Reinforcement Learning

RL

https://deeplizard.com/learn/video/nyjbcRQ-uQ8

What is RL?

- ML method based on rewarding desired behaviors and punishing undesired ones
  - “carrot and the stick”
- An RL agent is able to perceive and interpret its environment, take actions and learn through trial and error.
  - Agent seeks long-term and maximum overall reward to achieve an optimal solution.
  - These long-term goals help prevent the agent from stalling on lesser goals.

Benefits

Reinforcement learning is applicable to a wide range of complex problems that cannot be tackled with other machine learning algorithms. RL is closer to artificial general intelligence (AGI), as it possesses the ability to seek a long-term goal while exploring various possibilities autonomously. Some of the benefits of RL include:

- **Focuses on the problem as a whole.** Conventional machine learning algorithms are designed to excel at specific subtasks, without a notion of the big picture. RL, on the other hand, doesn't divide the problem into subproblems; it directly works to maximize the long-term reward. It has an obvious purpose, understands the goal, and is capable of trading off short-term rewards for long-term benefits.
- **Does not need a separate data collection step.** Training data is obtained via the direct interaction of the agent with the environment. Training data is the learning agent's experience, not a separate collection of data that has to be fed to the algorithm. This significantly reduces the burden on the supervisor in charge of the training process.
- **Works in dynamic, uncertain environments.** RL algorithms are inherently adaptive and built to respond to changes in the environment. In RL, time matters and the experience that the agent collects is not independently and identically distributed (i.i.d.), unlike conventional machine learning algorithms. Since the dimension of time is deeply buried in the mechanics of RL, the learning is inherently adaptive.

Challenges

While RL algorithms have been successful in solving complex problems in diverse simulated environments, their adoption in the real world has been slow. Here are some of the challenges that have made their uptake difficult:

- **RL agent needs extensive experience.**
  - RL methods autonomously generate training data by interacting with the environment. Thus, the rate of data collection is limited by the dynamics of the environment. Environments with high latency slow down the learning curve. Furthermore, in complex environments with high-dimensional state spaces, extensive exploration is needed before a good solution can be found.
- **Delayed rewards.**
  - The learning agent can trade off short-term rewards for long-term gains. While this foundational principle makes RL useful, it also makes it difficult for the agent to discover the optimal policy. This is especially true in environments where the outcome is unknown until a large number of sequential actions are taken. In this scenario, assigning credit to a previous action for the final outcome is challenging and can introduce large variance during training. The game of chess is a relevant example here, where the outcome of the game is unknown until both players have made all their moves.
- **Lack of interpretability.**
  - Once an RL agent has learned the optimal policy and is deployed in the environment, it takes actions based on its experience. To an external observer, the reason for these actions might not be obvious. This lack of interpretability interferes with the development of trust between the agent and the observer. If an observer could explain the actions that the RL agent tasks, it would help him in understanding the problem better and discovering limitations of the model, especially in high-risk environments.

RL Advantages

- Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.
- The model can correct the errors that occurred during the training process.
- In RL, training data is obtained via the direct interaction of the agent with the environment
- Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.
- Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.
- Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

RL Disadvantages

- Reinforcement learning is not preferable to use for solving simple problems.
- Reinforcement learning needs a lot of data and a lot of computation
- Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.
- Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

Applications

- Gaming
  - Likely the most common usage field for reinforcement learning. It is capable of achieving superhuman performance in numerous games.
  - Game theory.
  - Applicable to Finance
- Resource management
  - In enterprise resource management (ERM), RL algorithms can allocate limited resources to different tasks as long as there is an overall goal it is trying to achieve. A goal in this circumstance would be to save time or conserve resources.
  - Applicable to Finance
- Personalized recommendations
  - Applicable to Finance
- Robotics

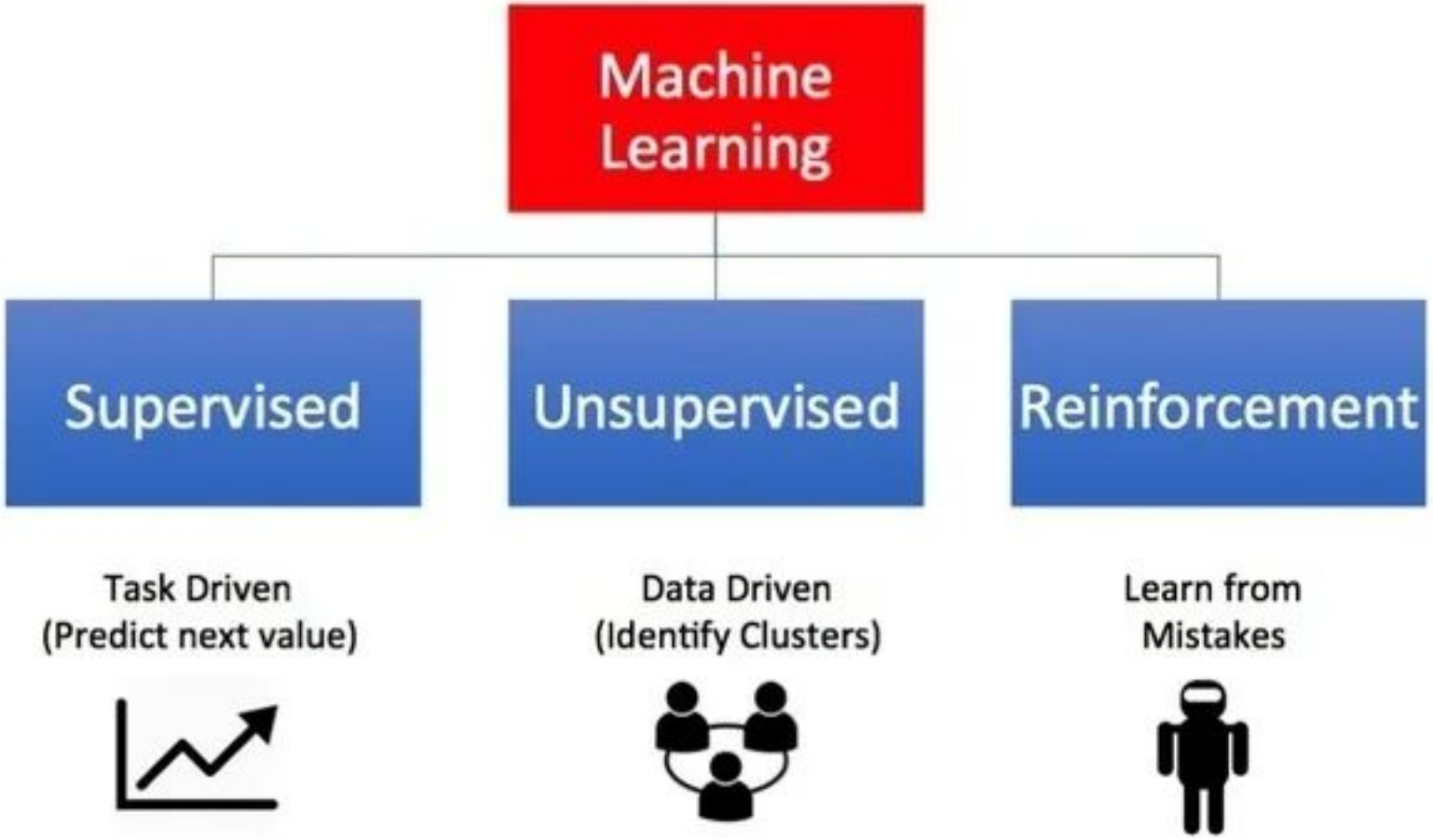
Applications

- **Gaming** - RL is quite widely used in building AI for playing computer games. AlphaGo Zero is the first computer program to defeat a world champion in the ancient Chinese game of Go. Others include ATARI games, Backgammon ,etc
- **Robotics** - In robotics and industrial automation, RL is used to enable the robot to create an efficient adaptive control system for itself which learns from its own experience and behavior. DeepMind's work on Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Policy updates is a good example of the same.
- Other applications of RL include abstractive text summarization engines, dialog agents(text, speech) which can learn from user interactions and improve with time, learning optimal treatment policies in healthcare and RL based agents for online stock trading.

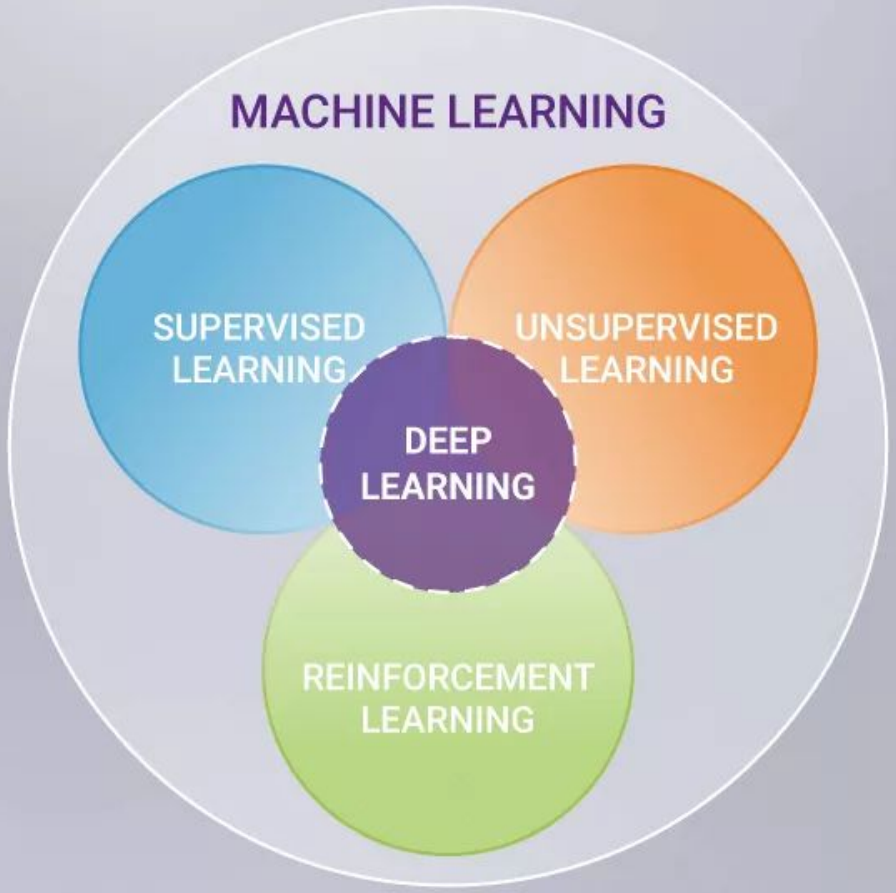
Robotics

https://www.youtube.com/watch?v=ZhsEKTo7V04

Types of Machine Learning



SYNOPSYS®

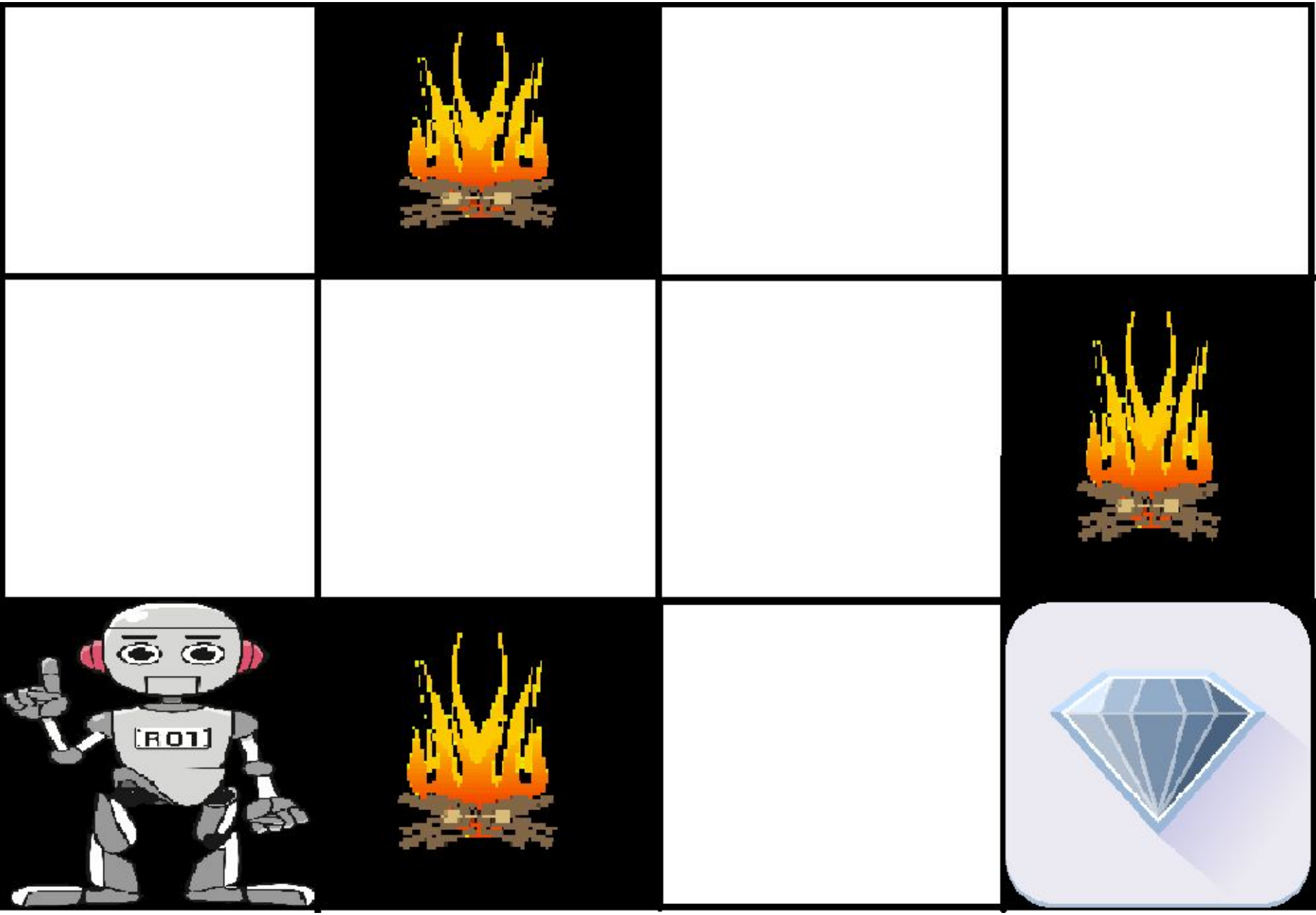

MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
DEEP LEARNING
REINFORCEMENT LEARNING

Supervised Learning vs RL

- Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where the feedback provided to the agent is correct set of actions for performing a task, reinforcement learning uses rewards and punishments as signals for positive and negative behavior.
- As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in the case of reinforcement learning the goal is to find a **suitable action model** that would maximize the total cumulative reward of the agent.

RL Problem Definition

- The problem is as follows:
  - We have an agent and a reward, with many hurdles in between.
  - The agent is supposed to find the best possible path to reach the reward.
  - The following problem explains the problem more easily.


R01

RL Problem Cont'd.

- The previous image shows the robot, diamond, and fire.
- The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fire.
- The robot learns by trying all the possible paths and then choosing the path which gives it the reward with the least hurdles.
- Each right step will give the robot a reward and each wrong step will subtract the reward of the robot.
- The total reward will be calculated when it reaches the final reward that is the diamond.

RL Main Points

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, the model will return a state and the user will decide to reward or punish the model based on its output.
- The model continues to learn.
- The best solution is decided based on the maximum reward.

Positive RL

- Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior.
- In other words, it has a positive effect on behavior.
- Advantages:
  - Maximizes Performance
  - Sustain Change for a long period of time
  - Too much Reinforcement can lead to an overload of states which can diminish the results

Negative RL

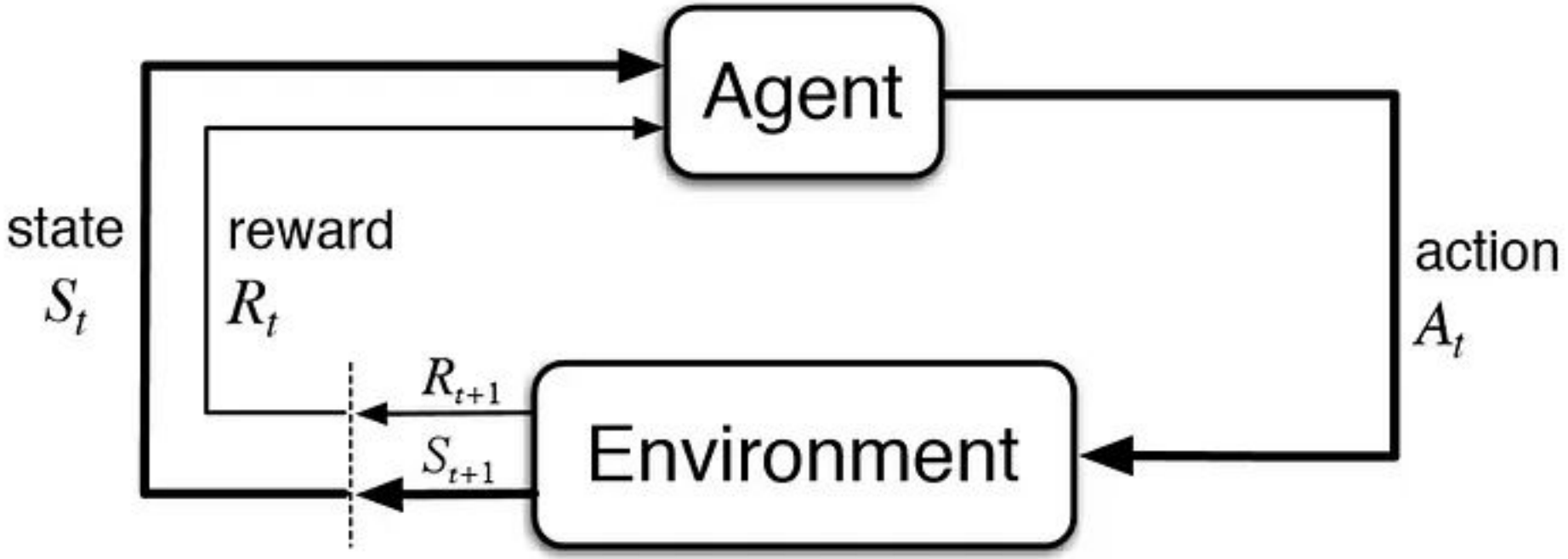
- Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.
- Advantages:
  - Increases Behavior
  - Provide defiance to a minimum standard of performance
  - It Only provides enough to meet up the minimum behavior

RL Terms

- Agent – the program controlling the object of concern (for instance, a robot).
- Environment — Physical world in which the agent operates
- State — Current situation of the agent
- Reward — Feedback from the environment
- Policy — Method to map agent's state to actions, e.g. the algorithm used by the agent to decide its actions.
- Value — Future reward that an agent would receive by taking an action in a particular state
  - What finance concept does this remind you?

Action-Reward Feedback

- The figure below illustrates the action-reward feedback loop of a generic RL model.



Exploration vs. Exploitation

- In order to build an optimal policy, the agent faces the dilemma of exploring new states while maximizing its overall reward at the same time.
- This is called Exploration vs Exploitation trade-off.
- To balance both, the best overall strategy may involve short term sacrifices.
- Therefore, the agent should collect enough information to make the best overall decision in the future.

RL applications in trading and finance

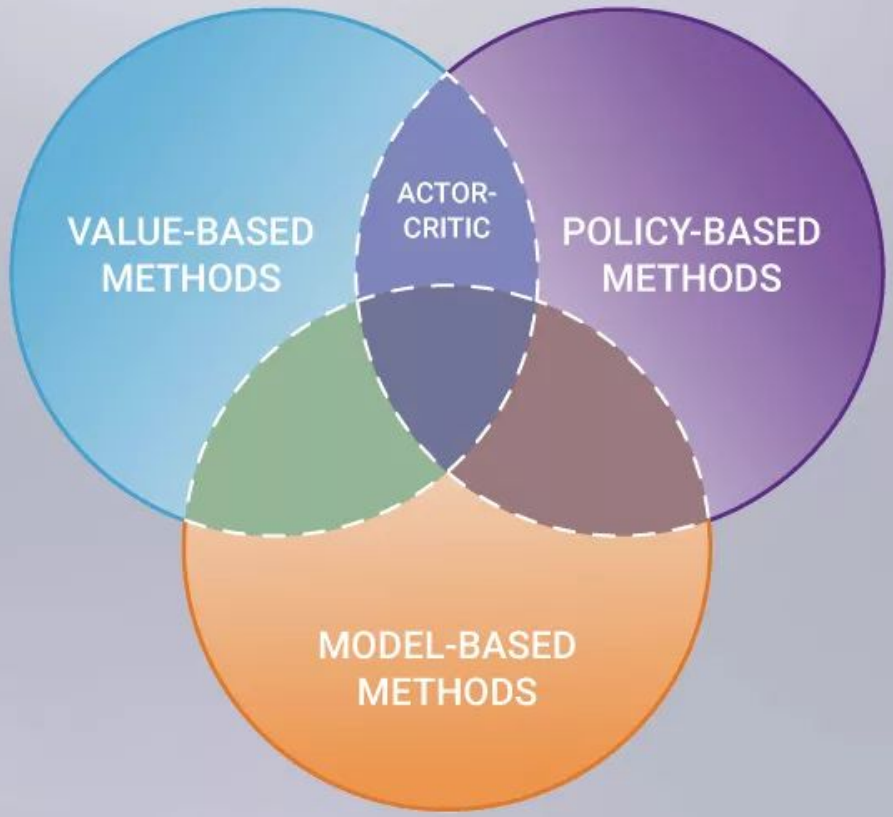
- Supervised time series models can be used for predicting future sales as well as predicting stock prices. However, these models don't determine the action to take at a particular stock price. Enter Reinforcement Learning (RL).
- An RL agent can decide on such a task; whether to hold, buy, or sell. The RL model is evaluated using market benchmark standards in order to ensure that it's performing optimally.
- This automation brings consistency into the process, unlike previous methods where analysts would have to make every single decision.
- IBM for example has a reinforcement learning based platform that has the ability to make financial trades. It computes the reward function based on the loss or profit of every financial transaction.
- Other examples: https://hackernoon.com/reinforcement-learning-10-real-reward-and-punishment-applications-d3203tqx

Finance Applications

- Market Simulation
  - Participants
    - https://en.wikipedia.org/wiki/Financial_market_participants
    - https://www.investopedia.com/terms/f/financial-market.asp
  - Agents
  - Synthetic Data (eg GAN generated)

RL Algorithms

SYNOPSYS®



RL Algorithms

- Model-based
  - Markov Decision Processes(MDPs)
- Model-free – real world environments are more likely to lack any prior knowledge of environment dynamics. Model-free RL methods come handy in such cases.
  - Q-learning
  - SARSA (State-Action-Reward-State-Action)
  - Deep Q-Networks(DQNs)
  - Deep Deterministic Policy Gradient(DDPG)

Model-based vs. Model-free

- Model-based, as it sounds, has an agent trying to understand its environment and creating a model for it based on its interactions with this environment. In such a system, preferences take priority over the consequences of the actions i.e. the greedy agent will always try to perform an action that will get the maximum reward irrespective of what that action may cause.

- On the other hand, model-free algorithms seek to learn the consequences of their actions through experience via algorithms such as Policy Gradient, Q-Learning, etc. In other words, such an algorithm will carry out an action multiple times and will adjust the policy (the strategy behind its actions) for optimal rewards, based on the outcomes.

Model-based vs. Model-free

- Think of it this way, if the agent can predict the reward for some action before actually performing it thereby planning what it should do, the algorithm is model-based. While if it actually needs to carry out the action to see what happens and learn from it, it is model-free.

- This results in different applications for these two classes, for e.g. a model-based approach may be the perfect fit for playing chess or for a robotic arm in the assembly line of a product, where the environment is static and getting the task done most efficiently is our main concern. However, in the case of real-world applications such as self-driving cars, a model-based approach might prompt the car to run over a pedestrian to reach its destination in less time (maximum reward), but a model-free approach would make the car wait till the road is clear (optimal way out).

Value-Based Algos

- Value-based algorithms consider optimal policy to be a direct result of estimating the value function of every state accurately.
- Using a recursive relation described by the Bellman equation, the agent interacts with the environment to sample trajectories of states and rewards.
- Given enough trajectories, the value function of the MDP can be estimated. Once the value function is known, discovering the optimal policy is simply a matter of acting greedily with respect to the value function at every state of the process. Some popular value-based algorithms are SARSA and Q-learning.

Policy-Based Algos

- Policy-based algorithms, on the other hand, directly estimate the optimal policy without modeling the value function. By parametrizing the policy directly using learnable weights, they render the learning problem into an explicit optimization problem.
- Like value-based algorithms, the agent samples trajectories of states and rewards; however, this information is used to explicitly improve the policy by maximizing the average value function across all states.
- Popular policy-based RL algorithms include Monte Carlo policy gradient (REINFORCE) and deterministic policy gradient (DPG). Policy-based approaches suffer from a high variance which manifests as instabilities during the training process. Value-based approaches, though more stable, are not suitable to model continuous action spaces.

Actor-Critic algorithm

- One of the most powerful RL algorithms, called the actor-critic algorithm, is built by combining the value-based and policy-based approaches.
- In this algorithm, both the policy (actor) and the value function (critic) are parametrized to enable effective use of training data with stable convergence.

RL

MARKOV DECISION PROCESSES (MDPS)

MDP Components

- Markov decision processes give us a way to formalize sequential decision making.
- This formalization is the basis for structuring problems that are solved with reinforcement learning.
- Components of an MDP:
  - Agent
  - Environment
  - State
  - Action
  - Reward

MDP Components

- In an MDP, we have a decision maker, called an **agent**, that interacts with the **environment** it's placed in.
- These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment's **state**.
- Given this representation, the agent selects an **action** to take.
- The environment is then transitioned into a new state, and the agent is given a **reward** as a consequence of the previous action.

Cumulative Reward

- This process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again, which creates something called a trajectory that shows the sequence of states, actions, and rewards.
- Throughout this process, it is the agent's goal to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the cumulative rewards it receives over time.
- **It is the agent's goal to maximize the cumulative rewards.**

MDP Notation

In an MDP, we have a set of states $\boldsymbol{S}$, a set of actions $\boldsymbol{A}$, and a set of rewards $\boldsymbol{R}$. We'll assume that each of these sets has a finite number of elements.

At each time step $t = 0, 1, 2, \cdots$, the agent receives some representation of the environment's state $S_t \in \boldsymbol{S}$. Based on this state, the agent selects an action $A_t \in \boldsymbol{A}$. This gives us the state-action pair (S_t, A_t).

Time is then incremented to the next time step $t + 1$, and the environment is transitioned to a new state $S_{t+1} \in \boldsymbol{S}$. At this time, the agent receives a numerical reward $R_{t+1} \in \boldsymbol{R}$ for the action A_t taken from state S_t.

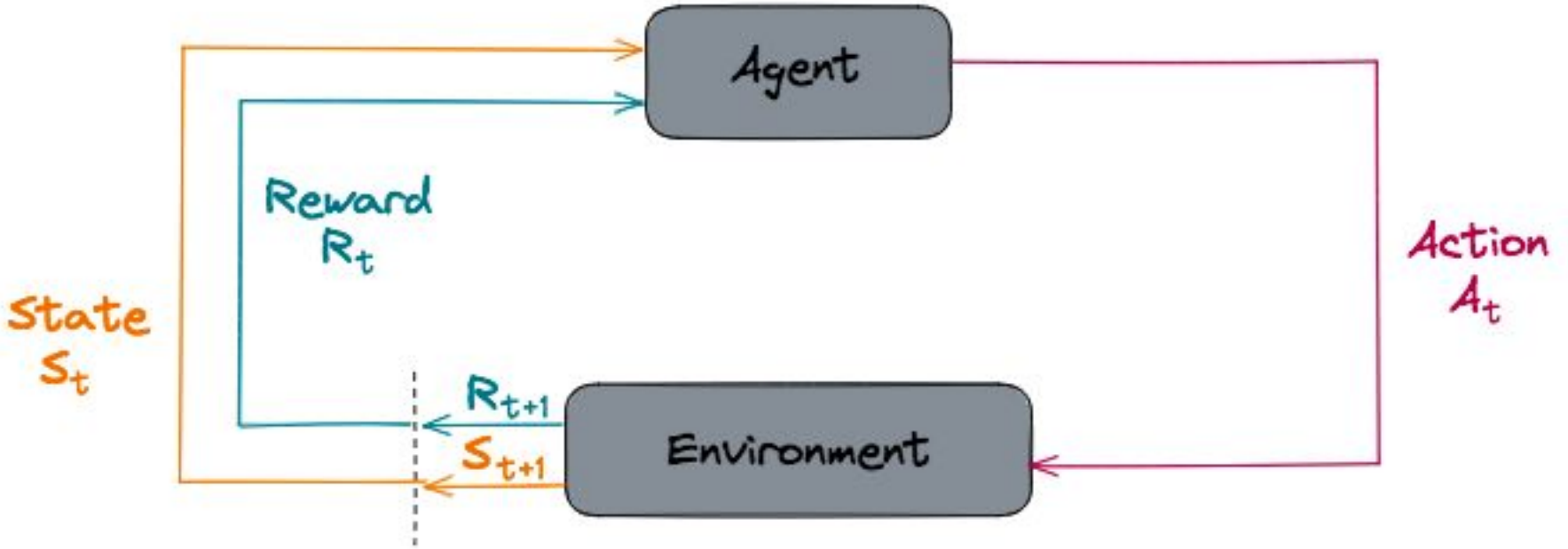
We can think of the process of receiving a reward as an arbitrary function f that maps state-action pairs to rewards. At each time t, we have

$$f(S_t, A_t) = R_{t+1}.$$

The trajectory representing the sequential process of selecting an action from a state, transitioning to a new state, and receiving a reward can be represented as

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$$

MDP State Diagram



- At time , the environment is in state.
- The agent observes the current state and selects action.
- The environment transitions to state and grants the agent reward.
- This process then starts over for the next time step,.

Transitions

- Transition : Moving from one state to another is called Transition.
- Transition Probability: The probability that the agent will move from one state to another is called transition probability.
- The Markov Property states that :
  “Future is Independent of the past given the present”
- The transition probabilities describe the dynamics of the world. They play the role of the next-state function in a problem-solving search, except that every state is thought to be a possible consequence of taking an action in a state.

Transition Probabilities

Since the sets $\boldsymbol{S}$ and $\boldsymbol{R}$ are finite, the random variables R_t and S_t have well defined probability distributions. In other words, all the possible values that can be assigned to R_t and S_t have some associated probability. These distributions depend on the *preceding* state and action that occurred in the previous time step $t-1$.

For example, suppose $s' \in \boldsymbol{S}$ and $r \in \boldsymbol{R}$. Then there is *some* probability that $S_t = s'$ and $R_t = r$. This probability is determined by the particular values of the *preceding* state $s \in \boldsymbol{S}$ and action $a \in \boldsymbol{A}(s)$. Note that $\boldsymbol{A}(s)$ is the set of actions that can be taken from state s.

Let's define this probability.

For all $s' \in \boldsymbol{S}$, $s \in \boldsymbol{S}$, $r \in \boldsymbol{R}$, and $a \in \boldsymbol{A}(s)$, we define the probability of the transition to state s' with reward r from taking action a in state s as:

$$p\left(s', r \mid s, a\right) = \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}.$$

RL Algo

Q LEARNING

Q Learning

- Q-learning is a reinforcement learning technique used for learning the optimal **policy** in a **Markov Decision Process**.
- Once we have our optimal Q-function we can determine the optimal policy by applying a reinforcement learning algorithm to find the action that maximizes for each state.
- The objective of Q-learning is to find a policy that is optimal in the sense that the expected value of the total reward over all successive steps is the maximum achievable.
- So, in other words, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair.
- https://deeplizard.com/learn/video/qhRNvCVVJaA

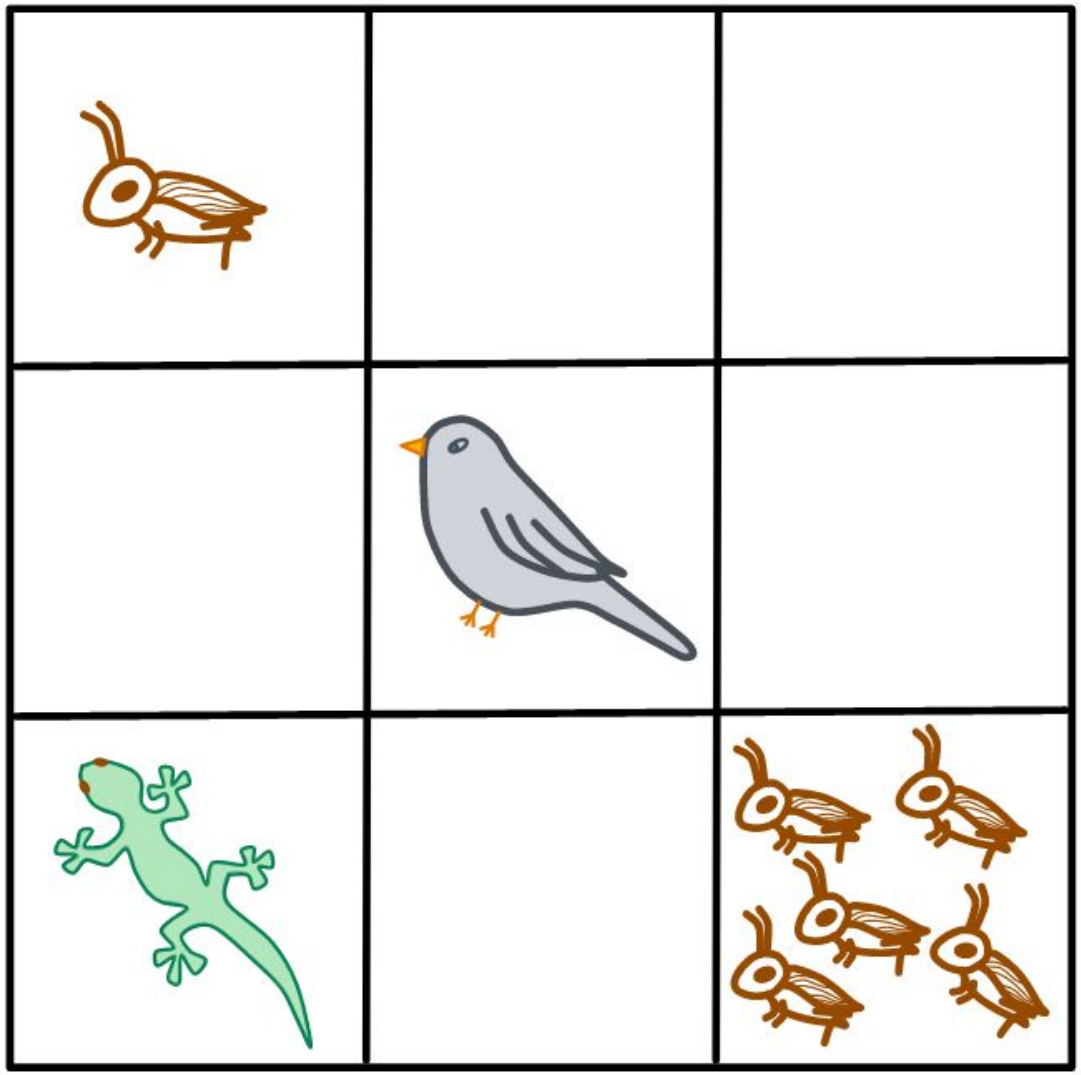
Q Value

- Q Value (Q Function): Usually denoted as Q(s,a) (sometimes with a π subscript, and sometimes as Q(s,a; θ) in Deep RL), Q Value is a measure of the overall expected reward assuming the Agent is in state s and performs action a, and then continues playing until the end of the episode following some policy π.
- Its name is an abbreviation of the word “Quality”, and it is defined mathematically as:

$$Q(s,a) = \mathbf{E}\left[\sum_{n=0}^{N} \gamma^n r_n\right]$$

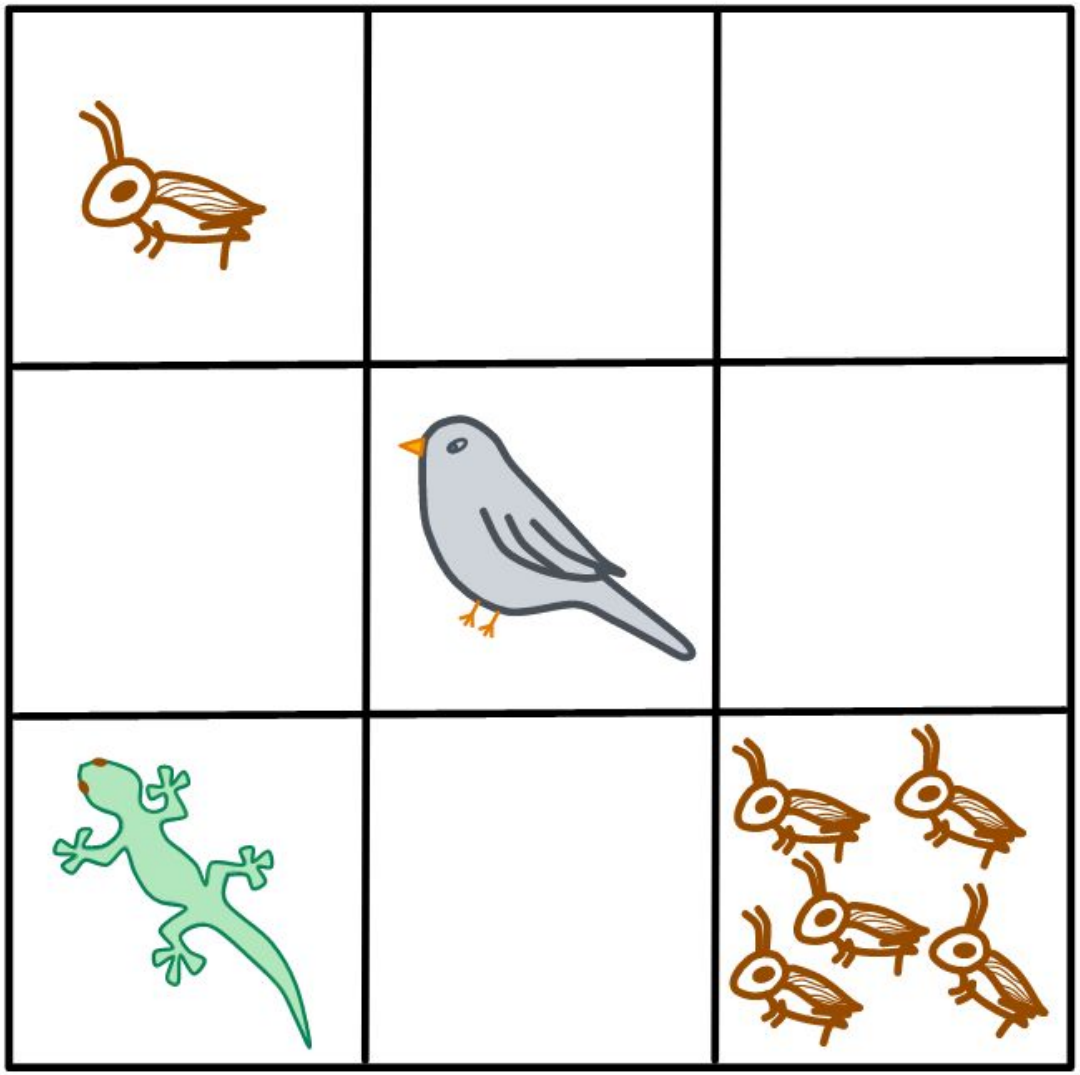
- where N is the number of states from state s till the terminal state, γ is the discount factor and r^0 is the immediate reward received after performing action a in state s.

The Lizard Game



- Suppose we have the following environment shown below.
- The agent in our environment is the lizard.
- The lizard wants to eat as many crickets as possible in the least amount of time without stumbling across a bird, which will, itself, eat the lizard.

The Lizard Game



- The lizard can move left, right, up, or down in this environment. These are the actions.
- The states are determined by the individual tiles and where the lizard is on the board at any given time.

State/Reward

- If the lizard lands on a tile that has one cricket, the reward is plus one point. Landing on an empty tile is minus one point. A tile with five crickets is plus ten points and will end the episode. A tile with a bird is minus ten points and will also end the episode.

State	Reward
One cricket	+1
Empty	- 1
Five crickets	+10 Game over
Bird	- 10 Game over

Initial State

- Now, at the start of the game, the lizard has no idea how good any given action is from any given state. It's not aware of anything besides the current state of the environment.
- In other words, it doesn't know from the start whether navigating left, right, up, or down will result in a positive reward or negative reward.
- Therefore, the Q-values for each state-action pair will all be initialized to zero since the lizard knows nothing about the environment at the start.
- Throughout the game, though, the Q-values will be iteratively updated using value iteration.

Storing Q-values in a Q-table

States	Actions: Left	Right	Up	Down
1 cricket	0	0	0	0
Empty 1	0	0	0	0
Empty 2	0	0	0	0
Empty 3	0	0	0	0
Bird	0	0	0	0
Empty 4	0	0	0	0
Empty 5	0	0	0	0
Empty 6	0	0	0	0
5 crickets	0	0	0	0

Updating the Q-table

- As just mentioned, since the lizard knows nothing about the environment or the expected rewards for any state-action pair, all the Q-values in the table are first initialized to zero.
- Over time, though, as the lizard plays several episodes of the game, the Q-values produced for the state-action pairs that the lizard experiences will be used to update the Q-values stored in the Q-table.
- As the Q-table becomes updated, in later moves and later episodes, the lizard can look in the Q-table and base its next action on the highest Q-value for the current state.

Episodes

- Now, we'll set some standard number of episodes that we want the lizard to play. Let's say we want the lizard to play five episodes. It is during these episodes that the learning process will take place.
- In each episode, the lizard starts out by choosing an action from the starting state based on the current Q-values in the table. The lizard chooses the action based on which action has the highest Q-value in the Q-table for the current state.
- But, wait... That's kind of weird for the first actions in the first episode, right? Because all the Q-values are set zero at the start, so there's no way for the lizard to differentiate between them to discover which one is considered better. So, what action does it start with?
- To answer this question, we'll introduce the trade-off between exploration and exploitation. This will help us understand not just how an agent takes its first actions, but how exactly it chooses actions in general.

Exploration vs. Exploitation

- Exploration is the act of exploring the environment to find out information about it. Exploitation is the act of exploiting the information that is already known about the environment in order to maximize the return.
- The goal of an agent is to maximize the expected return, so you might think that we want our agent to use exploitation all the time and not worry about doing any exploration. This strategy, however, isn't quite right.
- Think of our game. If our lizard got to the single cricket before it got to the group of five crickets, then only making use of exploitation, going forward the lizard would just learn to exploit the information it knows about the location of the single cricket to get single incremental points infinitely. It would then also be losing single points infinitely just to back out of the tile before it can come back in to get the cricket again.
- If the lizard was able to explore the environment, however, it would have the opportunity to find the group of five crickets that would immediately win the game. If the lizard only explored the environment with no exploitation, however, then it would miss out on making use of known information that could help to maximize the return.
- Given this, we need a balance of both exploitation and exploration.

DQN

- A more complex version of it, though far more popular, is the ***Deep Q-Network*** variant (which is sometimes even referred to simply as *Deep Q-Learning* or just *Q-Learning*). This variant replaces the state-action table with a neural network in order to cope with large-scale tasks, where the number of possible state-action pairs can be enormous. You can find a tutorial for this algorithm in this blogpost.

Gymnasium

- https://gymnasium.farama.org/
- Gymnasium is a standard API for reinforcement learning, and a diverse collection of reference environments

Further Reading

- https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e
- https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning
- https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
- https://www.geeksforgeeks.org/what-is-reinforcement-learning/
- https://www.synopsys.com/ai/what-is-reinforcement-learning.html
- https://deeplizard.com/learn/video/qhRNvCVVJaA