Chapter 08 – Linear Approximation

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Introduction

Introducing this new chapter, we can reason about the problems of tabular representations, which are:

- High memory
- Require much time and data
- Lack of generalization

The question is now that instead of using tables, we will represent V and Q with parametrized functions:

$$V(s, \boldsymbol{w}) \approx V^{\pi}(s)$$

and equivalently

$$Q(s, a, \boldsymbol{w}) \approx Q^{\pi}(s, a)$$

Where $\boldsymbol{w}$ are the weights and $V(s, \boldsymbol{w})$ can be any linear combination of features (Decision Tree, KNN, Neural Networks, etc.).

Now, how do we optimize V? We can iterate the optimization process of the approximated value function by, for example, minimizing the Mean Squared Value Error (MSVE):

$$MSVE(\boldsymbol{w}) = \sum_{\boldsymbol{s}} d(s) \cdot [Q(s, a) - Q_{\pi}(s, a, \boldsymbol{w})]^2$$

Where:

- $d(s)$ = It's the distribution over the states s. It's a value between [0, 1] such that $\sum_s d(s) = 1$
- $Q_{\pi}(s, a, \boldsymbol{w})$ = Approximation of Q
- $Q(s, a)$ = True value of Q

We seek for the best possible value of $\boldsymbol{w}^*$ such that $RMSE(\boldsymbol{w}^*) \leq RMSE(\boldsymbol{w})$ for all w.

2. Optimization through Gradient Descent

In order to get the best value of $\boldsymbol{w}$, we could use Gradient Descent which optimizes a function $f(x, \boldsymbol{w})$ by minimizing an error function $J(\boldsymbol{w})$. This is done iteratively, by updating w in the following way:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_t)$$

The question is how do we explicit $J(\boldsymbol{w}_t)$? Well, using the Mean Squared Error (MSE):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot \nabla_{\boldsymbol{w}} \left[\frac{1}{2} \cdot \left(f^* - f(x_t, \boldsymbol{w}_t)\right)^2\right] =$$
$$= \boldsymbol{w}_t - \alpha \cdot \left(f^* - f(x_t, \boldsymbol{w}_t)\right) \cdot \nabla_{\boldsymbol{w}} f(x_t, \boldsymbol{w}_t)$$

If we simply plug in our information that $f(x_t, \boldsymbol{w}_t) = Q(s, a, \boldsymbol{w})$ and that $f^* = Q^{\pi}$, we obtain that:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot \left({Q^{\pi}}_t - Q(s_t, a_t, \boldsymbol{w}_t)\right) \cdot \nabla_{\boldsymbol{w}} Q(s_t, a_t, \boldsymbol{w}_t)$$

2.1 Linear combination State-Action Value approximation

Now we need a way to explicit $Q(s_t, a_t, \boldsymbol{w}_t)$.

In order to do so, we might represent our state with a feature vector (which will be covered also later), in this form:

$$\boldsymbol{x}(s,a) = (x_1(s,a), \dots \ , x_n(s,a))^T$$

And by considering a parametric Q-function in the form $Q(s_t, a_t, \boldsymbol{w}_t) = \boldsymbol{w}^T \boldsymbol{x}(s_t, a_t)$, then we can reformulate everything by explicating our error function as:

$$J(w_t) = \mathbb{E}(Q^{\pi}(s_t, a_t) - \boldsymbol{w}^T \boldsymbol{x}(s_t, a_t))^2$$

Such that the final formula becomes:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot \left(Q^{\pi}{}_t - Q(s_t, a_t, \boldsymbol{w}_t)\right) \cdot \nabla_w \, Q(s_t, a_t, \boldsymbol{w}_t)$$

$$\Downarrow$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot \left(Q^{\pi}{}_t - \boldsymbol{w}^T \boldsymbol{x}(s_t, a_t)\right) \cdot \nabla_w \, \boldsymbol{w}^T \boldsymbol{x}(s_t, a_t)$$

$$\Downarrow$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \cdot \left(Q^{\pi}{}_t - \boldsymbol{w}^T \boldsymbol{x}(s_t, a_t)\right) \cdot \boldsymbol{x}(s,a)$$

The $\boldsymbol{x}(s,a)$ is the basically the gradient of $\boldsymbol{w}^T \boldsymbol{x}(s_t, a_t)$ w.r.t to w. In fact, this quantity is equal to $\boldsymbol{x}(s_t, a_t)$.

The $Q^{\pi}{}_t$ becomes our target y, since we don't know the true Q.

In forward-view. it is equal to:

- Monte-Carlo = G_t
- $TD(0)$ = $R_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1}, \boldsymbol{w})$
- Forward-view $TD(\gamma)$ = G_t^{λ}

For backward view $TD(\gamma)$, the update is composed as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \cdot \delta_t \cdot e_t$$

Where:

- $\delta_t = R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}, \boldsymbol{w}) - Q(s_t, a_t, \boldsymbol{w}_t)$
  - Here, as we did in standard TD, (*) is our TD target, while (*) is our approximated Q.
- $e_t = \gamma \cdot \lambda \cdot e_{t-1} + \nabla_w \, Q(s_t, a_t, \boldsymbol{w})$

Substantially, for all the weight vector, I use the gradient information (*) to say how much each weight is contributing to the prediction (instead of using the Indicator Function $\boldsymbol{I}$ as we did in Eligibility traces).

3. Least squares

Incremental methods use the normal Gradient Descent, which is simple but not efficient, due to the fact that we have to calculate it for every sample.

Batch methods instead, seek to find the best fitting value function over a batch of samples (as we do in Supervised Learning).

From now on, we'll refer to our target ${Q^\pi}_t = y_t$ // ${Q^\pi}_1 = y_1$ // $Q^\pi = y$.

Given the following variables:

- Q function approx. := $Q(s, a, \boldsymbol{w}) \sim y(s, a)$
- Experience D := $\{(s_1, a_1, y_1), (s_2, a_2, y_2), \dots, (s_n, a_n, y_n)\}$

The least squares algorithm finds a parameter vector $\boldsymbol{w}$, minimizing the following error through SGD or with the closed form solution (Linear version):

$$J(\boldsymbol{w}) = \sum_i (y_i - Q(s_i, a_i, \boldsymbol{w}))^2$$

3.1 Closed form solution

In case of the linear value function approx., then: $Q(s, a, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s, a)$

We would like to find the argmin of this quantity here:

$$argmin_{\boldsymbol{w}} [\sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}(s_i, a_i))^2]$$

Achieving that $\boldsymbol{w} = (X^T X)^{-\mathbf{1}} \cdot X \cdot y$, where:

- $(X^T X)^{-\mathbf{1}}$:= the pseudo-inverse of X
- $X = [(x(s_1, a_1), \dots, x(s_n, a_n))^T]$
- $y = [y_1, y_2, \dots, y_n]^T$

3.2 SGD with Experience Replay

Given experience $D = \{(s_1, a_1, y_1), (s_2, a_2, y_2), \dots, (s_n, a_n, y_n)\}$

We repeat the following process:

- Sample a batch from D = (s, a, y)
- Apply SGD update: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \cdot (y - Q(s, a, \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(s, a, \boldsymbol{w})$

3.3 Features

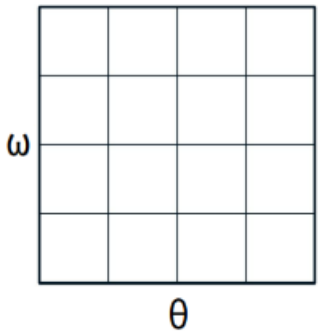
Features, in the context of RL play a very important role, as most of the times features should be inspected in combination and not independently.

For the pendulum task, we would like to know if the pendulum is swinging up or down.

Namely, we can know this through 2 features: Angle θ and Angular velocity ω.

3.3.1 Tile Coding

Considering the pendulum task again, it is a continuous 2D state space (θ and ω), so we can represent it through a grid:



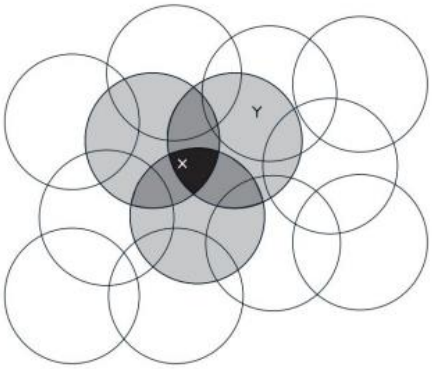
In tile coding, what we would like to do is having a binary feature representation of the space, which encodes the presence of the current state in a discretized space.

Basically, imagine this grid is composed by zeros (0). When the pendulum will have a certain angle and a certain angular velocity, a specific cell will be 1. It's like a one-hot encoding of the space:

$$[0, 0, 0, \dots, 1, 0, 0]$$

3.3.2 Coarse Coding

Not going very far from Tile Coding, we consider this as a generalization of it.



We can think of this as being in multiple space at a time, thus having multiple 1s in the one-hot encoding vector.

Representing x, it results in:

$$[0, 0, 0, \dots, 1, 1, 1, 0, 0]$$