**4/24/2023**

**Actor-Critic Algorithm**

* Actor-critic Algorithm – separate V/Q value calculation and policy learning. The difference is that both get updated simultaneously.
* **Sidenote:** Expanding TD equation to infinity converges to Monte-Carlo equations
* Actor-critic algorithm utilizes the preference metric (from MAB problem – softmax/gradient policy)
* In summary, two equations that are updated every step. Actor, which is the policy, and then the critic, which is the V or Q value.
  + Critic:
    - where is the TD-0 error:
  + Actor:
* From Prof. Mahdi’s HW3

Text, letter

Description automatically generated

* Very fast convergence due to the gradient being used directly compared to other algorithms (policy + value iteration, SARSA + Q-learning, etc.)
* Also model-free, similar to SARSA + Q-learning
* Any policy can be used (doesn’t have to be softmax). The only difference would have to be that the gradient would have to be computed accordingly, leading to a difference in the H or preference equation.

**Summary**

* MDP - <S, A, R, P(s’|s, a)>
* If MDP is known, model-based policies
  + Policy iteration
  + Value iteration
* If MDP is not known, model-free policies
  + Monte-Carlo
    - Generate a bunch of trajectories, estimate V values based on G\_ts
  + Temporal Difference learning
    - Instead of going depth-wise, choose single action (or more, depending on TD length)
      * SARSA
      * Q-learning
  + Actor-Critic
    - Similar to TD, but separation between policy and value function
* All these algorithms were tabularized, we had discrete tables. A real problem might not be tabularized. Actions, states, can be continuous (for example, car position/speed in Cartpole). Tabular algorithms no longer work ~ would converge too slowly/consume too much memory.
* **One solution:** approximate and discretize the action/state spaces. Tradeoff between closeness to true state space and time to converge.

**Continuous RL Algorithms**

* **Option 1:** Instead of learning Q(s, a), learn theta (set of constants?) i.e. Q\theta(s, a, \theta)
* **Option 2:** DNN – states and action as input, Q value output. Another network can be used for the policy. Typically, 2-3 layers are enough to get good estimates. Question: How is the network trained?
* **Option 3:** Wavelets/Fourier transforms
* **Option 4:** Simple polynomials
  + In this case, we’re using known basis functions (phi) to express our Q-values as polynomials
  + **Example:**
    - If phis are chosen as indicator function i.e.
      * – this is the tabular case
    - Any class of phis can be chosen

**Least Square Policy Iteration (LSPI)**

* Policy Evaluation PE:

Or

* Policy Improvement

For model-free approach:

Solving for Q,

If Q is large, not possible to compute inverse in reasonable amount of time

So we apply the approximation principles as follows:

See paper: <https://users.cs.duke.edu/~parr/jmlr03.pdf>

* Works well when state space is continuous,, but not as much when action nspace is continuous

**5/1/2023**

* Tabular Q-learning
* For neural network, we can have the following loss function
  + .
* We separate the problem into two networks. One only estimates the target (target network), whereas the other one is used for the model (model network). Periodically however, we sync the two.
* **Experience Replay**
  + Create a buffer to store past experiences of an agent, and repeatedly pass them onto the network. Similar to minibatch.
  + Becomes a supervised learning problem, where
    - is the target
  + Algorithm
    - Gather samples from environment
    - Sample a random batch
    - Train network, update every T steps
  + This is known as **Deep Q Learning with experience replay.**
    - Batch size, Buffer size, and learning rate are hyperparameters that need to be optimized
  + Not all transitions in a buffer are equally important, and so **prioritized experience replay**, each transition is assigned a priority based on the associated transition. Sampling within batch is made non-uniform based on a softmax of their priorities.
* Policy Gradiewnt Methods
  + Neural network, input state, output a (deterministic), or multiple a’s with a probability (distribution over actions)
  + See
  + Downside: need to figure out a way to update the network. See pictures from class
* REINFORCE algorithm
* Centralized training (training), Decentralized Execution (using policy)