Reinforcement Learning

Workshop 2: Approximation Methods (Part 1)

Netbrain.ml

Logistics

Based on

"Reinforcement Learning: An Introduction" by Sutton and Barto (Approximation Solution Methods)

Introduction to Reinforcement Learning Lecture series by David Silver (6-7)

Last time: Workshop 1: An introduction and tabular methods

Workshop 2: Approximation methods (Part 1)

Introduction to Approximation Methods

Next week: Workshop 3: Approximation methods (Part 2)

A Deeper Look into Approximation Methods

Slides and notebooks: https://netbrainml.github.io/workshop/

Motivation for Approximation Methods

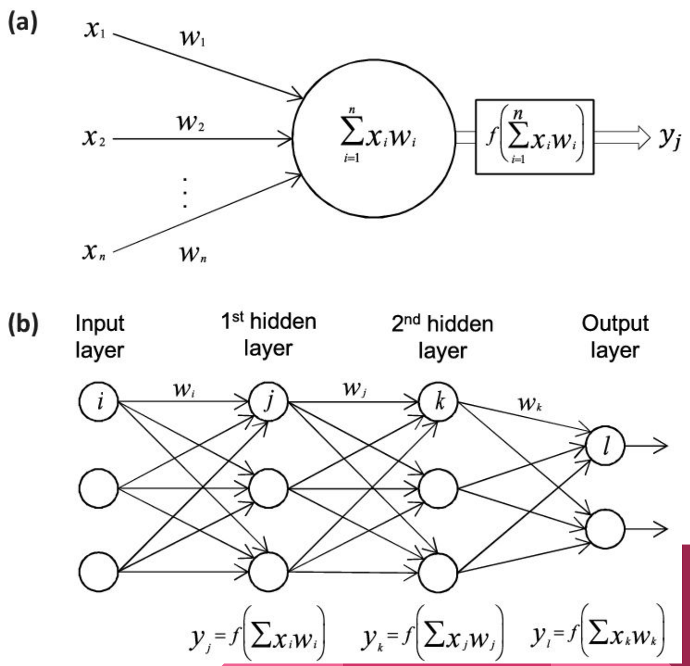
What is the issue with tabular methods?

For large scale RL tasks:

- Memory issue
- Need to explore and learn on all states for informed decision-making
  - LUT is unable to generalize.

Function approximation:

- Instead of a table, use some function as the value function and/or policy.
- For example, we can use a ML/DL model as the function approximator.
- In this workshop, we will focus more on using neural networks as our function approximator.



Overview

Approximate the value function:

Value Approximation Methods

- For this workshop series, we will focus on DQN methods with gradient-based optimization

Approximate the policy:

Policy Approximation Methods

- For this workshop series, we will focus on policy gradient methods
- We will only introduce one DFO method today but more next time

Approximate the policy and value function:

Actor Critic Approximation Methods

Approximate the model:

Model-based Approximation methods

Approximation Methods with Value Approximation

Value Approximation Method

Recall

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Let $\hat{v}(s,\mathbf{w})$ be the value function approximation parametrized by w, and let $v_\pi(s)$ be the target value function.

We define a loss function: $\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s)\Big[v_\pi(s) - \hat{v}(s,\mathbf{w})\Big]^2.$ where u(s) is some weighting.

We can use gradient-based optimization to adjust the weights

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t)\end{aligned}$$

MC and TD-Learning Approach

$$\overline{\mathrm{VE}}(\mathbf{w}) \doteq \sum_{s\in\mathcal{S}} \mu(s)\Big[v_\pi(s) - \hat{v}(s,\mathbf{w})\Big]^2.$$

If we use MC approach, we let $v_\pi(s)$ be the return
If we use TD-Learning or n-step bootstrapping, we let $v_\pi(s)$ be the TD target/n-step return.
Then we can update the weight vector using SGD:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t)\end{aligned}$$

For bootstrapping methods, we call the update semi-gradient. [0]

The following approaches are called Gradient MC and Semi-Gradient SARSA

[0] Baird, L. C., Klopf, A. H. (1993). Reinforcement learning with high-dimensional, continuous actions. Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147.

Experience Replay

Store past interactions with environment in a replay buffer

- “Each step of experience is potentially used in many weight updates, which allows for greater data efficiency.” [0]
- “Learning directly from consecutive samples is inefficient, due to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates.” [0]
- “When learning on-policy the current parameters determine the next data sample that the parameters are trained on.” [0]
- “By using experience replay the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters” [0]

[0] Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Graves, Alex & Antonoglou, Ioannis & Wierstra, Daan & Riedmiller, Martin. (2013). Playing Atari with Deep Reinforcement Learning.

Deep-Q Networks

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Graves, Alex & Antonoglou, Ioannis & Wierstra, Daan & Riedmiller, Martin. (2013). Playing Atari with Deep Reinforcement Learning.

Approximation Methods with Policy Approximation

Policy Approximation Method

Recall a policy is a function used to map state to action.

$$\pi(s) = a \qquad\qquad \pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$$

If we parameterized the policy, we have $\pi_\theta(a|s) = P[a|s]$

How can we update the policy so that we converge to an optimal policy?

Objective: Increase the likelihood of selecting actions with the highest expected return

Methods of updating policy:

Policy Gradient

DFO (Derivative Free Optimization)

- Cross Entropy Optimization (CE)

Policy Gradients

Find the gradient of the reward function wrt parameters of the policy, and perform gradient ascent.

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau \sim \pi_\theta}{\mathrm{E}} [R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) \\
&= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\
&= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \\
&= \underset{\tau \sim \pi_\theta}{\mathrm{E}} [\nabla_\theta \log P(\tau|\theta) R(\tau)] \\
\therefore \nabla_\theta J(\pi_\theta) &= \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \right]
\end{aligned}
$$

REINFORCE

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Cross Entropy Optimization

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration $= 1, 2, \ldots$ **do**
Collect n samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
Perform a noisy evaluation $R_i \sim \theta_i$
Select the top $p\%$ of samples (e.g. $p = 20$), which we'll call the **elite set**
Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new μ, σ.
end for
Return the final μ.

Approximation Methods with Actor Critic Methods

Actor Critic

Approximate both the policy and the value function.

Reducing variance of reward signal by using a value function approximation.

If we use gradient updates for the policy approximation (actor):

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right]$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right]$$

For the value function approximation (critic), we can follow the same procedure as before.

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s\in\mathcal{S}} \mu(s)\Big[v_\pi(s) - \hat{v}(s,\mathbf{w})\Big]^2.$$

Advantage Function

Advantages is another measure that can be considered as another version of Q-value with lower variance by taking the state-value off as the **baseline**.

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t)$$
$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})]$$
$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

Advantage Actor Critic

Use the advantage function instead of the value function approximation to calculate the gradient.

$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

$$\begin{aligned}\nabla_\theta J(\theta) &\sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\end{aligned}$$

Approximation Methods with Model-based Methods

Model Approximation with DYNA

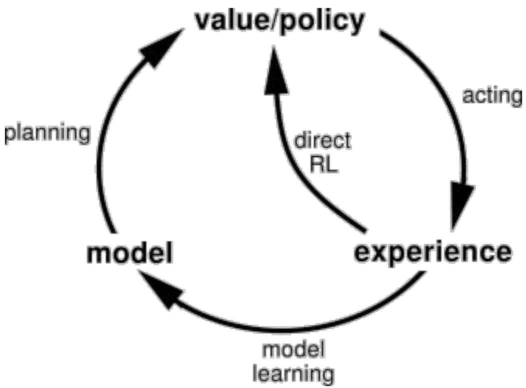
DYNA: Model-based approach to integrate learning and planning

Why might this approach be beneficial?

Approximate and learn the model

$$P(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

Optimize the policy/value function by planning with the model approximation.



DYNA-Q

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S, Q)

(c) Take action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Q-learning (steps a–d)

Model Update (step e)

Planning Step (step f)