**A Deeper Dive into Policies and Values in**

**GridWorld Reinforcement Learning**

**Introduction:**

The main objective in Reinforcement Learning (RL) entails training an agent (which serves as our decision-making entity) to interact with an environment (which encompasses GridWorld and its rules and rewards along with states) so the agent can optimize cumulative rewards collected during each interaction. The agent must discover both proper actions and situation quality assessments in order to accomplish its goal. Two core principles from RL appear as Policies and Values in a formalized state. RL algorithms require clear comprehension of how Policies and Values function individually and interact together for complete understanding of their operational principles.

**Understanding the Policy (π): The Agent's Strategy**

* **What it is:**

The agent produces its behavior guideline through a policy format. The policy determines which action the agent should carry out when located at a particular state in the GridWorld environment (specific cell). The contingency plan consists of complete guidance which functions as directions:

*"If you are at coordinates (x, y), then perform action Z."*

* **Representation in GridWorld:** As seen in the notebooks, a policy is often represented as a mapping (like a Python dictionary) that links each state (e.g., (0,0)) to a specific action ('up', 'down', 'left', 'right'). Crucially, terminal states (like the goal state (0,3)) and barrier states (like (1,1)) do not have actions associated with them in the policy, as the episode ends or movement is impossible from these states. The policies shown in the examples are *deterministic*, meaning for any given state, the policy specifies exactly one action. (In more general RL, policies can be *stochastic*, specifying probabilities for taking each action in a state).
* **Purpose and Role:** The policy directly controls the agent's actions. An agent equipped with a policy simply looks up its current state in the policy map and executes the prescribed action. The initial policies provided in the notebooks (like the one in IterativePolicyEvaluation.ipynb) are often arbitrary starting points. They represent an initial guess or a simple strategy that the RL algorithms will then evaluate and improve upon.
* **The Quest for Optimality (π\*):** While any set of state-action rules constitutes *a* policy, the ultimate aim of algorithms like Policy Iteration and Value Iteration is to discover an *optimal policy*, denoted π\*. This is the policy that, if followed, yields the maximum possible expected cumulative discounted reward from *any* starting state. Finding π\* means the agent has learned the best possible way to behave in the GridWorld to achieve its goal.

**Understanding Values (State-Value Function Vπ): The State's Worth**

* **What it is:**

The state-value function together with its policy provides the agent with information about how good it is to be in a specific state under a specific policy. This quantitative measure defines how acceptable a given situation becomes when giving a policy long-term consideration.

* **Formal Definition:**

Under a policy π the value of state s can be written as Vπ(s) and represents the agent's anticipated total reward amount that will be received through displaced reward while executing policy π starting from state s. The expected value consists of probabilities p(s', r | s, a) for state transitions to s' and reward reception of r given s, a together with stochastic policy probabilities π(a|s).

* **Representation in GridWorld:**

Every grid cell receives numerical representation called values. The print\_values() function visualizes these. The initialization process starts by assigning states a zero value which demonstrates ignorance about state worth as shown in ValueIteration.ipynb. The algorithms proceed to conduct multiple iterations that update these numerical scores.

* **The Role of the Discount Factor (γ):** The calculation of values relies heavily on the discount factor, gamma (γ), usually a value between 0 and 1 (e.g., 0.9 in the examples). Discounting serves several purposes:

*0< γ<1*

* + **Mathematical Convenience:** Ensures that the sum of rewards doesn't diverge to infinity in potentially cyclic environments.
  + **Preference for Near-Term Rewards:** Reflects that rewards obtained sooner are often more valuable than rewards obtained further in the future (akin to interest rates in finance or addressing uncertainty about the future). A γ of 0.9 means a reward received one step in the future is only worth 90% of the same reward received now; a reward two steps away is worth 0.9 \* 0.9 = 81%, and so on.

*γ = 0.9*

*Reward two steps away = 0.9\*0.9 = 81% and so on……*

* + **Adjusting Horizon:** A gamma closer to 1 results in an agent that is more far-sighted (valuing future rewards highly), while a gamma closer to 0 makes the agent more short-sighted (primarily concerned with immediate rewards).
* **Purpose and Utility:** State values are essential for decision-making and policy improvement. By comparing the values of potential next states (V(s')), an agent (or the algorithm improving the policy) can determine which actions lead to more promising situations in the long run. Values provide a quantitative basis for comparing states under a given behavioral strategy (policy).

**The Interplay: How Algorithms Leverage Policies and Values**

The algorithms presented demonstrate the dynamic relationship between policies

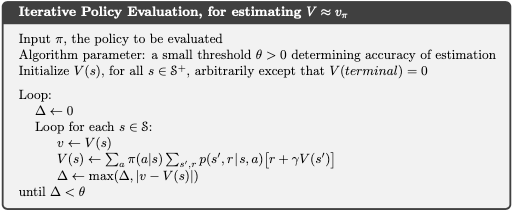
and values:

1. **Iterative Policy Evaluation (IPE):**
   * **Focus:** Given a fixed policy π, compute its corresponding state-value function V ≈ Vπ.
   * **Mechanism:**

The system implements Bellman equation for Vπ as it successively refines value estimation. At each iteration the method updates the value function of every state by evaluating expected returns from following the fixed policy π. The algorithm involves first summing the policy-determined actions when stochastic and then summing possible next states and rewards operated by environment dynamics p(s', r | s, a). The process continues until mathematical convergence is achieved when the maximum value change across states reaches the predetermined threshold level θ.

* + **Key Idea:** Policy is fixed -> Calculate how good that policy is (Values).

The formal algorithm is shown below:



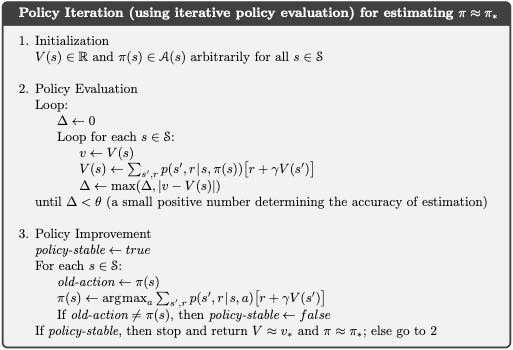
1. **Policy Iteration (PI):**
   * **Focus:** Find the optimal policy (π ≈ π\*) by alternating between evaluating and improving the policy.
   * **Mechanism:**
     + **(Policy Evaluation):** Use Iterative Policy Evaluation (as described above) to calculate the value function V ≈ Vπ for the *current* policy π. This step itself is an iterative loop converging when Δ < θ.
     + **(Policy Improvement):** Create a *new* policy by acting greedily with respect to the value function V just computed. For each state s, choose the action a that maximizes the expected one-step lookahead:

*argmax\_a Σ p(s',r|s,a)[r + γV(s')].*

Check if this new greedy policy is different from the old policy. If the policy remains unchanged (policy-stable is true) across all states, the algorithm has converged to the optimal policy π\* and optimal value function V\*. Otherwise, update the policy to this new greedy policy and return to the Policy Evaluation step. **(Iteration):** Repeat the Evaluation and Improvement steps. The process stops when the Policy Improvement step no longer changes the policy, indicating that the optimal policy (π\*) and its corresponding optimal value function (V\*) have been found.

* + **Key Idea:** Evaluate current strategy -> Use evaluation to find a better strategy -> Repeat until strategy is optimal.

The formal algorithm is shown below:



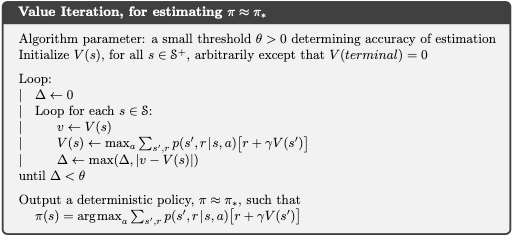
1. **Value Iteration (VI):**
   * **Focus:** Find the optimal value function (V ≈ V\*) directly, then extract the optimal policy (π ≈ π\*).
   * **Mechanism:** It combines evaluation and improvement into a single update step using the Bellman *optimality* equation. In each iteration, it sweeps through states, updating the value of each state s by taking the maximum expected return achievable by taking any action a from that state:

*V(s) ← max\_a Σ p(s',r|s,a)[r + γV(s')]*

This implicitly assumes the best action is taken. The iteration continues until the value function converges (Δ < θ).

* + **(Policy Extraction):** Once V ≈ V\* is found, the optimal policy π\* is extracted in a final step by choosing the action in each state that maximizes the expected return using these optimal values (the same argmax calculation as in Policy Improvement).
  + **Key Idea:** Iteratively update values assuming optimal actions are taken -> Once optimal values are found -> Extract the policy that achieves those values.

The formal algorithm is shown below:



**The Role of PolicyFromValues:**

This utility explicitly shows the *policy extraction* step. Given a set of converged values (either Vπ from IPE or V\* from VI), it constructs the corresponding policy by acting *greedily*. For each state, it examines all possible actions and selects the one that leads to the best combination of immediate reward and discounted value of the next state. This demonstrates how knowledge of state values can be directly translated into a concrete plan of action (a policy).

**In Conclusion:**

Policies and Values are two sides of the same coin in Reinforcement Learning for decision-making under uncertainty. A **policy (π)** is the agent's behavioral strategy (the "how-to"), while the **state-value function (Vπ)** quantifies the long-term consequence or goodness of being in a state under that specific policy (the "how-good"). Algorithms like Iterative Policy Evaluation, Policy Iteration, and Value Iteration provide systematic ways to compute, relate, and improve these components, ultimately aiming to discover an optimal policy that enables the agent to maximize its cumulative rewards in the environment. Understanding this fundamental relationship is key to grasping how agents learn to act intelligently.

**References**

 **Sutton, R. S., & Barto, A. G. (2018).**  
*Reinforcement Learning: An Introduction (2nd Edition).*  
MIT Press.

<http://incompleteideas.net/book/the-book.html>

This is the foundational text that explains policies, value functions, Bellman equations, and key algorithms like Policy Iteration and Value Iteration.

 **David Silver’s Reinforcement Learning Course (Lecture Slides and Videos)**  
University College London  
<https://www.davidsilver.uk/teaching/>

Highly recommended for visual and conceptual clarity around RL components such as state values, action values, and gridworld-style environments.

 **OpenAI Spinning Up in Deep RL**  
*Policy evaluation, policy improvement, value iteration concepts are explained in an applied context.*

<https://spinningup.openai.com/en/latest/>

 **GridWorld Environment – Classic Control Example**  
From gym and gymnasium packages in OpenAI  
<https://www.gymlibrary.dev/>

While modern implementations use more visual environments, the classic GridWorld example is a great minimal environment for theoretical reinforcement learning.

 **CS50’s Introduction to AI with Python**  
Harvard University  
 <https://cs50.harvard.edu/ai/>

Offers hands-on tutorials and labs on value iteration and policy-based decision-making using GridWorld environments.

 **GitHub Repositories – Practical Notebooks**  
Example repo with IPE, Policy Iteration, and Value Iteration using GridWorld:  
 <https://github.com/dennybritz/reinforcement-learning>

Includes Jupyter notebooks for everything described in the file, including IterativePolicyEvaluation.ipynb and ValueIteration.ipynb.