lab_ml: Lazy Machine Learning

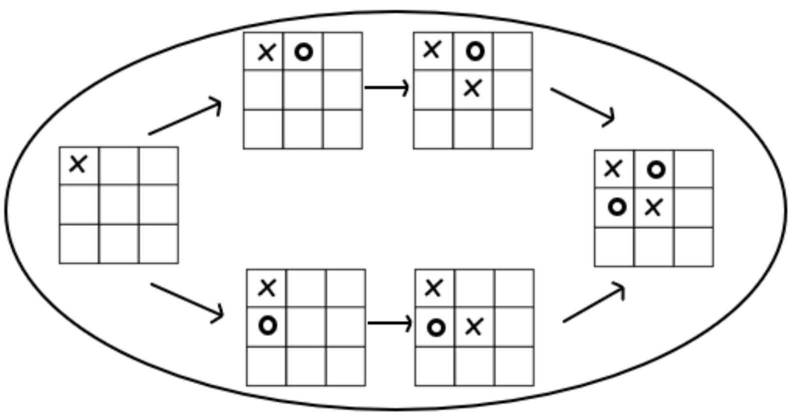
Welcome to Lab Machine Learning!

Overview:

In this lab you will learn how to teach computer how to learn to win a game. You will use a graph to represent a state space.

Using a graph as a state space:

Before an AI problem can be solved it must be represented as a state space. The state space is then searched to find a solution to the problem. A state space essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state and a goal state. Each state space takes the form of a tree or a graph. For visualization take a look partial state space for tic-tac-toe:

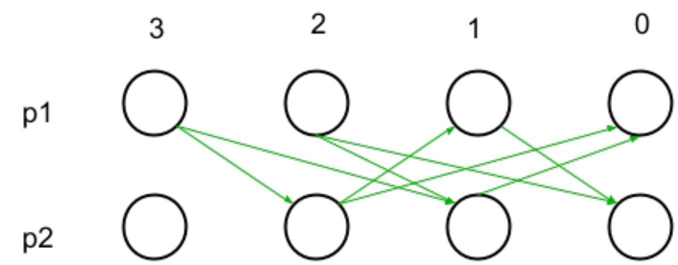


The Game of Nim

A game starts with **k** tokens. Players alternate turns with **Player 1** starting the game. Each turn, a player may pick up **1 or 2 tokens**. The player who picks up last token wins.

Exercise 1.1: How would you represent each state in this game?
HINT: What do we need to keep track of in each state?
Represent each of the states using the player number and the tokens.

Exercise 1.2: Connect the states in the following state space graph for a game with starting tokens **k = 3: Nim(3)**



Exercise 1.3: Which states are logically unreachable?
p1-2 and p2-3

Reinforcement learning:

Finally, we need to apply reinforcement learning. In reinforcement learning, an algorithm is rewarded for making a good decision and punished for making a poor decision. We will define a good decision as all decisions made by the player who won. Therefore, if Player 1 took the last token, all choices made by Player 1 are rewarded.

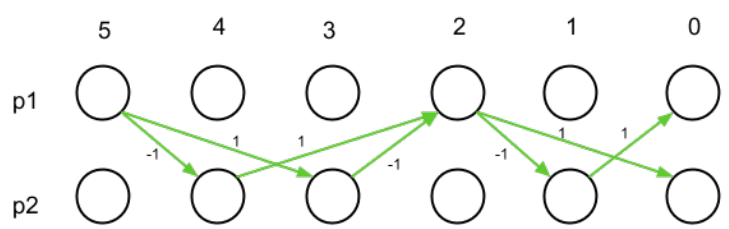
The reward is captured in our algorithm as the edge weight. When we consider a path through the graph, we can find that all edges along a path that has Player 1 winning (eg: the last vertex in the path goes to Player 2 with no tokens remaining, or "p2-0", meaning that Player 1 took the last token), then all choices made by Player 1 (edges where Player 1 is the source vertex) are rewarded by increasing the edge weight by +1 and all choices made by Player 2 are punished by changing the edge weight by -1.

Exercise 2.1:
Let's label the state *"Player 1 - 5 tokens available"* as `p1-5`.
What is the label of the state where `p1` wins? What about where `p2` wins?

`When p1 wins - p2-0`
`When p2 wins - p1-0`

Exercise 2.2: Given initial edge weights as 0, what will be updated edge weights after the next two games :

```
1. p1-5 -> p2-4 -> p1-2 -> p2-1 -> p1-0
2. p1-5 -> p2-3 -> p1-2 -> p2-0
```



Exercise 2.3: Given the following edge weights for a game `Nim(5)`, find how the trained players would play. Give the path they will follow.
Remember the start state is `p1-5`:

	p1-5	p1-4	p1-3	p1-2	p1-1	p1-0	p2-5	p2-4	p2-3	p2-2	p2-1	p2-0
p1-5								-3000	6000			
p1-4									9600	400		
p1-3										2	-1700	
p1-2											-15000	15000
p1-1												35000
p1-0												
p2-5		-5500	10000									
p2-4			7000	-10								
p2-3				-100	-15000							
p2-2					-18000	18000						
p2-1						32000						
p2-0												

`p1-5 -> p2-3 -> p1-2 -> p2-0`

AFTER YOU'RE DONE WITH LAB CODING:
Exercise 2.4: Would you prefer to go first or second in Nim(10)?

I would prefer to be the first player and take one token to end up in state p2-9.

In the programming part of this lab, you will:

- Using a graph as a state space
- Reinforcement learning
- How to teach a computer how to learn to win the game of Ni
- Implement next functions:
  - NimLearner constructor - which creates the vertices and edges for the state space of a game of Nim;
  - playRandomGame - which returns a random path through the graph of the state space as a vector<Edge>.
  - updateEdgeWeights - which updates the edge weights along a given path on the graph of the state space.

As your TA and CAs, we're here to help with your programming for the rest of this lab section! 😊