Q-learning for radio resource management

October 5, 2017

1 Reinforcement learning for RRM

The problem of online radio resource allocation is solved by formulating an associated Markov Decision Process and developing an online reinforcement learning algorithm to solve the MDP instead. In our problem we are concerned with assigning M communication channels to $M \subsetneq N$ controllers, such that the discounted future rewards are maximized. For our purposes, we assume that the RRM observes the state space of the dynamical system, further this state space also constitutes the state space of the associated MDP. The set of actions are subsets of $[N]$ of cardinality M. It is assumed that the RRM observes state s_t at time t and takes an action a_t according to some behavior distribution ρ. The RRM then observes reward r_t and the state transition s_{t+1}. The reward obtained is assumed to correspond to the last action. In other words, we do not allow for delayed rewards. This transition to s_{t+1} could be stochastic or deterministic depending on the actions taken by the RRM and the controllers. In other words we have $s_{t+1} \sim \mathcal{E}$. At time t_0 we are interested in maximizing

$$R_{t_0} = \sum_{t \geq t_0}^{\infty} \gamma^{t-t_0} r_t.$$

Q-learning is a useful methodology to solve problems of the above kind. It is based on finding Q-factors for every state-action pairs.

$$Q^*(s,a) := \max_{\pi} \mathbb{E}\left[R_t \mid s_t = s, a_t = a, \pi\right],$$

where π is a policy mapping states to actions. The Q-learning algorithm is based on the Bellman equation:

$$Q^*(s,a) = E_{s' \sim \mathcal{E}}[r + \gamma \max_{a' \in U(s')} Q^*(s',a') \mid s,a]$$

We use a function approximator to estimate the Q-factors, *i.e.,* $Q(s,a,\theta) \approx Q^*(s,a)$. The neural network function approximator with weights θ is referred to as a Q-Network. The Q-Network is trained by minimizing a sequence of loss functions $L_i(\theta_i)$ given by:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(s,a)}\left[(y_i - Q(s,a,\theta_i))^2\right],$$

where $y_i := \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \mid s, a \right]$, is the expected cost-to-go based on the latest update of the weights. Differentiating the loss function with respect to the weights:

$$\nabla_{\theta_i} L_i(\theta_i) = -2\, \mathbb{E}_{s,a \sim \rho(\cdot); s \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right].$$

It may be noted that we just take a sample rather than find the above expectation, when updating weights.

We briefly mentioned that the actions are picked according to some behavior distribution ρ. Specifically, we employ an ϵ-greedy approach to picking actions. In other words, w.p. ϵ we pick a random action and w.p. $1 - \epsilon$ we pick a greedy action. We also use an experience replay found in [3] to overcome biases and to have a stabilizing effect. We store the previous K experiences (s_t, a_t, r_t, s_{t+1}), where $t_0 - K + 1 \leq t \leq t_0$.

Algorithm 1. *Q-Learning for resource allocation*

1: Initialize the replay memory $\mathcal{D}$ to capacity K.
2: Initialize the weights, θ, of the Q-Network.
3: **for** the entire duration **do**
4: With probability ϵ select a random action a_t.
5: With probability $1 - \epsilon$ pick a_t that maximizes $Q^*(s_t, a; \theta)$.
6: Execute action a_t to obtain reward r_t and observe s_{t+1}.
7: Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.
8: Sample random mini-batch transitions from $\mathcal{D}$.
9: Corresponding to (s_j, a_j, r_j, s_{j+1}), set

$$y_j := r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta).$$

10: Perform a gradient descent step with loss given by $(y_j - Q(s_j, a_j; \theta))^2$.

end

References

[1] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *HotNets*, pages 50–56, 2016.

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *Nature*, 2013.