1 Hyper tuning using Q_learning

In order to tune the corresponding weights of distance, load , energy consumption, a varient of RL called Q learning is employed.

Q-Learning principles according to the MEC system:

1. **Environment:** The environment in which the agent operates is constructed from the set [$distance\_weight(W_d)$, $load\_weight(W_l)$, $EConsumption\_weight(W_{ec})$] in range between 0 to 1 with step value of 0.05. As a consequence, it is represented as 3 dimensional matrix of size n^3 with n=20.

2. **Reward:** The R_t rewards for each state in the environment is a metric that indicates the optimal weight assignment, which means that the particular combination of distance, load and energy consumption weights leads to lower application latency. It is constantly calculated from the average latency of mobile users(going through migration process).

   $$R_t = 1/A_t$$

   where, A_t represents the average latency of mobile users till that timestamp since beginning. The above equation only show that lower the average latency, higher the reward value assign to the current combination values of (W_d, W_l, W_{ec}).

Algorithm 1 Environment

INPUT: **Range** where, range of values between [0-1] (a multiple of 0.05)
OUTPUT: Rewards /* Reward Matrix */
for $W_d \in$ **Range do**
  for $W_l \in$ **Range do**
    for $W_{ec} \in$ **Range do**
      if $W_d + W_l + W_{ec} = 1$ **then**
        if current calculated A_t = Ideal A_t **then**
          /* yet to decide the value of MAX(R_t) */
          rewards[W_d][W_l][W_{ec}] $\leftarrow$ MAX(R_t)
        end if
        rewards[W_d][W_l][W_{ec}] $\leftarrow 1/A_t$
      end if
    end for
  end for
end for
return rewards

The training of the agent on the environment:

The training phase allows the agent to exploit the environment through the different actions it takes, by trying to improve its action strategy according to the rewards received.

1. **Action:** From any state W_d, W_l, W_{ec}, the agent has the possibility to execute the following 6 actions: stay in $W_d(|W_d)$, increment $W_d(+W_d)$, decrement $W_d(-W_d)$, stay in $W_l(|W_l)$, increment $W_l(+W_l)$, decrement $W_l(-W_l)$, stay in $W_{ec}(|W_{ec})$, increment $W_{ec}(+W_{ec})$, decrement $W_{ec}(-W_{ec})$(in multiple of 0.05). These actions are inserted in an Actions table, such as:

$$
\begin{aligned}
\textbf{Actions} = [&'|W'_d,' +W'_d,' -W'_d,' |W'_l,' +W'_l, \\
&' - W'_l,' |W'_{ec},' +W'_{ec},' -W'_{ec}]
\end{aligned}
$$

2. **Q_table:** The Q_table contains the Q values for each action-state (Q(s, a)), such that the lines of the table are the states that represent the set of combinations W_d, W_l, W_{ec} existing in the rewards matrix(algorithm 1), and the columns are the 6 actions that is described in Eq. 12. The Q values are initialized to 0 at the beginning of the training.

3. **Training functions:** To start the training, the agent must choose a random state (W_d, W_l, W_{ec}) in the environment (Algorithm 3). The state chosen by the agent must not be a terminal state that has a reward equal to $MAX(R_t)$ as indicated in algorithm 2. After choosing the departure state s = W_d, W_l, W_{ec}, the agent must perform an action according to the value of ϵ which indicates whether it is a random action or an action from the Q_table which maximizes the value Q(s, a) (algorithm 4). The execution of the action a at time t by the agent allows him to move to another state $s^{'}$ and receive a reward R_t according to the chosen action, as shown in algorithm 5.

4. **Agent Training:** Algorithm 6 gives the steps followed by the agent during its training with the parameters, fixed after multiple simulations, below, as well as the update of the Q_table at each state changes using the reward new state $s^{'}$. This is done using the reward R_t and the Q value of the current state Q($s^{'}$, a); the agent calculates the time difference (TD) that will be used for the computation of the new Q value of the previous state s (Q(s,a)).

   - ϵ: Exploration and exploitation time of environment(set as 0.9)
   - γ: Discount factor(set as 0.9)
   - α: Learning Rate(set as 0.9)
   - Get_Next_Action(W_d, W_l, W_{ec}, ϵ): Allows to determine the action to be executed that has the largest Q value in the Q_table.
   - Get_Next_State(W_d, W_l, W_{ec}, Action_Index): Allows to determine the next state from the executed action.

Algorithm 2 Is_Termination_State

INPUT: (W_d, W_l, W_{ec}) // Test if the state is terminal
OUTPUT: Boolean Value
if rewards$[W_d][W_l][W_{ec}] = MAX(R_t)$ **then**
    return ← True
end if
return ← False

Algorithm 3 Get_Starting_State

INPUT: range of values between [0-1].
OUTPUT: W_d, W_l, W_{ec}
For each $W_d, W_l, and\ W_{ec}$ choose a random real number(a multiple of 0.05) between 0 and 1;
if Is_Termination_State(W_d, W_l, W_{ec}) **then**
    return Get_Starting_State()
end if
return (W_d, W_l, W_{ec})

Algorithm 4 Get_Next_Action

INPUT: $(W_d, W_l, W_{ec}, \epsilon)$
OUTPUT: Action
if random() $< \epsilon$ **then**
    /* Choose the action that maximizes Q_{value} for (W_d, W_l, W_{ec}) */
    return action with max($Q_{values}[W_d, W_l, W_{ec}]$)
end if
return random action from actions table

Algorithm 5 Get_Next_State

/* current status and the selected action to perform */
INPUT: $(W_d, W_l, W_{ec}, Action\_Index)$
OUTPUT: $new\_W_d, new\_W_l, new\_W_{ec}$
if Actions[Action_Index]= $|W_d$ **then**
    $new\_W_d \leftarrow W_d$
end if
if Actions[Action_Index]= $+W_d$ and $W_d <= 1$ **then**
    $new\_W_d \leftarrow W_d$ + 0.05
end if
if Actions[Action_Index]= -W_d and $W_d >= 0$ **then**
    $new\_W_d \leftarrow W_d$ - 0.05
end if
if Actions[Action_Index]= $|W_l$ **then**
    $new\_W_l \leftarrow W_l$
end if
if Actions[Action_Index]= $+W_l$ and $W_l <= 1$ **then**
    $new\_W_l \leftarrow W_l$ + 0.05
end if
if Actions[Action_Index]= -W_l and $W_l >= 0$ **then**
    $new\_W_l \leftarrow W_l$ - 0.05
end if
if Actions[Action_Index]= $|W_{ec}$ **then**
    $new\_W_{ec} \leftarrow W_{ec}$
end if
if Actions[Action_Index]= $+W_{ec}$ and $W_{ec} <= 1$ **then**
    $new\_W_{ec} \leftarrow W_{ec}$ + 0.05
end if
if Actions[Action_Index]= -W_{ec} and $W_{ec} >= 0$ **then**
    $new\_W_{ec} \leftarrow W_{ec}$ - 0.05
end if
return $new\_W_d, new\_W_l, new\_W_{ec}$

Algorithm 6 Agent_Training

```
W_d, W_l, W_ec ← Get_Starting_State()
for iteration → [1 - total number of migration] do
    while Is_Termination_State(W_d, W_l, W_ec) = False do
        Action_Index ← Get_Next_Action(W_d, W_l, W_ec, ϵ)
        new_W_d, new_W_l, new_W_ec ← Get_Next_State(W_d, W_l, W_ec, Action_Index)
        reward ← rewards[new_W_d][new_W_l][new_W_ec]
        old_Q_value ← Q_values[W_d, W_l, W_ec, Action_Index]
        TD ← reward + (γ ∗ MAX_a(Q_values[W_d, W_l, W_ec])) - old_Q_value
        new_Q_value = old_Q_value + α*TD
        Q_values[W_d, W_l, W_ec, Action_Index] ← new_Q_value
    end while
end for
```