parity

technologies ltd

RustFest AI & ML Workshop

AIMED AT BEGINNERS OF RUST AND ML

Not suitable for you if you're a Rust or ML expert, start looking at `leaf` crate

LEARN TO WRITE A DECISION-TREE

We'll use the `id_tree` crate to create a decision tree for one of three games

LEARN ABOUT REINFORCEMENT LEARNING (Q-LEARNING)

We'll use the `renforce` crate to build a bot that learns a simple "taxi" game

parity
technologies ltd

Setup

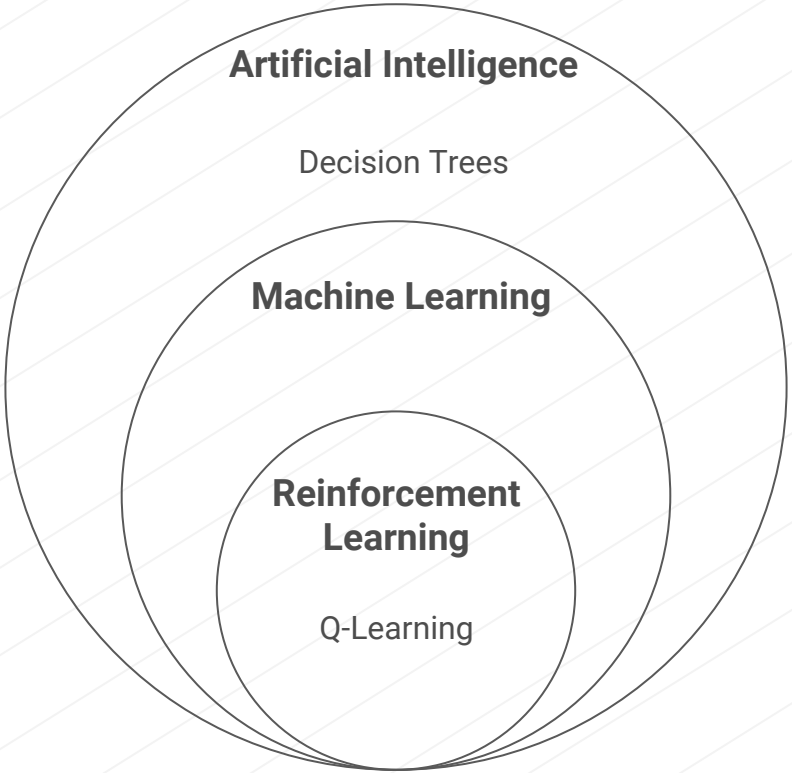
INSTALL RUSTUP
https://www.rustup.rs/

DOWNLOAD GAMES & SKELETON
Either download the zip of the source code from Github or:
`git clone https://github.com/folsen/rustfest2017/`

I will publish working solutions on a branch here at some point during the workshop

parity
technologies ltd

Difference between AI and ML (not rigorous)



parity
technologies ltd

Decision Trees

GENERAL IDEA

- Make every possible move, save the outcome of that move
- For every move made, make every possible subsequent move and save outcomes
- Recurse until game is over or some depth limit is hit

IN PRACTICE

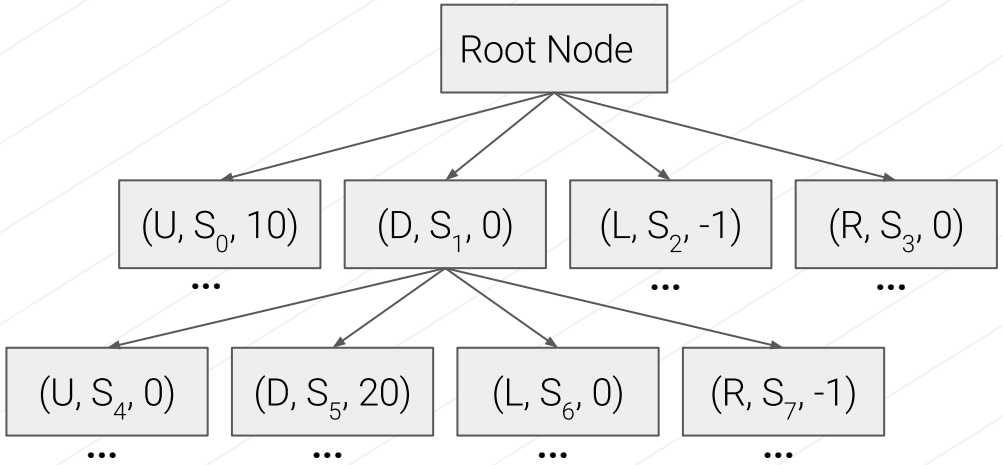
- We use an index-tree to allow arbitrary number of nodes per level
- Don’t add illogical moves to the tree (such as walking into a wall)
- Limit the depth to something computationally reasonable (# of nodes grows exponentially), pruning unused branches as you traverse the tree is very important.
- Come up with a utility function to be able to select moves with good scores

Decision Trees

A Node in the tree is some combination of the move taken, the resulting state (or whatever part is necessary to store from the state) and a score for that node.

The correct path to take through the tree will usually depend on the score at the leaf node (the end) of the tree.

There may be several moves that don't affect the score, and we may need to make suboptimal moves to reach an optimal state.



This tree grows in size at speed 4^n
Calculating n=10 moves ahead requires more than 1 million nodes

parity
technologies ltd

Let’s get coding!

Start implementing your decision tree for one of the games, roguelike is recommended but bejeweled is also fun.

I’ll interrupt about halfway to talk about Q-Learning

parity
technologies ltd

Q-Learning

GENERAL IDEA

- For a given state, take an action – random or selected by some strategy
- Record the utility of said action for given state
- Repeat for given number of iterations/episodes

IN PRACTICE

We'll use the `renforce` crate and provide it with ways to make observations on our environment through taking actions and calculating utility.

With more time, it would be feasible to build a Q-Learning library from scratch pretty easily, but for the sake of getting some nice results quickly, we'll use something off-the-shelf.

parity
technologies ltd

Q-Learning

Q-Learning is a model-free form of reinforcement learning. We don't give the algorithm any information about how to play the game, we just give it a reward for when it wins.

The algorithm is based around a state-action value (the Q-value). For any state, say a position `(x, y)`, we look at all the possible actions and what reward they would bring.

Even if only winning gives a reward, eventually a state-action matrix would be filled that scores every move with their expected utility (Q). Then we play the game selecting moves with highest expected utility.

State \ Action	Up	Right	Down	Left
(1,1)	-0.76	0.63	0.63	-0.75
(1,2)	-0.63	0.77	0.44	-0.86
(3,3)	-0.82	0.41	0.41	-0.82
(3,4)	-0.41	1	-0.41	-1

Goal is getting to (4,4).
Example uses Euclidean distance as reward function, produces immediate and accurate utility estimates, but not possible for all problems.

parity
technologies ltd

Q-Learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Action

State	Up	Right	Down	Left
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	1	0	0

1st episode

Action

State	Up	Right	Down	Left
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(3,3)	0.81	0.81	0.81	0.81
(3,4)	0	1	0	0

= α * γ * 1,
α = 0.9,
γ = 0.9

2nd episode

parity
technologies ltd

Get back to coding!

parity
technologies ltd

THANK YOU

fredrik@parity.io - marek@parity.io

parity
technologies ltd