Linear vs Non-linear Value Function Approximation in Reinforcement Learning

Emanda Hailu
GSR 5056/17

Reinforcement Learning Assignment IV Report
Submitted to Azmeraw Bekele

February 5, 2026

0.1 Value function approximation foundations

Reinforcement learning (RL) formalizes sequential decision-making as learning a policy that maximizes expected cumulative (discounted) reward, typically modeled as a Markov decision process (MDP). A central object is the **value function**—either the **state value** $v^\pi(s) = E[G_t \mid S_t = s]$ or **action value** $q^\pi(s, a) = E[G_t \mid S_t = s, A_t = a]$—which predicts long-run outcomes from a state (or state–action pair).

In realistic problems, exact tabular storage of v or q is often infeasible, so one learns an **approximate value function** $\hat{v}(s; \theta)$ or $\hat{q}(s, a; \theta)$ from data. The approximation is **policy evaluation** (estimating v^π or q^π for a fixed π) and/or **control** (improving π using the learned $\hat{q}$ or $\hat{v}$).

A crucial theoretical point is that bootstrapping methods (TD, SARSA, Q-learning) do not, in general, minimize supervised prediction error on Monte Carlo returns; instead they seek a **fixed point of a (projected) Bellman operator** when function approximation is present. For linear TD with on-trajectory sampling, convergence can be proven and the limit can be characterized as a projected solution, with approximation-error bounds under assumptions.

However, function approximation changes stability in profound ways. Even with linear approximation, divergence can occur in settings where samples are not generated "on-line" according to the Markov chain's own dynamics (or, more broadly, when the sampling distribution is mismatched). Moreover, non-linear approximators can introduce additional failure modes.

0.1.1 Visual intuition: linear vs non-linear approximators inside a TD-style backup

Below is a schematic showing the architectural distinction (parameterization) and where it appears in the RL update.

$$\text{TD-style target: } y = r + \gamma\,\hat{V}(s'; \theta_{\text{target}})$$
$$\text{or } y = r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_{\text{target}}).$$

```
Linear value:       phi(s) --> dot(w, phi(s)) = Vhat(s; w)
                    update: w <- w + alpha * delta * phi(s)

Neural value:       s --> [hidden layers] --> Vhat(s; theta)
                    update: theta <- theta + alpha * delta * grad_theta Vhat(s; theta)

Kernel value:       store {(s_i,a_i)}; Qhat(s,a)= Sum_i alpha_i k((s_i,a_i),(s,a))
                    update: alpha_i, dictionary size managed via sparsification / budget
```

This matters because the combination of **bootstrapping**, **function approximation**, and (often) **off-policy learning** is where instability and divergence risks become most acute, motivating specialized stable algorithms (for linear), and stabilizing heuristics (for deep/non-linear).

0.2 Linear approximation methods

A value function approximator is "linear" in RL when it is **linear in its adjustable parameters**:

$$\hat{v}(s; w) = w^\top x(s), \quad \hat{q}(s, a; w) = w^\top x(s, a).$$

This includes many architectures that are non-linear in inputs but still linear in parameters (e.g., radial basis features, Fourier features, tile coding), because the non-linearity is pushed into the fixed feature map $x(\cdot)$.

0.2.1 Common linear function classes used in RL

Tile coding / coarse coding (CMAC-style). The state is mapped to a sparse binary feature vector by overlapping tilings; learning becomes a sparse linear update. This tends to provide strong locality (limited interference) and is easy to implement for low-dimensional continuous states.

Radial basis function (RBF) features. A fixed set of Gaussian-like features provides smooth generalization over continuous spaces; still linear in weights. RBF features will reappear in the empirical section because they are a widely-used "strong linear baseline" in classic control benchmarks.

Least-squares / second-order linear TD methods. Methods such as LSTD and its variants replace incremental SGD-style updates with least-squares-style solutions, often improving sample efficiency but increasing computational and memory cost from $O(n)$ to $O(n^2)$ in the number of features.

0.2.2 Typical algorithms paired with linear approximation

TD(λ), SARSA(λ), Q-learning (linear). These are classic and computationally cheap $O(n)$ per step in feature dimension, but their convergence guarantees depend on sampling conditions; off-policy variants can be unstable even in linear settings.

LSPI / LSTDQ (linear architectures). LSPI is a data-efficient approximate policy iteration method using linear value-function approximation. The approach is designed to reuse data aggressively and avoid sensitive learning-rate schedules.

Gradient TD family (stable off-policy linear learning). To address off-policy instability with linear approximation, gradient-based TD algorithms define objective functions such as the mean-square projected Bellman error (MSPBE) and perform convergent stochastic approximation. These methods can have improved theoretical robustness but may trade off speed or add computational overhead.

0.3 Non-linear approximation methods

Non-linear approximation in RL refers to methods where $\hat{v}(\cdot;\theta)$ or $\hat{q}(\cdot;\theta)$ is **non-linear in parameters** (typical for neural networks), and also to **non-parametric** methods (typical for kernels/GPs) where complexity grows with data.

0.3.1 Neural networks as value approximators

Neural networks can represent complex, global non-linear mappings and can learn representations, especially from high-dimensional observations (pixels). However, early RL practice documented that naïvely combining TD-style targets with neural Q-functions can be unstable because a single update affects many states globally, potentially "destroying" previously learned regions.

Two major families of neural value approximation in the literature include:

Neural fitted value iteration / NFQ. Neural Fitted Q-Iteration (NFQ) is a batch fitted-Q approach that stores experience and repeatedly trains the neural regressor on the accumulated dataset, aiming to reduce destructive interference and improve data efficiency.

Deep Q-Networks (DQN). DQN uses experience replay to decorrelate samples and a target network to stabilize bootstrapped targets; these mechanisms are motivated as responses to divergence/instability in non-linear TD control.

0.3.2 Kernel methods and Gaussian-process value estimation

Kernel methods approximate value functions using similarity-weighted combinations of observed transitions; they can be very effective in low-dimensional continuous domains, but naïvely they scale poorly with data (time/memory typically growing at least quadratically).

GPSARSA / Gaussian process TD. GP-based approaches treat value estimation as Bayesian regression, providing uncertainty estimates and often strong sample efficiency in small-to-moderate domains, but they are limited by scaling and kernel choice.

Kernel-SARSA variants. The practical kernel-RL literature includes sparsification strategies to control memory growth while retaining good function quality and stability.

Tree-based and other non-linear regressors in fitted value iteration

Batch RL / fitted Q iteration (FQI) can, in principle, "plug in" many supervised learners (trees, neural networks, linear models). The tree-based batch RL literature shows that strong non-linear regressors can be used in fitted Bellman backups, making empirical performance depend heavily on regressor bias/variance and data coverage.

0.4 Benchmark environments and their RL challenges

The empirical comparison below focuses on three environments chosen to stress different aspects of approximation:

Environment	**State / observation**	**Action space**	**Stochasticity & key difficulty**	**Primary reason it's informative for approximation**
CartPole-style inverted pendulum balancing (continuous state, noisy actions)	Low-dimensional continuous state (angle, angular velocity)	Discrete (left / right / no force)	Action noise is injected; dynamics are non-linear	Shows sample efficiency and stability trade-offs for linear vs non-linear approximators in a classic control task with noise
MountainCar (continuous state, sparse/delayed rewards)	Continuous state (position, velocity)	Discrete (two actions in NFQ setting)	Sparse/delayed objective: must first move away to gain momentum	Highlights need for generalization and shaping; exposes over/under-generalization risks in global approximators
Breakout in the Arcade Learning Environment (ALE)	High-dimensional pixels (DQN uses $84 \times 84 \times 4$ preprocessed frames)	Discrete	Often treated as deterministic given emulator + initialization, but evaluation protocols introduce stochasticity; long-horizon visual credit assignment	Stresses representation learning and scalability, where linear vs deep differences are stark

A core caveat for cross-environment comparison is that different papers use different training budgets, evaluation truncation, and reward/cost conventions; the analysis below emphasizes **relative trends and mechanisms** more than absolute score comparability.

0.5 Empirical comparison across environments

0.5.1 Cross-environment summary table

Environment	Linear approximation representative	Non-linear representative	What "better" means here	Headline outcome
CartPole balancing (noisy actions)	LSPI / Q-learning with experience replay + fixed RBF basis (linear weights)	NFQ (MLP Q-function)	Longer balancing time; fewer samples to reach reliable success	Non-linear NFQ achieves reliable success with fewer training episodes than LSPI; linear + replay can also reach near-max performance but with more data
MountainCar	Fine discretized Q-table (equivalent to a very large linear one-hot feature map)	NFQ (MLP Q-function)	Fewer episodes/cycles to reach successful/near-optimal policy	NFQ attains near-optimal performance with orders-of-magnitude fewer episodes than a fine Q-table baseline; kernel/RBF approaches can also be strong in this regime
Breakout (ALE)	Linear layer baseline or engineered linear features (e.g., Blob-PROST)	DQN (CNN Q-function)	Higher game score; stability over long training	Deep CNN Q-function dramatically outperforms a simple linear baseline; strong engineered linear features close part of the gap but remain below DQN on this game

0.5.2 CartPole balancing: sample efficiency and robustness under action noise

Environment details (noise + dynamics). In the LSPI benchmark setting, CartPole balancing uses three discrete actions (left/right/no force) and adds uniform noise in $[-10, 10]$ to the chosen action. Episodes end when the pole falls past $\pm\pi/2$, and "success" is balancing up to a long horizon (e.g., 3000 steps).

Linear (fixed features + linear weights). In the LSPI experiments, the value function is approximated using a small set of radial basis features (a linear architecture), and performance is reported as average balancing steps versus number of short random training episodes (each ~ 6 steps). LSPI reaches about **2850 expected balancing steps** with **1000 training episodes**, while Q-learning with experience replay (same linear architecture) reaches **∼3000 expected balancing steps** with **700+ training episodes** and performs well after ∼400 episodes.

Non-linear (MLP via NFQ). NFQ trains a neural Q-function (MLP) using stored experience and repeated supervised fitting. In the same pole-balancing family of settings, NFQ achieves **100% successful trials (50/50)** once trained on **200 random episodes** (each ~ 6 cycles, roughly 1200 transitions), indicating high sample efficiency and robustness across random sample sets.

Kernel-based non-linear approximation (contextual evidence). Kernel-SARSA(λ) reports learning to “indefinitely” balance the pole in a small number of episodes even under strong action noise, outperforming tile coding and RBF coding in their experiments, while storing a small number of support points.

0.5.3 MountainCar: sparse reward, non-linear dynamics, and generalization pressure

Environment details. MountainCar requires the agent to reach a position threshold (e.g., ≥ 0.7) despite insufficient engine power to drive directly uphill; success typically requires first moving away from the goal to build momentum.

Non-linear (NFQ with MLP). NFQ reports that, averaged over 20 experiments, a **first successful policy** is achieved after **70.95 episodes / 2777 cycles**, and a best policy within 500 episodes achieves average cost **28.7**, with near-optimal cycles-to-goal.

Linear baseline (fine Q-table as a large linear model). NFQ reports that a **fine discretized Q-table** can reach **∼29.0 cycles-to-goal**, but required **300,000 episodes** (max length 300 cycles) and used a $250 \times 250 \times 2$ table. This can be understood as a linear value approximator with a very large one-hot feature map—expressive but sample-inefficient and memory-heavy because it generalizes poorly.

Kernel/RBF contextual evidence. In a SARSA(λ) comparison across function approximators, RBF coding and Kernel-SARSA(λ) reach an optimal policy in roughly the same number of episodes, while tile coding needs many more; Kernel-SARSA(λ) is also reported to use < 150 **stored samples**, substantially smaller than tile coding's feature dimension in their configuration.

0.5.4 Breakout in ALE: high-dimensional perception and the representation-learning regime

Environment details (ALE). The Arcade Learning Environment (ALE) provides an interface to many classic Atari 2600 games as RL environments and has become a widely used evaluation platform for general agents.

Non-linear (DQN). DQN uses a deep convolutional network to approximate the action-value function directly from pixels and motivates replay + target networks as solutions to instabilities of non-linear function approximation with Q-learning.

A direct linear baseline inside the DQN study. A commonly cited comparison replaces the convolutional network with a single linear layer and shows dramatically worse performance for the linear parameterization on Breakout under the same overall training machinery.

Strong engineered linear representations (shallow RL). Engineered linear feature sets inspired by DQN's inductive biases can be competitive with DQN across many ALE games, while also documenting cases (including Breakout) where DQN remains substantially stronger; these works often emphasize that shallow linear methods can be more computationally practical than deep RL.

0.6 When each approach tends to win

0.6.1 When linear approximation is usually the better engineering choice

Linear methods tend to dominate when the environment has **low-dimensional continuous state** and there exists a **well-aligned feature representation** (tilings, RBFs, Fourier features, handcrafted domain invariances). In this regime, linear architectures often provide:

- **Stable, interpretable learning dynamics** (convex objectives in supervised subproblems; transparent failure modes).
- **High sample efficiency** when paired with least-squares methods (LSTD/LSTDQ/LSPI), albeit with higher $O(n^2)$ costs in feature count.
- **Strong robustness** when features are local and updates have limited interference (tile coding/coarse coding, many RBF setups).

CartPole balancing (with noisy actions) is a strong example: linear methods with a small RBF basis can reach near-max balancing time, and experience replay can dramatically strengthen linear Q-learning behavior.

0.6.2 When non-linear approximation is usually necessary

Non-linear methods typically win when:

- The observation is **high-dimensional** (pixels, raw sensory streams), making handcrafted features brittle and making representation learning decisive.
- The optimal value function has **sharp discontinuities or complex interactions** that cannot be represented as a modest linear combination of features without an explosion in feature count.
- The task requires capturing **spatiotemporal invariances** or short-horizon memory in the value representation (e.g., using stacked frames in DQN).

0.6.3 Why stability is often harder for non-linear approximators

For TD control, the training target depends on the network being trained and on the policy induced by it. Replay buffers and target networks (or closely related stabilizers) are standard in deep Q-learning precisely because of instability risks.

Kernel methods occupy a middle ground: they can approximate rich non-linear functions and can work well in continuous domains, but must confront scaling and memory growth; practical algorithms therefore rely on sparsification/budgeting strategies to remain usable.

0.7 Practical recommendations and implementation checklist

0.7.1 A decision guide based on environment properties

If the environment is **low-dimensional and continuous** ($\leq\sim$ 10–20 dimensions) and you can construct meaningful features (tiles/RBFs/Fourier), start with **linear approximation** (often with eligibility traces and/or least-squares methods) because it is typically faster to debug and can be extremely sample efficient.

If the environment is **high-dimensional perceptual** (pixels, images) or requires learning invariances, you should expect that **non-linear (deep) approximation** is required; evidence from ALE-style domains shows that naïve linear parameterizations can be dramatically insufficient.

If data are expensive but the domain is low-dimensional continuous, consider **kernel/GP-style** value learning (or sparse kernel variants) as a principled non-linear alternative—especially if uncertainty estimates matter—while budgeting for memory/time scaling.

0.7.2 Implementation details that matter in practice

For **linear** approximators:

- Prefer **local features** (tiles/RBF) and normalize state variables; feature scaling strongly affects step-size stability.
- If you need off-policy learning, consider stable **gradient TD** methods rather than naïve off-policy TD/Q-learning, especially in large-feature regimes.
- If sample efficiency is paramount and n is modest, least-squares methods can help, but be explicit about $O(n^2)$ memory/time costs.

For **non-linear neural** approximators:

- Expect instability without safeguards; replay buffers and target networks (or closely related stabilizers) are standard in deep Q-learning precisely because of instability risks.
- Batch fitted-Q styles (NFQ/FQI) can improve data efficiency and reduce interference by repeatedly fitting targets on stored experience, at the cost of heavier supervised training loops.
- Monitor distribution shift and over-generalization; neural updates can change values globally, so small errors can propagate and alter behavior dramatically.

For **kernel methods**:

- Kernel choice and bandwidth are central; without sparsification, memory can scale poorly, so practical approaches often include explicit dictionary management.

0.7.3 Core references (primary sources and textbooks)

- Sutton & Barto, *Reinforcement Learning: An Introduction* (2018) for foundational definitions, linear approximation notation, and practical guidance on approximation and TD methods.
- Tsitsiklis & Van Roy for convergence characterization of TD with linear approximation and discussion of sampling-related divergence and non-linear hazards.
- Bradtke & Barto for least-squares TD (LSTD).
- LSPI and empirical comparisons against Q-learning variants for inverted pendulum and other control tasks.
- NFQ for data-efficient neural fitted Q iteration on classic benchmarks.

- DQN for deep value approximation from pixels and canonical stability mechanisms (replay, target network).
- ALE platform description and evaluation methodology, plus later protocol clarifications (stochasticity, reproducibility).