UCT and Beyond

David Silver

The Goal of This Talk

- What is UCT? (No theory!)
- Where does UCT belong in the family of reinforcement learning methods?
- What are the underlying ideas behind UCT?
- Can these ideas be applied in other ways?
- Beyond UCT, what next?

Introduction to UCT

- UCT is a **reinforcement learning** algorithm
- It learns a **value function** predicting the expected outcome from each position
- As value function improves, so does the **policy**
- Opponent model is self-play using same policy

Components of UCT

- Search: by sampling full trajectories
- Learning algorithm: Monte-Carlo
- Value function: Tabular
- Exploration: Tabular, counter based bonus
- Learning rate: Tabular, counter based
- Play-out policy: Pseudo-random

Sampling

- UCT “searches” by **sampling** games from the current position
- After each sample game, the value function is updated
- The opponent model is updated too (self-play)
- Sampling trajectories to update the value fn and model is called **Dyna** (Sutton, 1990)

Learning Algorithm

- UCT uses **Monte-Carlo** to update the value fn
- Only the result of the game is used
- **TD learning** updates the value function using data from all time-steps: bootstrapping
- TD is usually more efficient than MC
- MC is just a special case of TD (lambda = 1)

Value Function

- UCT uses a (partial) **tabular** value function
- The value of a state depends on the average result following that position
- Most reinforcement learning algorithms use **function approximation** to represent value fn
- The value function can be **initialised** to incorporate any prior knowledge

Exploration

- UCT explores by adding a **bonus** to the value
- Each state counts how many times it is visited
- The bonus at a state is a function of its counter
- There are many other RL strategies based on exploration bonuses (e.g. Sutton's Dyna+)
- Exploration bonuses can use function approximation too!

Learning Rate

- UCT uses a learning rate at each state that is inversely proportion to its counter
- This is optimal for stationary environments
- However, the policy is non-stationary
- Learning rate can also be adapted when using function approximation

Play-out Policy

- UCT uses a pseudo-random policy to play out games
- This is required whenever UCT leaves its knowledge base
- The play-out policy could be learned
- But if function approximation is used, the agent **never** leaves its knowledge base

A Proposal: Dynalite

- Search: by **DYNA** (sampling full trajectories)
- Value function: **LI**near + **T**abular + **E**phemeral
- Learning algorithm: TD learning
- Exploration: Linear + Tabular usage bonus
- Learning rate: Linear + Tabular usage
- Play out policy: none

VALUE FUNCTION

- Represent the value function as a **linear** combination of **features**
- **Tabular** is just a special case of **linear**
- Include states as features
- **Linear** features provide **generalisation**
- **Tabular** features provide **asymptotic** learning

Ephemerality

- UCT only forgets values due to memory limit
- DynaLite chooses to forget weights
- Each game old weights are **forgotten** and value function is **initialised** to learned value
- Weights then correspond to the value of a feature in the current **context**

USAGE

- UCT has a counter for each state tracking visits
- DynaLite counts the usage of each feature
- Exploration bonus and learning rate are linear functions of the usage

UCT .V. DYNALITE

- Advantages of UCT .v. DynaLite
    - Pseudo-random games are very fast
    - Step-size computation is simple and effective
    - Exploration bonus is simple and effective
    - Theoretical convergence guarantees under certain assumptions

UCT .V. DynaLite

- Potential advantages of DynaLite .v. UCT
  - Can generalise between different positions
  - Never leaves its knowledge-base
  - Knowledge is transferred between moves
  - Can use bootstrapping (TD methods)
  - Scales better to larger boards