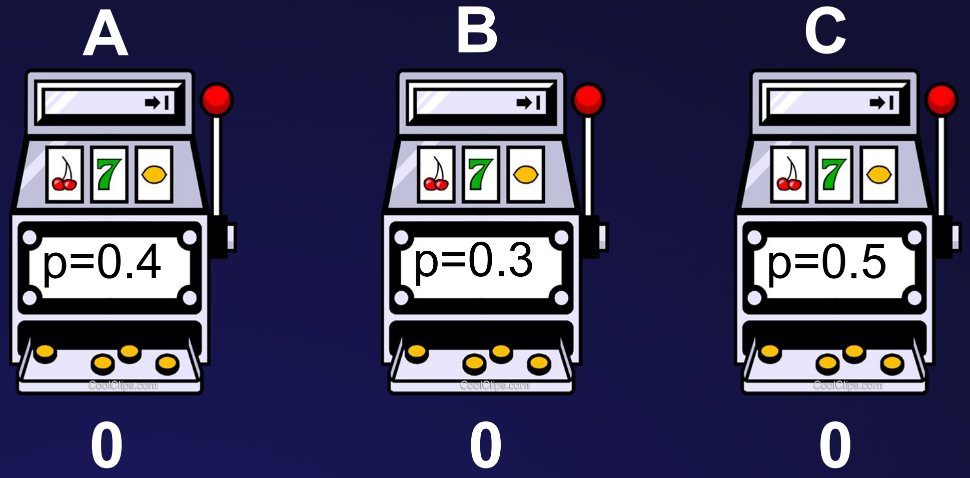
Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly



The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

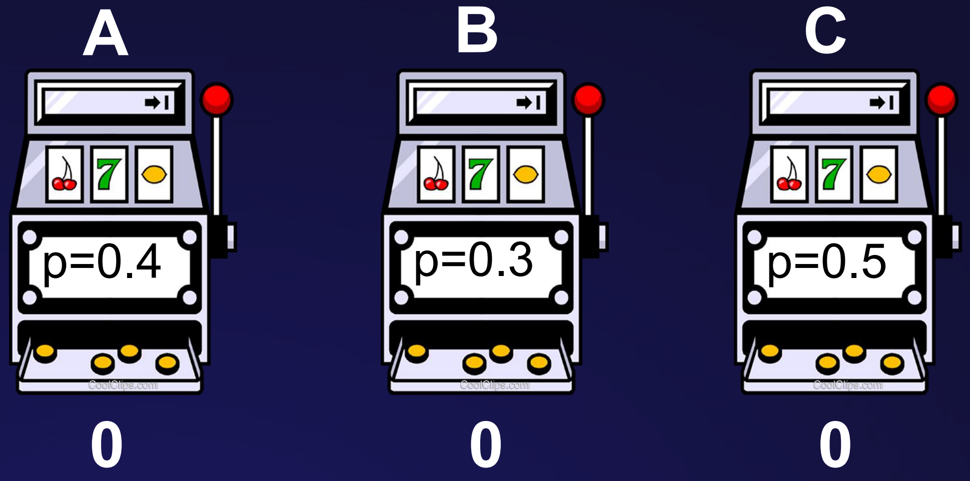
1. B: *0*



The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. B: *0*
2. A: *0*

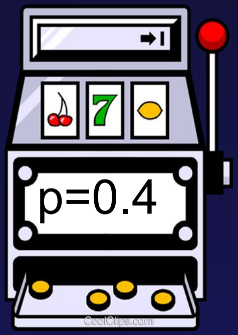


The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

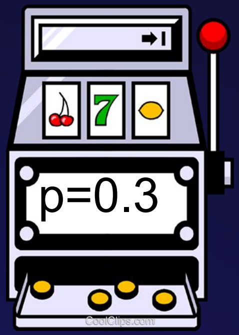
1. B: *0*
2. A: *0*
3. C: *0*

A



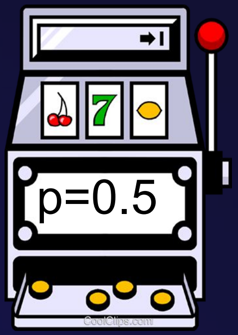
0

B



0

C



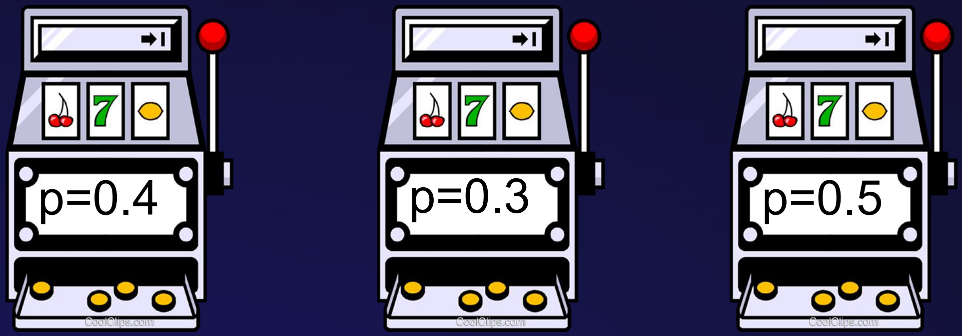
0

The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*

A B C

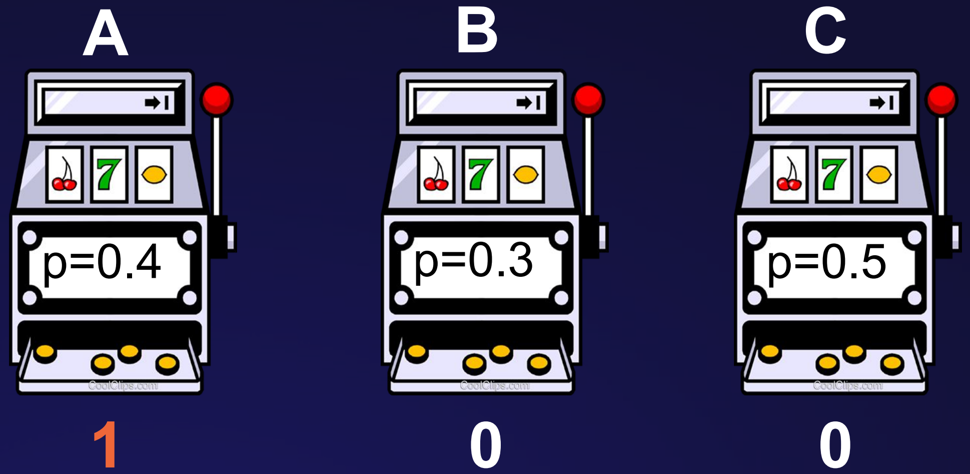


1 0 0

The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

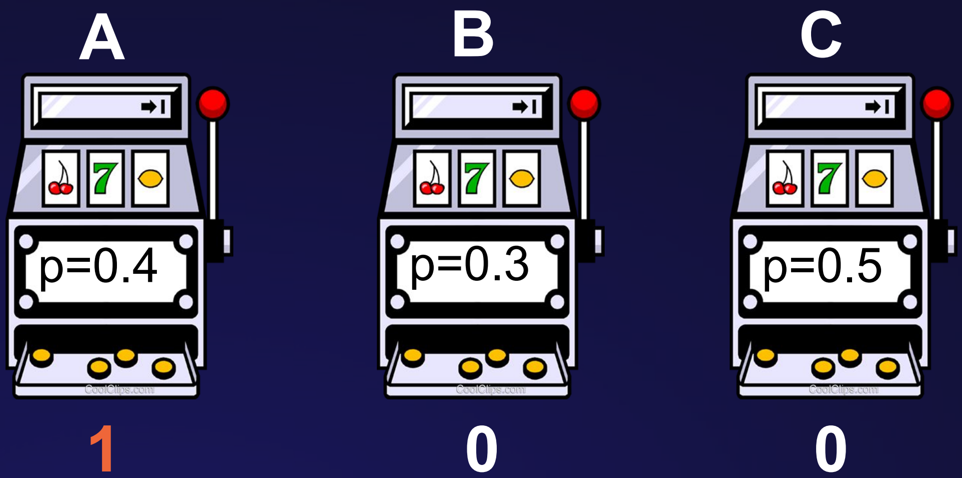
1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*
5. A: *0*



The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

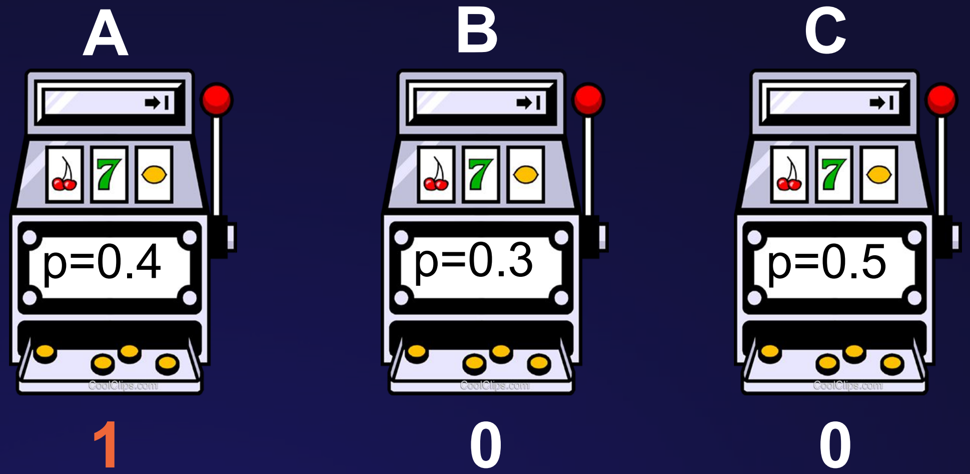
1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*
5. A: *0*
6. A: *0*



The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*
5. A: *0*
6. A: *0*
7. A: *0*

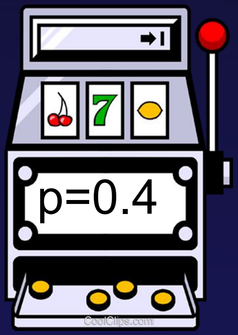


The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

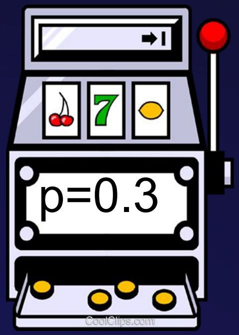
1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*
5. A: *0*
6. A: *0*
7. A: *0*
8. A: *1*

A



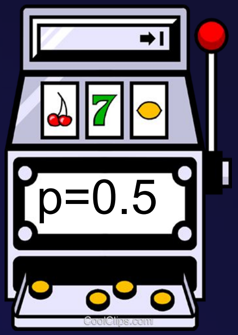
2

B



0

C

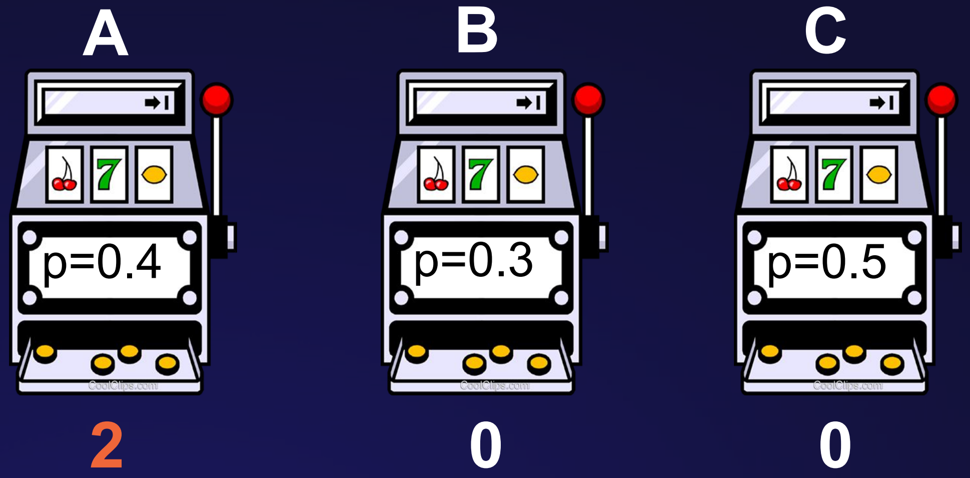


0

The bandit problem - *Greedy Policy*

Play the slot machine that has yielded the highest reward so far.
If two options have been equally good, pick randomly

1. B: *0*
2. A: *0*
3. C: *0*
4. A: *1*
5. A: *0*
6. A: *0*
7. A: *0*
8. A: *1*
9. A: *0*



Exploration vs Exploitation

- *Overview* -

Exploration vs Exploitation

Exploration

Performing actions that we suspect to be sub-optimal.
In order to attain more information about the environment.

Exploitation

Performing actions that we believe will maximise the total sum of rewards.

ε-Greedy

Simple, yet effective exploration algorithm

Perform what is to believed to be the optimal action,
but with probability ε perform a random action. $0 < \varepsilon < 1$.

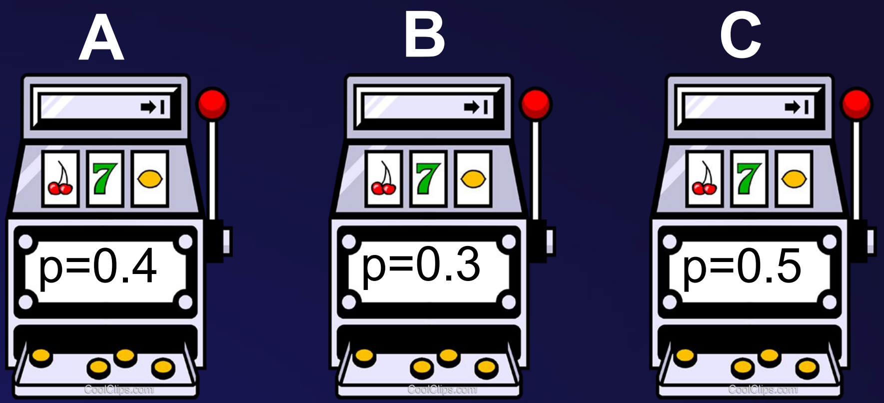
$\varepsilon = 0.1$ denotes that there is a *10%* chance we perform a random action.
This ensures that we are always given a certain amount of exploration.

The bandit problem: *ε-Greedy Policy*

*Play the slot machine that has yielded the highest **average** reward so far.*
If two options have been equally good, pick randomly.

But with ε=0.1 probability perform a random action

Turn	A	B	C
0	*0.5*	*0.5*	*0.5*

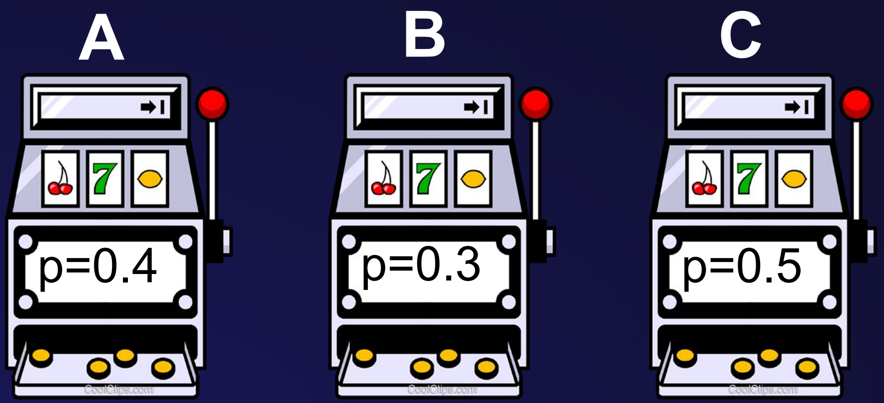


The bandit problem: *ε-Greedy Policy*

*Play the slot machine that has yielded the highest **average** reward so far.*
If two options have been equally good, pick randomly.

But with ε=0.1 probability perform a random action

Turn	A	B	C
0	0.5	0.5	0.5
10	0.25	0.0	0.0

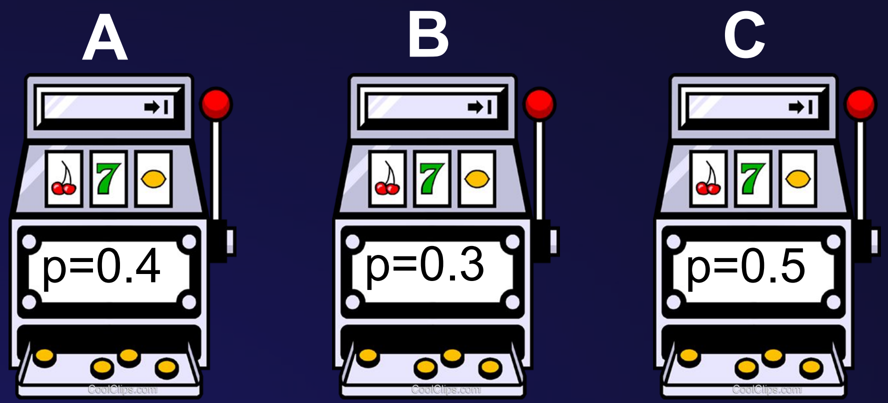


The bandit problem: ε-*Greedy Policy*

Play the slot machine that has yielded the highest ***average*** *reward so far.*
If two options have been equally good, pick randomly.

But with ε=0.1 probability perform a random action

Turn	A	B	C
0	*0.5*	*0.5*	*0.5*
10	*0.25*	*0.0*	*0.0*
100	*0.37*	*0.32*	*0.52*

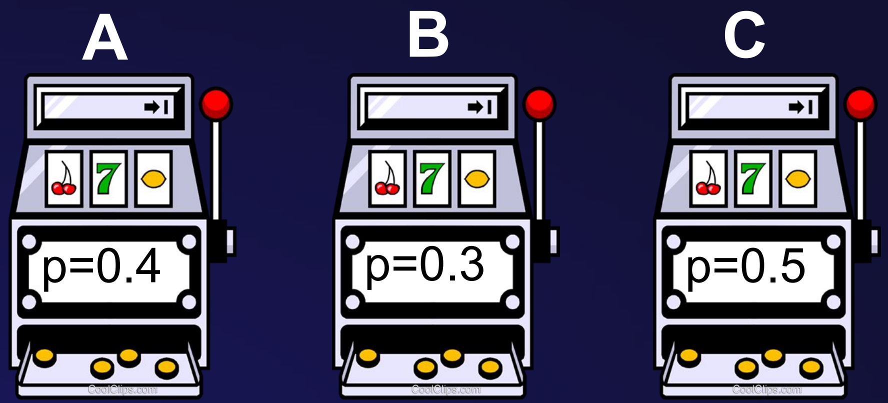


The bandit problem: ε-*Greedy Policy*

Play the slot machine that has yielded the highest ***average*** *reward so far.*
If two options have been equally good, pick randomly.

But with ε=0.1 probability perform a random action

Turn	A	B	C
0	0.5	0.5	0.5
10	0.25	0.0	0.0
100	0.37	0.32	0.52
10k	0.39	0.29	0.50



UCB1 Formula

More sophisticated algorithm, taking into account our uncertainty for certain actions

Pick the action that maximises the UCB score S

$$S = Q(a) + c\,U(a)$$

Exploitation

Exploration

Constant used to prioritise between the two

UCB1 Formula

More sophisticated algorithm, taking into account our uncertainty for certain actions

Pick the action that maximises the UCB score S

$$S = Q(a) + U(a)$$

$Q(a)$ = average reward received when performing action a

$$U(a) = \sqrt{\frac{2 \ln N}{n(a)}}$$

N = total number of actions performed

$n(a)$ = number of times action a has been performed

Deep Reinforcement Learning

Deep RL

Previous approaches stores a value for every action,
as actions depend on the current state, this does ***NOT*** scale!

Approximate number of states:

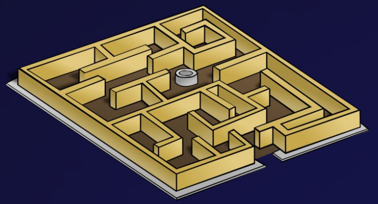
10^{52}

∞

∞

Deep RL

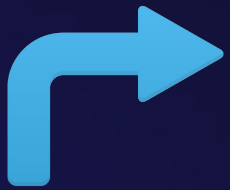
What if we instead used a Neural Network to learn a Policy function?



State

Policy

Neural Network



Action

Deep RL - *Atari Breakout*

Deep RL - *Atari Breakout*

State

210x160x3 pixels

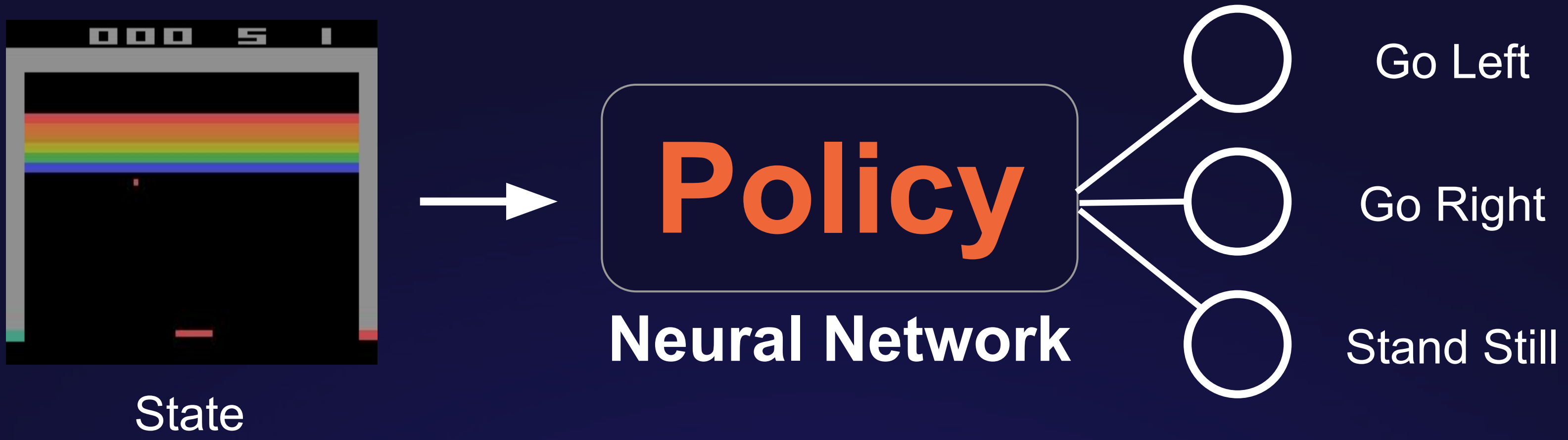
Actions:

Go left
Go Right
Stand Still

Rewards:

Hitting a brick
Finishing a level

Deep RL - *Atari Breakout*



Deep RL - *Atari Breakout*

Two problems arises:

Deep RL - *Atari Breakout*

Two problems arises:

How can we encode the game state so that it contains all the needed information and still being processable by a network.

Deep RL - *Atari Breakout*

Two problems arises:

How can we encode the game state so that it contains all the needed information and still being processable by a network.

If rewards are rare, how can we tell what actions contributed to what rewards?

Deep RL - *Atari Breakout*

How can we tell which way the ball is moving?

210x160x3 pixels

Deep RL - *Atari Breakout*

The state passed to the agent can contain additional information than what can currently be observed.

For example in Breakout, we could include the ***H*** last frames in the state. This gives the policy the ability to calculate the direction of the ball

*210x160x3x**H*** pixels

Credit Assignment Problem

If rewards are rare, how can we tell what actions contributed to what rewards?

Credit Assignment Problem

If rewards are rare, how can we tell what actions contributed to what rewards?

The perhaps hardest problem of Reinforcement Learning. Given a long time horizon and few rewards, how do we decide what actions were good and bad?

Credit Assignment Problem

If rewards are rare, how can we tell what actions contributed to what rewards?

→ *~42 Actions* →

Victory: ***+1***
Loss: ***-1***
Draw: ***0***

Solution 1: *Reward Shaping*

Introduce intermediate rewards that you think will contribute to a good solution

Taking opponents Queen

+1

Having Paddle under the ball

+1

Reward Shaping

Can greatly amplify the reward signal +1

Can introduce biases that could hinder the algorithm from finding the optimal policy. +1

Solution 2: *Computational Power*

By running sufficiently many trials, even the weakest reward signals can be sufficient.

Approximate Training time:

44 million Games played

200 years

65k years

AlphaZero

- *Case Study* -

Further Reading

- "An Introduction to Reinforcement Learning" - Great Textbook

- Deepmind Video Lectures:
  - *Deep Learning & Deep Reinforcement Learning*
  - *Reinforcement Learning*

- *Random Funny RL Videos:*
  - *Mar I/0 - Genetic Algorithms applied to Super Mario*
  - *How AlphaZero Works - Explanation video on AZ*
  - *AlphaGo - Netflix Documentary*
  - *History of Reinforcement Learning - Andrew Barto presentation*

Words of Wisdom