Multi Agent Systems

- Lab 7 -

Q-Learning with Value Function Approximation

Q-Learning Recap

- Value Function is more explicit in storing the value of executing ***an action*** in a ***given state: q(s, a)***

$$q^{\pi}(s,a)=E_{\pi}[R_{t+1}+\gamma R_{t+2}+\gamma^{2}R_{t+3}+\ldots|S_t=s,A_t=a]=E_{\pi}\left[\sum_{\tau=t+1}\gamma^{\tau-t-1}R_{\tau}|S_t=s,A_t=a\right]$$

- Instance of *model-free learning* – i.e. environment dynamics is unknown to the agent

- We tackled environments where number of states is small enough to use a ***tabular*** representation of the Q-Function

Q-Learning Recap

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

- Learning policy is greedy
- Play policy allows for exploration

procedure ε-Greedy (s, q, ε)
  with prob ε: return *random(A)*
  with prob 1-ε: return $\underset{a}{argmax}\, q(s,a)$
end

procedure Q-Learning (<S, A, γ>, ε)
  for all s in S, a in A **do**
    q(s,a) ← 0   // set initial values to 0
  end for
  for all episodes **do**
    s ← initial state
    while s not final state **do**
      pick action *a* using ε-Greedy (s, q, ε)
      execute *a* → get reward r and next state *s'*
      $q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma \max_{a'} q(s',a') - q(s, a))$
      s ← s'
    end while
  end for
  for all *s* in S **do**
    $\pi(s) \leftarrow argmax_{a\ in\ A}\ q(s, a)$
  end for
  return *π*

Q-Learning in continuous state space

- Many real world problems have enormous state and/or action spaces (e.g. robotics control, self driving)
- Tabular representation is not really appropriate
- Idea: Use a function to represent the value

Q-Learning with Linear Value Function Approximation – General Formulation

- Use *features* to represent state and action $x(s,a) = \begin{pmatrix} x_1(s,a) \\ x_2(s,a) \\ \dots \\ x_n(s,a) \end{pmatrix}$

- Q-function represented as ***weighted linear combination of features***

$$\hat{Q}(s,a,\boldsymbol{w}) = \boldsymbol{x}(s,a)^T \boldsymbol{w} = \sum_{j=1}^{n} x_j(s,a) w_j$$

- ***Learn*** weights **w** through stochastic gradient descent updates $\nabla_w J(w) = \nabla_w E_\pi[(Q^\pi(s,a) - \hat{Q}^\pi(s,a,\boldsymbol{w}))^2]$

Q-Learning with Linear Value Function Approximation – Simplified

- When action space ***A*** is *small* and *finite*
  consider a featurised representation of states only

$$x(s) = \begin{pmatrix} x_1(s) \\ x_2(s) \\ \dots \\ x_n(s) \end{pmatrix}$$

- Q-function represented as ***collection*** of *weighted linear combination of features* – ***one model per action***

$$\hat{Q}_a(s, \boldsymbol{w}) = \boldsymbol{x}(s)^T \boldsymbol{w} = \sum_{j=1}^{n} x_j(s) w_j, \forall a \in A$$

- ***Learn*** weights **w** through stochastic gradient descent updates

$$\nabla_w J(w) = \nabla_w E_\pi[(Q^\pi(s,a) - \hat{Q}_a^\pi(s, \boldsymbol{w}))^2]$$

Q-Learning with Linear Value Function Approximation – TD Target

- For Q-Function, instead of the actual gain per episode under current policy $Q^{\pi}(s,a)$ use **TD-target** $r+\gamma max_{a'}\hat{Q}(s',a',\boldsymbol{w})$

- ***Learn*** weights **w** through stochastic gradient descent updates

$$\nabla_w J(w)=\nabla_w(r+\gamma max_{a'}\hat{Q}_{a'}(s',\boldsymbol{w})-\hat{Q}_a(s,\boldsymbol{w}))^2$$

$$\Delta w=\alpha(r+\gamma max_{a'}\hat{Q}_{a'}(s',\boldsymbol{w})-\hat{Q}_a(s,\boldsymbol{w}))\nabla_w\hat{Q}_a(s,\boldsymbol{w})$$

$$\Delta w=\alpha(r+\gamma max_{a'}\hat{Q}_{a'}(s',\boldsymbol{w})-\hat{Q}_a(s,\boldsymbol{w}))x(s)$$

Q-Learning, Linear Approximation, TD target

procedure Q-Learning (<S, A, γ>, ε, estimator**)**
for all episodes **do**

s ← initial state
while s not final state **do**
pick action *a* using ε-Greedy (o_s, estimator, ε)
execute *a* → get reward r and next state *s'*
$x(s') = featurize(o_{s'})$
$[\hat{q}_{a1}(s'), \ldots, \hat{q}_{am}(s')] =$ ***estimator.predict(x(s'))***
$td_{target} = r + \gamma max_{a'} \hat{q}_{a'}(s')$
estimator.update(s, a, td_{target}*)*
s ← s'
end while
end for

for all *s* in S **do**
$\pi(s) \leftarrow argmax_{a \text{ in } A}\ q(s, a)$
end for
return *π*

Agent learns by observing consequences of actions it takes in the environment

Q-values adjusted through **temporal differences**

Learning is **off-policy**

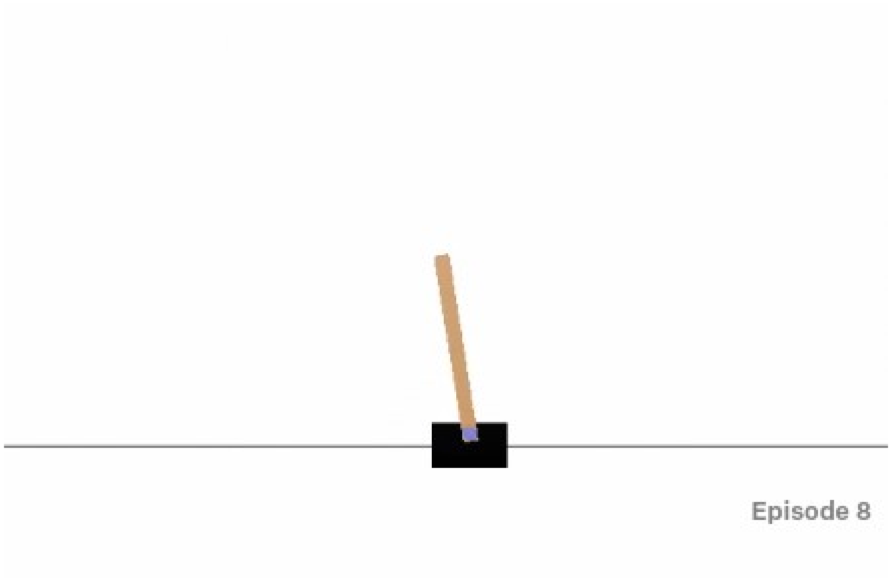
Learning policy is greedy

Play policy allows for exploration

procedure ε-Greedy (o_s, estimator, ε)
x(s) = featurize(o_s)
$\hat{q}$ = estimator(x(s))
with prob ε: return *random(A)*
with prob 1-ε: return $\underset{a}{argmax}\, \hat{q}_a(s)$
end

OpenAI Gym BlocksWorld Environment

- Cartpole-v1 environment in OpenAI Gymnasium:
  - Objective: keep a pendulum upright for as long as possible
  - 2 actions: left (force = -1), right (force = +1)
  - Reward: +1 for every timestep that the pole remains upright
  - Game ends when pole more the 15° from vertical OR cart moves > 2.4 units from center



OpenAI Gym BlocksWorld Environment

- **Q-function model setup**
  - Use a simple, 2 linear layer neural network as your q function estimator
    - Suggested model: **`Linear(state_size, 100) → activation → Linear(100, action_size)`**, **`state_size=4`** and **`action_size=2`**
    - Preinitialize the first layer model weights and biases as in the following
      - $w_{ij} \sim \sqrt{i \times 0.5}\, N(0,1),\ i=1..4,\ j=1..100$

        $b \sim uniform(0, 2\pi)$

- **Experiment setup –** run each experiment for a maximum of **2000 episodes**
  - **For the model optimization part**
    - **Explore** three different activation functions: *cos(x), sigmoid(x), tanh(x)*
    - **Explore** three values of the **SGD** learning rate: 1e-4, 5*1e-4, 1e-3
  - **For the RL part:**
    - **Explore** different values of ε – ε=0.0, ε=0.1, ε=decay(init=0.2, factor=0.9)
    - **Explore** different values of **learning rate (α):** `α=0.01, α=0.05, α=0.2`