Function Approximation

State Aggregation has several disadvantages:

1. Limited Precision
2. Complexity increases exponentially with dimensions of discretized states:

$$O(n^k) \tag{1}$$

where n is the number of segments (aggregated discrete states) and k is the number of dimensions.

Thus, we use a function f that is composed of weights w to approximate the value of states or state-actions. At every policy iteration, we adjust the weights until the function is sufficiently close to the optimal value function:

$$f_1(s|w) \rightarrow f_2(s|w) \rightarrow \cdots \rightarrow f_n(s|w) \approx v_*(s). \tag{2}$$

Linear Approximators

A linear approximator consists of the sum of each dimension of the state scaled by its respective parameter:

$$\begin{aligned} f(s|w) &= \hat{v}(s|w) \\ &= w \cdot s^T \\ &= w_1 \cdot s_1 + w_2 \cdot s_2 + w_3 \cdot s_3 + \cdots + w_n \cdot s_n \end{aligned} \tag{3}$$

where the state is represented by vector s and the parameters are represented by vector w below:

$$s = [s_1, s_2, s_3, \cdots, s_n], \text{ and} \tag{4}$$

$$w = [w_1, w_2, w_3, \cdots, w_n]. \tag{5}$$

Polynomial Approximators

A polynomial approximator consists of the sum of each dimension of the state and the exponents of the component up to a certain degree scaled by its respective parameter:

$$\begin{aligned} f(s|w) &= \hat{v}(s|w) \\ &= w \cdot s^T \\ &= w_1 \cdot s_1 + w_2 \cdot s_2 + w_3 \cdot s_3 + \cdots + w_n \cdot s_n \end{aligned} \tag{6}$$

where the state is represented by vector $\phi(s)$ and the parameters are represented by vector w below:

$$\phi(s) = [s_1, s_1^2, \cdots, s_1^j, s_2^1, \cdots, s_2^k, \cdots, s_m], \text{ and} \tag{7}$$

$$w = [w_1, w_2, w_3, \cdots, w_n] \tag{8}$$

where the each state components may have different number of exponents, and all the terms summed must be matched by the number of weights such that:

$$n = j + k + \cdots + m. \tag{9}$$

Neural Networks

Neural networks can approximate value functions by using a set of layers that contains artificial neurons that have parameters. The linear part of the neuron aggregates the inputs into a weight sum v_k w.r.t its parameters:

$$v_k = w_1 \cdot s_1 + w_2 \cdot s_2 + w_3 \cdot s_3 + \cdots + w_n \cdot s_n \tag{10}$$

which is consumed by the (usually) non-linear activation function:

$$y_k = \phi(v_k). \tag{11}$$

The whole operation can be re-expressed as:

$$y_k = \phi(\sum_{i=1}^{n} w_{k_i} \cdot x_i) \tag{12}$$

where w_{k_i} is the i-th weight of the k-th neuron of a given layer of the neural network.

For a given input vector $x = [x_1, x_2, x_3]$, the linear/aggregation portion layer that takes in x is represented by $3 \times k$ matrix w where k is the number of neurons/outputs of this layer:

$$[x_1, x_2, x_3] \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} = [y_1, y_2, y_3, y_4] \tag{13}$$

where each column vector of w represents the weights of the neuron of the layer.

The finally, aggregated vector goes through the activation function:

$$h = [\phi(y_1), \phi(y_2), \phi(y_3), \phi(y_4)]. \tag{14}$$

Overall, the whole (2-layer for example) neural network is represented by:

$$\hat{y} = \phi_2(\phi_1(x \cdot w_1) \cdot w_2). \tag{15}$$

Activation Functions

Activation functions helps approximate complex functions by adding non-linearity to the outputs of the neurons and, subsequently, the layers of the neural networks.

Activation Function	**Formula**	**Description**
Rectifier function	$\phi(x) = \max(x, 0)$	Propagates only positive outputs Mainly deployed in the hidden layers to help speed up the learning process of the neural network Can approximate complex functions

Activation Function	Formula	Description
Sigmoid	$S(s) = \frac{1}{1+e^{-x}}$	Compresses the input into $(0, 1)$ Usually in the last layer for classification Slows down the learning if put in the hidden layers

Stochastic Gradient Descent

Update rules for the parameters:

$$w_{t+1} = w_t - \alpha \nabla \hat{L}(w) \tag{16}$$

where $\nabla \hat{L}(w)$ is the gradient vector:

$$\nabla \hat{L}(w) = [\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \cdot, \frac{\partial L}{\partial w_n}] \tag{17}$$

where each component of the gradient vector is the partial derivative that indicates the direction to nudge each parameter of the neural network to so that the cost function $\hat{L}(w)$ increases. The parameters are updated within and across the layers using a technique called Backpropagation.

Cost Functions

Mean Absolute error takes the average of all the errors:

$$L(w) = \frac{1}{M} \sum_{i=1}^{N} |y_i - \hat{y}_i|. \tag{18}$$

Mean Squared error penalized the bigger errors more by squaring the error:

$$L(w) = \frac{1}{M} \sum_{i=1}^{N} [y_i - \hat{y}_i]^2. \tag{19}$$

We can marry Mean Squared error with Temporal-Difference error by getting the average of all the TD-errors squared:

$$\hat{L}(w) = \frac{1}{M} \sum_{i=1}^{N} [R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1} | w) - \hat{q}(S_t, A_t | w)]^2. \tag{20}$$

where the target is the bootstrapped $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1} | w)$, while the prediction is the current value $\hat{q}(S_t, A_t | w)$.