# Intro to Reinforcement Learning

Inspired by UMass CS687

David Koleczek

December 18, 2020

## What is Reinforcement Learning (RL)?

*Reinforcement learning is an area of machine learning, inspired by behaviorist psychology, concerned with how an agent can learn from interactions with an environment.*

–Wikipedia, Sutton and Barto (1998), Phillip Thomas

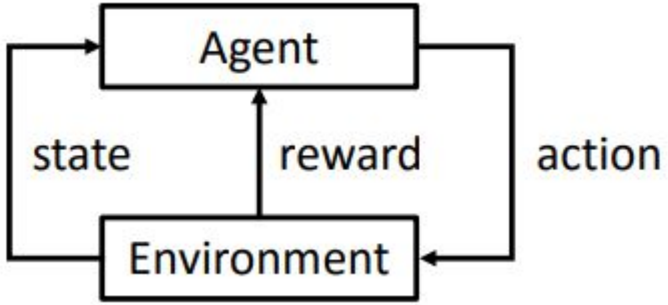


*Figure*: Agent-environment diagram.

Examples of agents include a child, dog, robot, **program**
Examples of environments include the world, lab, **software environment**

## What makes RL, RL?

**Evaluative Feedback**

- Rewards convey how “**good**” an agent’s actions are
    - Agent will try to maximize its total reward
- Agent is **not** told what to the best actions would have been

Where do we commonly tell an “agent” what it should have done (give instructive feedback)?

- Supervised Learning!
    - Features can be thought of as state of the environment
    - Target is like the best action given those features.

## What makes RL, RL?

**Sequential**

- The *entire* sequence of actions must be optimized to maximize the total reward the agent obtains.

What are the implications?

- Might require forgoing immediate rewards to obtain larger rewards later.
    - Stanford Marshmallow Experiment - would you take 1 cookie right now, or get 2 a month later?
- The way the agent selects actions changes the distribution of states it sees.
    - Means many RL problems are not provided as fixed datasets, but instead as code or descriptions of the environment
- (Anecdote) Makes RL algorithms hard to parallelize

# Example of an Agent-Environment

**State**: Position of the dog, coordinate like (2,1)

- The door is a special, terminal state ($s_\infty$)

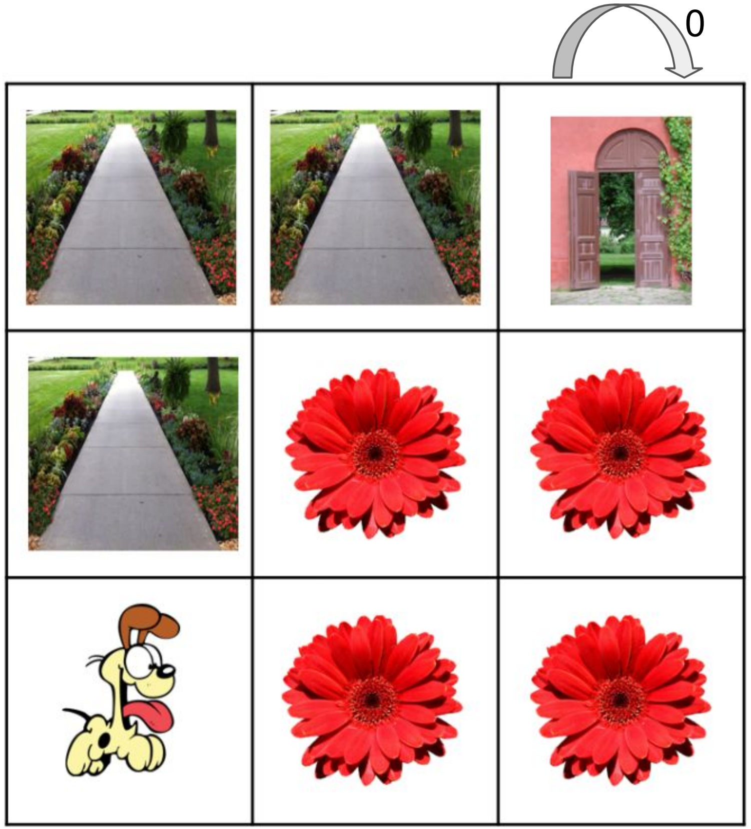
**Actions**: Move up, down, left, or right

**Environment Dynamics**

- With probability 0.10 the dog gets distracted and moves right regardless of the action selected

**Rewards**

- Let's think about some options...

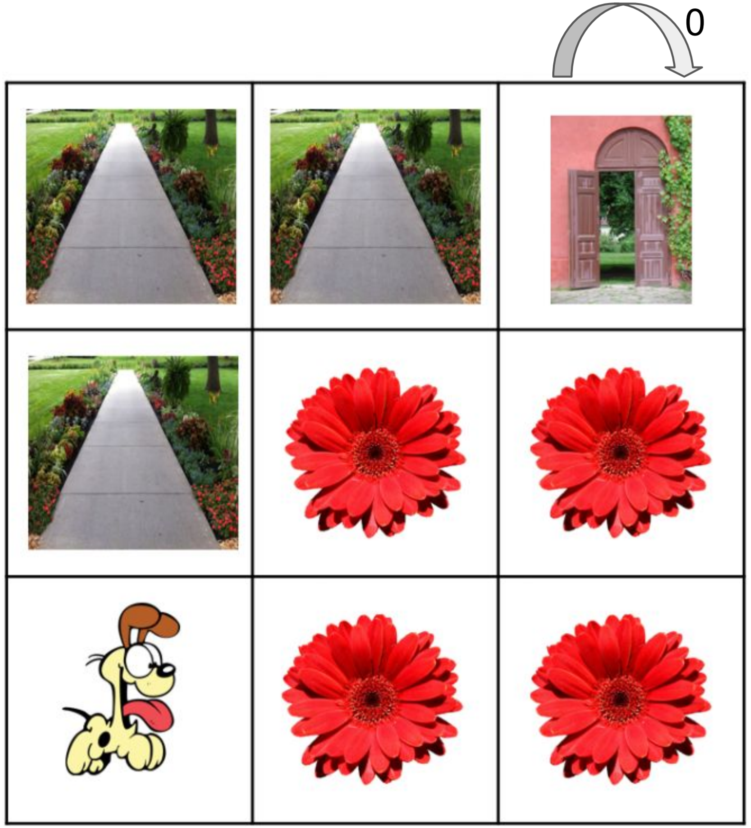


# Example of an Agent-Environment

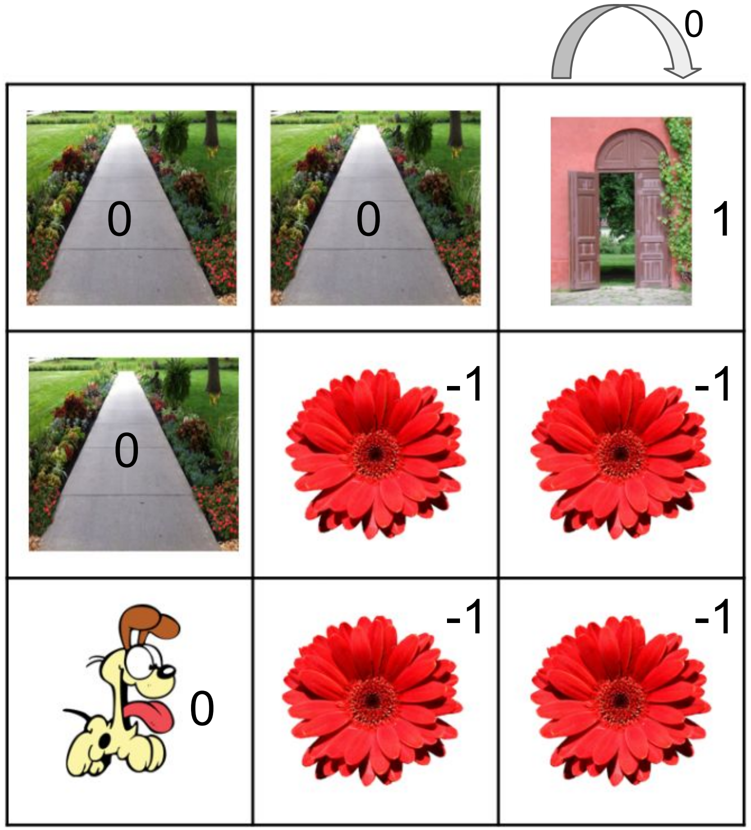
**Rewards**

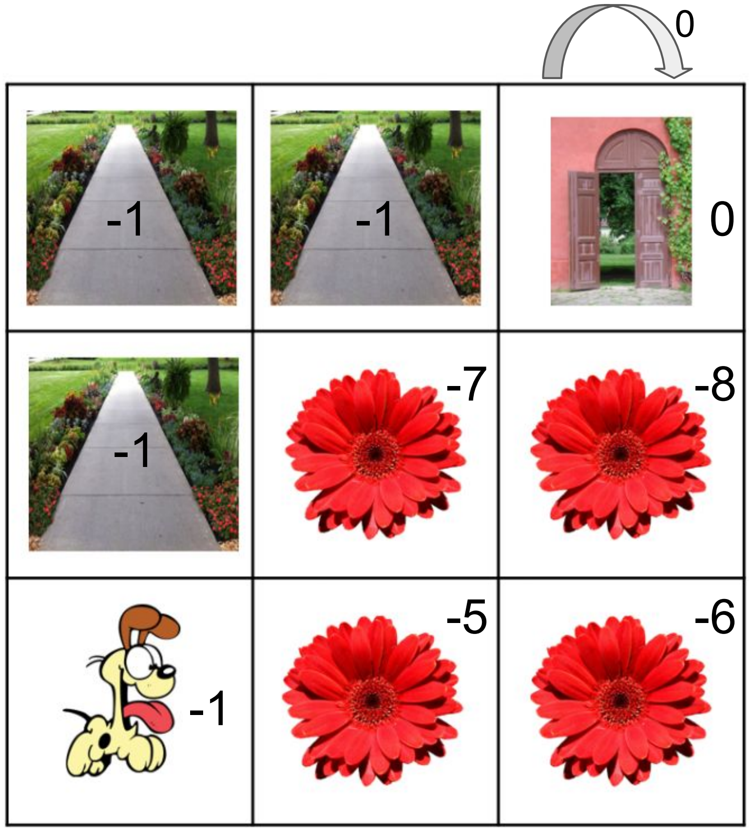
- We want to place rewards such that the dog will reach the door by taking the sidewalk and not ruining our flowers!

- **Question**: With this Reinforcement Learning setup, how would you assign rewards for each state so that the dog reaches the **door**?



# Example Solutions





## Takeaways on Rewards

Give rewards for what you want the agent to achieve, not for how you think the agent should achieve it

Rewards that are given to help the agent quickly identify what behavior is optimal is related to *reward shaping*.

Example blog post in robotics on how reward shaping can lead to undesired behavior

Pessimistic blog post on why getting RL to work and choosing rewards is hard

## Describing the Environment Formally

In order to reason about learning, we have to describe the environment and agent using math.
Of many mathematical models that can be used to describe an **environment**, *Markov Decision Processes* (MDPs) are the most common, simple, and flexible

- Capture a wide range of real and interesting problems

Specific definitions and notations vary, but usually share these common terms in a tuple:

- $(S, A, p, R, d_0, \gamma)$

## Describing the Environment Formally

$MDP := (S, A, p, R, d_0, \gamma)$

$S$ is the set of all possible states of the environment

- The **state** at **timestep** $t$ is $S_t$ and always takes values in $S$
- $s_\infty$ is the **terminal absorbing state**, a special state once the agent is in it, it can never leave and always gets a reward of 0

$A$ is the set of all possible actions an agent can take

- The action at timestep $t$ is $A_t$ and always takes values in $A$

## Describing the Environment Formally

*p* is the **transition function**, describes how the state of the environment changes

For all $s \in S$, $a \in A$, $s' \in S$, and $t \in \mathbb{N}_{\geq 0}$:

$$p(s, a, s') := \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$$

*R* describes how rewards are generated:

$$R(s, a) := \mathbf{E}[R_t \mid S_t = s, A_t = a]$$

*Note*: In RL at least *p* is not known by the agent, and *R* is usually also not known

- If *p* and *R* are known, this is planning, where the agent does not need to interact with its environment

## Describing the Environment Formally

$d_0$ is the initial state distribution

- For all s, $d_0(s) = \Pr(S_0 = s)$

$\gamma \in [0, 1]$ is a parameter called the *reward discount parameter*

## Describing the Agent Formally

A **policy** is a way that an agent can select actions

For all $s \in S$, $a \in A$, and $t \in \mathrm{N}_{\geq 0}$:

$$\pi(s, a) := \Pr(A_t = a \mid S_t = s)$$

An agent's goal is to find the **optimal policy**, $\pi^*$

## Describing the Agent Formally

Discounted Return $\rightarrow$ Objective function

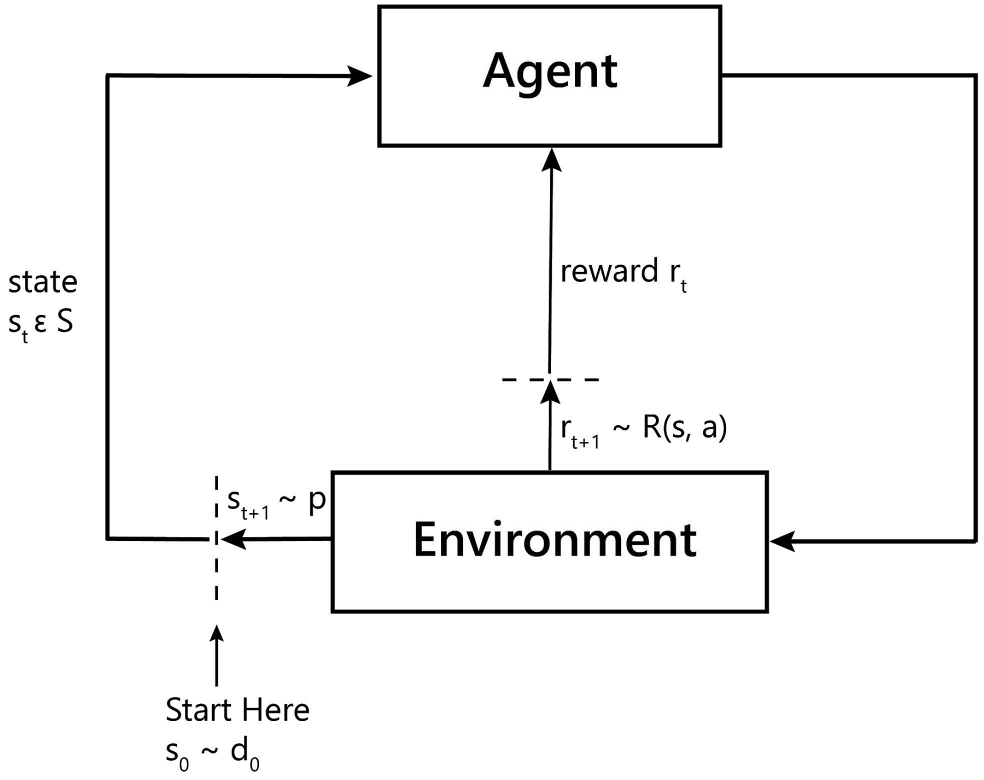
$G := \sum_{t=0}^{\infty} \gamma^t R_t$ $J(\pi) := \mathbf{E}[G|\pi]$

- Notice that $\gamma < 1$ means that rewards received during later timesteps are worth less to the agent.
  - Also ensures $J(\pi)$ is bounded, in practice lower $\gamma$ might result in faster learning

An optimal policy is any policy that satisfies

$$\pi^* \epsilon \arg\max_{\pi \epsilon \Pi} J(\pi)$$

# Concept Map



Agent's Goal:
Find $\pi^* \in \operatorname{argmax}_{\pi \in \Pi} J(\pi)$
$J(\pi) := \mathbf{E}[G \mid \pi]$
$G := \sum_{t=0}^{\infty} \gamma^t R_t$

# Preview of RL Algorithms: Finding an Optimal Policy

*Black-box optimization (BBO)*; not a class of algorithms specific to RL

Generic optimization algorithms that solve problems of the form: $\arg\max_{x \in \mathbb{R}^n} f(x)$

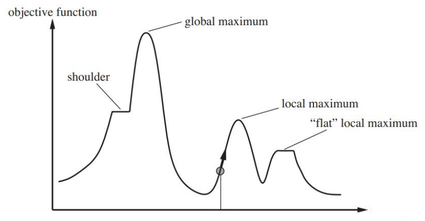
- Called *black-box* because they treat *f* as a block-box,
- i.e. do not leverage any knowledge about its structure

Fun Fact: Active area of research even in RL, largely for hyperparam optimization (*Bayes Opt*)

- Nvidia's Solution for the BBO competition at NeurIPS 2020

One of the simplest possible BBO algorithms applied to RL: *Hill-Climbing Search*

- Start with some initial policy (ex. uniform random)
- Run that policy to get G, the discounted return
- while(true): Generate a new policy (?), get G again
    - if (G of new policy > G of old policy): keep new policy

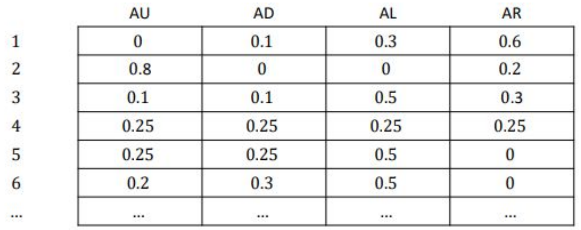


Russell & Norvig (2009), pg 129

# Preview of RL Algorithms

*Action-value function* or *Q-function* based methods

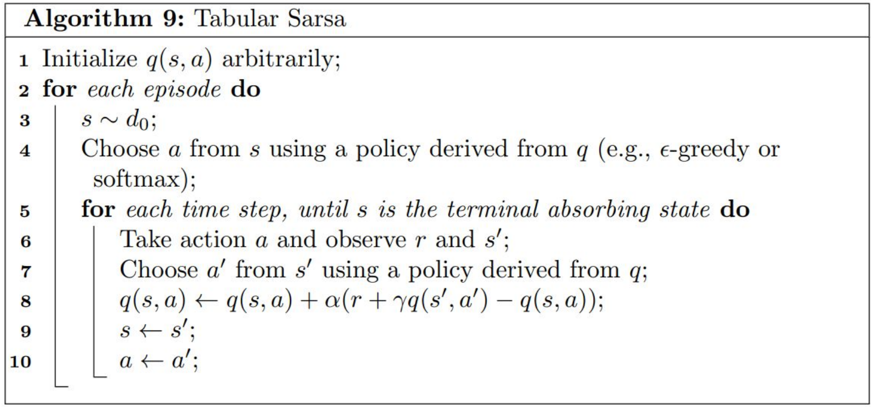
$$q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$
$$q^\pi(s, a) := \mathbf{E}[G_t | S_t = s, A_t = a, \pi],$$



| | AU | AD | AL | AR |
|---|---|---|---|---|
| 1 | 0 | 0.1 | 0.3 | 0.6 |
| 2 | 0.8 | 0 | 0 | 0.2 |
| 3 | 0.1 | 0.1 | 0.5 | 0.3 |
| 4 | 0.25 | 0.25 | 0.25 | 0.25 |
| 5 | 0.25 | 0.25 | 0.5 | 0 |
| 6 | 0.2 | 0.3 | 0.5 | 0 |
| ... | ... | ... | ... | ... |

Example *Q*-function: Each cell denotes expected discounted return from state *s* taking action *a* (values are not usually probabilities as shown here)

Algorithms: Sarsa, *Q*-Learning, Sarsa(λ), *Q*-Learning(λ)

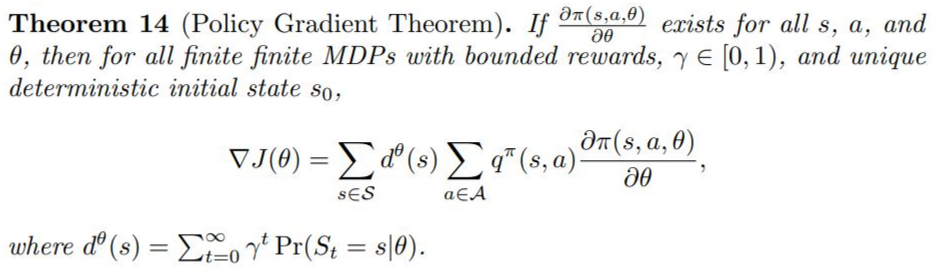


**Algorithm 9:** Tabular Sarsa

```
1  Initialize $q(s, a)$ arbitrarily;
2  for each episode do
3      $s \sim d_0$;
4      Choose $a$ from $s$ using a policy derived from $q$ (e.g., $\epsilon$-greedy or softmax);
5      for each time step, until $s$ is the terminal absorbing state do
6          Take action $a$ and observe $r$ and $s'$;
7          Choose $a'$ from $s'$ using a policy derived from $q$;
8          $q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a))$;
9          $s \leftarrow s'$;
10         $a \leftarrow a'$;
```

# Preview of RL Algorithms

Policy Gradient Methods

- Learn a parameterized policy that can select actions without consulting a value function
- Maximize the policy parameter such that the updates approximate gradient ascent on J: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}$
- 

**Theorem 14** (Policy Gradient Theorem). *If $\frac{\partial \pi(s,a,\theta)}{\partial \theta}$ exists for all $s$, $a$, and $\theta$, then for all finite finite MDPs with bounded rewards, $\gamma \in [0,1)$, and unique deterministic initial state $s_0$,*

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} d^\theta(s) \sum_{a \in \mathcal{A}} q^\pi(s,a) \frac{\partial \pi(s,a,\theta)}{\partial \theta},$$

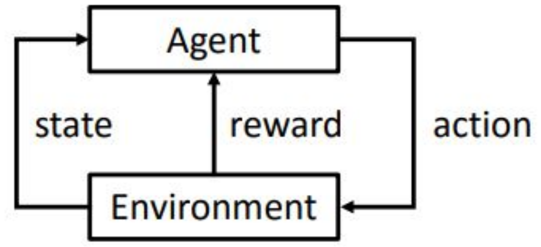
*where $d^\theta(s) = \sum_{t=0}^{\infty} \gamma^t \Pr(S_t = s | \theta)$.*

Algorithms: REINFORCE, REINFORCE w/Baseline, Actor-Critics

## Learning from Data

Frequently in RL environments used in research and examples

- Provide any action and immediately get a new state and reward
- Can “query” environment infinitely often (games, robot simulations)



OpenAI Gym Example

```
import gym
env = gym.make("CartPole-v1")
state = env.reset()
for _ in range(1000):
  action = my_action_selection() # your agent here
  state, reward, done, info = env.step(action)

  if done:
    state = env.reset()

env.close()
```

## Learning from Data

That type of setup is unrealistic in a lot of cases…

- (Opinion) A hard part of RL is setting up your problem
- Designing rewards, determining possible actions, state representation

What if we do have a fixed dataset?

- Equate it as needing to go through a process similar to feature engineering
- Raw data → data that can be used (well) by an algorithm

# Learning from Data

Have output from some *behavior policy*

- Goal is to improve that policy

| | | |
|---|---|---|
| Episode 1 | | |
| $S_0$ | $A_0$ | $R_0$ |
| $S_1$ | $A_1$ | $R_1$ |
| ... | ... | ... |
| Episode 2 | | |
| $S_0$ | $A_0$ | $R_0$ |
| $S_1$ | $A_1$ | $R_1$ |
| ... | ... | ... |
| Episode ... | | |

$\longrightarrow$

| | | | |
|---|---|---|---|
| Episode 1 | | | |
| $S_0$ | $A_0$ | $R_0$ | $S_1$ |
| $S_1$ | $A_1$ | $R_1$ | $S_2$ |
| ... | ... | ... | ... |
| Episode 2 | | | |
| $S_0$ | $A_0$ | $R_0$ | $S_1$ |
| $S_1$ | $A_1$ | $R_1$ | $S_2$ |
| ... | ... | ... | ... |
| Episode ... | | | |

Monte Carlo and Policy Gradient algorithms like REINFORCE

- Dataset on the left ends up being all you need

*Off-policy* algorithms like *Q*-Learning

- Also need the next state
  - Trivial to derive given the left dataset

*On-policy* algorithms like Sarsa

- Same data requirement as *Q*-Learning
- But assume you take actions according to the policy you are learning, which isn't the case since we have data from an arbitrary policy

## Learning from Data

Common Case: All you have is data and a problem to solve!

Example: Automated trading in financial/cryptocurrency markets[Inspiration 1 and 2]

- Trading happens in a double auction with an open order book
- Buyers and sellers get matched so they can trade with each other on an exchange

Data:

- Trade history (for a single asset on an exchange)
  - Each trade has a timestamp, size, price, and direction (buy/sell)
- Order book
  - List of who is willing to buy or sell and at what price, alongside a volume
  - Asks (offers): People willing to sell
  - Bids: People willing to buy
  - Note: difference between the best ask and best bid is called the spread and best ask > best bid or else a trade would have already happened

## Learning from Data

**Our Goal**: Create an agent that takes actions in the exchange to maximize profits

**State**: History of all exchange events (trades and order book) and current account balance at the current timestep

- Unknown to me: Good ways to parameterize/encode the history into a vector

**Actions**: Issue a *market* order: buy or sell at the best price(s) possible *right now*
**Reward**: Realized PnL (Profit and Loss)

- Ex.) If buy order then: reward is the inverse of value purchased

Create a “dataset” of sequential states that you can apply an RL algorithm to, which will give you a policy (mapping states to actions).
Agent will provide actions based on what RL algorithm you choose. Environment can provide the rewards as you take actions.

## Alternative to Supervised Learning? Think twice...

We might be tempted to take a supervised learning problem and convert it into an RL problem in order to apply sophisticated RL methods

Example for classification:

- State as the features
- Action as the label
- Reward is 1 if correct, -1 otherwise

Although possible and a valid use of RL, this will very likely give you bad results!

## RL or Supervised?

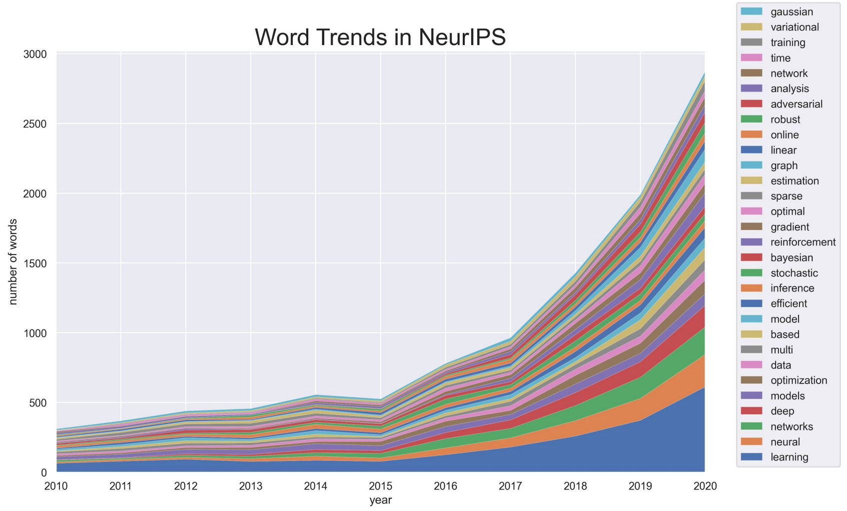
1.) Type 1 Diabetes treatment, where the goal is to determine how much insulin an insulin pump should inject in order to keep a person’s blood glucose levels near optimum levels. Data is current blood glucose + carbs.

2.) Digital marketing, where the goal is to present ads on a website that are likely to be clicked by the user. Assume that we have some knowledge about the user, like their age and gender.

3.) Determining the composition of rocks using spectroscopy. For this task the agent observes spectra collected from a rock and makes a decision about what minerals it believes are present. There are data sets containing examples of spectra of rocks with known compositions, but these data sets are small.

# Conclusion

- Hopefully have some foundational knowledge needed to figure out if RL could a good solution to some problem
- Able to dig deeper into textbooks or papers, lots I didn't cover.
- RL is still new, but growing fast!
- Lots of room for new discoveries/applications, or library development (state of RL is libraries is poor, unlike deep learning)


Word Trends in NeurIPS
number of words
year
gaussian
variational
training
time
network
analysis
adversarial
robust
online
linear
graph
estimation
sparse
optimal
gradient
reinforcement
bayesian
stochastic
inference
efficient
model
based
multi
data
optimization
models
deep
networks
neural
learning