A quick guide to Markov Decision Processes and Reinforcement Learning

Oleg Szehr
Dalle Molle Institute for Artificial Intelligence (IDSIA), Switzerland
oleg.szehr@idsia.ch

May 17, 2022

Recap: Contextual Bandits and Markov Decision Processes

Contextual Bandit:

k-armed bandit with *non-stationary* reward distributions. Agent receives *context* information as a hint about the current expected reward

Goal:

Find action that maximizes expected reward given context!

Markov Decision Process:

As contextual k-armed bandit but agent's actions do influence the next state of the environment/context

Goal:

Find rule of action to maximize accumulated reward over the planning horizon.

In this lecture, two examples:

1. Neural Linear Bayes Contextual Bandit
2. Alpha Go (Neural MCTS)

The Neural Contextual Bandit

Recall the Feed-Forward Neural Network (NN):

- *Neural Net:* Function $NN : \mathbb{R}^{N_1} \to \mathbb{R}^{N_L}$, concatenation of L layers
$$NN(x) = F_L \circ .... \circ F_2 \circ F_1(x)$$
- *Layer:* Function $F_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ of the form
$$F_l = \underbrace{\sigma}_{\substack{\text{activation func.}\\ \mathbb{R} \to \mathbb{R}}} \underbrace{\circ}_{\substack{\text{per}\\ \text{component}}} \underbrace{W_l}_{\substack{\text{affine func.}\\ \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}}}$$
- *Optimization Method:* Given sample $\{(x_i, y_i)\}_{i=1}$, tune all W_l to min. summed loss $\sum_i loss(NN(x_i), y_i)$

The Neural Contextual Bandit

Recall the Feed-Forward Neural Network (NN):

- *Neural Net:* Function $NN : \mathbb{R}^{N_1} \to \mathbb{R}^{N_L}$, concatenation of L layers
$$NN(x) = F_L \circ .... \circ F_2 \circ F_1(x)$$
- *Layer:* Function $F_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ of the form
$$F_l = \underbrace{\sigma}_{\substack{\text{activation func.}\\ \mathbb{R}\to\mathbb{R}}} \quad \underbrace{\circ}_{\substack{\text{per}\\ \text{component}}} \quad \underbrace{W_l}_{\substack{\text{affine func.}\\ \mathbb{R}^{N_l}\to\mathbb{R}^{N_{l+1}}}}$$
- *Optimization Method:* Given sample $\{(x_i, y_i)\}_{i=1}$, tune all W_l to min. summed loss $\sum_i loss(NN(x_i), y_i)$

How to solve contextual bandit problem?:

1. *Associate context and expected rewards*
   Set up specific model: Linear, RKHS, Neural Networks...
2. *Manage exploration*
   ϵ-Dithering, Thompson Sampling, UCB1,...

Neural Linear Bayes Contextual Bandit

Idea: Thomp. Sampl. on top of Neur. Net! But precisely?

1. If rewards deterministic:
   - Supervised learning of $(context, action) \to reward$ mapping
     In principle could train on history:
     $\mathcal{H}_t = \{((s_1, a_{k_1}), R(s_1, a_{k_1})), ..., ((s_t, a_{k_t}), R(s_t, a_{k_t}))\}$
     (context at time t: $s_t \in \mathbb{R}^d$)
   - Train NN taking input s_t and returns all rewards of k actions:
     $$NN : \mathbb{R}^d \to \mathbb{R}^k \text{ and } NN(s_t) = (\hat{R}(s_t, a_1), ..., \hat{R}(s_t, a_k))$$
   - Greedy choice: $a_{k_t} = \text{argmax } NN(s_t)$

Neural Linear Bayes Contextual Bandit

Idea: Thomp. Sampl. on top of Neur. Net! But precisely?

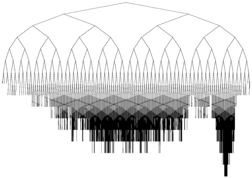
1. If rewards deterministic:
   - Supervised learning of $(context, action) \to reward$ mapping
     In principle could train on history:
     $\mathcal{H}_t = \{((s_1, a_{k_1}), R(s_1, a_{k_1})), ..., ((s_t, a_{k_t}), R(s_t, a_{k_t}))\}$
     (context at time t: $s_t \in \mathbb{R}^d$)
   - Train NN taking input s_t and returns all rewards of k actions:
     $$NN : \mathbb{R}^d \to \mathbb{R}^k \text{ and } NN(s_t) = (\hat{R}(s_t, a_1), ..., \hat{R}(s_t, a_k))$$
   - Greedy choice: $a_{k_t} = \text{argmax } NN(s_t)$
2. If rewards random:
   - Manage exploration via Thomp. Samp., non-linearity via NN
   - Assume affine dependency between context representation of NN and expected reward (at any time t):
     $$\mathbb{E}[R(a_k)|NN(s) \in \mathbb{R}^k] = W(NN(s))$$
   - Bayesian linear model for W: $W(x) = (\theta^*)^T x + \epsilon$ and
     $\pi_t(\theta, \sigma^2) = \pi_t(\theta)\pi_t(\theta|\sigma^2) \to$ Blackboard
   - Thompson sampling: Choose actions according to their posterior probability of being optimal

Recap: Monte Carlo Tree Search (MCTS)

Setting: Full-fledged MDP problem, finite action space.

We want: Planning algo. to optimize accumulated rewards.

Key Idea: Build specific, restricted, asymmetric decision tree



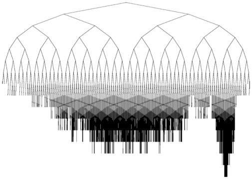
Recap: Monte Carlo Tree Search (MCTS)

Setting: Full-fledged MDP problem, finite action space.

We want: Planning algo. to optimize accumulated rewards.

Key Idea: Build specific, restricted, asymmetric decision tree

- MCTS involves two policies:
  1. *Default policy*: Used to value tree leafs
  2. *Tree policy*: Selects or creates leaf nodes from given tree
- Tree policy's purpose:
  - Adding new nodes → Exploration
  - Simulation of promising line → Exploitation



Recap: Monte Carlo Tree Search (MCTS)

Setting: Full-fledged MDP problem, finite action space.

We want: Planning algo. to optimize accumulated rewards.

Key Idea: Build specific, restricted, asymmetric decision tree

- MCTS involves two policies:
  1. *Default policy*: Used to value tree leafs
  2. *Tree policy*: Selects or creates leaf nodes from given tree
- Tree policy's purpose:
  - Adding new nodes → Exploration
  - Simulation of promising line → Exploitation

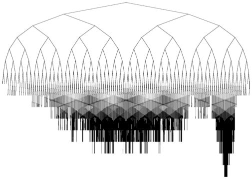
Tree-policy: Employ ordinary k-**armed bandit**!

- TS take actions according to their posterior prob. of being optimal
- $UCB1$ policy maximizes

  $UCB1_a = \bar{R}_a + c_{n,n_a}$ with $c_{n,l} = \sqrt{\frac{2 \ln n}{l}}$

  $\bar{R}_a$ average realized reward,

  n_a number of chosen a
- TS, $UCB1$ resolves exploration management, optimal regret $\mathcal{O}(log(n))$



Monte Carlo Tree Search (MCTS)

MCTS: Iterative improvement of tree policy and Bellman updates

1. *Selection:* Apply tree policy recursively to descend through tree until most relevant node
2. *Expansion:* Child node added to tree (according to tree policy)
3. *Simulation:* From new node default policy is applied to simulate reward
4. *Backpropagation:* Simulation result used to update statistics of ancestor nodes through the tree (count in $UCB1_i$)

```
function MonteCarloPlanning(state)
repeat(computationBudget): search(state,0)
return bestAction(state,0)

function search(state,depth)
if Terminal(state): return 0
if Leaf(state): return Evaluate(state)
action = selectAction(state,depth)
(nextState, reward) = simulateAction(state,action)
q = reward + γ search(nextstate,depth+1)
update(state,action,q,depth)
return q
```

MCTS Characteristics and Extensions

Characteristics:

1. *Asymptotic:* Converges to best decision if enough resources (Equiv. MinMax in P-games)
2. *Aheuristic:* No domain-specific knowledge, no eval. function (Chess: heuristics are available, $\alpha\beta$ very strong. Go: not)
3. *Anytime:* Tree statistics are update immediately. Small error prob. if stopped prematurely ($\rightarrow$ careful 'trap states' in Chess)

MCTS Characteristics and Extensions

Characteristics:

1. *Asymptotic:* Converges to best decision if enough resources (Equiv. MinMax in P-games)
2. *Aheuristic:* No domain-specific knowledge, no eval. function (Chess: heuristics are available, $\alpha\beta$ very strong. Go: not)
3. *Anytime:* Tree statistics are update immediately. Small error prob. if stopped prematurely ($\rightarrow$ careful 'trap states' in Chess)

Extensions: (exist in any direction!)

1. Tree policy:
   a) *Bandits:* UCB-Tuned, Bayesian Bandit, Continuous actions,...
   b) *Selection:* Domain Specific, Large Branching Issues (FPU), add pre-search, transposition Tables, history heuristics,...
   c) *Other*: Proof-number search, Pruning, Policy repr. (NN),...
2. Simulation: Rule-based policy, Policy representation (NN)...
3. Back-propag.: Weighted rewards,...

Most interesting extension: Neural Networks...

Neural MCTS: Towards AlphaGo, Alpha0,...

What is missing to be world's Chess champion?

- *Imitation Learning:* Mimic given *expert* policy ϵ by *apprentice* policy $\alpha \to$ *supervised Learning*
- *Expert Iteration Algorithm:* Iterate mutual improvement of expert and apprentice
  - Sample from expert ϵ_{t-1}. Mimic ϵ_{t-1} by α_t (imitation). Improve expert $\epsilon_t = \epsilon_t(\alpha_t)$ using apprentice estimates...
  - Expert provides values. Apprentice provides generalization and fast access.

Neural MCTS: Towards AlphaGo, Alpha0,...

What is missing to be world's Chess champion?

- *Imitation Learning:* Mimic given *expert* policy ϵ by *apprentice* policy $\alpha \rightarrow$ *supervised Learning*
- *Expert Iteration Algorithm:* Iterate mutual improvement of expert and apprentice
  - Sample from expert ϵ_{t-1}. Mimic ϵ_{t-1} by α_t (imitation). Improve expert $\epsilon_t = \epsilon_t(\alpha_t)$ using apprentice estimates...
  - Expert provides values.
    Apprentice provides generalization and fast access.
- *AlphaZero achitecture:*
  - Expert iteration scheme:
    Expert $\rightarrow$ MCTS, Apprentice $\rightarrow$ Deep conv. NN
  - Purpose of Apprentice:
    Operational: faster access to values, generalization capability of NN
    Search improvement: guidance of MCTS, accurate node valuation

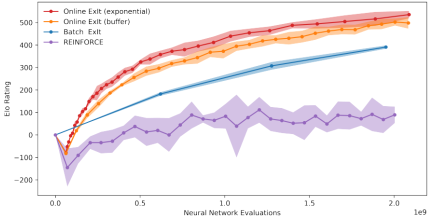
Neural MCTS: Towards AlphaGo, Alpha0,...II

Tree Policy NN:

- Trained on average MCTS play targets:
  $Loss_T = -\sum_a \frac{n(a,s)}{n} \log \alpha(a|s)$,
  $n(a,s)$ number of times a played from s, n number of plays.
- Usage:
  1. Bias tree policy
     $UCBNN_a(s) = USB1_a(s) + weight_a \frac{\alpha(a|s)}{n(s,a)+1}$

Valuation NN:

- Value NN :
  $Loss_V = -(z - V(s))^2$,
  z value estimates from MCTS.
- Usage:
  1. Reduce search depth
  2. Replace inaccurate rollout-based value estimation

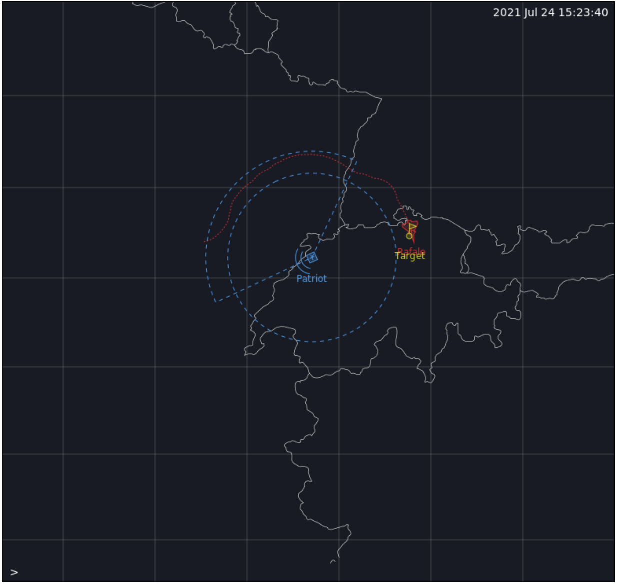


Value NN and Policy NN covered in single multitask NN:

- Regularization and higher speed

Neural MCTS: Example Scenario

Military example: Reach target without getting caught by radar



Neural MCTS: Example Scenario

Military example: Trained Neural Networks