
Value-based Reinforcement Learning via Mode Connectivity

L^AT_EX im Corporate Design der TU Darmstadt

Master thesis by Henrik Metternich

Date of submission: March 2, 2026

1. Review: Ahmed Hendawy
 2. Review: Carlo D'Eramo
 3. Review: Jan Peters
- Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Henrik Metternich, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 2. März 2026



Henrik Metternich

Contents

1. Introduction	8
2. Background	10
2.1. Reinforcement Learning	10
2.1.1. Problem Formulation	10
2.1.2. Value-based Reinforcement Learning	11
2.2. Abstraction of the Learning Process	14
2.2.1. Standard Optimization	14
2.2.2. Optimization in Reinforcement Learning	15
2.2.3. The Training Operator	15
2.3. Neural Network Subspaces	17
2.3.1. Subspaces	17
2.3.2. Connectors	18
2.3.3. Mode Connectivity	19
2.3.4. Training the Subspace	21
3. Related Work	23
3.1. Loss Landscapes and Optimization Geometry	23
3.2. Important Observations in Supervised Learning	23
3.3. Mode Connectivity and Subspaces in Reinforcement Learning	26
4. Methodology	28
4.1. Algorithms	28
4.2. Implementation Details	30
4.2.1. Algorithms	30
4.2.2. Environments	31
4.2.3. Efficient Subspace Implementation	31

5. Landscape Analysis	33
5.1. Ensembling	33
5.2. Flat Maxima	35
5.3. Mode Connectivity	38
6. Subspace Learning	41
6.1. Training the Subspace	41
6.2. Experiments	43
6.2.1. Different Subspaces	44
6.2.2. Without Multi-Step Return	45
6.2.3. Feature Similarity	46
6.2.4. Utilizing the Subspace during Training	47
6.2.5. Ablations on the Target Computation	49
6.3. Analyzing the Landscape	50
6.4. Discussion	52
7. Conclusion	54
A. Appendix	59



Acronyms

DQN Deep Q-Network

PPO Proximal Policy Optimization

A2C Advantage Actor-Critic

MDP Markov Decision Process

RL Reinforcement Learning

SL Supervised Learning

SGD Stochastic Gradient Descent

PQN Parallel Q-Network

MC Monte Carlo

NN Neural Network

NNS Neural Network Subspace

Abstract

Value-based reinforcement learning often suffers from high variance and overestimation bias, issues that can be mitigated by computationally expensive ensemble methods. Recent insights from supervised learning suggest that independent models are often connected by low-loss pathways, prompting the question of whether similar topological properties exist under the non-stationary optimization dynamics of reinforcement learning algorithms. This thesis first conducts a comprehensive landscape analysis of deep Q-networks, empirically confirming the existence of flat maxima and non-linear mode connectivity in value-based agents. Building upon these geometric insights, we explore subspace learning as a highly efficient alternative to traditional ensembles, aiming to parameterize an infinite continuum of policies from a single training run. To achieve this, we introduce an algorithmic approach designed to simultaneously optimize subspace anchors while actively enforcing structural diversity. However, our experiments across multiple Atari environments reveal that the resulting continuous subspaces are predominantly flat, allowing models to achieve parameter diversity without true functional divergence. Ultimately, while this work demonstrates the mechanical feasibility of subspace learning in reinforcement learning, it highlights that unlocking true ensemble benefits will require advanced regularization techniques that explicitly enforce behavioral heterogeneity.

Zusammenfassung

Wertbasiertes Reinforcement Learning leidet häufig unter hoher Varianz und Overestimation Bias, was typischerweise durch rechenintensive Ensemble-Methoden gemindert wird. Jüngste Erkenntnisse aus dem überwachten Lernen deuten darauf hin, dass unabhängige Modelle oft durch Low-Loss Pathways verbunden sind. Dies wirft die Frage auf, ob ähnliche topologische Eigenschaften auch unter der instationären Optimierungsdynamik des Reinforcement Learnings existieren. Diese Arbeit führt zunächst eine umfassende Landschaftsanalyse von Deep Q-Networks durch und bestätigt empirisch die Existenz flacher Maxima sowie nichtlinearer Mode Connectivity in wertbasierten Agenten. Aufbauend auf diesen geometrischen Erkenntnissen untersuchen wir Subspace Learning als hocheffiziente Alternative zu herkömmlichen Ensembles mit dem Ziel, ein unendliches Kontinuum von Policies aus einem einzigen Trainingsdurchlauf zu parametrisieren. Um dies zu erreichen, stellen wir einen algorithmischen Ansatz vor, der darauf ausgelegt ist, gleichzeitig Subspace-Anker zu optimieren und dabei aktiv strukturelle Diversität zu erzwingen. Unsere Experimente in verschiedenen Atari-Umgebungen zeigen jedoch, dass die resultierenden kontinuierlichen Subspaces durchweg flach sind. Dies ermöglicht es den Modellen, eine Parameterdiversität ohne echte funktionale Divergenz zu erreichen. Letztendlich demonstriert diese Arbeit zwar die mechanische Machbarkeit von Subspace Learning im Reinforcement Learning, verdeutlicht jedoch auch, dass die Erschließung echter Ensemble-Vorteile fortschrittliche Regularisierungstechniken erfordern wird, die explizit Verhaltensheterogenität erzwingen.

1. Introduction

Reinforcement Learning (RL) has emerged as a highly successful paradigm for solving complex sequential decision-making problems. By allowing an agent to learn optimal behaviors through trial-and-error interactions with its environment, RL has driven significant breakthroughs in diverse domains, ranging from continuous control in robotics to mastering superhuman strategies in complex games.

Within the broader landscape of RL algorithms, methods are fundamentally categorized into policy-based, value-based, and actor-critic approaches. Value-based reinforcement learning, in particular, focuses on estimating the expected cumulative return of states or state-action pairs to derive an optimal policy. At the core of many value-based methods is Q-learning. The integration of deep neural networks into this framework led to the development of the Deep Q-Network (DQN), which successfully stabilized the use of high-capacity non-linear function approximators. Since then, numerous extensions have been introduced to address inherent algorithmic flaws, yielding variants such as Double DQN (DDQN), Dueling DQN, and Rainbow DQN, which collectively improve learning stability and data efficiency.

Despite these significant advancements, relying on a single neural network for value function approximation introduces persistent challenges. A single approximation is notoriously prone to high variance and overestimation bias, a scenario in which the network consistently overestimates the true action values due to the greedy maximization step in the Bellman equation. To mitigate these issues, ensemble-based methods have emerged as a robust solution. By maintaining multiple independent Q-value estimators and aggregating their predictions, ensembles effectively smooth out individual approximation errors, reduce variance, and provide a more reliable target for network updates.

However, the primary limitation of traditional ensemble methods is their prohibitive computational expense. Achieving an ensemble of n distinct approximations strictly requires initializing, storing, and independently training n separate neural networks.

This scales the computational burden linearly with the ensemble size, making large-scale ensembling impractical for resource-intensive reinforcement learning tasks.

Recent investigations into the loss landscape of deep neural networks have unveiled intriguing geometric properties that offer a potential workaround to this computational bottleneck. Contrary to the traditional view of optimization landscapes as fields of strictly isolated local minima, empirical evidence demonstrates the phenomenon of mode connectivity. It has been shown that independent modes, or distinct high-performing parameter configurations, in the weight space are often connected by continuous, low-loss pathways. These pathways can be systematically discovered through targeted optimization directly in the weight space.

By leveraging these geometric properties, we can construct a low-dimensional parameter subspace defined by a small set of k structural anchors. Because the pathways between these anchors represent viable parameter configurations, this subspace effectively parameterizes an infinite continuum of functional models. Crucially, this structure allows us to sample and evaluate varied points along the continuous subspace, effectively generating an infinite ensemble of models from a single unified training run.

While subspace learning and mode connectivity have seen substantial exploration and success within the realm of supervised learning, where datasets are static and objectives are stationary, their application to reinforcement learning remains largely uncharted. The inherent non-stationarity of the RL training loop, where the optimization target shifts as the agent’s policy evolves, presents unique challenges for maintaining stable geometric structures in the parameter space.

In this work, we propose to bridge this gap by introducing and evaluating subspace methods specifically tailored for value-based reinforcement learning. We systematically investigate the loss landscape of Q-networks to understand how non-stationary optimization affects the topological properties of the weight space. Furthermore, we explore practical, computationally efficient methods to leverage these learned continuous subspaces to construct robust ensembles. By doing so, we seek to harness the theoretical benefits of ensemble methods, namely improved stability, variance reduction, and better performance, without the prohibitive computational costs typically associated with training multiple independent agents.

2. Background

2.1. Reinforcement Learning

Reinforcement Learning (RL) [23] is a paradigm of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives rewards or penalties for its actions, and its goal is to learn a policy that maximizes the cumulative reward over time. Unlike Supervised Learning (SL), RL agents learn from trial and error without explicit supervision, making it suitable for complex sequential decision-making problems.

2.1.1. Problem Formulation

Reinforcement Learning problems are commonly formalized as Markov Decision Processes (MDPs) [21], which provide a mathematical framework for modeling sequential decision-making in stochastic environments. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, $\mathcal{P}(s'|s, a)$ is the state transition probability function, $\mathcal{R}(s, a, s')$ is the reward function, and γ is the discount factor that determines the decay of the influence of future rewards. The agent interacts with the environment in discrete time steps. At each time step t , the agent observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to its policy $\pi(a|s)$, receives a reward $r_{t+1} = \mathcal{R}(s_t, a_t, s_{t+1})$, and transitions to the next state s_{t+1} according to the transition probabilities $\mathcal{P}(s_{t+1}|s_t, a_t)$. Figure 2.1 illustrates the interaction between the agent and the environment in an RL setting.

The objective of an RL agent is to find an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected cumulative discounted reward, often denoted as $J(\pi)$. The policy dictates the

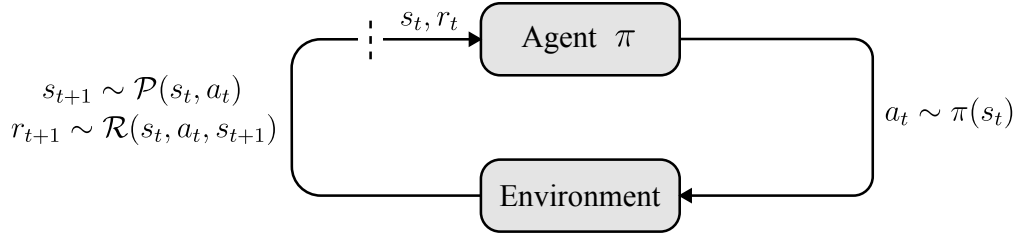


Figure 2.1.: The Reinforcement Learning loop, where an agent interacts with an environment by taking actions based on its policy, receiving observations and rewards in return.

action to take in each state. The objective function is typically defined as follows, where τ denotes a trajectory generated by policy π .

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right] \quad (2.1)$$

The behavior of an RL agent's policy π also induces a distribution over states, known as the state distribution $d^\pi(s)$. This distribution represents the probability of visiting each state $s \in \mathcal{S}$ when following policy π . For a fixed policy, the state distribution can converge to a stationary distribution in ergodic MDPs. The non-stationarity discussed earlier arises because the agent's policy changes during training, which in turn alters this state distribution, making the optimization target dynamic.

2.1.2. Value-based Reinforcement Learning

Value-based reinforcement learning constitutes a class of algorithms focused on estimating the value of states or state-action pairs to determine an optimal policy. The core idea is to learn how good it is for an agent to be in a given state, or how advantageous it is to perform a specific action within that state. This evaluation is formalized through two essential value functions: the state-value function $V^\pi(s)$, which represents the expected cumulative discounted reward starting from state s and following policy π , and the action-value function $Q^\pi(s, a)$, which estimates the expected return starting from state s , taking action a , and thereafter following policy π .

The foundation of value-based methods lies in the Bellman equation, which recursively decomposes the value function into the immediate reward and the discounted value of

the successor state. The objective is to satisfy the Bellman optimality equation, defined for the optimal action-value function $Q^*(s, a)$ as follows.

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \mid s, a \right]$$

Once Q^* is known, the optimal policy π^* can be derived by selecting the action that maximizes the Q-value in each state greedily: $\pi^*(s) = \arg \max_a Q^*(s, a)$.

Q-Learning and Deep Q-Networks (DQN)

Q-learning [27] is a fundamental value-based algorithm that seeks to learn the optimal action-value function $Q^*(s, a)$ directly. While traditional Q-learning maintains a lookup table for state-action pairs, this approach becomes computationally intractable in high-dimensional or continuous state spaces. To address this, Deep Q-Network (DQN) [19] utilizes deep neural networks as non-linear function approximators, denoted as $Q_\theta(s, a)$, allowing the agent to generalize experiences across similar states.

However, combining RL with non-linear function approximation is notoriously unstable due to the strong temporal correlation between consecutive samples and the moving target problem, where the optimization targets depend on the same weights being updated. To mitigate these issues, DQN introduces two critical mechanisms, the replay buffer and the target network.

The replay buffer stores the agent's experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ in a memory \mathcal{D} , from which mini-batches are sampled uniformly at random during training. This breaks temporal correlations and mimics the i.i.d. setting of supervised learning. Simultaneously, the target network maintains a separate set of weights $\bar{\theta}$ updated only periodically to compute a stable target $y_i = r + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\theta}}(s', a')$. The online network is then optimized by minimizing the mean squared error (MSE) loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[(y_i - Q_\theta(s, a))^2 \right] \tag{2.2}$$

Numerous DQN variants have since been developed, such as Double DQN [25] to reduce overestimation bias and Dueling DQN [26] to better generalize across action-independent state values.

Parallel Q-Networks (PQN)

Traditional value-based methods like the Deep Q-Network often suffer from computational bottlenecks because they strictly alternate between environment data collection and network optimization. This sequential process severely limits scalability and training speed. A natural solution to this inefficiency is the parallelization of data collection across multiple environments. While parallel actor-critic methods, such as Proximal Policy Optimization (PPO) [22], have successfully leveraged this distributed approach for years, bringing these benefits to purely value-based methods has historically been challenging.

Algorithm 1 Parallel Q-Network (PQN)

```
1: Initialize Q-network  $\theta$ , and initial states  $s_0^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
2:  $t_{\text{total}} \leftarrow 0$ 
3: while  $t_{\text{total}} < n_{\text{steps}}$  do
4:   for  $t = 0 \dots n_{\text{rollout}} - 1$  do
5:     Sample actions  $a_t^i \sim \epsilon\text{-greedy}(Q_\theta(s_t^i, \cdot))$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
6:     Execute  $a_t^i$ , observe  $r_t^i, s_{t+1}^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
7:   end for
8:   Compute  $R_t^{\lambda, i} = r_t^i + \gamma \left[ \lambda R_{t+1}^{\lambda, i} + (1 - \lambda) \max_{a'} Q_\theta(s_{t+1}^i, a') \right]$  for all  $t$  and  $i$ 
9:   for epoch =  $1 \dots n_{\text{epochs}}$  do
10:    Shuffle data over time  $t$  and environment  $i$  axes to form  $M$  minibatches
11:    for each minibatch  $m$  do
12:       $\theta \leftarrow \theta - \eta \nabla_\theta \left( \frac{1}{M} \sum_{(s_t^i, a_t^i) \in m} \left( Q_\theta(s_t^i, a_t^i) - R_t^{\lambda, i} \right)^2 \right)$ 
13:    end for
14:  end for
15:   $s_0^i \leftarrow s_{n_{\text{rollout}}}^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
16:   $t_{\text{total}} \leftarrow t_{\text{total}} + (n_{\text{rollout}} \times n_{\text{envs}})$ 
17: end while
```

Recently, Gallici et al. [9] addressed this gap by proposing Parallel Q-Networks (PQN). PQN introduces a drastically simplified architecture that entirely removes both the replay buffer and the target network. These two components were previously thought essential for stabilizing off-policy Q-learning. To maintain stability without these mechanisms, PQN relies predominantly on the application of layer normalization within the neural network. This architectural choice serves as the primary stabilizer, while the approach is

additionally supported by the use of λ -returns for value estimation. Algorithm 1 details the complete training procedure.

2.2. Abstraction of the Learning Process

To analyze the geometric properties of the resulting neural networks, we must first formalize the process by which they are generated. While deep RL shares the computational backbone of gradient-based updates with SL, the underlying dynamics differ fundamentally. In order to unify these processes under a common framework, we will first describe the standard optimization process in SL and then contrast it with the non-stationary optimization dynamics of RL. Finally, we will introduce the concept of a training operator that abstracts away the specific dynamics and allows us to analyze the resulting neural networks as outputs of a deterministic mapping from randomness to parameters.

2.2.1. Standard Optimization

Optimization can be generally defined as the problem of finding the minimum of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. We seek a parameter vector $\theta \in \mathbb{R}^n$ such that the scalar value $f(\theta)$ is minimized.

$$\min_{\theta} f(\theta) \tag{2.3}$$

In the context of supervised learning, we define a neural network as a function $g_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$ parameterized by θ . The objective function f becomes the expected loss \mathcal{L} over a fixed dataset \mathcal{D} .

$$f(\theta) = \mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}}[\ell(g_{\theta}(\mathbf{x}), y)] \tag{2.4}$$

Although the landscape of f is often non-convex and the optimization process is stochastic, due to mini-batch sampling and random initialization, the objective function itself is stationary. The geometry of the loss landscape is fixed prior to training, and the optimizer simply navigates this static terrain. Though \mathcal{L} is the primary objective function, the true goal is to find parameters that generalize well to unseen data, which is often measured by accuracy \mathcal{A} that is estimated on a separate test set $\mathcal{D}_{\text{Test}}$.

2.2.2. Optimization in Reinforcement Learning

A fundamental challenge in Reinforcement Learning (RL) is the inherent loss of stationarity. Unlike supervised learning paradigms that rely on static, independently and identically distributed datasets, the data distribution in RL is deeply coupled with the agent's behavior. Specifically, as the policy is updated during training, the underlying state-action visitation distribution continuously shifts. Consequently, the optimization objective becomes a moving target.

To illustrate this, consider the Deep Q-Network (DQN) algorithm. At any time step i , we minimize the DQN loss function, which is dependent on the current network parameters θ^i .

$$f^i(\theta^i) = \mathcal{L}_{\text{DQN}}(\theta^i) \approx \mathbb{E}_{s \sim d^{\pi_{\theta^i}}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \bar{\theta}^i) - Q(s, a; \theta^i) \right)^2 \right]$$

A gradient step on the current parameters θ^i yields an updated policy, which fundamentally shifts the discounted state visitation distribution $d^{\pi_{\theta}}$. Because the subsequent objective function f^{i+1} is evaluated over this new distribution, the learning process can suffer from severe instability or divergence. This non-stationarity breaks standard gradient descent convergence guarantees, which is precisely why stabilizing mechanisms, such as the target network parameterized by $\bar{\theta}^i$ seen in the equation above, are necessary.

2.2.3. The Training Operator

Despite these dynamical differences, we can unify both processes under a single functional abstraction. Whether through static minimization or dynamic fixed-point iteration, the training process can be viewed strictly as a deterministic mapping from a source of randomness to a final parameter configuration.

Let ξ represent the complete state of stochasticity in the learning process. This variable $\xi \in \Xi$ encapsulates all random factors involved in a single training run. This includes, but is not limited to the random initialization of the network weights, the sequence of mini-batch indices sampled from the dataset or replay buffer, and the stochastic transitions of the environment and exploration noise in RL. Then we can define a training operator \mathcal{T} that takes this randomness as input and produces the final trained parameters θ as output in the following way.

Definition 2.1 (Training Operator) *Let Ξ be the space of all possible realizations of randomness in the training process. The training operator \mathcal{T} is a function that maps a specific realization of randomness $\xi \in \Xi$ to the final trained parameters $\theta \in \mathbb{R}^n$ after completing the training procedure.*

Under this formalism, a trained neural network is a realization of the random variable θ . This abstraction allows us to state properties of the solution space without explicitly modeling the complex, non-stationary dynamics of the RL training loop. DQN can be expressed as a specific instance of this training operator \mathcal{T}_{DQN} , where ξ includes the random initialization of the Q-network, the sequence of mini-batches sampled from the replay buffer, and the stochastic transitions of the environment and exploration noise.

So far the training operator \mathcal{T} has been defined as a mapping from the space of randomness to the final parameter configuration after training. However, we are also interested in properties of the training process at intermediate stages. Therefore we extend the definition of the training operator to capture the parameter state at any point during the training process, not just at the end.

Definition 2.2 (Stepwise Training Operator) *Let $\Theta = \mathbb{R}^n$ be the parameter space and Ξ be the space of stochastic realizations (excluding initialization). We define the extended training operator $\mathcal{T}^{(T)}$ as a function that maps an initial parameter configuration θ_0 and a randomness realization ξ to the parameter state after T optimization steps:*

$$\mathcal{T}^{(T)} : \Xi \times \Theta \rightarrow \Theta, \quad \theta^T = \mathcal{T}^{(T)}(\xi, \theta^0) \quad (2.5)$$

By leveraging this stepwise definition, we can naturally decompose the entire training process into sequential phases. Specifically, we can split a training procedure of T total steps into two distinct parts. The first phase consists of T_1 initial optimization steps, followed by a second phase of T_2 steps. Mathematically, this composition is expressed as

$$\mathcal{T}^{(T)}(\xi) = \mathcal{T}^{(T_2)}(\xi, \mathcal{T}^{(T_1)}(\xi)) \quad \text{where} \quad T = T_1 + T_2 \quad (2.6)$$

This sequential decomposition is particularly useful for analyzing the trajectory of the parameters during optimization. It allows us to treat the intermediate parameter state produced after T_1 steps as the initialization for the remaining T_2 steps. Consequently, we can study the behavior and stability of the learning algorithm from any arbitrary midpoint within the overall training loop without needing to redefine the underlying stochastic processes.

2.3. Neural Network Subspaces

The parameters of a Neural Network (NN) can typically be represented as a collection of tensors of different shapes. By flattening and concatenating these tensors, we can represent all parameters of a neural network as a single vector $\theta \in \mathbb{R}^n$ in a high-dimensional space, where n is the total number of parameters in the network. We define a Neural Network Subspace (NNS) as a subset $\Theta \subset \mathbb{R}^n$ of this high-dimensional parameter space. However, representing subspaces as arbitrary subsets of \mathbb{R}^n is neither practical nor efficient. We will instead use low-dimensional parameterizations to define these subspaces more effectively. We will first introduce the general concept of subspaces before discussing a specific type of subspace called connectors.

2.3.1. Subspaces

Instead of defining the NNS Θ in the high-dimensional space \mathbb{R}^n directly, we can define it using a lower-dimensional parameterization. We can achieve this by selecting a set of k parameter vectors $\{\theta_i\}_{i=1}^k$ in \mathbb{R}^n , which we will refer to as anchors. These anchors can be thought of as representative points in the parameter space that span the subspace of interest. By combining these anchors using a set of weight vectors $w \in \mathbb{R}^k$, we can define the neural network parameters $\theta(w)$ as a weighted sum of the anchors, where w_i is the i -th component of the weight vector w .

$$\theta(w) = \sum_{i=1}^k w_i \theta_i \quad (2.7)$$

Using this reparameterization we can propose an alternative definition of a NNS as the set of all parameter vectors that can be expressed as a weighted combination of the anchors. This approach allows us to reparameterize a subspace of neural network parameters using a tuple $\langle \{\theta_i\}_{i=1}^k, \mathcal{W} \rangle$, where $\mathcal{W} \subset \mathbb{R}^k$ is the set of valid weight vectors that define the subspace.

Definition 2.3 (Neural Network Subspace) *A neural network subspace Θ is defined as a subset of the parameter space \mathbb{R}^n that can be parameterized by a set of k anchor points $\{\theta_i\}_{i=1}^k$ and a set of valid weight vectors $\mathcal{W} \subset \mathbb{R}^k$ such that:*

$$\Theta = \{\theta(w) | w \in \mathcal{W}\} \quad (2.8)$$

where $\theta(w) = \sum_{i=1}^k w_i \theta_i$.

Using this parameterization, we can efficiently explore and optimize within the subspace of neural network parameters defined by the anchors and the weight vectors. Figure 2.2 illustrates different types of neural network subspaces that can be defined using this definition.

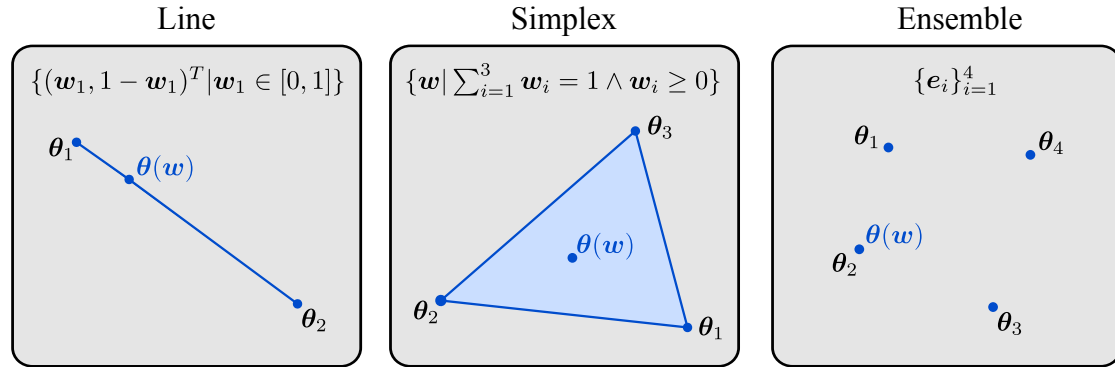


Figure 2.2.: Different types of neural network subspaces including a line, simplex, and ensemble. **Left:** Simplex subspace spanned by 3 solutions. **Middle:** Line subspace connecting 2 solutions. **Right:** Ensemble of 4 represented as a subspace.

2.3.2. Connectors

The definition of a neural network subspace is still quite general and can encompass a wide variety of structures in the parameter space. However, in the context of analyzing the loss landscape and understanding the relationships between different solutions, we are particularly interested in a specific type of subspace that we will refer to as a connector. A connector is a special type of subspace that connects two distinct points in the parameter space, which we will denote as θ_1 and θ_k . We can think of a connector as a continuous path in the parameter space that links these two points, allowing us to analyze the properties of the loss landscape along this path and understand how the two solutions are related to each other. This continuous path can be parameterized by a single scalar parameter $\alpha \in [0, 1]$, which allows us to define the connector as a function $\theta(\alpha)$ that maps to points in the parameter space, where $\theta(0) = \theta_1$ and $\theta(1) = \theta_k$.

$$\boldsymbol{\theta}(\alpha) = \boldsymbol{\theta}(c(\alpha)) = \sum_{i=1}^k c_i(\alpha) \boldsymbol{\theta}_i \quad (2.9)$$

We will model connectors as a specific type of neural network subspaces. More specifically we will model the set \mathcal{W} by a continuous function $c(\alpha) : [0, 1] \rightarrow \mathbb{R}^k$ that maps the scalar parameter α to a weight vector in \mathbb{R}^k . Using this parameterization, we can define a connector as a specific type of neural network subspace that connects two distinct points in the parameter space through a continuous path defined by the function $c(\alpha)$.

Definition 2.4 (Connector) A connector is a specific type of neural network subspace that connects two distinct points $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_k$ in the parameter space. It is defined as a continuous function $\boldsymbol{\theta}(\alpha) : [0, 1] \rightarrow \mathbb{R}^n$ such that $\boldsymbol{\theta}(0) = \boldsymbol{\theta}_1$ and $\boldsymbol{\theta}(1) = \boldsymbol{\theta}_k$.

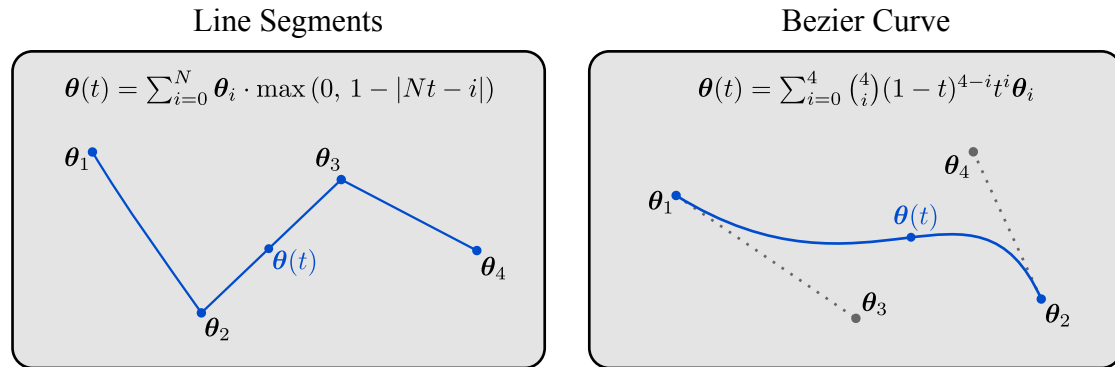


Figure 2.3.: Different types of neural network connectors including line segments, and a bezier curve. **Left:** Polyline subspace spanned by 4 solutions. **Right:** Bezier Curve subspace connecting 2 solutions.

2.3.3. Mode Connectivity

Mode-connectivity is a property of the loss landscape of neural networks that describes the existence of low-loss paths connecting different local minima in the parameter space. In order to properly define mode connectivity, we will first introduce the concept of barrier

height and super level sets. The barrier height between two points in the parameter space is a measure of the maximum loss encountered along any continuous path connecting those two points. It quantifies the difficulty of transitioning from one solution to another in terms of the loss landscape.

Definition 2.5 (Barrier Height) *Given an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and a connector $(\{\theta_i\}_{i=1}^k, c)$ connecting two endpoints θ_1 and θ_k , the barrier height $B_c(\theta_1, \theta_k)$ is defined as the difference between the maximum value of the objective function along the connector and the maximum of the objective function at the endpoints:*

$$B_c(\theta_1, \theta_k) = \max_{\alpha \in [0,1]} f(\theta(\alpha)) - \max(f(\theta_1), f(\theta_k)) \quad (2.10)$$

Conversely, when analyzing minimization problems, it is useful to define the barrier depth. The barrier depth measures the maximum drop in the objective function along the path relative to the minimum value at the endpoints.

Definition 2.6 (Barrier Depth) *Given the same objective function and connector, the barrier depth $D_c(\theta_1, \theta_k)$ is defined as:*

$$D_c(\theta_1, \theta_k) = \min(f(\theta_1), f(\theta_k)) - \min_{\alpha \in [0,1]} f(\theta(\alpha)) \quad (2.11)$$

To provide a relative measure of the barrier that accounts for the scale of the objective function, we use the normalized barrier height. This is calculated by dividing the barrier height by the maximum objective value at the endpoints (assuming strictly positive objective values), allowing for fair comparisons across different regions of the landscape:

$$\bar{B}_c(\theta_1, \theta_k) = \frac{B_c(\theta_1, \theta_k)}{\max(f(\theta_1), f(\theta_k))} \quad (2.12)$$

With these barrier metrics established, we can isolate specific regions of the parameter space using level sets.

Definition 2.7 (Super-Level Set) *Given an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and a threshold $\eta \in \mathbb{R}$, the super-level set $L_{\geq \eta}$ is defined as the set of all points in the parameter space where the objective function value is greater than or equal to η :*

$$L_{\geq \eta} = \{\theta \in \mathbb{R}^n \mid f(\theta) \geq \eta\} \quad (2.13)$$

By the same logic, a sub-level set $L_{\leq\eta}$ contains all points where the objective function evaluates to less than or equal to the threshold η :

$$L_{\leq\eta} = \{\boldsymbol{\theta} \in \mathbb{R}^n \mid f(\boldsymbol{\theta}) \leq \eta\} \quad (2.14)$$

Depending on whether the goal is maximization (e.g., maximizing a reward) or minimization (e.g., minimizing a loss), mode connectivity can be framed using either super-level or sub-level sets. We can now formally define mode connectivity by evaluating whether the path mapped by a connector remains entirely within one of these specified level sets.

Definition 2.8 (Mode Connectivity) *Two solutions $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_k$ are said to be mode-connected with respect to a super-level set if there exists a connector $(\{\boldsymbol{\theta}_i\}_{i=1}^k, \mathcal{c})$ such that all points along the path defined by the connector remain within the same super-level set $L_{\geq\eta}$ for some threshold η .*

Analogously, for a minimization objective, two solutions $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_k$ are mode-connected if there exists a connector such that all points along the path remain within a sub-level set $L_{\leq\eta}$. This ensures that the transition between the two modes never incurs a penalty exceeding the threshold η .

2.3.4. Training the Subspace

To train the subspace, our primary goal is to optimize the parameters $\{\boldsymbol{\theta}_i\}_{i=1}^k$ with respect to a specific objective function f , which is defined over the parameter space \mathbb{R}^n . We achieve this optimization by computing the expected loss.

Conceptually, the training objective for the subspace is defined as the expectation of the loss calculated over the points within the subspace defined by \mathcal{W} . Therefore, any standard objective function f can be transformed into a subspace-specific training objective, denoted as $f_{\mathcal{W}}$, by taking the expectation of f over the points in that subspace.

$$f_{\mathcal{W}}(\{\boldsymbol{\theta}_i\}_{i=1}^k) = \mathbb{E}_{\mathbf{w} \sim \mathcal{W}} \left[f \left(\sum_{i=1}^k \mathbf{w}_i \boldsymbol{\theta}_i \right) \right]$$

In the specific context of connectors, we refine this approach by defining the objective as the expectation of the loss over the trajectory of the connector \mathcal{c} . Here, the path is

parameterized by a scalar α , representing the interpolation along the curve between the endpoints in the parameter space. The objective is then calculated by sampling α from a distribution.

$$f_c(\{\theta_i\}_{i=1}^k) = \mathbb{E}_{\alpha \sim U[0,1]} \left[f \left(\sum_{i=1}^k c_i(\alpha) \theta_i \right) \right]$$

While we assume a uniform distribution $U[0,1]$ here for simplicity, this formulation holds for any arbitrary distribution $q(\alpha)$, allowing for non-uniform importance sampling along the connector. In addition to training the anchors, it is also possible to optimize the mixing coefficient α while keeping the anchors fixed. This restricts the optimization to the one-dimensional subspace defined by the connector, allowing us to pinpoint the best solution between the two base models.

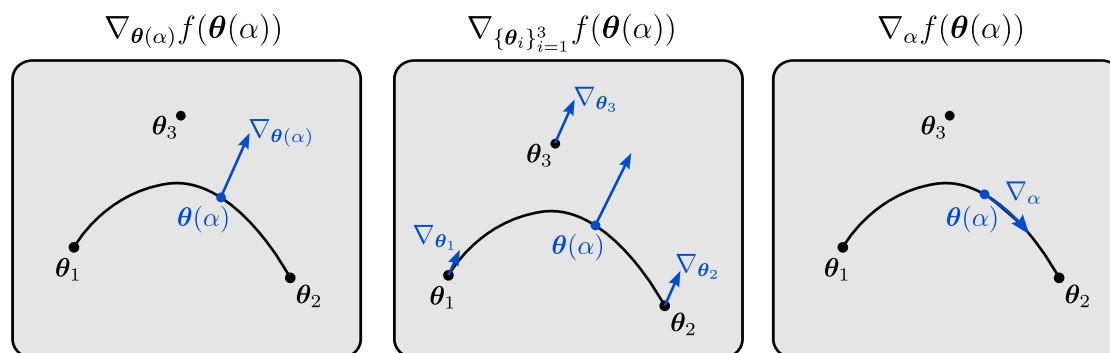


Figure 2.4.: Visualizing distinct gradients in a parameterized connecting path. **Left:** The parameter space gradient evaluated at a specific point on the curve. **Middle:** Gradients with respect to the underlying control points. **Right:** The one-dimensional gradient with respect to the scalar path parameter, representing the curve’s tangent.

Figure 2.4 provides a visual intuition for how this optimization process unfolds. Specifically, it contrasts the standard parameter space gradient with the specialized updates required for our approach. Additionally, it also illustrates the one-dimensional gradient with respect to the scalar path parameter α , which represents the tangent direction along the connector.

3. Related Work

3.1. Loss Landscapes and Optimization Geometry

Research into the shape of neural network loss landscapes has provided key insights into how models learn. Goodfellow et al. [13] suggested that adversarial examples are caused mainly by the linear behavior of deep networks in high-dimensional spaces, rather than extreme complexity. However, Frankle [6] updated this view, showing that for modern deep architectures, the path from initialization to the solution is rarely simple or convex. To help visualize these complex surfaces, Li et al. [17] introduced new plotting techniques, demonstrating that ResNets have much smoother landscapes compared to older architectures like VGG. Additionally, Fort and Jastrzebski [4] described the global loss landscape as a set of connected "wedges," where different hyperparameters guide the model into different regions.

Recent work has also focused on how the landscape shape affects sparse training and final performance. Evci et al. [3] analyzed why training sparse networks from scratch is difficult, finding that good sparse solutions are often separated by barriers that are hard to cross without temporarily adding more parameters. Finally, vonoswald2020neural used the geometry of the loss landscape late in training to improve generalization, showing that averaging weights found in the final stages acts as an efficient ensemble.

3.2. Important Observations in Supervised Learning

Improving the generalization and robustness of deep learning models is a central goal in supervised learning. One of the most effective strategies to achieve this is combining multiple models. The following observations highlight several key insights about combining

models and the geometry of the loss landscape in supervised learning, which have implications for understanding how to improve performance and generalization. Unless specified otherwise, we assume that $\theta_0^T \leftarrow \mathcal{T}_{\text{SL}}(\xi_0)$ and $\theta_1^T \leftarrow \mathcal{T}_{\text{SL}}(\xi_0)$ are two independently trained models that achieve high performance on the same task, and $g(\theta, x)$ represents the output of a model with parameters θ given input x .

One of the most well-known techniques for improving performance is ensembling, which combines the predictions of multiple models to produce a more accurate and robust output.

Observation 3.1 (Ensembling in Output Space [10]) *Ensembling θ_0^T and θ_1^T in output space, by averaging their predictions $\hat{y} = \frac{1}{2}(g(\theta_0^T, x) + g(\theta_1^T, x))$, boosts the accuracy \mathcal{A} , calibration, and robustness to distribution shift. This is attributed to functional diversity, i.e. two models making different errors that can be mitigated through averaging.*

While output ensembling provides strong performance benefits, it requires storing and running multiple models during inference, which is often computationally prohibitive. A natural question is whether we can achieve similar benefits more efficiently by combining the models into a single network through parameter averaging. However, for independently trained networks, this naive approach typically fails.

Observation 3.2 (Ensembling in Weight Space [7, 5]) *Ensembling θ_0^T and θ_1^T in weight space, by averaging their parameters $\hat{y} = g(\frac{1}{2}(\theta_0^T + \theta_1^T), x)$, fails. The resulting model performs significantly worse than either of the original models, indicating that solutions are not linearly connected in weight space, and that the loss landscape contains linear barriers between these solutions.*

The failure of weight averaging suggests that independent solutions normally reside in distinct basins of attraction separated by high-loss regions. However, this rule is not absolute. When models are not independent, or when they are explicitly trained to stay within a specific geometric region, averaging parameters can actually improve performance. This is particularly true when identifying "flat" minima, where the loss function is relatively insensitive to small perturbations in weights.

Observation 3.3 (Flat Region [15]) *If parameters $\theta_1, \theta_2, \dots, \theta_k$ lie at the periphery of a flat region in the loss landscape, then $\mathcal{A}(\frac{1}{k} \sum_{i=1}^k \theta_i) > \frac{1}{k} \sum_{i=1}^k \mathcal{A}(\theta_i)$, where \mathcal{A} is the accuracy of the model. This suggests that averaging parameters in a flat region can lead to improved performance compared to individual models.*

The success of averaging within a flat region naturally leads to an exploration of the conditions under which multiple models converge into the same basin to allow for direct linear interpolation. We know from Observation 3.2 that independently trained models are separated by high-loss barriers, but altering the training process fundamentally changes this geometry. Researchers have specifically found that networks sharing their initial optimization trajectory before branching off tend to remain within the same broad basin, which allows a straight line between them to maintain low loss.

Observation 3.4 (Linear Mode-Connectivity [7]) *There exists a linear connector when part of the optimization trajectory is shared. Let θ^t be the parameters at training step t . Starting from this shared point, we continue training two separate models to obtain $\theta_0^T = \mathcal{T}_{SL}^{T-t}(\xi_0, \theta^t)$ and $\theta_1^T = \mathcal{T}_{SL}^{T-t}(\xi_1, \theta^t)$, that are linearly connected, meaning that the loss along the linear path between them remains low.*

More broadly, when solutions are found independently, they are rarely linearly connected. However, researchers have demonstrated that the barriers separating these modes are not absolute. By moving beyond straight lines, it is possible to find low-loss paths that connect seemingly isolated solutions.

Observation 3.5 (Non-Linear Mode-Connectivity [10, 2]) *For two solutions θ_0^T and θ_1^T , there exists a non-linear curve $\theta(\alpha)$ in weight space such that $\theta(0) = \theta_0^T$, $\theta(1) = \theta_1^T$, and the loss along this curve remains low, even when a linear path between them has high loss. This indicates that while solutions may not be linearly connected, they can still be connected through more complex paths in the loss landscape.*

In summary, these observations highlight that the loss landscape in supervised learning is rich with geometric structure, where high-performing solutions are not isolated but rather connected through non-linear paths or, under specific conditions like flat minima, combined directly in weight space. These insights provide a foundation for understanding how geometry influences generalization and robustness. A key question then arises whether these geometric properties transfer to Reinforcement Learning, where the objective shifts from minimizing static prediction error to maximizing cumulative reward in a dynamic environment.

3.3. Mode Connectivity and Subspaces in Reinforcement Learning

While mode connectivity and subspaces have been extensively studied in supervised learning, their exploration in the context of RL is relatively limited. However, there are a handful of papers that investigate those in RL, particularly focusing on the geometry of policy spaces and how it can be leveraged for improved performance and adaptability.

Gaya et al. [12] address the scalability-plasticity dilemma in Continual Reinforcement Learning, where agents must learn a sequence of tasks without suffering from catastrophic forgetting or requiring intractable model growth. To solve this, they introduce Continual Subspace of Policies (CSP), a method that incrementally constructs a policy subspace by adaptively adding new basis directions only when necessary, allowing the agent to represent diverse behaviors within a compact structure. Experiments on the Brax [8] and Continual World [29] benchmarks demonstrate that CSP achieves superior performance compared to fixed-size and progressively growing baselines while exhibiting positive transfer and scaling sublinearly with the number of tasks.

Gaya et al. [11] propose learning a continuous subspace defined as a Line of Policies within the actor parameter space to address online adaptation in Deep Reinforcement Learning. By building upon algorithms like Proximal Policy Optimization (PPO) [22] and Advantage Actor-Critic (A2C) [18], they train this subspace so that any policy sampled from a convex combination of anchor parameters performs well on the training task while exhibiting diverse behaviors. This geometric structure allows the agent to implicitly capture an infinite set of strategies and enables efficient adaptation to unknown test dynamics by optimizing the mixing coefficients within the learned subspace. Empirical results demonstrate that searching within this pre-trained actor subspace significantly improves robustness and performance on unseen environments compared to single-policy baselines without requiring auxiliary discriminators.

Hendawy et al. [14] introduce STAR, a Multi-Task Reinforcement Learning algorithm that leverages mode connectivity to effectively share knowledge across diverse skills without increasing inference-time complexity. By enforcing a linear connection in parameter space between single-task policies and a central multi-task policy, the method ensures that high performance is maintained along the interpolation path. Empirical results on MuJoCo [24] and Metaworld [32] benchmarks demonstrate that this geometric constraint induces implicit regularization, enabling the multi-task policy to surpass related baselines while achieving competitive performance against complex mixture-of-expert approaches.

All three papers demonstrate that leveraging mode connectivity in RL can lead to improved performance, adaptability, and knowledge sharing across tasks, highlighting the potential of geometric approaches in advancing reinforcement learning methodologies. Yet, the exploration of mode connectivity in RL is still in its early stages, and there remain many open questions regarding the underlying geometry of policy spaces, the impact of different architectures and training regimes on connectivity, and how these insights can be further exploited for more efficient learning and generalization in complex environments.

4. Methodology

This chapter outlines the proposed methodology, bridging the theoretical algorithms with their practical software implementation. We first introduce the underlying mechanics of subspace learning and the specific reinforcement learning algorithms evaluated in this study. Subsequently, we detail the high-performance engineering required to make this approach computationally tractable, including native JAX acceleration, environment parallelization, and optimized subspace layer evaluations.

4.1. Algorithms

To effectively train the continuous subspace, we must adapt the standard Q-learning objective to account for the parameterized network weights. In the case of DQN, the primary objective function for the network update is defined as an expectation over both the transition dynamics and the sampled subspace parameters

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{w} \sim \mathcal{W}, (s, a, r, s', d) \sim \mathcal{D}} \left[(Q_{\theta(\mathbf{w})}(s, a) - y)^2 \right]$$

During the environmental interaction phase, we sample actions by drawing a single parameter configuration \mathbf{w} at the beginning of each episode and acting greedily with respect to that specific network instantiation. The following pseudocode outlines the high-level adaptations applied to the standard Deep Q-Network to create our Subspace Deep Q-Network (SubDQN) algorithm.

Algorithm 2 Subspace Deep Q-Network (SubDQN)

```
1: Initialize Q-networks  $\{\theta_i\}_{i=1}^k$  and  $\{\bar{\theta}_i\}_{i=1}^k$ , replay buffer  $\mathcal{D}$ , state  $s_0$  and  $w_0$ 
2: while  $t_{\text{total}} < n_{\text{steps}}$  do
3:   Sample action  $a_t \sim \epsilon\text{-greedy}(Q_{\theta(w_t)}(s_t, \cdot))$ , execute  $a_t$ , observe  $r_t, s_{t+1}$ , and  $d_t$ 
4:   Update  $w_t \sim \mathcal{W}$  if the current episode has terminated
5:   Store transition  $(s_t, a_t, r_t, s_{t+1}, d_t)$  in  $\mathcal{D}$ 
6:   Sample random minibatch  $m$  of transitions  $(s_j, a_j, r_j, s_{j+1}, d_j) \sim \mathcal{D}$ 
7:   Sample  $w_j \sim \mathcal{W}$  and compute  $y_j = r_j + (1 - d_j)\gamma \max_{a'} Q_{\bar{\theta}(w_j)}(s_{j+1}, a')$ 
8:    $\{\theta_i\}_{i=1}^k \leftarrow \{\theta_i\}_{i=1}^k - \eta \nabla_{\theta} \left( \frac{1}{|m|} \sum_{j \in m} \left( Q_{\theta(w_j)}(s_j, a_j) - y_j \right)^2 \right)$ 
9:   if  $t_{\text{total}} \bmod n_{\text{target}} = 0$  then
10:     $\{\bar{\theta}_i\}_{i=1}^k \leftarrow \{\theta_i\}_{i=1}^k$ 
11:   end if
12:    $t_{\text{total}} \leftarrow t_{\text{total}} + 1$ 
13: end while
```

This algorithm introduces a few key deviations from a standard DQN implementation. Foremost, instead of maintaining a single set of parameters, we initialize and optimize a set of underlying anchors $\{\theta_i\}_{i=1}^k$ that collectively define the subspace. The behavioral policy is determined by sampling a specific network configuration w_t for each episode, which ensures temporally consistent exploration. Furthermore, during the optimization step, a distinct parameter configuration w_j is sampled for every transition in the minibatch to compute both the target value y_j and the corresponding Q-value prediction.

Importantly, these subspace modifications are agnostic to the underlying value-based algorithm and can be readily applied to other methods, such as Parallel Q-Networks (PQN). When specifically utilizing connector-type subspaces, we adapt our sampling strategy to align with the geometric structure of the connection. Instead of sampling from a general high-dimensional distribution \mathcal{W} , we parameterize the subspace using an interpolation weight α and sample from a one-dimensional distribution, such as $\alpha \sim \text{Uniform}(0, 1)$, or more generalized distributions defined over the segment between two modes. We denote these specific connector-based approaches as ConDQN and ConPQN.

Finally, we employ this exact same algorithmic strategy to conduct our baseline mode connectivity experiments. The sole distinction in the mode connectivity setting is that the weights of the endpoints are strictly frozen during optimization, whereas our subspace learning approach trains all parameters simultaneously. We refer to this variant as MCPQN.

4.2. Implementation Details

4.2.1. Algorithms

Landscape analysis inherently requires mapping the loss landscape of a given model, a process that relies on evaluating the objective function across various points in the parameter space. In standard supervised learning, this procedure is relatively straightforward, as the loss can be efficiently computed over a fixed, static dataset for any arbitrary set of parameter values.

In the context of reinforcement learning (RL), however, this evaluation becomes highly computationally expensive. Because there is no static dataset, estimating the performance of a specific parameter configuration requires executing the corresponding policy within an environment to simulate trajectories and accumulate empirical rewards.

To mitigate this computational bottleneck and make landscape evaluation efficient, we implement the entire training process of our algorithms natively in JAX. This end-to-end approach allows us to heavily leverage hardware acceleration and large-scale parallelization. Beyond the significant performance improvements, this implementation strategy yields a much cleaner and highly modular codebase. It allows us to elegantly decouple the core algorithmic logic from the environment interactions and other low-level implementation details. The following is an example showing the two-step process of training for mode connectivity, providing a side-by-side comparison of the mathematical formalization and its highly concise JAX implementation.

Formalization	Implementation
$\theta_1 \leftarrow \mathcal{T}_{\text{PQN}}(\xi_1, \theta_1)$	<code>alg_state1 = pqn.train(key1, pqn.init(key1))</code>
$\theta_2 \leftarrow \mathcal{T}_{\text{PQN}}(\xi_2, \theta_2)$	<code>alg_state2 = pqn.train(key2, pqn.init(key2))</code>
$\theta \leftarrow \mathcal{T}_{\text{MCPQN}}(\xi, \theta_1, \theta_2)$	<code>alg_state = mcpqn.train(key, alg_state1, alg_state2)</code>

Figure 4.1.: Side-by-side comparison of the formalization of the training process for the Subspace Deep Q-Network (SubDQN) algorithm and its implementation in JAX. The left side shows the mathematical formalization, while the right side presents the corresponding JAX code.

4.2.2. Environments

While implementing RL algorithms entirely in JAX offers substantial performance benefits, an end-to-end approach introduces a major drawback. The environments themselves must also be implemented natively in JAX to maintain hardware acceleration. Unfortunately, this is not the case for many standard benchmarks, such as the widely used environments provided in the Gymnasium library.

Maintaining a separate codebase solely to interact with these environments is impractical, as it would require duplicating the implementation and maintenance of all algorithms and experimental setups. To resolve this compatibility issue, we utilize the `gxm`¹ library, which was developed specifically within the context of this thesis. The `gxm` library provides a JAX-compatible interface designed to wrap standard CPU-based environments in a JAX-friendly manner. Crucially, this wrapper enables the use of JAX’s vectorizing map (`vmap`) transformations, allowing for seamless parallelization of environment steps. Ultimately, `gxm` provides a unified interface that accommodates both standard CPU environments and popular native JAX-based environments, ensuring a streamlined and highly maintainable codebase.

```
import gxm

env = gxm.make("Gymnasium/ALE/Breakout-v5")
env_state, timestep = env.init(key)
action = env.action_space.sample(key)
env_state, timestep = env.step(key, env_state, action)
```

Figure 4.2.: Example of wrapping a standard CPU-based environment using the `gxm` library. The wrappers are compatible with JAX’s `vmap` transformations, allowing for efficient unified parallelization of environment interactions.

4.2.3. Efficient Subspace Implementation

Naively computing the shared weights $\theta(w)$ is computationally efficient only when applying the identical weight configuration w to a large batch of inputs x . However, when a setup requires batching over both diverse inputs and varying weights w simultaneously,

¹<https://github.com/huterguier/gxm>

explicitly materializing the shared weights θ_w for each individual input necessitates a separate forward pass. This approach scales poorly and is computationally expensive.

To circumvent this bottleneck, we can leverage the linearity of the individual network layers. This mathematical property allows us to compute the forward pass for any given combination of input and weight without ever explicitly materializing the shared parameterization $\theta(w)$.

Specifically, this optimization is achieved by first computing the output of each basis layer independently for a given input. Once these intermediate outputs are evaluated, they are subsequently combined using the respective weight configuration w . For example, consider a standard linear layer defined by a set of basis parameters $(\mathbf{W}_i, \mathbf{b}_i)$ for $i \in \{1, \dots, n\}$, applied to an input \mathbf{x}^j with corresponding weights w^j . The efficient forward pass can be mathematically formulated as follows:

$$\mathbf{y}^j = \left(\sum_{i=1}^n w_i^j \mathbf{W}_i \right) \mathbf{x}^j + \sum_{i=1}^n w_i^j \mathbf{b}_i = \sum_{i=1}^n w_i^j (\mathbf{W}_i \mathbf{x}^j) + \sum_{i=1}^n w_i^j \mathbf{b}_i = \sum_{i=1}^n w_i^j (\mathbf{W}_i \mathbf{x}^j + \mathbf{b}_i) \quad (4.1)$$

This approach extends naturally to convolutional layers, where the forward pass is computed by independently applying each basis convolution to the input and combining the resulting feature maps according to the weights w^j . Without this optimization, the computational cost of evaluating a single forward pass would scale linearly with the batch size, making the approach computationally infeasible.

5. Landscape Analysis

This chapter investigates whether well-documented optimization phenomena from supervised learning hold true for value-based reinforcement learning. Unlike supervised learning, which minimizes a stationary loss to improve accuracy \mathcal{A} , reinforcement learning presents a complex and non-stationary optimization problem where the data distribution actively shifts as the policy evolves. Given this inherent complexity, we aim to determine if training under these non-stationary conditions still yields similar structural properties regarding our true objective of maximizing the expected return J . To better understand these dynamics, we analyze the geometric optimization landscape of value-based reinforcement learning algorithms.

5.1. Ensembling

Ensembling multiple models in the output space is a prominent technique in reinforcement learning, particularly within value-based methods where function approximation inherently introduces estimation bias and variance. The literature features various approaches that leverage ensembles to significantly improve both the performance and the stability of these algorithms. Because individual models often converge to different local optima, they tend to make uncorrelated errors during training. Aggregating their predictions harnesses this functional diversity, effectively smoothing out inaccuracies and preventing the greedy maximization step from exploiting isolated overestimations. This mechanism is formalized in Observation 5.1, which illustrates how averaging action values translates directly to an improved policy.

Observation 5.1 (Ensembling in Output Space) *Ensembling θ_0^T and θ_1^T in output space, by averaging their Q-functions $\hat{y} = \frac{1}{2}(Q_{\theta_0^T}(s, \cdot) + Q_{\theta_1^T}(s, \cdot))$ increases J , the expected return of the greedy policy of the ensembled Q-function. This is attributed to functional diversity, i.e. two models making different errors that can be mitigated through averaging.*

Several prominent approaches in the literature actively exploit the principles outlined in Observation 5.1 to enhance both stability and exploration. Foundational work on ensemble methods by Wiering and Van Hasselt [28] established that combining multiple value estimators directly mitigates individual approximation errors and yields more robust policies. Building upon these concepts for deep reinforcement learning, Osband et al. [20] introduced Bootstrapped DQN, which maintains an ensemble of Q_θ networks trained on different data bootstraps to capture epistemic uncertainty and drive deep exploration. Similarly, UCB exploration strategies have been successfully adapted by utilizing the variance across an ensemble of Q_θ functions to construct upper confidence bounds for action selection [1]. Beyond exploration, ensembling is highly effective for variance reduction and bias control. Maxmin Q-learning [16] utilizes a Q-ensemble to compute a more conservative target \bar{y} by taking the minimum over multiple action value estimates, effectively controlling the overestimation bias that naturally arises during the greedy maximization step. Together, these methods highlight how functional diversity within an ensemble acts as a critical mechanism for improving the performance of value-based agents.

In contrast to ensembling in the output space, directly averaging model parameters in weight space is generally ineffective for independently trained networks. The loss landscape of deep neural networks is highly non-convex and characterized by multiple isolated local minima. Consequently, independent solutions are rarely linearly connected, a phenomenon well-documented in supervised learning where linear interpolation between converged networks typically traverses regions of high loss.

This isolation is visually confirmed in Figure 5.1, which projects three individually trained Q_θ functions onto a 2D plane. The heatmap clearly demonstrates that the individual high-performance modes are separated by regions of significantly lower return. Crucially, the midpoint between these distinct solutions exhibits very poor performance, reinforcing the premise that the optimization landscape is highly non-convex and that independently found optima are not linearly connected. This empirical evidence leads directly to Observation 5.2.

Observation 5.2 (Ensembling in Weight Space [7, 5]) *Ensembling θ_0^T and θ_1^T in weight space, by averaging their parameters $\bar{y} = Q_{\frac{1}{2}(\theta_0^T + \theta_1^T)}(s, \cdot)$, fails. The resulting model performs significantly worse than either of the original models, indicating that solutions are not linearly connected in weight space, and that the landscape of the optimization problem is highly non-convex with multiple isolated minima.*

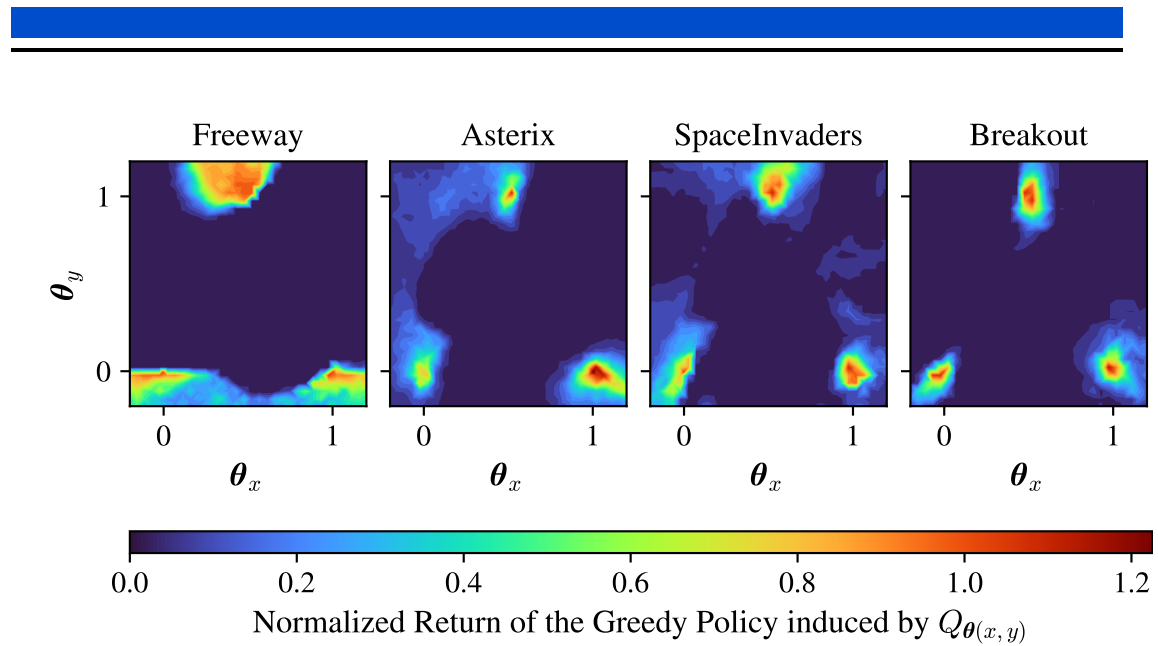


Figure 5.1.: Three individually trained Q-functions projected onto a 2D plane. The color of the heatmap corresponds to the return of the greedy policy of the Q-function at that point in the plane.

Because independently trained models follow distinct optimization trajectories, their learned representations are fundamentally misaligned, causing the averaged weights to collapse into high-loss regions. However, while naively ensembling these independent solutions fails, there are specific topological settings where weight-space averaging becomes highly effective.

5.2. Flat Maxima

In supervised learning, flat minima are a well-documented topological feature frequently associated with improved generalization performance. In these regions, the model's predictions remain robust to small perturbations in the parameter space. To understand if this desirable property transfers to value-based reinforcement learning, we investigate the prevalence of flat regions within the RL objective landscape and whether exploiting them can lead to enhanced policy performance.

Observation 5.3 (Flat Region) *If parameters $\theta_1, \theta_2, \dots, \theta_k$ lie at the periphery of a flat region in the loss landscape, then $J(\frac{1}{k} \sum_{i=1}^k \theta_i) > \frac{1}{k} \sum_{i=1}^k J(\theta_i)$, where J is the expected return of the greedy policy of the Q-function at that point in weight space.*

By identifying and averaging weights that lie on the periphery of such a flat maximum, the resulting ensembled model effectively centers itself within the robust region. This centering not only smooths the action value estimates but ultimately yields a higher expected return than the individual peripheral models, providing a powerful mechanism for improving generalization in reinforcement learning [30].

To empirically validate the existence of these flat regions, we can visualize the parameter space by constructing a simplex subspace bounded by specifically optimized peripheral weights. Figure 5.2 illustrates this across several MinAtar environments.

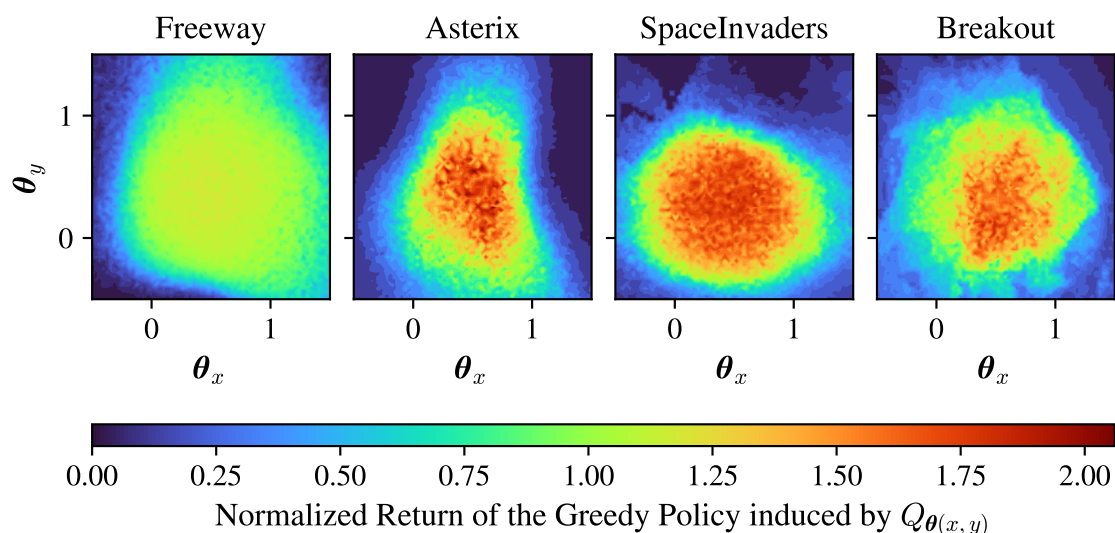


Figure 5.2.: Simplex Q-function subspaces trained with DQN on four different MinAtar games with 3 anchors. The anchors are marked with black markers, and the simplex vertices are marked with white markers. The color of the heatmap corresponds to the return of the greedy policy of the Q-function at that point in the simplex.

The heatmaps clearly show that the high-return areas are not confined to sharp, isolated

peaks. Instead, they span broad, continuous regions within the simplex, confirming the presence of accessible flat maxima in the optimization landscape.

Having visualized the topology of this subspace, we next quantitatively track the performance dynamics within it. Because the peripheral anchors were explicitly optimized to construct this flat region, our objective here is not to claim an immediate algorithmic advantage, but rather to empirically verify the mechanics of Observation 5.3. By plotting the actual episodic returns over the course of training, we can directly demonstrate that the geometric center of this simplex—obtained by averaging the trained boundary weights—consistently yields a higher and more stable performance than the individual anchors bounding it.

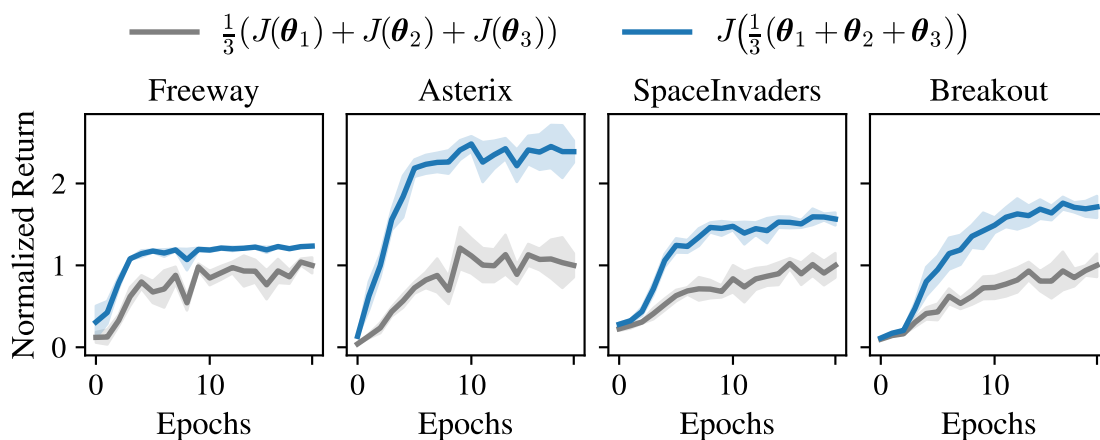


Figure 5.3.: Empirical demonstration of the flat region observation across four MinAtar environments. The normalized return of the weight-averaged model (blue) consistently outperforms the average return of the individual peripheral models (grey). This highlights the performance benefits of centering parameters within a flat maximum.

As demonstrated in Figure 5.3, the empirical results align perfectly with Observation 5.3. Throughout the training process, the ensembled model situated at the center of the flat maximum consistently achieves a higher normalized return than the average of its peripheral counterparts.

5.3. Mode Connectivity

The central inquiry of this section revolves around whether we can successfully find a connector between two independently trained solutions within the loss landscape. To systematically investigate this, we first study an alternative setting where we share a portion of the initial training trajectory before diverging to continue training two separate models. In this shared-trajectory scenario, we consistently find a linear connector between the two resulting models. This phenomenon is illustrated in Figure 5.4.

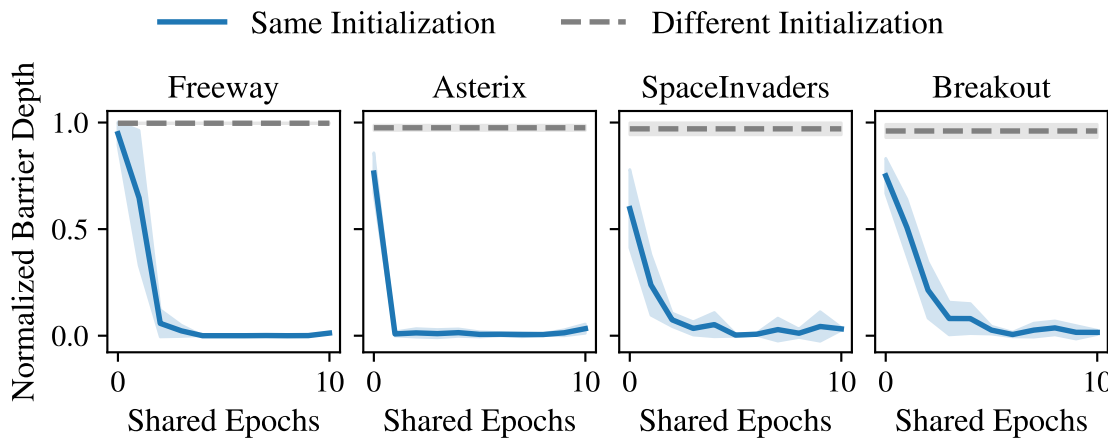


Figure 5.4.: Normalized barrier depth as a function of shared training epochs across four environments. The plot compares the resulting barrier depth between models that share the same initialization and models that start from different initializations.

As we systematically increase the number of shared training epochs, the resulting barrier depth steadily decreases. This trend strongly indicates that the two models become increasingly linearly connected the longer their initial optimization paths overlap. Furthermore, the data reveals that even a shared initialization without subsequent shared training steps can have a profound and significant effect on reducing the final barrier depth. These empirical findings directly lead to the following formal observation.

Observation 5.4 (Linear Mode-Connectivity) *There exists a linear connector when part of the optimization trajectory is shared. Let θ^t be the parameters at training step t . Starting*

from this shared point, we continue training two separate models to obtain $\theta_0^T = \mathcal{T}_{QL}^{T-t}(\xi_0, \theta^t)$ and $\theta_1^T = \mathcal{T}_{QL}^{T-t}(\xi_1, \theta^t)$ that are linearly connected, meaning that the return along a linear path between them remains low.

Building upon these insights, we now shift our focus to the more challenging setting where we train two models entirely independently from the start and subsequently attempt to find a valid connector between them. For this experiment, we first train two separate Q-functions independently until convergence. We then parameterize and train a quadratic Bezier curve specifically to connect these two distinct points in the parameter space. By visualizing this learned curve within the two-dimensional plane spanned by the two solutions, we can effectively demonstrate that the curve forms a fully connected path characterized by a consistently low loss. Figure 5.5 displays the heatmap of the return along this learned curve.

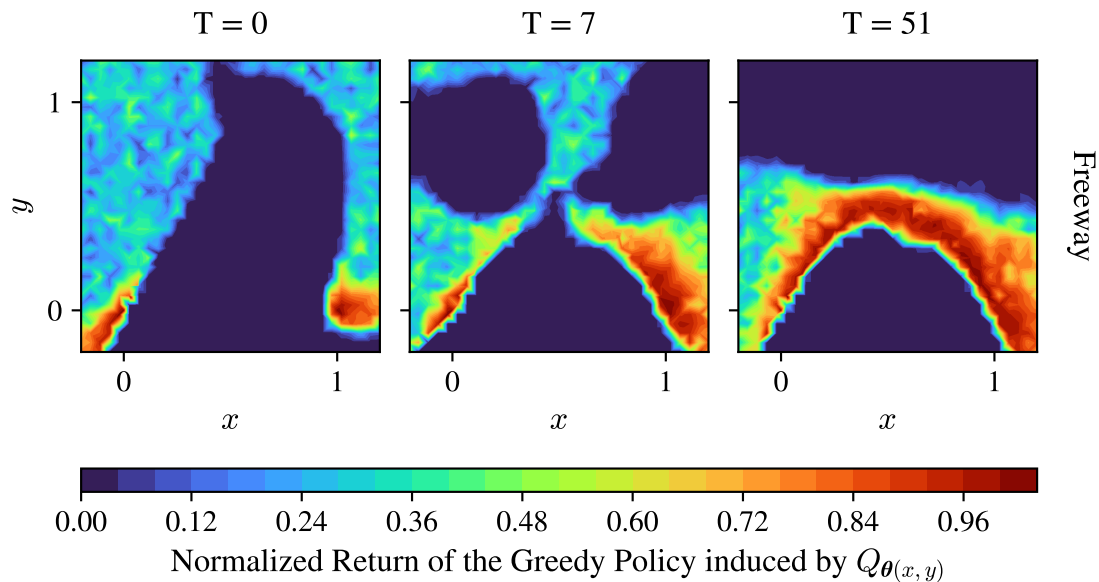


Figure 5.5.: Normalized return landscape across interpolation weights during training. This heatmap visualizes the agent’s performance along the learned quadratic Bezier curve. The curves are embedded in the 2D plane. The individual axes show the learned subspace at different training epochs.

Initially, the two independent models remain entirely separated by regions of high loss in the parameter space. However, as we train the third subspace anchor defining the Bezier

curve, the models become progressively more connected. This optimization continues until they ultimately achieve a fully connected solution pathway. We have successfully demonstrated this non-linear connectivity phenomenon across multiple environments, and a comprehensive visualization encompassing all four MinAtar environments can be found in Figure A.3.

Based on these comprehensive experiments, we can conclude that linear mode connectivity indeed exists within the domain of value-based reinforcement learning. However, this linear connectivity is by no means guaranteed and depends heavily on the specific training procedure and shared initialization dynamics. Despite evaluating a limited number of environments and random seeds, our findings provide compelling evidence that completely independent modes can still be effectively connected.

Observation 5.5 (Non-Linear Mode-Connectivity) *For two solutions θ_0^T and θ_1^T , there exists a non-linear curve $\theta(\alpha)$ in weight space such that $\theta(0) = \theta_0^T$, $\theta(1) = \theta_1^T$, and the loss along this curve remains low, even when a linear path between them has high loss. This indicates that while solutions may not be linearly connected, they can still be connected through more complex paths in the loss landscape.*

In summary, our empirical investigation demonstrates that while the loss landscape of value-based reinforcement learning models may not inherently support linear paths between independent solutions, it is sufficiently rich to harbor complex non-linear connectors. By leveraging parameterized curves such as quadratic Bezier pathways, we can successfully navigate the high-dimensional weight space to link disparate local minima without suffering significant performance degradation. This confirms that mode connectivity is a practical and observable phenomenon in reinforcement learning, which ultimately paves the way for advanced subspace learning techniques and more robust policy ensembles.

6. Subspace Learning

While the concept of mode connectivity provides valuable theoretical insights into the loss landscape of neural networks, making it a practical tool for reinforcement learning presents significant challenges. Specifically, methods applied after independent training are not guaranteed to successfully find a connected path between the solutions. Furthermore, the optimization process required to discover these connections is often computationally prohibitive. To address these limitations, we shift our approach from searching for connections after training to proactively learning a continuous subspace of policies that are connected by design.

6.1. Training the Subspace

To establish this continuous subspace, we utilize a connector subspace, which is specifically designed to connect two points within the parameter space. Unlike traditional mode connectivity approaches that first train independent anchor models and subsequently attempt to connect them, our method trains the entire subspace simultaneously without fixing any anchors. To prevent the models within this subspace from collapsing into a single identical solution, we incorporate a diversity loss during the training process, as in [30]. This loss function calculates the pairwise cosine similarity of the network parameters to effectively encourage structural diversity across the learned subspace.

$$\mathcal{L}_{similarity} = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^N \cos^2(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) \quad (6.1)$$

Although the theoretical foundation of this continuous subspace framework is broadly applicable across various standard Q-learning methodologies, we formalize our approach in this work through the Connector Parallel Q-Networks (ConPQN) algorithm. This specific formulation allows us to effectively establish and analyze the continuous parameter

subspace within the broader context of value-based reinforcement learning. By adapting this architecture, ConPQN provides a robust mechanism to simultaneously train the connected policy space while seamlessly navigating the complex dynamics of the underlying Q-function.

Algorithm 3 Connector Parallel Q-Network (ConPQN)

```

1: Initialize Q-network  $\{\theta_i\}_{i=1}^k$ , and points  $\alpha_0^i$  and initial states  $s_0^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
2:  $t_{\text{total}} \leftarrow 0$ 
3: while  $t_{\text{total}} < n_{\text{steps}}$  do
4:   for  $t = 0 \dots n_{\text{rollout}} - 1$  do
5:     Sample actions  $a_t^i \sim \epsilon\text{-greedy}(Q_{\theta(\alpha_t^i)}(s_t^i, \cdot))$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
6:     Execute  $a_t^i$ , observe  $r_t^i, s_{t+1}^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
7:     Update  $\alpha_{t+1}^i$  if the corresponding environment terminated
8:   end for
9:   Compute  $R_t^{\lambda, i}(\alpha_{i,t}) = r_t^i + \gamma \left[ \lambda R_{t+1}^{\lambda, i}(\alpha_{i,t}) + (1 - \lambda) \max_{a'} Q_{\theta(\alpha_{i,t})}(s_{t+1}^i, a') \right]$ 
10:  for epoch =  $1 \dots n_{\text{epochs}}$  do
11:    Shuffle data over time  $t$  and environment  $i$  axes to form  $M$  minibatches
12:    for each minibatch  $m$  do
13:       $\mathcal{L}_{\text{total}} \leftarrow \frac{1}{M} \sum_{(s_t^i, a_t^i) \in m} \left( Q_{\theta(\alpha_{i,t})}(s_t^i, a_t^i) - R_t^{\lambda, i}(\alpha_{i,t}) \right)^2 + \beta \mathcal{L}_{\text{similarity}}(\{\theta\}_{i=1}^k)$ 
14:       $\{\theta\}_{i=1}^k \leftarrow \{\theta\}_{i=1}^k - \eta \nabla_{\{\theta\}_{i=1}^k} \mathcal{L}_{\text{total}}$ 
15:    end for
16:  end for
17:   $s_0^i \leftarrow s_{n_{\text{rollout}}}^i$  for  $i \in \{1, \dots, n_{\text{envs}}\}$ 
18:   $t_{\text{total}} \leftarrow t_{\text{total}} + (n_{\text{rollout}} \times n_{\text{envs}})$ 
19: end while

```

Algorithm 3 outlines the complete procedure for the Connector Parallel Q-Network (ConPQN). The algorithm operates by seamlessly integrating the exploration of a continuous parameter space with standard reinforcement learning rollouts.

The process begins by defining the specific geometry of the connector subspace, which can take the form of a simple linear path or a more complex structure such as a Bézier curve. Based on the chosen subspace formulation, we initialize the parameter set $\{\theta\}_{i=1}^k$, where k corresponds to the number of structural anchor points required to define the subspace geometry.

During the data collection phase, we sample trajectories across multiple parallel environments. To ensure the network learns a robust continuum of policies, each environment i is assigned a distinct interpolation coefficient α_t^i . This coefficient dictates the exact network configuration within the continuous subspace used for action selection via an ϵ -greedy policy. To maintain diverse exploration, the assigned α values are updated or resampled at the conclusion of each episode.

Following the rollout, the algorithm computes the multi-step temporal difference returns $R_t^{\lambda,i}(\alpha_{i,t})$. This step represents the most critical and computationally demanding component of the ConPQN architecture. To perform a mathematically sound gradient update for a specific point in the subspace, the multi-step return must be evaluated using the exact same α that was used during the trajectory generation.

Evaluating these returns across a dense continuum of the subspace is computationally prohibitive. To mitigate this computational bottleneck, we typically evaluate the returns using a small, discrete set of α values per rollout. In our experiments, we found that sampling five uniformly spaced α values provides sufficient coverage of the subspace to ensure effective learning and regularization without incurring an overwhelming computational cost.

Finally, the network parameters are updated using a composite loss function, denoted as $\mathcal{L}_{\text{total}}$. This objective function combines the standard Q-learning temporal difference error with the structural regularization provided by the diversity loss. As previously stated in Equation 6.1, the diversity loss computes the average pairwise cosine similarity of the network parameters across the different sampled points. By incorporating this term, we actively prevent the distinct anchor points from collapsing into a single, identical solution, thereby maintaining the desired continuous diversity across the learned subspace.

6.2. Experiments

To evaluate the efficacy of our proposed method, we conduct comprehensive experiments across a suite of 15 Atari games. For each environment, we run the algorithms using 5 distinct random seeds and report the Interquartile Mean (IQM) along with the 95% confidence intervals to ensure robust statistical evaluation. Throughout these experiments, we utilize the default hyperparameters established in the original PQN paper by [9].

We benchmark our ConPQN approach against several baselines. The primary baseline is the standard PQN algorithm. Additionally, we compare against an ensemble variant

of PQN. This ensemble consists of two separate Q-functions trained on the exact same dataset but initialized differently and subjected to independent training noise. Finally, drawing inspiration from [30], we analyze a Standardized Ensemble of Two (SEO2). This specific evaluation metric computes the average prediction of the Q-functions at interpolation weights α and $1 - \alpha$. We plot the performance of this SEO2 variant to thoroughly investigate the functional diversity captured within our learned subspace.

6.2.1. Different Subspaces

We evaluate a basic implementation of the Connector Parallel Q-Networks (ConPQN) utilizing two distinct geometric structures for the subspace, namely a line and a Bezier curve. The line subspace connects the endpoints directly as previously depicted in Figure 2.2. In contrast, the second approach employs a quadratic Bezier curve subspace. This structure resembles the curves shown earlier in Figure 2.3, though it specifically utilizes exactly three anchor points rather than four to define the continuous path. The empirical results for these configurations are presented in Figure 6.1.

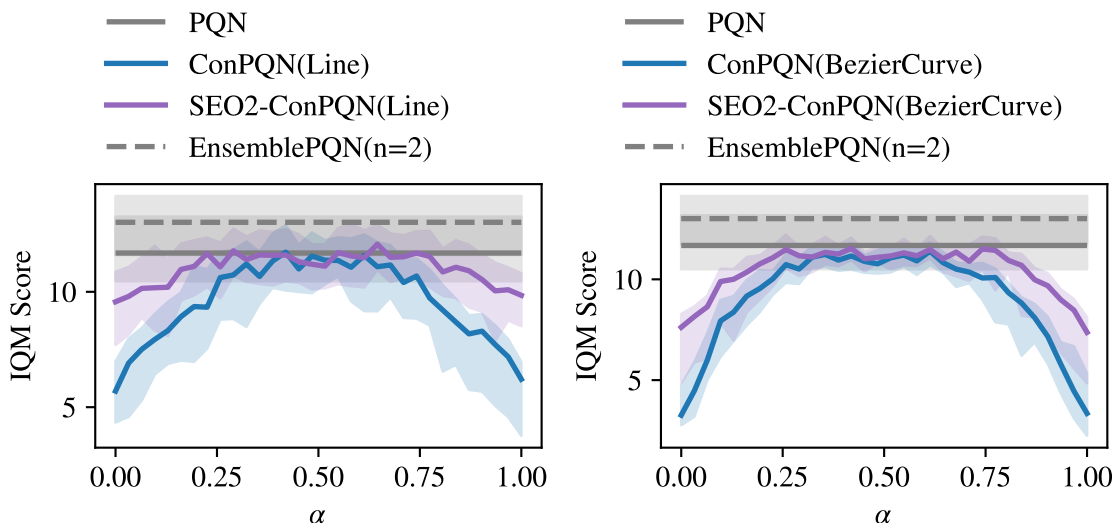


Figure 6.1.: IQM of human-normalized scores on 15 Atari games (5 seeds); shaded regions show confidence intervals. We compare ConPQN and SEO2-ConPQN against PQN and EnsemblePQN baselines across interpolation weights α , using a Bezier curve (**Left**) and line (**Right**) subspace.

The performance metrics indicate that both subspace approaches perform roughly on par with the standard PQN baseline. Between the two variants, the simple line subspace achieves slightly better results than the quadratic Bezier curve. However, both ConPQN implementations ultimately fall short of the performance achieved by the EnsemblePQN baseline.

Furthermore, analyzing the Standardized Ensemble of Two (SEO2) reveals an important characteristic of the learned representations. The SEO2 performance curve lacks the expected distinct peaks at the extreme ends of the subspace. This observation strongly suggests that the learned subspace lacks sufficient functional diversity and fails to capture significantly different policies across the varying interpolation weights α . This fundamental lack of diversity motivates the need for further ablation studies. Subsequent sections will thoroughly investigate the factors contributing to this phenomenon and explore potential modifications to the training procedure and the diversity loss function that could successfully enhance the overall diversity of the learned continuous subspace.

6.2.2. Without Multi-Step Return

One notable drawback of learning continuous subspaces with PQN is the computational overhead associated with the return calculations. Specifically, the multi-step return must be computed across all interpolation weights, α_i . This requirement presents a difficult trade-off since evaluating a large number of α_i along the connector becomes highly computationally expensive, whereas using a small number fails to adequately cover and regularize the learned subspace.

To investigate the model’s dependency on this computation, we conduct an ablation study replacing the multi-step λ -return with a simple one-step return for all α values, intentionally ignoring the resulting off-policy issues. We evaluate this ablated ConPQN model against standard PQN and PQN ensemble baselines, both of which are similarly restricted to utilizing a one-step return. The empirical results, illustrated in Figure 6.2, reveal a noticeable contrast in robustness between the architectures. While relying on a one-step return does not significantly impact the performance of the standard PQN and EnsemblePQN, it leads to a substantial performance degradation for the ConPQN.

One plausible explanation for this discrepancy is that the multi-step return provides a much more stable target during the update process, which acts as a crucial regularizer when learning the continuous subspace. This target instability proves to be uniquely detrimental to the ConPQN architecture because the target values are derived directly from the evolving

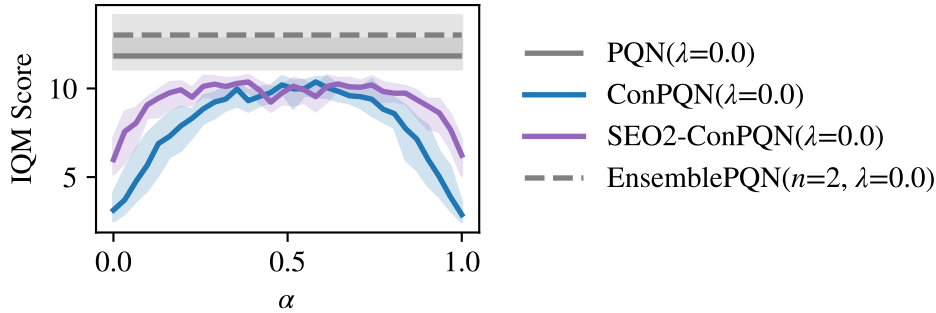


Figure 6.2.: IQM of human-normalized scores on 15 Atari games over 5 seeds, where shaded regions indicate confidence intervals. We compare ConPQN and SEO2-ConPQN utilizing one-step return and a Bezier curve subspace against PQN and EnsemblePQN baselines across interpolation weights α .

subspace itself. Consequently, without the variance-reducing and smoothing effects of multi-step returns, the bootstrapping targets are highly susceptible to large margins of error for certain α values, ultimately destabilizing the overall learning process.

6.2.3. Feature Similarity

Analogous to the approach presented by [30], we conduct ablation studies to evaluate the impact of the diversity loss on the model’s performance. Instead of regularizing the network by calculating the cosine similarity of the parameters, we investigate an alternative method: regularizing the similarity of the features across varying interpolation weights, α .

Specifically, for a given state and action pair, we extract the features from the penultimate layer of the network for different α along the connector. We then regularize the similarity of these extracted features by applying a modified cosine similarity loss, defined as follows.

$$\mathcal{L}_{feature} = \beta |\alpha_j - \alpha_k| \cos^2(\phi_j, \phi_k) \quad (6.2)$$

In this formulation, ϕ_j and ϕ_k represent the penultimate layer features corresponding to the interpolation parameters α_j and α_k , respectively. Notably, this loss is scaled by the absolute distance between the parameters, $|\alpha_j - \alpha_k|$. This scaling factor is crucial because it encourages greater feature diversity between models that are located further apart within the learned subspace. Conversely, heavily penalizing the similarity of models

with closely situated α values can be detrimental, as it disrupts the continuity and leads to a less smooth subspace. The empirical results of this ablation study are illustrated in Figure 6.3. As observed in the figure, substituting parameter similarity with feature

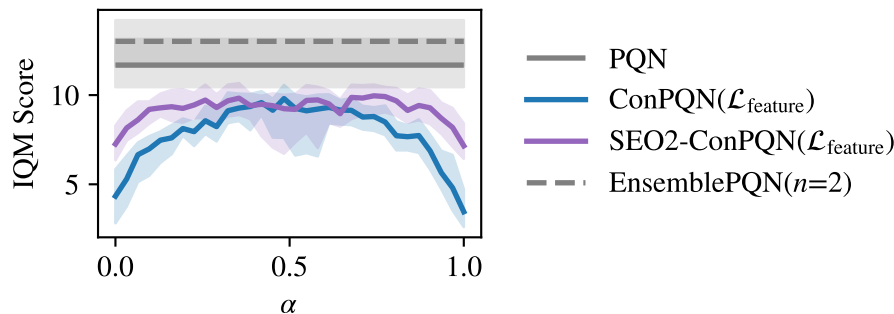


Figure 6.3.: IQM of human-normalized scores on 15 Atari games over 5 seeds, where shaded regions indicate confidence intervals. We compare ConPQN and SEO2-ConPQN utilizing the feature similarity for regularization and a Bezier curve subspace against PQN and EnsemblePQN baselines across interpolation weights α .

similarity does not yield a significant positive impact on the performance of ConPQN. In practice, enforcing this feature-level diversity regularization actually leads to a significant degradation in overall performance.

6.2.4. Utilizing the Subspace during Training

As previously discussed, training a continuous subspace ideally offers advantages comparable to training an ensemble of independent policies but incurs a fraction of the computational cost. We hypothesize that these advantages can be leveraged not only during evaluation but also actively during the training phase. To this end, we investigate two distinct methods of utilizing the learned subspace to stabilize the learning process.

The first method involves computing the Q-values for multiple interpolation weights, α , and subsequently averaging their predictions. This aggregated prediction serves as a more stable bootstrapping target for the network updates. Mathematically, the modified target y can be expressed as

$$y = r + \gamma \max_{a'} \frac{1}{N} \sum_{i=1}^N Q(\alpha_i)(s', a') \quad (6.3)$$

where N represents the number of sampled α_i along the connector. The empirical results for this target averaging approach are presented in Figure 6.4. Contrary to our expectations,

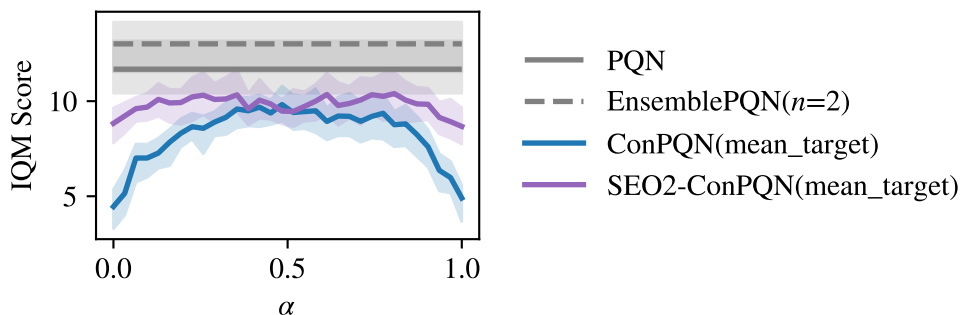


Figure 6.4.: IQM of human-normalized scores on 15 Atari games over 5 seeds, where shaded regions indicate confidence intervals. We compare ConPQN and SEO2-ConPQN utilizing mean target averaging and a Bezier curve subspace against PQN and EnsemblePQN baselines across interpolation weights α .

utilizing the mean target across the subspace fails to yield any performance improvements. As depicted in the figure, this approach can even lead to a degradation in the overall learning efficiency in certain cases, resulting in lower average return over all games.

The second method we explore focuses on the behavioral aspect of the agent. Instead of modifying the update target, we utilize the subspace to determine a more robust action during the exploration and data gathering phase. Similar to the previous method, we compute the Q-values for multiple α values and average them to select a stable action. The action a chosen at state s is formalized as

$$a = \arg \max_a \frac{1}{N} \sum_{i=1}^N Q_{\alpha_i}(s, a) \quad (6.4)$$

Figure 6.5 illustrates the results of applying this mean action selection strategy.

Much like the target averaging strategy, leveraging the subspace for action selection does not translate into enhanced performance. One plausible explanation for this failure is a lack of sufficient diversity within the learned subspace itself. If the individual networks corresponding to different α weights produce highly correlated Q-values, averaging them will not provide significantly distinct or superior actions for a given state compared to a single network evaluation.

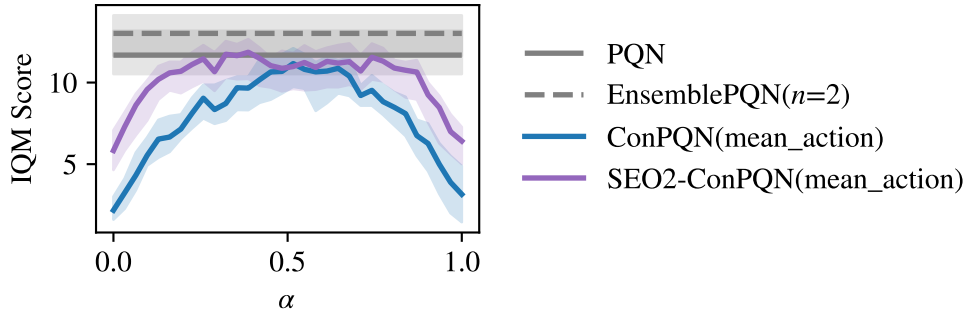


Figure 6.5.: IQM of human-normalized scores on 15 Atari games over 5 seeds, where shaded regions indicate confidence intervals. We compare ConPQN and SEO2-ConPQN utilizing mean action selection and a Bezier curve subspace against PQN and EnsemblePQN baselines across interpolation weights α .

6.2.5. Ablations on the Target Computation

As discussed in the preceding analysis of one-step returns, calculating the multi-step return remains crucial for stable learning but proves computationally expensive when evaluated across all interpolation weights α . To address this computational bottleneck, we conduct an ablation study where we intentionally disregard the resulting off-policy discrepancies. Specifically, we sample new α values at every timestep while computing the standard λ -return. This modification ensures that the immediate one-step return remains on-policy, although the subsequent bootstrapping steps inherently become off-policy. The empirical results for this modified target computation are presented in Figure 6.6.

Interestingly, this specific ablation yields a visible increase in overall performance. The midpoint of the learned subspace now performs on par with the EnsemblePQN baseline, marking a significant improvement over the original ConPQN architecture. Furthermore, the Standardized Ensemble of Two (SEO2) achieves slightly better results than the EnsemblePQN baseline itself. However, it is important to note that this comparison is not entirely equitable, given that our Bezier curve implementation utilizes three anchor points whereas the baseline ensemble relies on only two independently trained networks.

Despite these performance gains, analyzing the SEO2 curve indicates that the subspace still lacks substantial diversity. The absence of two distinct performance peaks at the extreme ends of the interpolation range suggests that the learned policies do not differ significantly from one another. To investigate this persistent lack of diversity further, subsequent

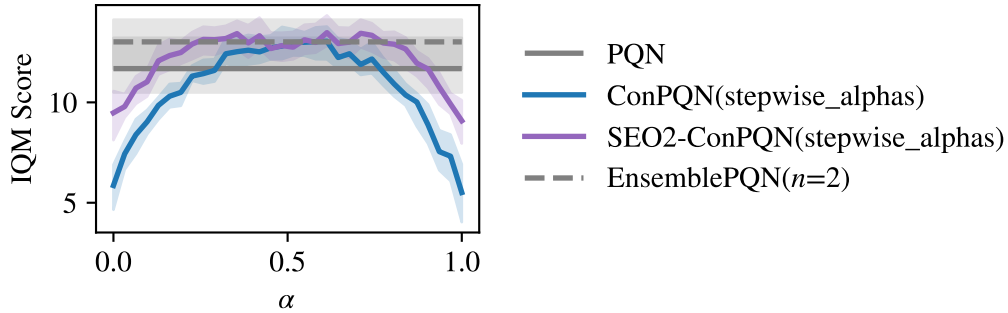


Figure 6.6.: IQM of human-normalized scores on 15 Atari games over 5 seeds, where shaded regions indicate confidence intervals. We compare ConPQN and SEO2-ConPQN utilizing a stepwise selection of α and a Bezier curve subspace against PQN and EnsemblePQN baselines across interpolation weights α .

analyses will examine the loss landscape of the subspace to evaluate its flatness and overall structural properties.

6.3. Analyzing the Landscape

To further understand these dynamics, we conduct a visual analysis of the loss landscape associated with the learned subspace. We generate a heatmap of this loss landscape by projecting the anchors of the subspace onto a two-dimensional plane. By evaluating the return of the greedy policy across this 2D plane, we can effectively visualize the performance landscape of the learned continuous subspace. Figure 6.7 illustrates this visualization method.

While the heatmaps appear visually distinct across different games, they all share a prominent common characteristic. Specifically, the landscapes are consistently very flat across the entire learned subspace. Most notably, the line connecting the endpoints remains connected with respect to the barrier depth metric. This pervasive flatness most likely explains the observed lack of diversity within the learned subspace and provides a clear rationale for the failure of the mean action and mean target aggregation approaches discussed previously.

Beyond this general flatness, we can make several other important game-specific observations. In environments such as Boxing, FishingDerby, and Pong, the subspace is notably flat

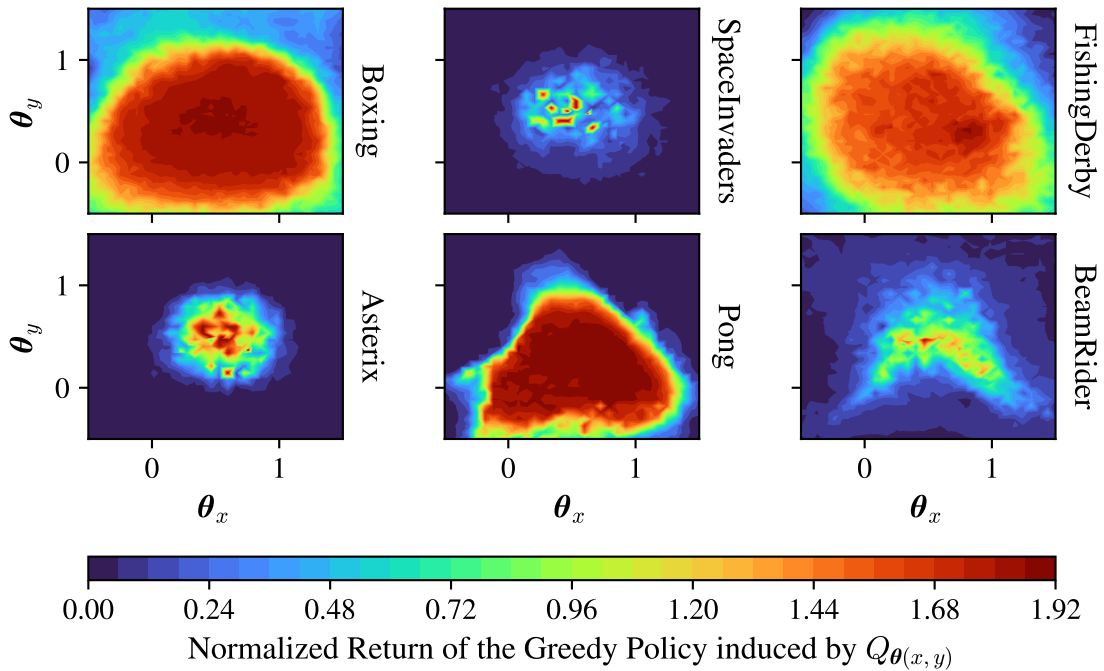


Figure 6.7.: 2D Q-function parameter subspaces trained with DQN across six different Atari games. The color of the heatmap corresponds to the normalized return of the greedy policy induced by the Q-function at that specific point in the subspace.

but maintains a relatively high and consistent return across the entire region. Conversely, on SpaceInvaders and Asterix, the high-performing subspace region is extremely narrow. In these environments, the return drops drastically as we move closer to the endpoints. Furthermore, on SpaceInvaders, this narrow subspace appears highly inconsistent and exhibits a significant amount of noise in the return values across the interpolation weights. This instability most likely accounts for the poor performance of both ConPQN and SEO2-ConPQN in these specific games. Individual results for all games are provided in Figure A.1.

Interestingly, the Beamrider environment offers hints of a Bezier curve shape within its subspace. However, this geometric structure is not prominently visible in the return landscape itself, which remains very flat across the entire area. Figure A.3 provides the full suite of heatmaps for all evaluated games.

Overall, this loss landscape analysis yields valuable insights into the structural properties of the learned subspace and its direct implications for policy diversity and performance across diverse environments. The inherent flatness of the landscape emerges as a key factor contributing to the observed lack of diversity and ultimately limits the effectiveness of strategies that rely on averaging multiple policies across the subspace.

6.4. Discussion

The empirical evaluation of the Connector Parallel Q-Networks (ConPQN) algorithm presents a nuanced view of subspace learning in value-based reinforcement learning. While our method successfully learns a continuous parameter subspace that maintains a connected policy, the resulting functional diversity within this space remains fundamentally limited. The performance of ConPQN generally matches the standard PQN baseline but struggles to realize the theoretical benefits of a true policy ensemble.

Our ablation studies and landscape analyses provide critical insights into the underlying causes of this limitation. The loss landscapes across the evaluated Atari environments exhibit a pervasive flatness. Because the parameter space allows for paths where the policy behavior remains largely unchanged, the networks naturally converge to solutions that satisfy the parameter diversity loss structurally without actually diversifying their functional behavior. This directly explains why leveraging the subspace for mean target computation or mean action selection failed to yield performance improvements. Averaging the outputs of functionally identical policies provides no variance reduction or ensemble advantage.

Furthermore, our experiments highlight a significant challenge regarding target stability. The ConPQN architecture proved highly sensitive to the removal of multi-step returns, which indicates that bootstrapping from a dynamically shifting subspace inherently destabilizes the learning process. Although the stepwise alpha sampling ablation offered a partial remedy and boosted performance, it did not solve the root issue of functional homogeneity. Similarly, substituting parameter similarity with feature similarity proved detrimental, emphasizing that naive regularization techniques are insufficient to force distinct behavioral modes in Q-learning.

Ultimately, these findings suggest that while learning continuous subspaces is mechanically feasible, simply regularizing parameter or feature similarity is inadequate for achieving true behavioral diversity. Future research must explore more sophisticated diversity


metrics, perhaps focusing directly on the state-action visitations or the specific action distributions of the individual policies. Developing regularizers that explicitly penalize functional correlation rather than just structural similarity will be crucial for unlocking the full potential of subspace learning in reinforcement learning environments.

We have already conducted preliminary studies exploring these advanced directions. To directly encourage functional diversity across the subspace, we implemented an explicit regularizer based on the Jensen-Shannon divergence between the Boltzmann policies induced by the Q-functions at varying interpolation weights α . Furthermore, we investigated training the structural anchor models separately to force a more significant initial divergence before establishing the connector. We also tried to generalize this control mechanism by sampling the interpolation weights from a Beta distribution, which allows us to dynamically widen or narrow the probability density to regularize the extent of exploration and diversity across the subspace. Ultimately, our preliminary findings indicate that while these targeted interventions do succeed in increasing the desired behavioral diversity, they concurrently render the overall learning process highly unstable.

7. Conclusion

This thesis investigated the optimization landscape of value-based reinforcement learning, specifically exploring if subspace learning could provide a computationally efficient alternative to traditional ensembles. Through our empirical analysis, we established that the Q-function landscape exhibits topological properties similar to those in supervised learning, confirming the presence of both flat maxima and mode connectivity. While we explored subspace learning as a method to generate continuous parameterizations, the practical benefits were not as significant as expected. Our experiments revealed that optimizing continuous subspaces tends to yield solutions trapped within uniformly flat minima, achieving parameter diversity without true functional or behavioral divergence. Consequently, learning subspaces does not inherently bridge distinct, high-performing modes. Ultimately, while we confirmed the mechanical feasibility of subspace learning in this context, unlocking its true ensemble potential will require advanced regularization techniques. Future work must investigate new methods to explicitly discover diverse mode-connected solutions with behavioral heterogeneity without sacrificing the computational efficiency that initially motivated this approach.

Building upon these findings, several promising avenues for future research emerge. First, a more rigorous investigation into the diversity within the learned subspaces is necessary, utilizing alternative behavioral and functional metrics beyond standard return landscapes to better quantify mode distinctness. Furthermore, future work could explore mechanisms to dynamically control this diversity during training. For instance, by parameterizing the sampling distribution, such as adapting the shape parameters of a Beta distribution, one could explicitly govern the structural breadth of the learned region. A wider distribution would facilitate the training of distant anchor points, encouraging greater diversity, whereas a narrower distribution would allow the subspace to collapse toward a refined and highly performing area. Finally, the concept of building a “model soup” [31] from diverse Q functions presents an intriguing direction. By averaging the weights of multiple models trained with varying RL specific hyperparameters, it may be possible to construct



a single robust Q network that leverages the strengths of multiple distinct modes without the inference overhead of a traditional ensemble.

Bibliography

- [1] Richard Y Chen et al. “Ucb exploration via q-ensembles”. In: *arXiv preprint arXiv:1706.01502* (2017).
- [2] Felix Draxler et al. “Essentially no barriers in neural network energy landscape”. In: *International conference on machine learning*. PMLR. 2018, pp. 1309–1318.
- [3] Utku Evci et al. “The Difficulty of Training Sparse Neural Networks”. In: *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*. 2019.
- [4] Stanislav Fort and Stanislaw Jastrzebski. “Large scale structure of neural network loss landscapes”. In: *Advances in Neural Information Processing Systems 32* (2019).
- [5] Stanislav Fort et al. “Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel”. In: *Advances in Neural Information Processing Systems 33* (2020), pp. 5850–5861.
- [6] Jonathan Frankle. “Revisiting” Qualitatively Characterizing Neural Network Optimization Problems”. In: *NeurIPS 2020 Workshop: Deep Learning through Information Geometry*. 2020.
- [7] Jonathan Frankle et al. “Linear mode connectivity and the lottery ticket hypothesis”. In: *International conference on machine learning*. PMLR. 2020, pp. 3259–3269.
- [8] C Daniel Freeman and Erik Frey. “Brax-A Differentiable Physics Engine for Large Scale Rigid Body Simulation”. In: (2021).
- [9] Matteo Gallici et al. “Simplifying Deep Temporal Difference Learning”. In: *The Thirteenth International Conference on Learning Representations*. 2025.
- [10] Timur Garipov et al. “Loss surfaces, mode connectivity, and fast ensembling of dnns”. In: *Advances in neural information processing systems 31* (2018).
- [11] Jean-Baptiste Gaya, Laure Soulier, and Ludovic Denoyer. “learning a Subspace of Policies for Online Adaptation in Reinforcement Learning”. In: *International Conference of Learning Representations (ICLR) 2022*. 2022.

-
-
- [12] Jean-Baptiste Gaya et al. “Building a Subspace of Policies for Scalable Continual Learning”. In: *International Conference of Learning Representations*. 2023.
- [13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: (2015).
- [14] Ahmed Hendawy et al. “It is All Connected: Multi-Task Reinforcement Learning via Mode Connectivity”. In: *Eighteenth European Workshop on Reinforcement Learning*.
- [15] Pavel Izmailov et al. “Averaging weights leads to wider optima and better generalization”. In: *arXiv preprint arXiv:1803.05407* (2018).
- [16] Qingfeng Lan et al. “Maxmin Q-learning: Controlling the Estimation Bias of Q-learning”. In: (2020).
- [17] Hao Li et al. “Visualizing the loss landscape of neural nets”. In: *Advances in neural information processing systems* 31 (2018).
- [18] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PmlR. 2016, pp. 1928–1937.
- [19] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.
- [20] Ian Osband et al. “Deep exploration via bootstrapped DQN”. In: *Advances in neural information processing systems* 29 (2016).
- [21] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [22] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347.
- [23] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.
- [24] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.
- [25] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.
- [26] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.

-
-
- [27] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.
 - [28] Marco A Wiering and Hado Van Hasselt. “Ensemble algorithms in reinforcement learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.4 (2008), pp. 930–936.
 - [29] Maciej Wołczyk et al. “Continual world: A robotic benchmark for continual reinforcement learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28496–28510.
 - [30] Mitchell Wortsman et al. “Learning neural network subspaces”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11217–11227.
 - [31] Mitchell Wortsman et al. “Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time”. In: *International conference on machine learning*. PMLR. 2022, pp. 23965–23998.
 - [32] Tianhe Yu et al. “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning”. In: *Conference on robot learning*. PMLR. 2020, pp. 1094–1100.

A. Appendix

This appendix provides supplementary visualizations and extended empirical results to support the findings presented in the main text. Specifically, we detail the individual performance trajectories of both PQN and ConPQN across the complete suite of evaluated Atari environments (Figure A.1). Furthermore, we illustrate the iterative training progress of the learned Bezier curve connectors between independently trained Q-functions on MinAtar (Figure A.2), and provide comprehensive 2D heatmaps detailing the return landscapes within the learned flat Q-function parameter subspaces across various Atari games (Figure A.3).

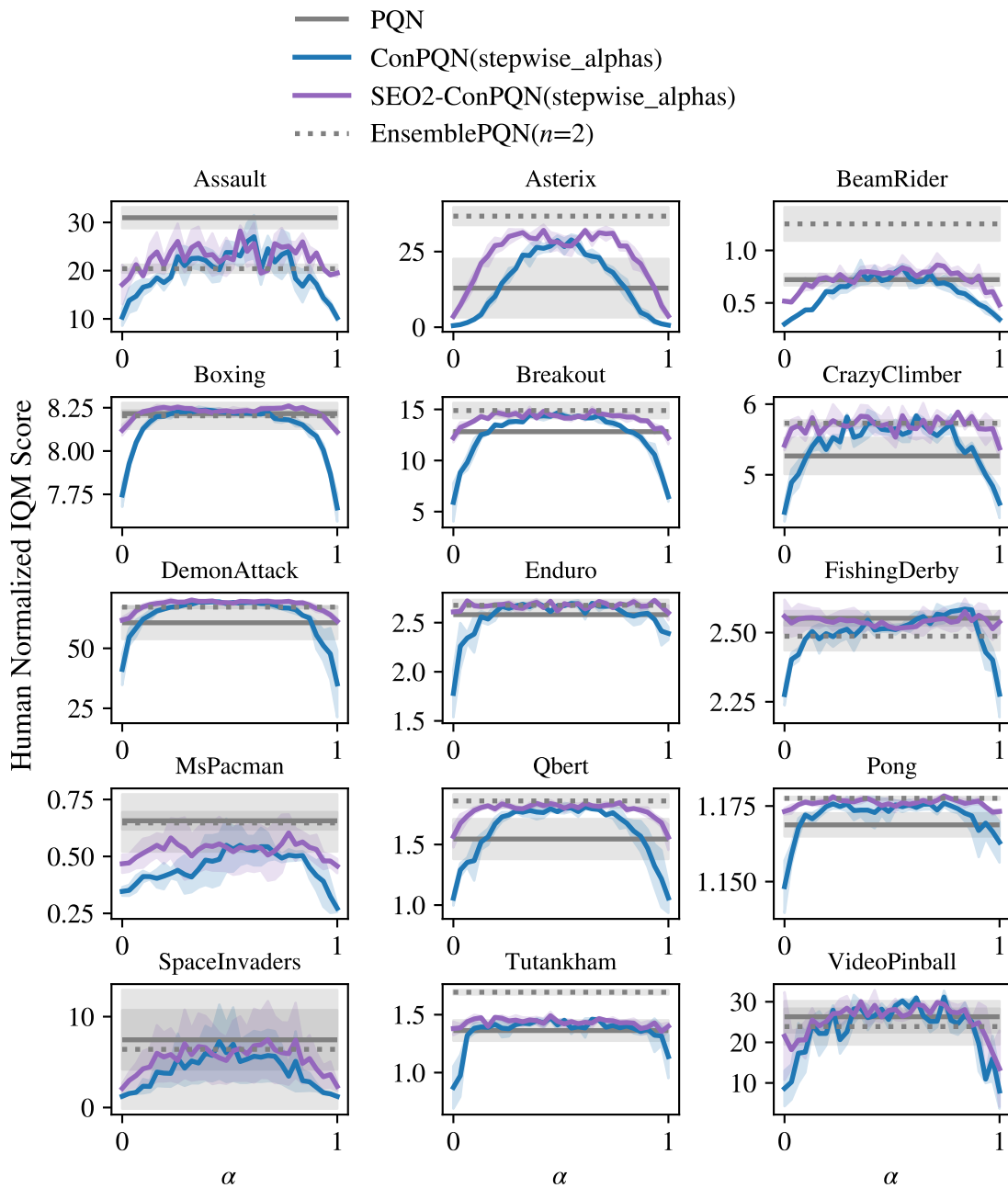


Figure A.1.: Individual performance of PQN and ConPQN using stepwise α sampling on 15 Atari games over 5 seeds. Shaded areas indicate standard deviation. All scores are human-normalized.

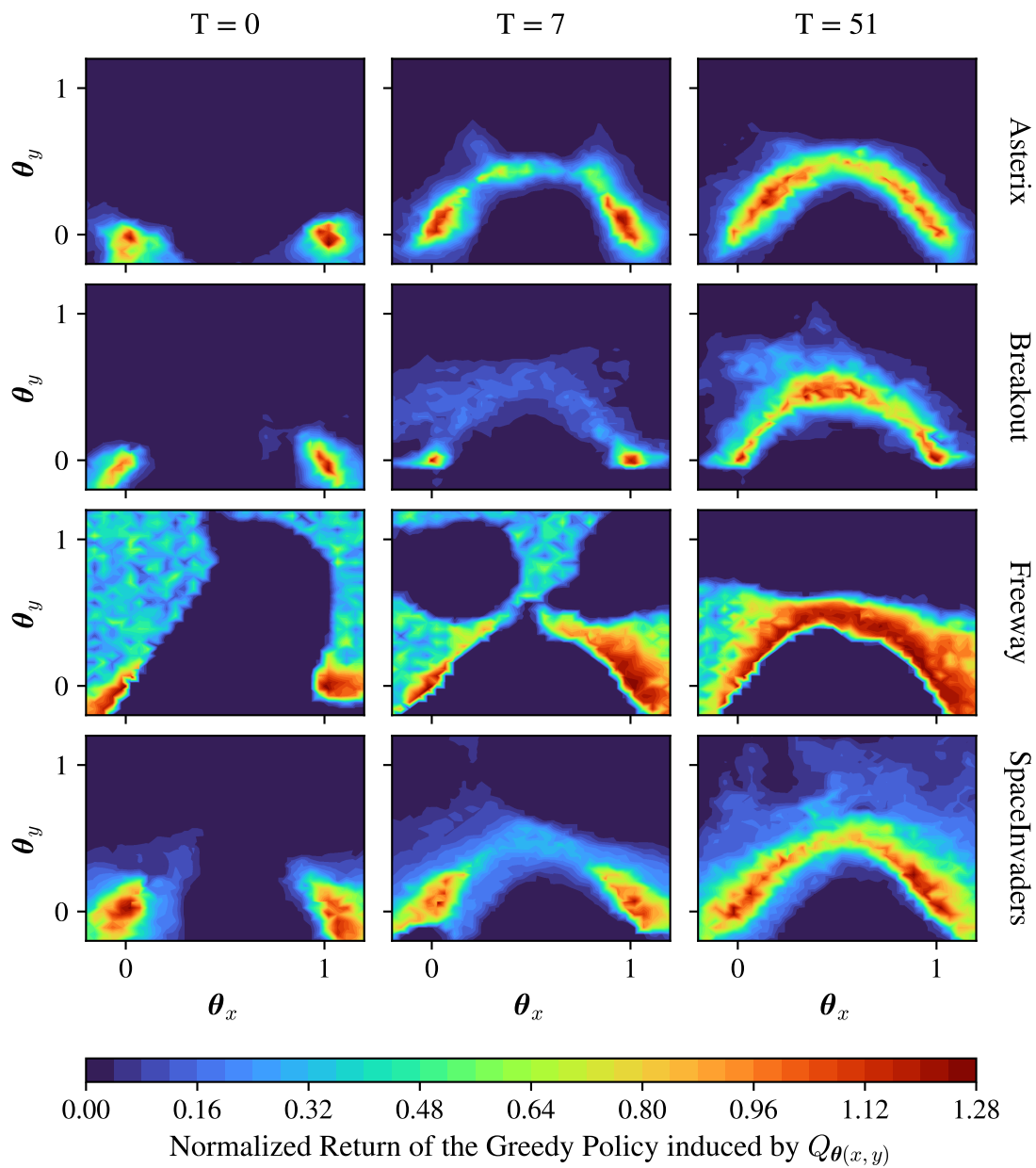


Figure A.2.: Connecting two independently trained Q-functions using a quadratic Bezier Curve on four MinAtar games. The plot shows the learned connector's performance along the curve, after 0, 7, and 50 iterations of training. The returns are normalized by the performance of the two endpoints.

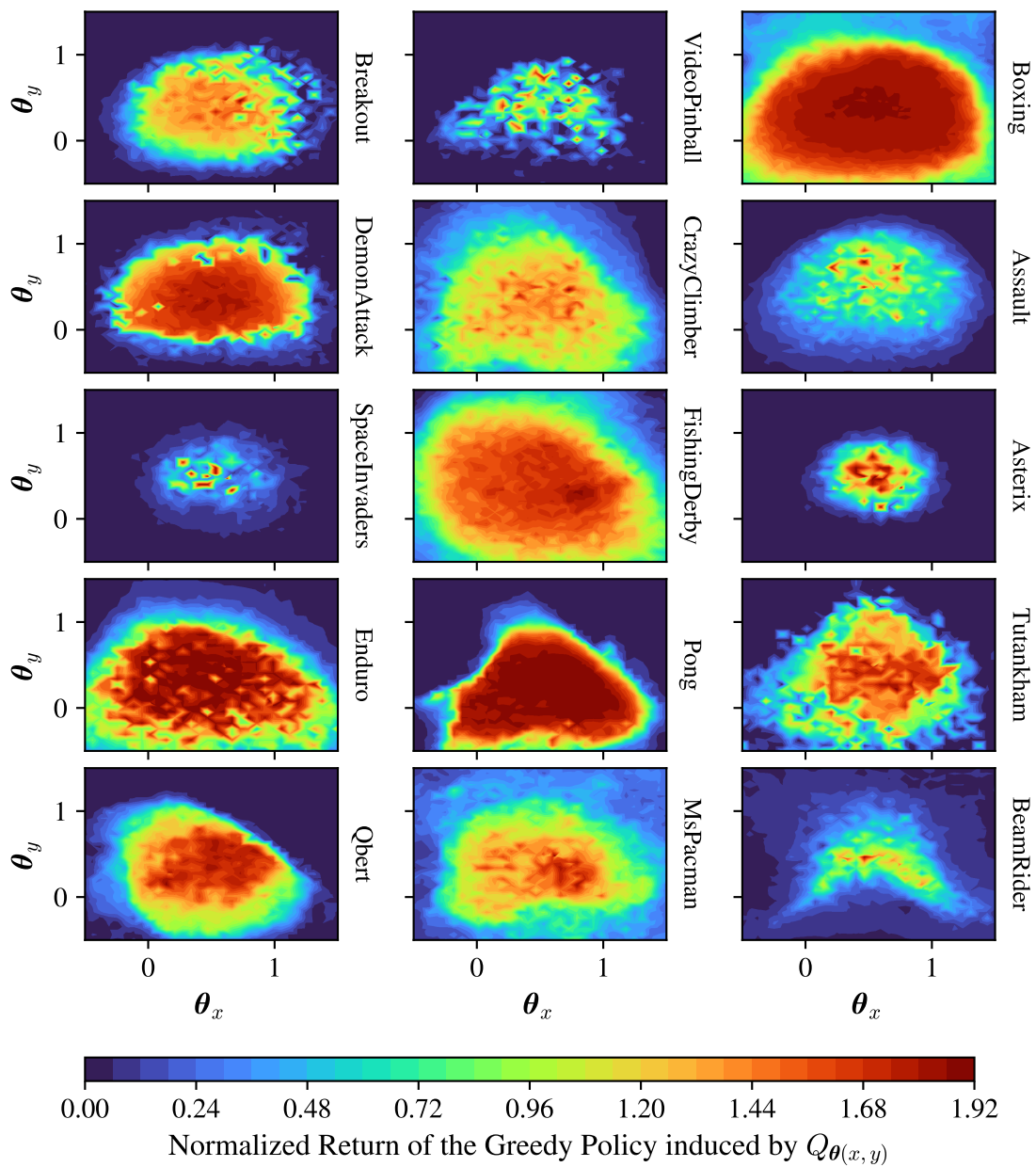


Figure A.3.: 2D Q-function parameter subspaces trained with DQN across 15 different Atari games. The color of the heatmap corresponds to the normalized return of the greedy policy induced by the Q-function at that specific point in the subspace.