Energy efficient UDN design using reinforcement learning

DGIST

BongSang Kim
JaeHyun Lee

05/July/2019

UGRP Meeting

Things to-do

- Setup python development env. for Linux/Windows
- Study RL
- Study gym, tensorflow, pytorch in github
- Divide Objective into RL/Env code

What have done

- Design RL code – MCTS
- Design Env code – Develop UDN and so on

should be done

Study RL

- **DQN**
  - **Approximate action-value(Q) function** **by deep neural network**
  - **Update by the** **gradient of loss function**

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
    end for
end for

Study RL

- **Actor-Critic**
  - **Actor: update θ which approximate policy**
  - **Critic: update w which approximate action-value function q(s,a)**
  - **Could reduce learning time by approximating q function**
  - **Actor-Critic follows an <u>approximate</u> policy gradient**

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s,a) = \phi(s,a)^\top w$

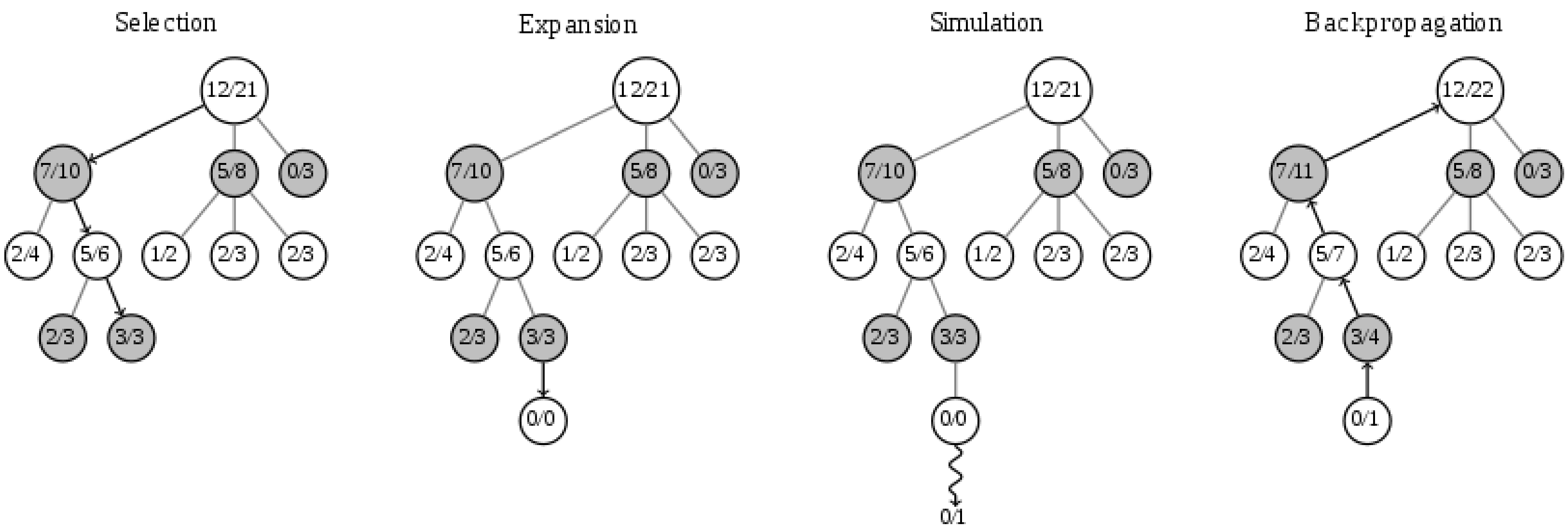
  Critic Updates w by linear TD(0)

  Actor Updates θ by policy gradient

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

RL design - MCTS

- **Monte-Carlo Tree Search**
- **State, node visit count, value**



- **Why MCTS?**
  - **Easy to come up with RL model with our environment model**

Initial RL design – MCTS

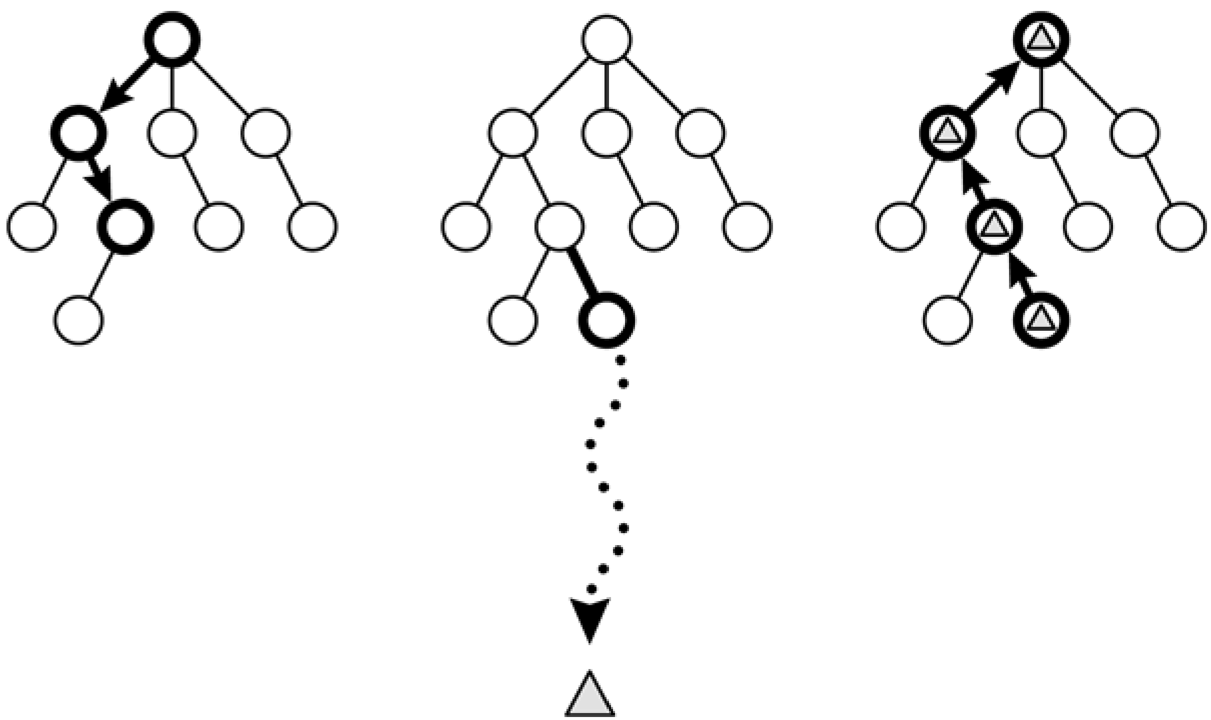
- **Pseudo code**

```
def MCTS(node=root_node, threshold, terminal)
    initialize root node
    for search tree:
        calculate the rewards
        if rewards < threshold:
            break
        elif:
            moves to next nodes
        if node arrives terminal:
            end for
        update policy(backpropagate)
    return best policy|reward
```

Future Plan(next week)

- **MCTS design**
- n BS, m UE, distributed in random.
- State: Matrix of BS on/off [discrete, $S_i = 1 \;\equiv i^{\text{th}}$ BS on, $i = 1, 2, \ldots, \text{n}$]
- Action: Matrix of change BS on/off state [discrete]
- Reward: SNR, $\propto D^{\alpha}$ [$\alpha = -4$ (NLoS)]
- User-BS association rule: nearest association
- D: Distance between User and BS matrix
  - $D_{ij} = (BS_{ix} - UE_{jx})^2 + (BS_{iy} - UE_{jy})^2$
  - [BS, UE is location matrix of BS and User]
- Initial policy: $\pi_0 = P(a|s) = 0.5$
- Node: initialize with $\text{BS}_i = 1$ for every i
- Constraint: one action, one BS condition change
- Optimize policy – update by backpropagation

Future Plan(next week)



Future plan(one month)

- Program RL code by Tensorflow Library
- Apply UDN at our Environment
  - Consider LoS, NLoS, Interference
  - Set BS density for UDN
  - Consider all BS state change concurrently
  - Update reward
    - consider Energy Consumption
    - SNR
    - Terminal condition - reach at Terminal Node or SNR < threshold θ
- Update RL code for programmed env code

Any Questions?

THANK YOU