Cheatsheet for Advanced Machine Learning **V2**
Marcel van Laar

1 Intro & Bandits

Let's say you have 4 different options. For each choice there is a reward, and there is also a long-term reward. At each time step t the agent chooses an action A_t and receives some reward R_t

- There are k available options
- The expected reward of some action a is as follows (this is often defined as "action value"):

$$Q_t(a) = \frac{\text{Sum of R when } a \text{ chosen before } t}{\text{\# times } a \text{ chosen before } t} \quad \text{(Expected reward of a)}$$

Given the estimated value of all possible actions, we take the action with the largest $Q_t(a)$ (greedy)

$$A_t = argmax_a(Q_t(a)) \tag{1}$$

1.1 Saving rewards

You do not need to save all historical rewards of an action, to estimate Q_t We just upadte Q_{t-1} to Q_t when a new reward R_{t-1} comes in:

$$Q_t = Q_{t-1} + \frac{1}{t-1}(R_{t-1} - Q_{t-1}) \tag{2}$$

This is cheap in terms of memory and computation. A general form we will see often:

$$\text{NewEstimate} = \text{OldEstimate} + \text{stepSize(Target - Oldestimate)} \tag{3}$$

1.2 Greedy vs other approaches

The reward distribution may not be linear. You need to look at exploration vs exploitation of the reward structure. This is an important tradeoff in RL, at each point you can either:

1. Exploit the information you have: pick the action believed to be best
2. Explore: try out another action, maybe get new information

By taking a greedy approach, we are only diong exploitation. A straightforward way to add some exploration, is called the $\epsilon - greedy$ action selection:

- Most of the time we are greedy
- For each action, with some small probability ϵ we select an action uniformely at random
- As t goes to infinity, so does the number of times each action is tried. So our estimates $Q_t(a)$ will converge slowly to $q_*(a)$.

1.3 A k-armed bandit algorithm

ε-greedy k-armed bandit algorithm

Initialize $Q(a) \leftarrow 0$ and $N(a) \leftarrow 0$ for all actions $a = 1, \ldots, k$
for each step $t = 1, 2, \ldots, T$ **do**
  With probability ε select a random action A_t
  Otherwise select $A_t = \arg\max_a Q(a)$
  Take action A_t and observe reward R_t
  Update:
    $N(A_t) \leftarrow N(A_t) + 1$
    $Q(A_t) \leftarrow Q(A_t) + \frac{1}{N(A_t)}(R_t - Q(A_t))$

1.4 How does ϵ affect learning?

The parameter ϵ is related to the exploration strategy of the multi-armed bandits algorithm. A smalll ϵ (less exploration) can lead to eventual better performance.

- ϵ too small: Needs a long time to find optimal solution, likely to get stuck in suboptimal action
- ϵ too large: in the long run, keeps making wrong moves even if it has already found the most optimal action.

2 Markov Decision Process

3 Dynamic Programming

Iterative Policy Evaluation Algorithm

Input: policy π to be evaluated
Initialize $V(s) = 0$ for all $s \in \mathcal{S}^+$ (arbitrarily, except $V(terminal) = 0$)
repeat
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small threshold determining accuracy)
Output: $V \approx v_\pi$

4 Monte Carlo

First-visit MC prediction Algorithm

Input: policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ empty list for all $s \in \mathcal{S}$
repeat
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t does not appear in $S_0, S_1, \ldots, S_{t-1}$ [a] **then**
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$
until forever

[a] Note that without this line, this algo becomes full MC prediction

This algorithm tries to estimate $v_\pi(s)$, the value of some state s under policy π, given a set of episodes obtained by following policy π and passing through s. Note that s may be visited multiple times in the same episode. We call each visit of some episode s a 'visit' of that state, and first-visit just evaluates following first visits to s in the episode, averaging them. Every-visit MC prediction is just the same algorithm, without the check for S_t having occurred earlier in the episode.

4.1 Monte Carlo Control

Monte Carlo Control keeps track of an alternate approximate value function, and that value function is then altered every iteration to more closely approximate the real value function.

Policy improvement of this control algorithm is done by making the policy greedy with respect to the current value function.

For any action-value function q, the corresponding greedy policy is the one that, foreach $s \in S$, chooses the action with maximal value:

$$\pi(s) \doteq argmax_a q(s, a) \tag{4}$$

Policy improvement can then be done by constructing each π_{k+1} as the greedy policy, with respect to q_{pi_k}

Monte Carlo ES (Exploring Starts) Algorithm

Initialize:
  $Q(s, a) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  $\pi(s) \leftarrow$ arbitrary policy for all $s \in \mathcal{S}$
  $Returns(s, a) \leftarrow$ empty list for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
repeat
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if (S_t, A_t) does not appear in $(S_0, A_0), (S_1, A_1), \ldots, (S_{t-1}, A_{t-1})$ **then**
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
until forever

5 TD Learning

TD Learning is a combination of ideas from Monte Carlo and Dynamic Programming. We update estimates based in part on other learned estimates, without waiting for a final outcome.

Monte Carlo methods need to wait until the end of the episode until determining $V(S_t)$ increment, because only then do we know what G_t is going to be. But! TD methods wait only until the next time step, at $t+1$ they immediately form a target and make a useful update using R_{t+1}

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{5}$$

Note that this is a special case of the TD(γ) algorithm:

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$
repeat
  Initialize S
  repeat
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal
until forever

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$
repeat
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  repeat
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal
until forever

5.1 On-policy vs. Off-policy

All control methods need to behave non-optimally in order to find out optimal behaviour. The **on-policy** approach to this is a compromise: we learn action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach, is to use two policies, one learned about and becomes optimal, and one that defines exploratory behaviour. These are called the *target policy* and the *Behaviour policy*. In case we say that learning data is "off" the target policy, we are doing **off-policy learning**

5.2 Samples of on-policy vs off-policy methods

5.2.1 On-policy methods

- **SARSA** - State-Action-Reward-State-Action, updates Q-values using the policy being followed
- **Policy Gradient methods** - REINFORCE, Actor-Critic, A3C, PPO, TRPO
- **Monte Carlo Control** - Uses same policy for exploration and learning
- **TD(0)** - Temporal Difference learning with same policy
- **n-step SARSA** - Multi-step version of SARSA
- **SARSA(λ)** - Eligibility trace version of SARSA
- **Actor-Critic** - Policy gradient with value function baseline
- **A3C** - Asynchronous Advantage Actor-Critic
- **PPO** - Proximal Policy Optimization
- **TRPO** - Trust Region Policy Optimization
- **MCTS** - Monte Carlo Tree Search variants

5.2.2 Off-policy methods

- **Q-Learning** - Uses max Q-value for updates regardless of policy followed
- **Expected SARSA** - Updates using expected value over all actions
- **DQN** - Deep Q-Network, neural network version of Q-learning
- **Double DQN** - Addresses overestimation bias in DQN
- **Dueling DQN** - Separates state value and action advantage
- **Rainbow DQN** - Combination of multiple DQN improvements
- **DDPG** - Deep Deterministic Policy Gradient
- **TD3** - Twin Delayed Deep Deterministic Policy Gradient
- **SAC** - Soft Actor-Critic with entropy regularization
- **Off-policy Actor-Critic** - Uses importance sampling
- **Retrace(λ)** - Safe off-policy correction
- **Tree-backup** - Multi-step off-policy without importance sampling
- **QR-DQN** - Quantile Regression DQN for distributional RL
- **C51** - Categorical distributional RL algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$
repeat
  Initialize S
  repeat
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal
until forever

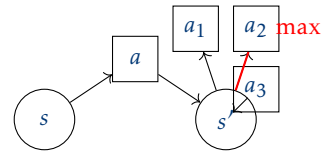
Rewritten, the update formula would be:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R_{t+1} + \gamma max_a(Q_{S+1},a) - Q(S_t,A_t)] \tag{6}$$

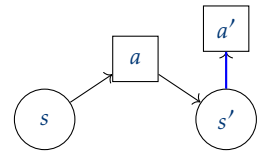
So the same as SARSA, except now grabbing the action taken in the next state with our policy, we grab the $Q(S,a)$ when taking some maximal action a in the next step

5.3 Backup Diagrams

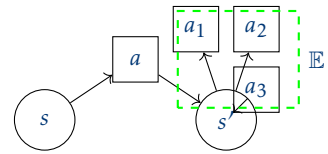
The following diagrams show how different TD methods update their value estimates:



Q-learning



SARSA



Expected SARSA

- **Q-learning**: Uses $\max_a Q(s',a)$ (off-policy)
- **SARSA**: Uses $Q(s',a')$ where a' is chosen by policy (on-policy)
- **Expected SARSA**: Uses $\sum_a \pi(a|s')Q(s',a)$ (expected value)

5.4 Expected SARSA

Expected Sarsa uses expected value under the current policy:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1},a) - Q(S_t,A_t)] \tag{7}$$

6 Function Approximation

7 n-step TD

n-step TD for estimating $V \approx v_\pi$

Input: policy π to be evaluated
Algorithm parameters: step size $\alpha \in (0,1]$, positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$, except that $V(terminal) = 0$
repeat
  Initialize and store $S_0 \neq terminal$
  $T \leftarrow \infty$
  for $t = 0, 1, 2, \ldots$ **do**
    if $t < T$ **then**
      Take action according to $\pi(\cdot|S_t)$
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      if S_{t+1} is terminal **then**
        $T \leftarrow t+1$
    $\tau \leftarrow t - n + 1$ (time whose estimate is being updated)
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      if $\tau + n < T$ **then**
        $G \leftarrow G + \gamma^n V(S_{\tau+n})$ (bootstrap)
      $V(S_\tau) \leftarrow V(S_\tau) + \alpha[G - V(S_\tau)]$
until $\tau = T - 1$

So, when calculating, for example for a 3-step TD:

$$V_{new}(S) \leftarrow V_{old}(S) + \alpha * ((R_t + \gamma * R_{t+1} + \gamma^2 * R_{t+2}) + \gamma^3 * V_o \tag{8}$$

8 General assignments

8.1 features + tiles

You have 5 tilings of $10 * 10$ tiles. This means you have $10 * 10$ features in each tiling (one tile = one feature), and total $10 * 10 * 5 = 500$ features in all tilings.

8.2 On tilings

For each tiling, each state will be on exactly one tile from that tiling, there are no overlap tiles. So, for that tile, there is a one-hot "1" in the vector and the rest is 0. So, for 5 tilings, there are 5 1's in x(s).

8.3 Picking a good step-size α

A rule of thumb for this is:

$$a \doteq (\tau \mathbb{E}[\mathbf{x}^T \mathbf{x}])^{-1} \tag{9}$$

Supposing you want to learn about τ experiences with a substantially similar feature vector, with $\mathbf{x}$ being a random feature vector chosen from the same distribution as input vectors will be in the Stochastic Gradient Descent.

Say we have 5 tilings, for each feature vector $\mathbf{x}$ there will be exactly 1 active tile in each tiling. That means $\mathbb{E}[\mathbf{x}^t \mathbf{x}] = 5$, regardless of distribution. so, with $\tau = 10$:

$$\alpha = \frac{1}{10 \times 5} = 0.02 \tag{10}$$

9 Policy Gradient

9.1 Stochastic Gradient Descent

In SGD, we assume that states appeach in examples with the same distribution μ, over which we are trying to minimize $\overline{VE}$. A grood strategy is to minimize error, by adjusting weight vector after each sample in the direction that would reduce the most error:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)]^2 \tag{11}$$

where α is step-size, $\nabla f(\mathbf{w})$, is for any scalar expression $f(\mathbf{w})$ that is a function of a vector. It is the column vector of partial derivatives, with respect to the vector, or if you will, the gradient:

$$\nabla f(\mathbf{w}) \doteq (\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w_d})^T \tag{12}$$

SGD methods are 'gradient methods' because the overall step in $\mathbf{w}_t$ is proportional to the negative gradient of the example's squared error! The general SGD method for state-value prediction:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[U_t - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t) \tag{13}$$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$

Algorithm parameters: step size $\alpha > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily

repeat

  Initialize S (start of episode)

  repeat

    $A \leftarrow$ action given by π for S

    Take action A, observe R, S'

    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla\hat{v}(S, \mathbf{w})$

    $S \leftarrow S'$

  until S is terminal

until forever