Why Move to Parameterized Functions?

In reinforcement learning, when the state or state-action space is **large or continuous**, we can't represent value functions exactly in a table. So we **approximate** the value function using **parameterized functions** like:

- Linear combinations of features
- Neural networks
- Other function approximators

1. Value Function Approximation

We aim to approximate the true value function $v_\pi(s)$ with a parameterized function $\hat{v}(s, w)$, where w are learnable parameters.

2. Prediction with Function Approximation

- The **goal** is to learn w so that $\hat{v}(s, w) \approx v_\pi(s)$
- We minimize **mean squared error** between estimated and true value over a distribution of states.

3. Gradient Descent Methods

- Update weights using **stochastic gradient descent (SGD)**:

$$w_{t+1} = w_t + \alpha \cdot \left(U_t - \hat{v}(S_t, w_t)\right) \cdot \nabla_w \hat{v}(S_t, w_t)$$

where U_t is a target (e.g., a Monte Carlo return or TD target)

4. Control with Function Approximation

- For **action-value function approximation**, we use $\hat{q}(s, a, w)$
- In control settings, we use approximated action-value functions to inform policy improvement (e.g., ϵ\epsilonϵ-greedy policies)

Generalization vs. Discrimination

1. **Generalization**
   - **Definition**: The ability of a function approximator to apply learned knowledge from one state (or state-action pair) to **similar** states.

- **Why it matters**: In large or continuous state spaces, the agent cannot visit every state. Generalization allows the agent to **perform well in unseen or infrequent situations**.
- **Example**: If the agent learns that a red button gives high reward in one room, it might generalize that red buttons are good in other rooms too.

2. **Discrimination**
   - **Definition**: The ability to **distinguish** between different states (or state-action pairs) and assign them **different values or policies**.
   - **Why it matters**: Some states might look similar but require different actions. Over-generalization can lead to **incorrect decisions**.
   - **Example**: A red button in one room gives reward, but in another room it triggers a trap — the agent must **discriminate** based on more than just the color.

3. **The Trade-Off**
   - You want **just enough generalization** to reduce learning time and improve performance in new areas.
   - You also need **sufficient discrimination** to avoid overgeneralizing and making mistakes.
   - The design of **feature representations**, **neural network architectures**, and **training data** influences this balance.

Goal of Value Estimation

Estimate the **value function**:

- $v_\pi(s)$: expected return following policy π from state s
- or $q\_\pi(s, a)$: expected return from state-action pair (s, a)

The Supervised Learning Analogy

In **supervised learning**, we are given:

- **Input**: x
- **Target**: y

- **Model**: $f(x, w)$ approximating the mapping from input to target

In **value estimation**, we interpret:

- **Input**: state s (or state-action pair (s, a))
- **Target**: estimate of return (from Monte Carlo or TD learning)
- **Model**: $\hat{v}(s, w)$ or $\hat{q}(s, a, w)$, a parameterized function

We then **minimize the prediction error** just like in supervised learning.

Loss Function (Mean Squared Error)

$$LOSS = \mathbb{E}_s\left[\left(\hat{v}(s, w) - v_{target}(s)\right)^2\right]$$

- Where $v_{target}(s)$ is the **learning target**, such as:
  - Monte Carlo return: $G_t = R_{t+1} + R_{t+2} + \cdots$
  - TD target: $R_{t+1} + \gamma \hat{v}(S_{t+1})$

Gradient Descent Update

$$w \leftarrow w - \alpha \cdot \nabla_w \left(\hat{v}(s, w) - v_{target}(s)\right)^2$$

Or equivalently:

$$w \leftarrow w - \alpha \cdot \delta_t \cdot \nabla_w \hat{v}(s, w)$$

Where $\delta_t = v_{target} - \hat{v}(s, w)$

The Objective for On-Policy Prediction

Estimate the **value function** $v_\pi(s)$ accurately **under the same policy** π that is being followed (on-policy), using **function approximation**.

$$\hat{v}(s, w) \approx v_\pi(s)$$

Objective Function (Mean Squared Value Error):

The agent minimizes the **expected squared error** between predicted and true values, weighted by the **on-policy state distribution** $d_\pi(s)$

$$MSVE(w) = \sum_s d_\pi(s)[v_\pi(s) - \hat{v}(s, w)]^2$$

- $d_\pi(s)$: steady-state distribution over states when following π
- The function approximation should focus on performing well **where the agent spends most of its time** (according to d_π).

Training with Gradient Descent

- **Target** can be a **Monte Carlo return** or a **TD target**
- Weight update rule (for TD learning):

$$w_{t+1} = w_t + \alpha \cdot \delta_t \cdot \nabla_w \hat{v}(S_t, w_t)$$

where $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)$

TD Objective

The TD learning **does not directly minimize** a fixed loss function like Monte Carlo methods do (e.g., MSE between return and estimate).
Instead, TD methods aim to **minimize the expected TD error** over time.

So, the effective objective is:

$$\min_w \mathbb{E}_\pi[\delta_t^2]$$

where:

- w are parameters of the value function $\hat{v}(s, w)$
- δ_t is the TD error
- The expectation is over states, actions, rewards from policy π

TD Update Rule (for function approximation)

$$w_{t+1} = w_t + \alpha \cdot \delta_t \cdot \nabla_w \hat{v}(S_t, w_t)$$

Linear TD

Linear TD is a special case of TD learning where the **value function is approximated as a linear combination of features**. It's simple, efficient, and often used as a theoretical foundation in reinforcement learning research.

We want to estimate the value function $\hat{v}(s, w) \approx v_\pi(s)$, but instead of using a table, we use:

$$\delta_t = R_{t+1} + \gamma w_t^T x(S_{t+1}) - w_t^T x(S_t)$$

Where:

- $x(s) \in \mathbb{R}^d$ is a **feature vector** for state s
- $w \in \mathbb{R}^d$ is a **weight vector**

TD(0) Update Rule (Linear Case)

Linear TD(0) **does not** minimize the mean squared value error directly. Instead, it minimizes an approximation known as the:

$$MSBPE(w) = \|\Pi(T^\pi \hat{v}) - \hat{v}\|_D^2$$

Where:

- $T^\pi \hat{v}$: Bellman operator applied to the current estimate
- Π: projection onto the space of linear functions spanned by features
- D: diagonal matrix representing the state distribution d_π