Notes for Meeting

Starting too narrow, don't know what I'm applying machine learning techniques to

Structure of problem is not well understood

deep learning vs reinforcement learning

state vs. stateless model

Different Kinds of Neural Networks

Introduction to Reinforcement Learning

3.1 k-armed Bandit Problem

We would like to maximize expected reward based on k different actions, assuming you do not know the true rewards. For each action A_t that takes a value a, we then have $q_*(a) = E(R_t | A_t = a)$. We maintain an estimate of $q_*(a)$, $Q_t(a)$. We can either take a "greedy" action by choosing what we know to be the highest, or "explore" by choosing a different action. Thus in the short run, we do better by being greedy, but in the long run potentially better by exploring, so what actions we take depends on how many time steps remain.

3.2 Action-Value Methods

The simplest way to estimate $Q_t(a)$ is to replace the population expectation with sample averages:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_t 1(A_t = a)}{\sum_{i=1}^{t-1} 1(A_t = a)}$$

which is consistent for $q_*(a)$ by the law of large numbers. In this sense, the greedy action A_t can be described by $\text{argmax}_a Q_t(a)$. Further, instead of always being greedy, we can choose to explore with probability ε and use an ε-greedy method. The advantage is that asymptotically, every action is sampled, with little harm to optimality. Positive values of ε almost always outperform pure greediness, and this is especially true for nonstationary rewards.

When it comes to estimating the value of an action, we can use the formula earlier to get

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

However, it is more computationally efficient to use an updating rule instead of holding records for every selection:

$$\mathit{NewEstimate} \longleftarrow \mathit{OldEstimate} + \mathit{StepSize}[\mathit{Target} - \mathit{OldEstimate}]$$

Under the previous estimator, we would have $Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$, and Q_1 is an arbitrary selection.

3.3 Nonstationary Problem and Initial Values

One way R changing over time is to provide more weight to recent rewards, which can be done by using a constant step-size parameter $\alpha \in (0, 1]$. Our updating formula then becomes $Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$.

It is clear that with constant step size, the bias induced by the initial choice for Q_1 does not go away. This has advantages and disadvantages; on the one hand, it can be used to encourage exploration early on (the method has more leeway to choose non-greedy actions in the beginning), but at the expense of making that assumption. We refer to this as an optimistic initial value, and it is a simple but effective way to explore with stationary rewards.

3.4 Upper-Confidence-Bound and Gradient Bandit Algorithms

Instead of randomly exploring ε often, we can take into account how fruitful that action may be. If we choose actions according to:

$$A_t = \operatorname*{argmax}_a \left[Q_t(a) c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Further, we may be able to train the algorithm to have a preference over its actions; i.e. for each action value a, there is a preference $H_t(a)$. Then the probability of taking that action, based on the softmax function, is:

$$P(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

The learning/updating method for these preferences is based on Stochastic Gradient Ascent[1]:

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \pi_t(A_t))$$
$$H_{t+a}(a) = H_t(a) - \alpha(R_t - \overline{R_t})$$

Demeaning the rewards serves to check if the latest reward is high or low compared to baseline, and if it is high, then the preference is positively updated, and vice versa.

Policies and Markov Decision Processes

In our framework, there is an agent and an environment, and at each time step, the agent makes a decision by selecting an action. Then, the environment is updated and changed, and the agent selects another action. The means by which the agent gains information about the environment is a representation, known as a state $S_t \in \mathcal{S}$; the consequent action is $A_t \in \mathcal{A}$ with a reward $R_t \in \mathcal{R}$. A_t, S_t, and R_t are random variables that take on where $\mathcal{A}, \mathcal{S}, \mathcal{R}$ are the set of all actions and states.

In the case of a finite Markov Decision Process, the random variables have discrete probability distributions, and we can describe the joint probability that the next state and reward takes on a certain value as a function of the previous state and action:

$$p(s', r|s, a) = P(S_t = s', R_t = r'|S_{t-1} = s, A_{t-1} = a)$$

which satisfies:

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1$$

We have essentially assumed that the probability for each value of S_t and R_t depends only on the immediate past, S_{t-1} and A_{t-1}, which is what makes this a Markov process. This has ramifications for what the state represents: it must contain "The state must include information about all aspects of the past agent–environment interaction that make a difference for the future"

The agent seeks to maximize expected return G_t, some function of the rewards. It is nice to use a discounted sum of the form:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \ \gamma \in [0, 1]$$

[1]It can be shown that the expected value of the update is equal to the expected value of the gradient of the reward

From this, we can see the recursive relationship for returns:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

A policy $\pi : \mathcal{S} \to \mathbb{R}$ is a mapping from a state to a probability of selecting an action, where if the agent is following the policy π, $P(A_t = a|S_t = s) = \pi(a|s)$. The state-value function v_π is the expected return in the state s of following π:

$$v_\pi(s) = E(G_t|S_t = s)$$

and the action-value function q_π:

$$q_\pi(s,a) = E(G_t|S_t = s, A_t = a)$$

The value functions can be estimated by sample averages of the rewards received under certain policies, actions, and states, i.e. if we keep averages of actual returns in all the states we encounter, we can estimate v_π, and if we keep seperate averages depending on our actions, we estimate q_π (Monte-Carlo Methods).

4.1 The Bellman Equation

Recall the definition of a (discrete) conditional expectation, if X is a random variable taking on values $\{x_i\}_{i\in I}$:

$$E(X|Y) = \sum_{i\in I} x_i P(X = x_i|Y)$$

Furthermore, we can use the recursive relationship for returns we observed earlier to derive a condition for any state s and its possible successor states:

$$\begin{aligned} v_\pi(s) &= E(R_{t+1} + \gamma G_{t+1}|S_t = s) \text{ since } G_t = R_{t+1} + \gamma G_{t+1} \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r + \gamma E(G_{t+1}|S_{t+1} = s')\right] \\ &\boxed{v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r|s,a)[r + \gamma v_\pi(s')]} \end{aligned}$$

The last line is the Bellman Equation: at any given state s, the agent will take an action based on the policy π. Then, the environment will respond by presenting a new state s' and reward r. What we obtain is the weighted average over all the possible s' and r defined by the Markovian dynamics of the system established earlier.

4.2 Optimality

Our goal with a reinforcement learning problem is essentially to find the highest yielding policy. Value functions provide a partial ordering over policies: π is better than or equal to π' if $v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in \mathcal{S}$. The overall best policy is denoted π^*, and its state value function is called the optimal state-value function v_*, defined as:

$$v_*(s) = \max_\pi v_\pi(s),\ \forall s \in \mathcal{S}$$

and also gives rise to an optimal action-value function q_*:

$$q_*(s,a) = \max_\pi q_\pi(s,a),\ \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Since this gives the expected return for taking action a in state s and thereafter following an optimal policy, we can rewrite q_* in terms of v_*:

$$q_*(s,a) = E(R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a)$$

We can apply this notion of optimality to the Bellman equation, and get the Bellman optimality equation, which expresses the fact that the value function for a given state must be the expected return from choosing the best action for that state:

$$\begin{aligned} v_*(a) &= \max_{a \in \mathcal{A}(s)} q_{\pi^*}(s,a) \\ &= \max_a E(R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a) \\ &= \max_a \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a)[r + \gamma v_{\pi^*}(s')] \end{aligned}$$

The next sections will focus on actually estimating the value functions

Monte Carlo Methods and Temporal Difference Learning

5.1 Actor Critic Method Briefly

The actor critic method is a policy gradient method that attempts to learn an optimal policy using two different processes, the actor and the critic. The actor follows a parametrized policy $\pi_\theta(a|s)$, where $\theta = \text{argmax}\, J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$. $J(\theta)$

- two neural networks, actor network and critic network
- actor network has parameter θ to estimate the policy $\pi_\theta(s, a)$, critic network has parameter w to estimate $q(s, a)$ and update the policy
-