VarShare: Principled Task-Specialization via Variational Weight Adapters in Multi-Task RL

Research Proposal

November 2025

Abstract

The primary challenge in Multi-Task Reinforcement Learning (MTRL) is the *Stability-Plasticity Dilemma*: agents must be plastic enough to learn task-specific nuances but stable enough to maintain and transfer shared dynamics across environments. Recent state-of-the-art approaches, such as those presented in the paper *Bigger, Regularized, Categorical (BRC)*, demonstrate that scaling model capacity can mitigate interference. However, these architectures typically rely on unconstrained task embeddings. We argue that this lack of constraint leads to inefficient transfer (over-specialization) and poor zero-shot initialization. To address this, we introduce **VarShare**. By modeling task-specific parameters as variational residual adapters in weight space, we utilize the Evidence Lower Bound (ELBO) to automatically arbitrate between shared knowledge and task-specific adaptation. We demonstrate analytically that this formulation acts as an adaptive regularizer that enforces sharing by default while permitting specialization only when the reward signal outweighs the information cost.

1 Introduction and Motivation

The goal of Multi-Task Reinforcement Learning (MTRL) is to train a single generalist agent capable of solving diverse tasks while leveraging shared structures to improve sample efficiency.

1.1 The Current State of the Art

Recent advancements, most notably the work *Bigger, Regularized, Categorical: High-Capacity Value Functions are Efficient Multi-Task Learners*, have shifted the focus from complex gradient projection algorithms to architectural scaling. The BRC agent uses massive residual networks (BroNet) and categorical cross-entropy loss to absorb the conflicting gradients inherent in MTRL.

1.2 The Limitation: Unconstrained Specialization

While effective, current high-capacity architectures typically condition the network on a learned task embedding vector z_m. These embeddings are updated end-to-end via backpropagation without explicit constraints. This leads to two critical failure modes:

1. **Redundant Parameterization (Inefficient Transfer):** Because the embeddings are unconstrained, two identical tasks (e.g., "Walk Fast" and "Walk Slow") may learn divergent embeddings. This causes the network to treat them as distinct problems, preventing data from one task from improving the representation of the other.
2. **Initialization Failure:** When a new, unseen task is introduced, it is typically initialized with a random embedding. This results in a random policy, failing to leverage the "average" competency the agent has already acquired.
3. **Gradient Interference:** Without a mechanism to dampen task-specific shifts, high-magnitude gradients from one task can distort the shared backbone, causing negative transfer to other tasks.

1.3 The Proposed Solution

We propose **VarShare**, a framework that formalizes the trade-off between sharing and specializing using **Variational Inference (VI)**. Inspired by the probabilistic principles found in the paper *Variational*

Continual Learning, we treat task-specific parameters not as deterministic values, but as random variables constrained by a shared prior. This introduces a "soft tie" that pulls all tasks toward a shared base model unless the data strictly demands specialization.

2 Intuition: The Adaptive Rubber Band

To understand the core mechanism of VarShare, we employ a physical analogy. Consider the relationship between the **Shared Base** (θ) and the **Task-Specific Parameters** (ϕ_m) for task m.

In VarShare, ϕ_m is not a free-floating point; it is connected to the origin (zero) by a "rubber band" (the KL-Divergence penalty). The Shared Base θ effectively acts as the reference point for all tasks.

- **The Anchor (Prior):** The system assumes by default that $\phi_m = 0$. In this state, the task relies 100% on the shared parameters θ.
- **The Force (Likelihood):** The RL loss (Temporal Difference error) exerts a "pull" on ϕ_m. It drags the parameters away from zero to minimize prediction error.
- **The Elasticity (Variance):** Unlike L2 regularization, which acts as a rigid steel spring, VarShare learns the *variance* σ_m. If the task is noisy or conflicts with the base, the agent can "loosen the spring" (increase σ_m) to absorb the tension without aggressively shifting the mean.

Mechanism: If tasks agree, the Shared Base θ moves to solve them. The RL "force" on ϕ_m vanishes, and the rubber band snaps ϕ_m back to zero (Automatic Sharing). If tasks disagree, θ gets stuck in the middle. The RL force on ϕ_m persists, stretching the band and allowing ϕ_m to settle at a new non-zero value (Necessary Specialization).

3 Methodology: Variational Weight-Space Adapters

We propose implementing VarShare using **Weight-Space Composition**. Instead of combining the outputs of two distinct networks, we inject task-specific probabilistic residuals directly into the weight matrices of the shared backbone.

3.1 Mathematical Formulation

Let $W \in \mathbb{R}^{d_{in} \times d_{out}}$ be a weight matrix in a layer of the shared backbone. For a specific task m, we define the effective weight matrix W_m as:

$$W_m = \theta + \phi_m \tag{1}$$

Where:

- θ: The deterministic shared parameters (Standard weights).
- ϕ_m: The stochastic task-specific residual parameters.

3.2 Probabilistic Definitions

We define a hierarchical structure to govern ϕ_m:

1. The Prior (Hypothesis of Similarity): Before seeing data for task m, we assume it requires no modification to the base. We place a centered Gaussian prior on ϕ_m. Crucially, this prior is **fixed** at zero relative to the base θ:

$$p(\phi_m) = \mathcal{N}(\mathbf{0}, \sigma_{prior}^2 \mathbf{I}) \tag{2}$$

This fixed prior creates the permanent "gravity" that pulls task parameters back to the shared base.

2. The Variational Posterior (The Learned Adaptation): We cannot calculate the true posterior analytically. Instead, we approximate it using a variational distribution q_m, parameterized by learnable tensors μ_m (mean) and ρ_m (parameterized variance):

$$q_m(\phi_m) = \mathcal{N}(\mu_m, \text{diag}(\sigma_m^2)) \quad \text{where } \sigma_m = \text{softplus}(\rho_m) \tag{3}$$

Here, μ_m and ρ_m are explicit parameters in the neural network, updated via backpropagation.

- μ_m: Represents the learned *shift* or specialization.
- σ_m: Represents the *uncertainty* or "looseness" of that specialization.

3.3 The Objective: Evidence Lower Bound (ELBO)

We maximize the ELBO, which balances task performance against the cost of specialization:

$$\mathcal{L}(\theta, \mu_m, \sigma_m) = \underbrace{\mathbb{E}_{\phi_m \sim q_m}[\log p(\mathcal{D}_m|W_m)]}_{\text{(A) RL Performance (Likelihood)}} - \beta \underbrace{D_{KL}(q_m(\phi_m)||p(\phi_m))}_{\text{(B) Complexity Cost (Regularizer)}} \tag{4}$$

- **Term A (Likelihood):** The standard Categorical Cross-Entropy loss used in BRC. It encourages μ_m to move to reduce TD error.
- **Term B (KL Divergence):** A closed-form penalty. Since the prior is centered at 0, this term minimizes μ_m^2 and penalizes deviations of σ_m from σ_{prior}.

To optimize Term A, we use the **Reparameterization Trick**. During the forward pass, we sample noise $\epsilon \sim \mathcal{N}(0, I)$ and compute the weights as:

$$W_{m,\text{sample}} = \theta + (\mu_m + \sigma_m \odot \epsilon)$$

4 Analysis: Why Sharing is Preferred

A critical requirement of VarShare is that the model should prefer updating the shared parameters over the task-specific ones when possible. We analyze the gradient flow to prove this system favors sharing.

4.1 Cost Asymmetry

Let J be the negative ELBO (the Loss we minimize).

- **Gradient w.r.t Shared Base θ:** The KL penalty depends only on ϕ_m, not θ.

  $$\nabla_\theta J = \nabla_\theta(\text{RL Loss}) + 0$$

  Updating θ is "free" in terms of complexity cost.

- **Gradient w.r.t Task Parameters μ_m:** The KL term depends quadratically on μ_m.

  $$\nabla_{\mu_m} J = \nabla_{\mu_m}(\text{RL Loss}) + \beta \frac{\mu_m}{\sigma_{prior}^2}$$

  Every step μ_m takes away from zero incurs a direct penalty gradient pushing it back.

4.2 Signal Amplification (The Crowd Effect)

In Multi-Task Learning, we update using batches containing data from all M tasks.

- The Shared Base θ receives gradients summed across **all** tasks: $\sum_{m=1}^{M} \nabla L_m$.
- The Task Parameter μ_m receives gradients only from task m: ∇L_m.

If tasks possess shared structure, the gradient on θ is M times stronger than the gradient on μ_m. Combined with the zero penalty for θ, the optimizer will move θ rapidly to solve the shared structure. Once θ solves the task, the gradient ∇L_m vanishes, and the KL penalty drives $\mu_m \to 0$. Specialization only occurs if the gradients on θ cancel out (conflict), leaving μ_m as the only way to reduce loss.

5 Comparison with L2 Regularization

It is important to distinguish VarShare from simple L2 weight decay ($\lambda||\phi||^2$). While both penalize the magnitude of weights, VarShare adapts the "stiffness" of the penalty via the learnable variance σ_m.

1. **Conflict Absorption:** If Task A wants $\phi = +5$ and Task B wants $\phi = -5$, L2 regularization forces $\phi \approx 0$, causing underfitting on both. VarShare can increase the variance σ_m. This allows the distribution to "cover" the necessary value without shifting the mean, signaling uncertainty rather than failure.
2. **Auto-Exploration:** Since ϕ_m is sampled during training ($W = \theta + \mu + \sigma\epsilon$), high uncertainty ($\sigma$) leads to diverse weight samples. This acts as parameter-space noise (similar to Thompson Sampling), driving exploration on difficult tasks automatically. L2 regularization is deterministic and lacks this property.

6 Implementation Proposal

We integrate VarShare into the standard residual architecture (BroNet) using a **Variational Residual Adapter**.

For every linear layer in the network defined by $y = \theta x + b$:

1. **Storage:** We store the shared matrix θ (size $D \times D$) and task-specific parameters $\mu_{1:M}, \rho_{1:M}$ (each size $D \times D$).
2. **Forward Pass (Task m):**
   - Compute shared output: $y_{shared} = x\theta^T$.
   - Compute variance: $\sigma_m = \log(1 + \exp(\rho_m))$.
   - Sample noise: $\epsilon \sim \mathcal{N}(0, I)$.
   - Compute specific weight: $\phi_{sample} = \mu_m + \sigma_m \odot \epsilon$.
   - Compute specific output: $y_{spec} = x\phi_{sample}^T$.
   - Final output: $y = y_{shared} + y_{spec} + b$.
3. **Backward Pass:** Gradients flow to θ, μ_m, ρ_m. The KL penalty is added to the loss based on μ_m and σ_m.

7 Refinements for Stability

To ensure stability in the noisy context of Reinforcement Learning, we adopt two specific technical refinements:

- **Flipout Estimator:** Standard sampling uses the same noise ϵ for an entire mini-batch, which causes high gradient variance. The "Flipout" estimator uses uncorrelated noise for each example in the batch efficiently, significantly stabilizing training.
- **Post-Training Pruning:** Since the KL term actively pushes unnecessary parameters to zero, we can apply a simple threshold post-training. If $|\mu_m| < \epsilon$, we set $\mu_m = 0$ and remove the parameter from memory. This results in a highly compressed model where tasks only store the sparse "diffs" from the shared base.