Bayesian Experience Reuse for Learning from Multiple Demonstrators

Michael Gimelfarb[*] , **Scott Sanner**[*] and **Chi-Guhn Lee**
Department of Mechanical and Industrial Engineering, University of Toronto
mike.gimelfarb@mail.utoronto.ca, ssanner@mie.utoronto.ca, cglee@mie.utoronto.ca

Abstract

Learning from Demonstrations (LfD) is a powerful approach for incorporating advice from experts in the form of demonstrations. However, demonstrations often come from multiple sub-optimal experts with conflicting goals, rendering them difficult to incorporate effectively in online settings. To address this, we formulate a quadratic program whose solution yields an adaptive weighting over experts, that can be used to sample experts with relevant goals. In order to compare different source and target task goals safely, we model their uncertainty using normal-inverse-gamma priors, whose posteriors are learned from demonstrations using Bayesian neural networks with a shared encoder. Our resulting approach, which we call Bayesian Experience Reuse, can be applied for LfD in static and dynamic decision-making settings. We demonstrate its effectiveness for minimizing multi-modal functions, and optimizing a high-dimensional supply chain with cost uncertainty, where it is also shown to improve upon the performance of the demonstrators' policies.

1 Introduction

Learning from demonstrations (LfD) is a powerful approach for incorporating advice from experts in the form of demonstrations to accelerate the learning of new skills. However, existing work in LfD often assume that demonstrations are generated from a single agent with a single goal [Argall *et al.*, 2009]. In practice, data can be available from multiple sub-optimal agents with conflicting goals. For example, when learning to operate a vehicle autonomously from demonstrators [Bojarski *et al.*, 2016], different drivers can have different goals (destinations), needs (safety) and experience levels. Relying on demonstrators whose goals are misaligned with the new target task can lead to unintended or dangerous behaviors, and can be minimized by actively learning to trust the most relevant demonstrators.

In this paper, we focus on LfD with multiple conflicting demonstrators for solving static and dynamic optimization problems. Following existing work [Gao *et al.*, 2018], demonstrations in our setting also contain immediate rewards, e.g. (s, a, r, s'), an instance of LfD referred to as *reinforcement learning from demonstrations* (RLfD). Our setting also differs from traditional work in LfD, in that the goal is to *improve upon* the demonstrator rather than simply mimic its behaviors. In order to measure the similarity between demonstrators' reward functions, we parameterize them as linear functions in a common feature space. Furthermore, we take a Bayesian approach by modeling their uncertainty using Normal-Inverse-Gamma priors. These quantities are modeled as Bayesian neural networks with a shared encoder, and trained end-to-end from demonstrations in an online manner. We then formulate a quadratic program whose solution yields a probability distribution over the demonstrators. This allows demonstrators to be sampled directly, while incorporating uncertainty in the estimates of their reward functions. Furthermore, being Bayesian allows us to avoid premature convergence, be more robust to non-stationary, sparse or limited data [Bishop, 2006], and trade off the mean and variance of the reward estimates in a principled way (Theorem 1).

In order to transfer demonstrations to new tasks in LfD, one approach is to pre-train the learner directly on the source data [Cruz Jr *et al.*, 2017], or learn and reuse auxiliary representations from the source data such as policies [Fernández *et al.*, 2010]. However, the former can be ineffective when demonstrations assume conflicting goals, while meaningful policies can be difficult to solicit from the latter when demonstrators are limited, sub-optimal or exploratory in nature [Nicolescu and Mataric, 2003]. On the other hand, experience collected from a failed or inexperienced demonstrator can be just as valuable as an experienced one [Grollman and Billard, 2011]. We present an algorithm called *Bayesian Experience Reuse* (BERS), for directly reusing multiple demonstrations in a way that is consistent with the learned weighting over the source and target task goals (Algorithm 1). While tailored for LfD, our approach is quite general and can be applied in other areas such as multi-task learning.

2 Background

2.1 Reinforcement Learning

Decision-making in this paper can be summarized in a *Markov decision process* (MDP), formally defined as a five-

[*] Affiliate to Vector Institute, Toronto, Canada.

tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where: $\mathcal{S}$ is a set of states, $\mathcal{A}(s)$ is a set of possible actions in state s, $P(\cdot|s,a)$ gives the next-state s' distribution upon taking action a in state s, $R(s,a,s')$ is the corresponding reward, and $\gamma \in [0,1]$ is a discount factor. The objective of an agent is to find a policy $\mu : \mathcal{S} \to \mathcal{A}$ that maximizes the long-run expected return $Q^\mu(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \,|\, s_0 = s, a_0 = a]$, where $a_t = \mu(s_t)$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$.

In the *reinforcement learning* (RL) setting, neither P nor R are known. Instead, the agent interacts with the environment using a randomized exploration policy μ^e, collecting rewards and observing state transitions. In order to learn optimal policies, *temporal difference* methods first learn $Q(s,a)$ and use it to recover an optimal policy, while *policy gradient* methods parameterize and recover an optimal policy μ^* directly. *Actor-critic* methods learn a critic $Q(s,a)$ and actor policy $\mu(s)$ simultaneously [Sutton and Barto, 2018].

2.2 Common Feature Representations

In our problem setting, each *task* is associated with an unknown function $y : \mathcal{X} \to \mathbb{R}$ on some domain $\mathcal{X}$. In the RL setting for example, $y(\mathbf{x}) = R(s,a,s')$ are reward functions. Given a feature map $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^d$, a function y can be expressed as a linear combination $y(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$, $\forall x \in \mathcal{X}$, where $\mathbf{w} \in \mathbb{R}^d$ is a fixed vector.

We are interested in the problem of transferring demonstrations (s,a,r,s') between tasks in a domain $\mathcal{M}^{\boldsymbol{\phi}}$ on a common $\mathcal{X}$,

$$\mathcal{M}^{\boldsymbol{\phi}} = \left\{ y \,:\, \exists \mathbf{w} \in \mathbb{R}^d \text{ s.t. } y(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w},\ \forall \mathbf{x} \in \mathcal{X} \right\}. \tag{1}$$

In the RL setting, $\mathcal{M}^{\boldsymbol{\phi}}$ could include all MDPs with shared dynamics and different rewards R_i. In (1), we have explicitly assumed that the (unknown) state features $\boldsymbol{\phi}$ are shared among tasks. This is not a restrictive assumption in practice, as given a set of tasks $T_1, T_2 \dots T_n \in \mathcal{M}^{\boldsymbol{\phi}}$, we may trivially define $\phi_k(\mathbf{x}) = y_{T_k}(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{X}$ for each $k = 1, 2 \dots n$. In practice, however, different rewards may share common features. Pooling different sets of basis functions into a common basis in this way will also allow us to represent, and thus compare, conflicting goals consistently. The challenge is to learn suitable common embeddings $\boldsymbol{\phi}$ and posterior distributions for $y(\mathbf{x})$, and leverage them for measuring task similarity.

3 Bayesian Experience Reuse

The agent is presented with sets of demonstrations $\mathcal{D}_1, \mathcal{D}_2, \dots \mathcal{D}_N$ sampled from respective source tasks $T_1, T_2, \dots T_N \in \mathcal{M}^{\boldsymbol{\phi}}$, that are represented as collections of labeled pairs $(\mathbf{x}_t, y_t)$. The agent would like to leverage these demonstrations to solve a new task, $T_{target} \in \mathcal{M}^{\boldsymbol{\phi}}$, for which a limited but gradually growing set of demonstrations $\mathcal{D}_{target}$ is available.

In order to make optimal use of the source tasks during training, the agent should learn to favor demonstrators whose underlying reward representation is *closest* to the target task reward. By *actively* learning to trust relevant demonstrators and avoiding the irrelevant ones, an agent can maximize the benefit associated with pre-training on the corresponding demonstrations. Furthermore, reward representations are more natural and more robust for measuring task similarity than discounted value functions, because values are more sensitive to small changes in rewards. For instance, given a constant perturbation in rewards of ΔR, the corresponding value function will change by $\frac{\Delta R}{1-\gamma}$ that is considerably greater than ΔR in the non-myopic setting.

3.1 Bayesian Regression with Common Features

In order to facilitate the learning of reward representations, we learn a shared feature space $\boldsymbol{\phi}$, together with Bayesian regressions $\mathbb{P}(\mathbf{w}_i|\mathcal{D}_i)$ and $\mathbb{P}(\mathbf{w}_{target}|\mathcal{D}_{target})$ for the source and target tasks respectively, such that $y_t^i \approx \boldsymbol{\phi}(\mathbf{x}_t^i)^\top \mathbf{w}_i$ for all i and $(\mathbf{x}_t^i, y_t^i) \in \mathcal{D}_i$ and $y_t^{target} \approx \boldsymbol{\phi}(\mathbf{x}_t^{target})^\top \mathbf{w}_{target}$ for all $(\mathbf{x}_t^{target}, y_t^{target}) \in \mathcal{D}_{target}$. As we will show, the shared feature representation $\boldsymbol{\phi}$ is critical in order to allow meaningful comparisons between source $\mathbf{w}_i$ and target $\mathbf{w}$.

In order to tractably learn the features $\boldsymbol{\phi}$ as well as the corresponding posterior distributions, we parameterize $\boldsymbol{\phi}(\mathbf{x}) \approx \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x})$ using a deep neural network (encoder) with weight parameters $\boldsymbol{\theta}$, and model $\mathbf{w}_1, \dots \mathbf{w}_N, \mathbf{w}_{target}$ using the *normal-inverse-gamma* conjugate prior:

$$\begin{aligned} y_i(\mathbf{x}) &= \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x})^\top \mathbf{w}_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_i^2) \\ \mathbf{w}_i &\sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 \boldsymbol{\Lambda}_i^{-1}), \quad \sigma_i^2 \sim \text{InvGamma}(\alpha_i, \beta_i). \end{aligned} \tag{2}$$

We note that a Gaussian prior on the rewards is quite reasonable, and has been successfully applied in other areas such as exploration [Janz *et al.*, 2019]. The joint posterior $\mathbb{P}(\mathbf{w}_i, \sigma_i^2|\mathcal{D}_i)$ now factors as

$$\mathbb{P}(\mathbf{w}_i, \sigma_i^2|\mathcal{D}_i) \propto \mathbb{P}(\mathbf{w}_i|\sigma_i^2, \mathcal{D}_i)\, \mathbb{P}(\sigma_i^2|\mathcal{D}_i), \tag{3}$$

where $\mathbf{w}_i|\sigma_i^2, \mathcal{D}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2 \boldsymbol{\Lambda}_i^{-1})$ and $\sigma_i^2|\mathcal{D}_i \sim \text{InvGamma}(\alpha_i, \beta_i)$, where:

$$\begin{aligned} \boldsymbol{\Lambda}_i &= \boldsymbol{\Lambda}_i^0 + \boldsymbol{\Phi}_i^\top \boldsymbol{\Phi}_i, \quad \boldsymbol{\mu}_i = \boldsymbol{\Lambda}_i^{-1} \left(\boldsymbol{\Lambda}_i^0 \boldsymbol{\mu}_i^0 + \boldsymbol{\Phi}_i^\top \mathbf{y}_i \right), \\ \alpha_i &= \alpha_i^0 + \frac{1}{2}|\mathcal{D}_i|, \\ \beta_i &= \beta_i^0 + \frac{1}{2} \left(\mathbf{y}_i^\top \mathbf{y}_i + (\boldsymbol{\mu}_i^0)^\top \boldsymbol{\Lambda}_i^0 \boldsymbol{\mu}_i^0 - \boldsymbol{\mu}_i^\top \boldsymbol{\Lambda}_i \boldsymbol{\mu}_i \right), \end{aligned} \tag{4}$$

and where $\boldsymbol{\Phi}_i$ is the matrix of state features $\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}_t^i)$ and $\mathbf{y}_i$ is the vector of observations y_t^i in $\mathcal{D}_i$ [Bishop, 2006]. We have also assumed that, conditioned on data $\mathcal{D}_i$, the weights $\mathbf{w}_i$ and variances σ_i^2 are mutually independent between tasks, a very mild assumption in practice. Adapting the *neural-linear* approach of [Ober and Rasmussen, 2019; Snoek *et al.*, 2015], we update $\boldsymbol{\theta}$ by gradient ascent on the *marginal log-likelihood* function for each head i,

$$\begin{aligned} \log \mathbb{P}(\mathbf{y}_i|\mathcal{D}_i) &= |\mathcal{D}_i| \pi + \log \Gamma(\alpha_i^0) - a_i^0 \log \beta_i^0 \\ &\quad + \frac{1}{2} \log \det \boldsymbol{\Lambda}_i^0 - \log \Gamma(\alpha_i) + \alpha_i \log \beta_i - \frac{1}{2} \log \det \boldsymbol{\Lambda}_i, \end{aligned} \tag{5}$$

where the key quantities are provided by (4) and depend implicitly on $\boldsymbol{\theta}$ through $\boldsymbol{\Phi}_i$.

Now, parameter sharing allows $\boldsymbol{\phi}$ to be learned, refined and transferred seamlessly from source to target tasks, and provides a form of transfer in its own right. However, our main contribution is to use the posterior distributions over experts' goals, $\mathbf{w}_i$ and $\mathbf{w}_{target}$, to transfer demonstrations.

3.2 Expert Selection via Quadratic Programming

In order to derive a Bayesian decision rule for source task selection, we first observe that source tasks that are most similar to the target task — and hence those that lead to better transfer — should have $\mathbf{w}_i$ closest to the true target $\mathbf{w}_{target}$. In our setting, we instead have uncertain estimates for $\mathbf{w}_i$ and $\mathbf{w}_{target}$ modelled as random variables. We therefore look for a weighting $\sum_{i=1}^{N} a_i \mathbf{w}_i$ that is *closest* to $\mathbf{w}_{target}$, while factoring in the uncertainty in these estimates.

More specifically, suppose that the posterior distributions of $\mathbf{w}_i$ for each $i = 1, 2 \dots N$ and $\mathbf{w}_{target}$ have been estimated from past data. Our goal is to weight the $\mathbf{w}_i$ in such a way that the weighted sum, $\sum_i a_i \mathbf{w}_i$, is closest to $\mathbf{w}_{target}$ in expectation. In other words, we seek $\mathbf{a}$ that minimizes the following optimization problem:

$$\min_{\mathbf{a} \in \mathcal{P}} \mathcal{L}(\mathbf{a}) = \mathbb{E}\left[\|\mathbf{w}_{target} - \sum_{i=1}^{N} a_i \mathbf{w}_i\|_2^2 \,\middle|\, \mathscr{D} \right], \qquad (6)$$

where $\mathscr{D} = \bigcup_i \mathcal{D}_i \cup \mathcal{D}_{target}$ is the union of all source and target demonstrations, and $\mathcal{P}$ is a convex polyhedron. Specifically, our goal is to sample source tasks according to $\mathbf{a}$, so we restrict $\mathbf{a}$ to a discrete probability distribution by setting $\mathcal{P} = \{\mathbf{a} \in \mathbb{R}^N : \mathbf{1}^\top \mathbf{a} = 1, \mathbf{a} \geq \mathbf{0}\}$. In other applications, such as regression problems [Pardoe and Stone, 2010], we may set $\mathcal{P} = \mathbb{R}^N$, or incorporate other constraints on expert selection depending on the problem.

The following result will facilitate the optimization of (6) under our previous assumptions.

Theorem 1. *Suppose $\mathbf{w}_1 \dots \mathbf{w}_N$ and $\mathbf{w}_{target}$ are defined by the posterior (3). Then, for every $\mathbf{a} \in \mathbb{R}^N$,*

$$\mathcal{L}(\mathbf{a}) \propto \|\boldsymbol{\mu}_{target} - \sum_{i=1}^{N} a_i \boldsymbol{\mu}_i\|_2^2 + \sum_{i=1}^{N} a_i^2 \left(\frac{\beta_i}{\alpha_i - 1} \right) \mathrm{tr}(\boldsymbol{\Sigma}_i), \qquad (7)$$

with equivalence up to terms constant in $\mathbf{a}$.

Hence, we have shown that *optimizing the expected error (6) is equivalent to optimizing the error in the posterior means plus a penalty equal to the product of the noise and posterior variances*. The penalty term prevents the posterior $\mathbf{a}$ from concentrating all its probability mass on a single demonstrator, whose benefit is demonstrated experimentally.

To simplify (7) further, we define $\mathbf{M} = [\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_N] \in \mathbb{R}^{d \times N}$ and $\mathbf{S} \in \mathbb{R}^{N \times N}$ the diagonal matrix with entries $\frac{\beta_i}{\alpha_i - 1} \mathrm{tr}(\boldsymbol{\Sigma}_i)$, $i = 1, 2 \dots N$. Rewriting (7) in this new notation and invoking Theorem 1, we obtain the following *quadratic program* (QP):

$$\begin{aligned} \min_{\mathbf{a}} \quad & -\boldsymbol{\mu}^\top \mathbf{M}\mathbf{a} + \frac{1}{2}\mathbf{a}^\top (\mathbf{M}^\top \mathbf{M} + \mathbf{S})\mathbf{a} \\ \text{subject to} \quad & \mathbf{1}^\top \mathbf{a} = 1, \quad \mathbf{a} \geq \mathbf{0}. \end{aligned} \qquad (8)$$

Here, $\mathbf{M}^\top \mathbf{M} + \mathbf{S}$ is positive definite, since it is the sum of the positive semi-definite matrix $\mathbf{M}^\top \mathbf{M}$ and the positive definite matrix $\mathbf{S}$. Hence, the above QP can be formulated and solved exactly using an off-the-shelf solver in polynomial time in N, independent of the dimension d and the number of demonstrations $|\mathcal{D}_i|$. Hence, *(8) remains tractable when the domain complexity is high or the number of demonstrations is large*. This is no longer the case when the second order term $\mathbf{S}$ is omitted, since (8) can become rank-deficient and lack a unique solution.

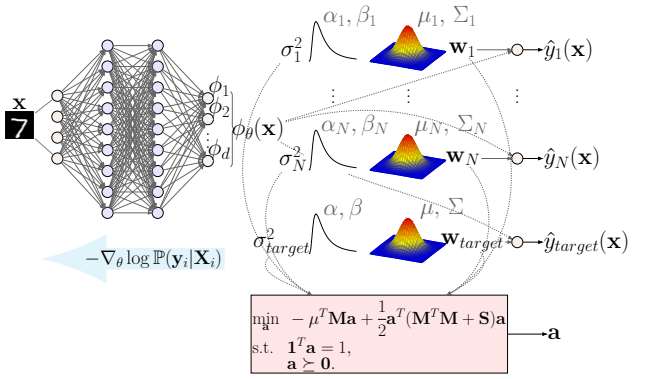


Figure 1: Bayesian multi-headed neural-linear model with shared encoder (MLP) and aggregated QP decision layer.

In the case of very large N, warm starts could be effective since the posterior changes smoothly over time (as we demonstrate experimentally), to use neural networks [Amos and Kolter, 2017], or optimize (8) directly via gradient descent. Our framework is agnostic to how (8) is solved, so we leave these investigations for future work.

The full architecture is summarized conceptually in Figure 1. Here, Bayesian heads with parameters $\{(\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i, \alpha_i, \beta_i)\}_{i=1}^N$ and $(\boldsymbol{\mu}, \boldsymbol{\Lambda}, \alpha, \beta)$ are maintained for source and target tasks respectively, while sharing the encoder parameters $\boldsymbol{\theta}$. Periodically, these estimates are used to construct and solve (8). The outputs $\hat{y}_i$ and $\hat{y}_{target}$ can also be used for making predictions, such as in regression problems. In this paper, our goal instead is to use the posterior distributions over task goals, and the corresponding QP solution, to rank and transfer the most relevant source demonstrations.

3.3 Bayesian Experience Reuse

A simple, yet effective, approach for transferring demonstrations from a single source is to pre-train the learning agent on the demonstrations [Cruz Jr *et al.*, 2017]. However, when data originates from multiple demonstrators with differing goals, some interaction or prior knowledge about the target environment is necessary in order to determine which data to use for pre-training. Without assuming any prior knowledge about the target environment, we train the agent on the source data in an offline manner while concurrently learning the target task online. Thus, the source demonstrations provide an effective exploration bonus, by enriching the agent's training data with novel experiences that might otherwise never be observed in the target task.

More specifically, in each episode of target task learning m, we sample a source task $T_i \in \mathcal{M}^\Phi$ according to $\mathbf{a}$ obtained from (8), and train the agent on experiences drawn from the corresponding data $\mathcal{D}_i$. In order for the target agent to improve beyond the demonstrator and generalize correctly to the target task, the agent must eventually learn from target demonstrations rather than source data. So, we adapt the sample annealing idea in [Fernández *et al.*, 2010], by grad-

Algorithm 1 Bayesian Experience Reuse (BERS)

Require: $\{\mathcal{D}_i\}_{i=1}^{N}$, $T_{N+1} = T_{target} \in \mathcal{M}^{\phi}$, O_{base}, $\mathcal{D}_{N+1} = \mathcal{D}_{target} = \emptyset$, $\boldsymbol{\theta}$, $\{\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i, \alpha_i, \beta_i\}_{i=1}^{N+1}$, p_m, $\mathbf{a}$
  pre-train $\boldsymbol{\theta}$, $\{\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i, \alpha_i, \beta_i\}_{i=1}^{N}$ on $\{\mathcal{D}_i\}_{i=1}^{N}$ using (4), (5)
  for episode $m = 1, 2, \ldots$ **do**
    for step $t = 1, 2, \ldots T$ of episode m **do**
      explore T_{target} using O_{base} and collect $d = (\mathbf{x}, y)$
      $\mathcal{D}_{target} = \mathcal{D}_{target} \cup d$
      sample $\text{train\_on\_source\_data} \sim \text{Bernoulli}(p_m)$
      if $\text{train\_on\_source\_data} = \text{true}$ **then**
        sample $i_t \sim \mathbf{a}$ and experience $\mathcal{B} \subset \mathcal{D}_{i_t}$
      else
        sample experience $\mathcal{B} \subset \mathcal{D}_{target}$
      end if
      train O_{base} on $\mathcal{B}$
    end for
    train $\boldsymbol{\theta}$, $\{\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i, \alpha_i, \beta_i\}_{i=1}^{N+1}$ on $\{\mathcal{D}_i\}_{i=1}^{N+1}$ using (5)
    solve QP (8) to obtain the solution $\mathbf{a}$
  end for
  return O_{base}

ually decreasing the fraction of time $p_m \in [0, 1]$ that the agent trains on source data. The resulting approach, which we call *Bayesian Experience Reuse* (BERS), is outlined in Algorithm 1.

In particular, we define O_{base} as a learning algorithm for solving the target task, assumed to be a static or dynamic optimization problem in this work. Hence, O_{base} is either a static optimization algorithm or an reinforcement learning algorithm. We first pre-train the N source heads on the source demonstrations to learn ϕ and the posteriors for $\mathbf{w}_1 \ldots \mathbf{w}_N$. In each episode, we explore the target task and collect data, putting them in $\mathcal{D}_{target}$. At each time step, the agent either trains on a batch of source demonstrations from task $i \sim \mathbf{a}$ with probability p_m, or a batch of target demonstrations with probability $1 - p_m$. At the end of each episode, we refine ϕ and the posterior distributions for all tasks and recompute $\mathbf{a}$. With simple modifications, BERS can be applied in multi-task settings by maintaining a separate QP solution per task.

4 Empirical Evaluation

In order to demonstrate the effectiveness of BERS, we consider two problems: (1) the search for the minimum of static but high-dimensional multi-modal functions, and (2) the dynamic control of a complex supply chain network with stochastic demand[1].

4.1 Static Optimization of Multi-Modal Functions

We first consider the problem of finding the minimum of a smooth but highly complex multi-modal function. LfD can be useful in this setting because the known solution of one function can be used as an initial "guess" when starting the search for the minimum of another similar function.

[1]The appendix can be found at https://github.com/mike-gimelfarb/bayesian-experience-reuse.

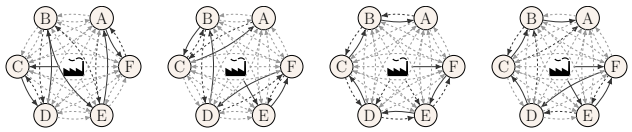


Figure 2: Supply chain source and target task configurations.

More specifically, we use the 10-dimensional Rosenbrock, Ackley and sphere functions as source tasks, and the Rastrigin function as the target task (please see appendix for definitions and processing). We also consider the simpler setting in which one of the source functions is the ground truth. As the base learning agent O_{base}, we use Differential Evolution (DE) (please see appendix for details)[2]. The search is limited to $x_i \in [-4, 4]$ for all $i = 1, 2 \ldots 10$. The global minimums of the functions are: $x^*_{Rosenbrock} = \mathbf{1}$, $x^*_{Ackley} = \mathbf{0}$, $x^*_{Sphere} = -\mathbf{2}$ and $x^*_{Rastrigin} = -\mathbf{2}$. Since the sphere and Rastrigin functions are locally similar around the minimum point, a successful LfD experiment should exploit the structure of the functions when optimizing the Rastrigin function. We also consider the multi-task setting by optimizing the Rastrigin function simultaneously with the other source functions, to demonstrate the versatility of BERS. In the latter setting, we maintain a separate QP solution per task. In both cases, the best solution found for each source task to date is transferred directly to O_{base} (the appendix details how the solutions are transferred).

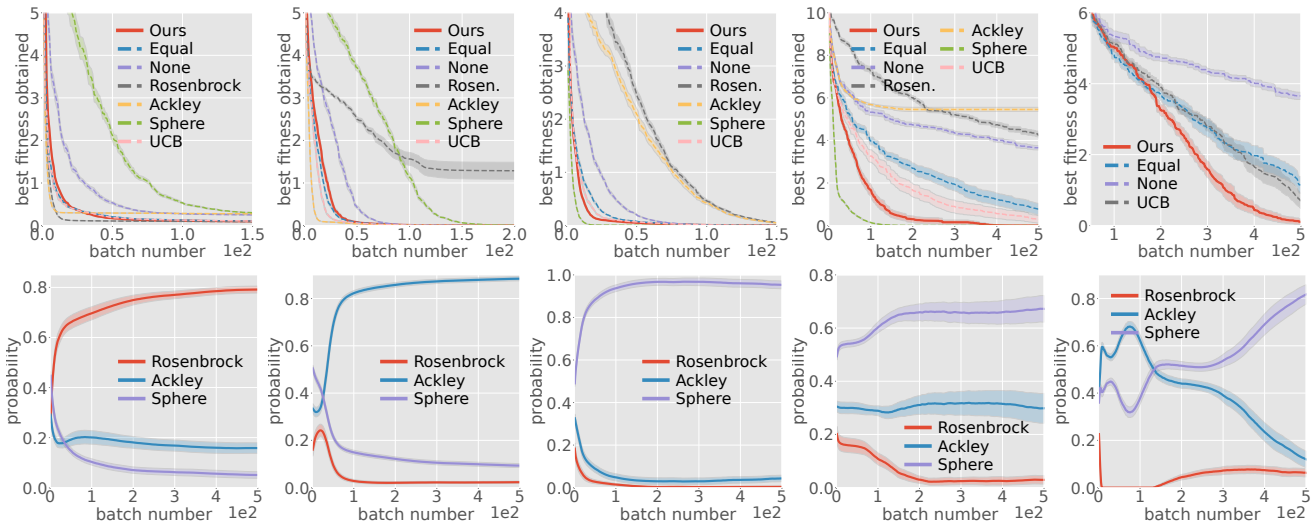
We consider the following set of baselines in order to compare the performance of BERS (**Ours**): (1) the UCB algorithm [Auer, 2002] with asymptotically optimal convergence (**UCB**), in which the reward is the improvement in the function value after transferring a solution from one of the source functions, (2) the equally-weighted prior $\mathbf{a} = [\frac{1}{N}, \ldots \frac{1}{N}]$ (**Equal**), (3) individual demonstrators (**S1**, **S2**…), and (4) standard DE without transfer (**None**). Figure 3 illustrates the function value of the best solution found to date, and the value of $\mathbf{a}$, after each iteration.

4.2 Dynamic Control of a Supply Chain

A supply chain network for the production and distribution of a single product consists of a central factory and $K = 6$ warehouses, denoted A, B …F. The factory can manufacture up to 35 units of inventory per day, and the factory and the warehouses can each store up to 50 units of inventory at any given time. A very large fleet of trucks is available to move inventory between points in the network. Each truck can deliver up to 4 units of inventory between any two points in the network, and takes a single day regardless of location.

Demand for each warehouse A, B …F, in units per day, is Poisson-distributed with respective means $\{7, 6, 6, 5, 5, 5\}$. Demand that cannot be fulfilled is lost forever. The selling price per unit of inventory is 0.6, the production cost is 0.1,

[2]Another option is to use Bayesian optimization (BO). However, this is more suitable for expensive functions with relatively low numbers of local optima, such as for hyper-parameter optimization.



(a) Rosenbrock (b) Ackley (c) Sphere (d) Rastrigin (T) (e) Rastrigin (MT)

Figure 3: Best function values (top row) and weights $\mathbf{a}$ (bottom row) in the transfer (T) and multi-task (MT) learning settings for static function optimization, with each source and target task as ground truth. Averaged over 20 trials with shaded standard error bars.

and the storage cost per unit per day is 0.03 for the factory and each warehouse. However, the cost of dispatching a truck is not fixed, but depends on the source and destination node.

There are two kind of routes: *cheap* routes are easy to navigate and incur a cost of 0.03 per truck, whereas *expensive* routes have difficult terrain and tariffs and cost 1.50 or 3.00, depending on source or destination. The company does not know the cost of each route in advance, but has identified three likely scenarios, summarized in Figure 2. Here, cheap routes are indicated with lighter arrows, while more expensive routes are indicated with darker arrows. As before, we will take the company's estimates as source tasks *and* as the ground truth, and also consider the setting where the target task is different from any of the source tasks. Please note that this problem is quite similar to the one in [Kemmer *et al.*, 2018], but our version is considerably more challenging.

We solve this problem using reinforcement learning, where the state is the current stock in the factory and warehouses, and actions are modelled as follows: (1) one continuous action for production as a proportion of the maximum; (2) $K+1$ actions for proportions of factory stock to ship to each warehouse (including to keep at the factory); and (3) K actions per warehouse, for proportions of warehouse stocks to ship to all other warehouses (including itself). This leads to a $2 + K + K^2 = 44$-dimensional continuous action space. In order to tractably solve this problem, we use the actor-critic algorithm DDPG [Lillicrap *et al.*, 2016] as O_{base} (further details are provided in the appendix).

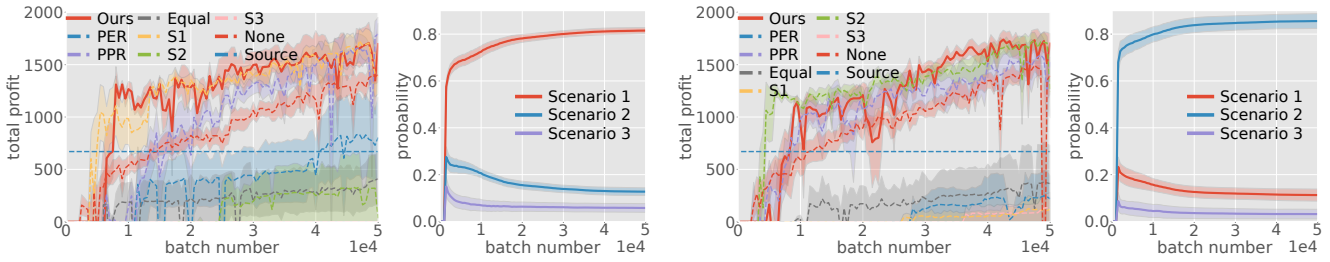
We evaluate BERS (**Ours**) against: (1) prioritized experience replay initialized with demonstrations from all source tasks [Hou *et al.*, 2017] (**PER**), and (2) a state-of-the-art policy reuse algorithm [Li and Zhang, 2018] (**PPR**). In the latter choice, a source policy is trained using the same architecture as the actor network for DDPG for 50 epochs using the cross-entropy loss, and used for exploration. Figure 4 illustrates the total profit achieved during each episode of testing using the greedy policy, and the corresponding QP weights.

4.3 Discussion

On the static optimization task, BERS performs comparatively similar to the single best demonstrator, because it is able to identity the most suitable source task after observing a small number of target demonstrations (Figure 3). While UCB is a strong baseline, BERS finds the solution in less iterations while incurring less variability. As postulated, the solution to (8) favours the Sphere function when solving the Rastrigin function, because they are structurally the most similar, and this leads to a quicker identification of the global minimum for the Rastrigin function.

On the supply chain task, BERS also achieves results similar to the single best expert, and does slightly better than PPR. While their asymptotic performance on Scenario 1 is similar, BERS achieves better jump-start performance on Scenario 1 and better asymptotic performance on Scenarios 2 and 3. One reason for this is that PPR, despite being trained on the same data as BERS, must learn a policy from noisy observations. Despite this, both BERS and PPR are able to quickly surpass the performance of the policy that originally generated the source data (horizontal line in Figure 4). On the other hand, PER was not able to obtain satisfactory performance, because PER prioritizes experiences by TD error that is not suitable for ranking demonstrations with different rewards. On the other hand, BERS learns a common feature embedding that allows for consistent comparison between tasks (Appendix).

Finally, on the target scenario, we can see that the weights assigned to Scenarios 1 and 2 are roughly equal, which makes sense as the target task shares some similarities with both of the aforementioned scenarios (Figure 2). By mixing two source tasks, BERS is able to perform substantially better on the target task than the two source tasks (S2 and S3) in isola-



(a) Scenario 1 as Ground Truth

(b) Scenario 2 as Ground Truth

(c) Scenario 3 as Ground Truth

(d) Target Task

Figure 4: Total testing reward per episode (left) and weights assigned to source tasks (right) over epochs using DDPG for the Supply Chain problem, with each source and target task as ground truth. Averaged over 5 trials with shaded standard error bars.

tion. Also, by adopting a Bayesian treatment, it enjoys stable convergence of the task weights $\mathbf{a}$ on all experiments.

5 Related Work

Most work in LfD incorporates demonstrations from a single expert [Argall *et al.*, 2009]. Some papers in the area of RLfD relax this assumption to a single *sub-optimal* demonstrator and use pre-training [Gao *et al.*, 2018; Hester *et al.*, 2018], reward shaping [Suay *et al.*, 2016], ranking [Wang and Taylor, 2017], or other approaches. However, these papers cannot learn from multiple demonstrators with conflicting goals. Papers on this topic typically assume multiple near-optimal demonstrators, so that recovering policies [Barreto *et al.*, 2017; Fernández *et al.*, 2010; Madarasz and Behrens, 2019] is possible. While our paper shares some similarities with these, it is fundamentally different. First, this stream of literature studies policy transfer, whereas we study LfD. BERS does not learn auxiliary representations for source demonstrators' behaviors (e.g. value functions or policies), allowing it to incorporate sub-optimal exploration data. Furthermore, BERS learns a latent representation of the task goals in an online setting not studied in prior work.

More generally, our approach is related to *multi-modal learning*, in which a common representation of multiple heterogeneous data sources is learned [Hausman *et al.*, 2017; Tsai *et al.*, 2019]. However, to our knowledge, papers on this topic have not been applied to our problem setting. Furthermore, our learned weightings over demonstrators could be seen as a form of *attention* [Zadeh *et al.*, 2018]. The idea of learning shared features is inspired by both encoder sharing [Flet-Berliac and Preux, 2019] and uncertainty quantification [Azizsoltani *et al.*, 2019; Brown *et al.*, 2020]. Finally, our approach shares some of the similarities of *Bayesian policy reuse* [Rosman *et al.*, 2016], by formulating the problem of policy selection as a Bayesian choice problem. However, our work differs in that we apply Bayesian inference for LfD instead of policy transfer. Our work is the first to apply these ideas towards LfD from multiple demonstrators.

6 Conclusion

We studied the problem of LfD with multiple sub-optimal demonstrators with different goals in a Bayesian setting. We proposed a multi-headed Bayesian neural network to efficiently learn consistent representations of the source and target reward functions from the demonstrations. Reward functions were parameterized as linear models, whose uncertainty was modeled using Normal-Inverse-Gamma priors and updated using Bayes' rule. A QP formulation ranked the demonstrators while trading off the mean and variance of the uncertainty in the learned reward representations, and Bayesian Experience Reuse (BERS) was proposed to incorporate demonstrations directly when learning new tasks. Empirical results show that BERS can successfully transfer experience from conflicting demonstrators.

References

[Amos and Kolter, 2017] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, pages 136–145, 2017.

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Auer, 2002] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[Azizsoltani *et al.*, 2019] Hamoon Azizsoltani, Yeo Jin Kim, Markel Sanz Ausin, Tiffany Barnes, and Min Chi. Unobserved is not equal to non-existent: using gaussian processes to infer immediate rewards across contexts. In *IJCAI*, pages 1974–1980, 2019.

[Barreto *et al.*, 2017] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *NeurIPS*, pages 4055–4065, 2017.

[Bishop, 2006] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[Bojarski *et al.*, 2016] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv:1604.07316*, 2016.

[Brown *et al.*, 2020] Daniel Brown, Russell Coleman, Ravi Srinivasan, and Scott Niekum. Safe imitation learning via fast bayesian reward inference from preferences. In *ICML*, pages 1165–1177. PMLR, 2020.

[Cruz Jr *et al.*, 2017] Gabriel V Cruz Jr, Yunshu Du, and Matthew E Taylor. Pre-training neural networks with human demonstrations for deep reinforcement learning. *arXiv:1709.04083*, 2017.

[Fernández *et al.*, 2010] Fernando Fernández, Javier García, and Manuela Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

[Flet-Berliac and Preux, 2019] Yannis Flet-Berliac and Philippe Preux. Merl: Multi-head reinforcement learning. In *NeurIPS Workshop*, 2019.

[Gao *et al.*, 2018] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *ICLR Workshop*, 2018.

[Grollman and Billard, 2011] Daniel H Grollman and Aude Billard. Donut as i do: Learning from failed demonstrations. In *ICRA*, pages 3804–3809. IEEE, 2011.

[Hausman *et al.*, 2017] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph J Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *NeurIPS*, pages 1235–1245, 2017.

[Hester *et al.*, 2018] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *AAAI*, 2018.

[Hou *et al.*, 2017] Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *SMC*, pages 316–321. IEEE, 2017.

[Janz *et al.*, 2019] David Janz, Jiri Hron, Przemysław Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: exploration and uncertainty in temporal difference learning. In *NeurIPS*, pages 4509–4518, 2019.

[Kemmer *et al.*, 2018] Lukas Kemmer, Henrik von Kleist, Diego de Rochebouët, Nikolaos Tziortziotis, and Jesse Read. Reinforcement learning for supply chain optimization. In *EWRL*, pages 1–9, 2018.

[Li and Zhang, 2018] Siyuan Li and Chongjie Zhang. An optimal online method of selecting source policies for reinforcement learning. In *AAAI*, 2018.

[Lillicrap *et al.*, 2016] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[Madarasz and Behrens, 2019] Tamas Madarasz and Tim Behrens. Better transfer learning with inferred successor maps. In *NeurIPS*, volume 32, pages 9029–9040, 2019.

[Nicolescu and Mataric, 2003] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *AAMAS*, pages 241–248, 2003.

[Ober and Rasmussen, 2019] Sebastian W Ober and Carl Edward Rasmussen. Benchmarking the neural linear model for regression. *arXiv:1912.08416*, 2019.

[Pardoe and Stone, 2010] David Pardoe and Peter Stone. Boosting for regression transfer. In *ICML*, pages 863–870, 2010.

[Rosman *et al.*, 2016] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. Bayesian policy reuse. *Machine Learning*, 104(1):99–127, 2016.

[Snoek *et al.*, 2015] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, pages 2171–2180, 2015.

[Suay *et al.*, 2016] Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pages 429–437, 2016.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Tsai *et al.*, 2019] Yao-Hung Hubert Tsai, Paul Pu Liang, Amir Zadeh, Louis-Philippe Morency, and Ruslan Salakhutdinov. Learning factorized multimodal representations. In *ICLR*, 2019.

[Wang and Taylor, 2017] Zhaodong Wang and Matthew E Taylor. Improving reinforcement learning with confidence-based demonstrations. In *IJCAI*, pages 3027–3033, 2017.

[Zadeh *et al.*, 2018] Amir Zadeh, Paul Pu Liang, Soujanya Poria, Prateek Vij, Erik Cambria, and Louis-Philippe Morency. Multi-attention recurrent network for human communication comprehension. In *AAAI*, page 5642, 2018.

Appendix

Proof of Theorem 1

Proof. Given the true values of the variances $\mathbf{v} = [\sigma_1^2, \dots \sigma_N^2, \sigma_{target}^2]$, and given $\mathscr{D}$, the vectors $\mathbf{w}_1, \dots \mathbf{w}_N, \mathbf{w}_{target}$ are normally distributed with respective means $\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_N, \boldsymbol{\mu}_{target}$ and covariances $\sigma_1^2\boldsymbol{\Sigma}_1, \dots \sigma_N^2\boldsymbol{\Sigma}_N, \sigma_{target}^2\boldsymbol{\Sigma}_{target}$ (equation (3)). Then, for any $\mathbf{a} \in \mathbb{R}^N$,

$$\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i \,|\, \mathbf{v}, \mathscr{D} \sim \mathcal{N}(\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i, \sigma_{target}^2 \boldsymbol{\Sigma}_{target} + \sum_i a_i^2 \sigma_i^2 \boldsymbol{\Sigma}_i)$$

and ignoring terms independent of $\mathbf{a}$,

$$\begin{aligned}\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,|\, \mathbf{v}, \mathscr{D}\Big] &= \operatorname{tr}(\sigma_{target}^2 \boldsymbol{\Sigma}_{target} + \sum_i a_i^2 \sigma_i^2 \boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2 \\ &\propto \sum_i a_i^2 \sigma_i^2 \operatorname{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2.\end{aligned}$$

Then, taking expectation with respect to $\mathbf{v}$,

$$\begin{aligned}\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,\Big|\, \mathscr{D}\Big] &= \mathbb{E}\left[\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,\Big|\, \mathbf{v}, \mathscr{D}\Big] \,\Big|\, \mathscr{D}\right] \\ &\propto \sum_i a_i^2 \,\mathbb{E}[\sigma_i^2 | \mathcal{D}_i] \operatorname{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2 \\ &= \sum_{i=1}^N a_i^2 \left(\frac{\beta_i}{\alpha_i - 1}\right) \operatorname{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_{i=1}^N a_i \boldsymbol{\mu}_i\|_2^2,\end{aligned}$$

where in the last step we used (3) again and the expectation of $\mathrm{InvGamma}(\alpha_i, \beta_i)$. □

Hyper-Parameters for the Neural-Linear Model

Architecture: We set $\mathbf{w}_i \sim \mathcal{N}(0, \mathbf{I})$, $\sigma_i^2 \sim \mathrm{InvGamma}(1, 1)$. For the encoder, we set $d = 20$, use L2 regularization with penalty $\lambda = 10^{-4}$, a learning rate of 10^{-4}, and initialize weights using Glorot uniform. The encoder has two hidden layers with 200 ReLU units each (300 in the first layer for the supply chain problem), and $\tanh$ outputs. We also include a constant bias term in the feature map $\boldsymbol{\phi}$ when learning $\mathbf{w}$.

Training: Prior to transfer, we first train the neural linear model on 4000 batches of size 64 sampled uniformly from source data. During target training, we train on batches of size 64 sampled from source and target data after each iteration (generation for optimization, episode for supply chain) to avoid catastrophic forgetting of features (one batch for optimization and 20 for supply chain). QPs are solved from scratch at the end of each iteration using the `cvxopt` package [1].

Details for the Static Function Optimization Benchmark

Functions: We consider both transfer and multi-task learning settings. For the transfer experiment, we use the Rosenbrock, Ackley and Sphere functions as the source tasks,

$$\begin{aligned}f_{Rosenbrock}(\mathbf{x}) &= \sum_{i=1}^{9} 100[(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \\ f_{Ackley}(\mathbf{x}) &= -20 \exp\left(-0.2\sqrt{\frac{1}{10}\sum_{i=1}^{10} x_i^2}\right) - \exp\left(\frac{1}{10}\sum_{i=1}^{10}\cos(2\pi x_i)\right) + 20 + \exp(1) \\ f_{Sphere}(\mathbf{x}) &= \sum_{i=1}^{10}(x_i + 2)^2,\end{aligned}$$

and the Rastrigin function

$$f_{Rastrigin} = 100 + \sum_{i=1}^{10}[(x_i + 2)^2 - 10\cos(2\pi(x_i + 2))],$$

as the target task. We also consider each of the source tasks as the ground truth to see whether we can identify the correct source task in an ideal controlled setting. In the multi-task setting, we did not observe any advantages of our algorithm nor the baselines when solving the Rosenbrock, Ackley or Sphere functions, so we focus on the quality of solution obtained for the Rastrigin function only in the main paper.

Data Processing: For Rosenbrock, sphere and Rastrigin functions, we transform outputs using $y \mapsto \sqrt{y}$ so they become approximately equally scaled. For Rosenbrock, we subsequently also divide outputs by 10. This ensures that the outputs of all functions are approximately equally-scaled, to demonstrate whether we can actually "learn" each function from the data rather than distinguish them according to scale alone.

Solver Settings: To optimize all functions, we use the Differential Evolution (DE) algorithm [2]. The pseudo-code of this algorithm is outlined in Algorithm 2. Here, CR is the crossover probability and denotes the probability of replacing a coordinate in an existing solution with the candidates (crossover), F is the differential weight and controls the strength of the crossover, and NP is the size of the population (larger implies a more global search). We use standard values $CR = 0.7, F = 0.5$ and $NP = 32$ for reporting experimental results, unless indicated otherwise. The search bounds are also set to $[-4, +4]$ for all coordinates; all points that are generated during the initialization of the population and the crossover are clipped to this range.

Algorithm 2 Differential Evolution (DE)

Require: $f : \mathbb{R}^D \to \mathbb{R}$, $CR \in [0, 1]$, $F \in [0, 2]$, $NP \geq 4$
  initialize NP points $\mathcal{P}$ uniformly at random in the search space
  for generation $m = 1, 2, \ldots$ **do**
    for agent $\mathbf{x} \in \mathcal{P}$ **do**
      randomly select three candidates $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{P}$ that are distinct from each other and from $\mathbf{x}$
      pick a random index $R \in \{1, 2 \ldots D\}$
      for $i = 1, 2 \ldots D$ **do**
        pick $r_i \sim U(0, 1)$
        if $r_i < CR$ or $i = R$ **then**
          set $y_i = a_i + F(b_i - c_i)$
        else
          set $y_i = x_i$
        end if
        if $f(\mathbf{y}) \leq f(\mathbf{x})$ **then**
          replace $\mathbf{x}$ in $\mathcal{P}$ with $\mathbf{y}$
        end if
      end for
    end for
  end for
  return the agent in $\mathcal{P}$ with the least function value

Transfer Learning: We first generate demonstrations $(\mathbf{x}_i, f(\mathbf{x}_i))$ from each source function f using DE (Algorithm 2) and configuration above until a fitness of 0.15 is achieved. Respectively, these have sizes 17693, 6452 and 5853 for each source task. We further transform outputs for training and prediction with the neural-linear model using a log-transform $y \mapsto \log(1 + y)$ to limit the effects of outliers and stabilize the training of the model. When solving the target task, the batch $\mathcal{B}$ is a singleton set containing the best solution from the source data, and O_{base} is trained by replacing the first of the three sampled candidates prior to crossover with probability $p_m = 0.99^m$, where m is the index of the current generation (following the structure of Algorithm 1). This guarantees that the new swarm will possess the traits of the source solutions with high probability, but still maintain diversity so the solution can be improved further.

Multi-Task Learning: We solve all source and target functions simultaneously, sharing the best solutions between them using the mechanisms outlined above for the transfer learning experiment. One QP is maintained for each task to learn weightings over the other tasks excluding itself. Here, we set $p_m = 0.3$, so that the best obtained solutions can be consistently shared between tasks over time.

Details for the Supply Chain Benchmark

Solver Settings: We use the Deep Deterministic Policy Gradient (DDPG) Algorithm [3] to solve this problem. The critic network has 300 hidden units per layer. We also use L2 penalty $\lambda = 10^{-4}$, learning rates 10^{-4} and 2×10^{-4} for actor and critic respectively, $U(-3 \times 10^{-3}, 3 \times 10^{-3})$ initialization for weights in output layers, discount factor $\gamma = 0.96$, horizon $T = 200$, randomized replay with capacity 10000, batch size of 32, and explore using independent Gaussian noise $\mathcal{N}(0, \sigma_t^2)$, where σ_t is annealed from 0.15 to 0 linearly over 50000 training steps.

Transfer Learning: We considered the transfer from multiple source tasks (Scenarios 1, 2 and 3) to a single target task (Target Task) as indicated in Figure 2. We also consider each source task as a ground truth. To collect source data, we train DDPG with the above hyper-parameters on each source task for 30000 time steps, then randomly sub-sample 10000 observations. This last step demonstrates whether our approach can learn stable representations with limited data and incomplete exploration

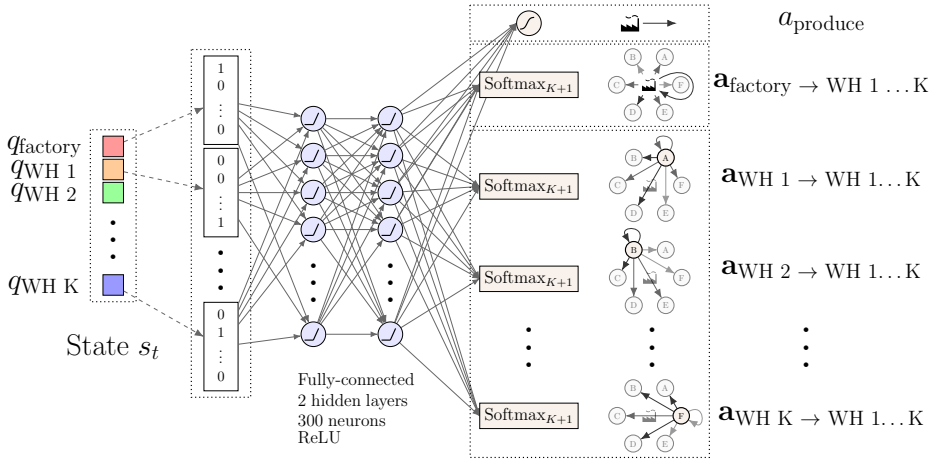


Figure 5: Actor network.

trajectories. Since regression is sensitive to outliers, when training the neural linear model, we remove 2.5% of the samples with the highest and lowest rewards (we also exclude observations collected from the target task that lie outside any of these bounds when training the neural-linear model). In order to implement transfer, we set $p_m = 0.95^m$, where m is the current episode number. This provides a reasonable balance between exploration of source data and exploitation of target data. In accordance with Algorithm 1, we sample batches $\mathcal{B}$ of size 32 uniformly at random from the source data.

Prioritized Experience Replay: We evaluated the performance of Prioritized Experience Replay (PER) [4] on the Supply Chain benchmark. In summary, PER is a replay buffer that ranks experiences using the Bellman error, defined for the DDPG algorithm for a transition (s, a, r, s') as

$$\delta = r + \gamma Q'(s', \mu'(s')) - Q(s, a),$$

where Q is the critic network, and Q' and μ' are the target critic and target actor networks, respectively.

The source demonstrations are combined into a single data set and shuffled. The capacity of the buffer is also fixed to the total number of source demonstrations from all tasks (around 28,000 examples). We then load all source demonstrations into the replay buffer prior to target task training with initial Bellman error $\delta = 1$. During target training, observed transitions are immediately added to the replay buffer and override the oldest experience. In this way, the agent is able to maximize the use of the source data at the beginning of training but eventually shift emphasis to target task data.

Figure 6 demonstrates the composition of each batch (of size 32) according to source (Scenario 1, Scenario 2, Scenario 3, Target Task) for each ground truth. As illustrated in all four plots, in early stages of training, batches are predominantly composed of source data, while in later stages, they consist entirely of target data. Figure 6 shows that PER is unable to favor demonstrations from the source task that correspond to the ground truth, in all four problem settings. One possible explanation of this is that an implicit assumption of PER is violated, namely that experiences are drawn from the same distribution of rewards, whereas in our experiment, experiences can come from tasks with contradictory goals. In this case, experiences corresponding to the ground truth do not necessarily have larger Bellman error (in fact, the opposite may be true).

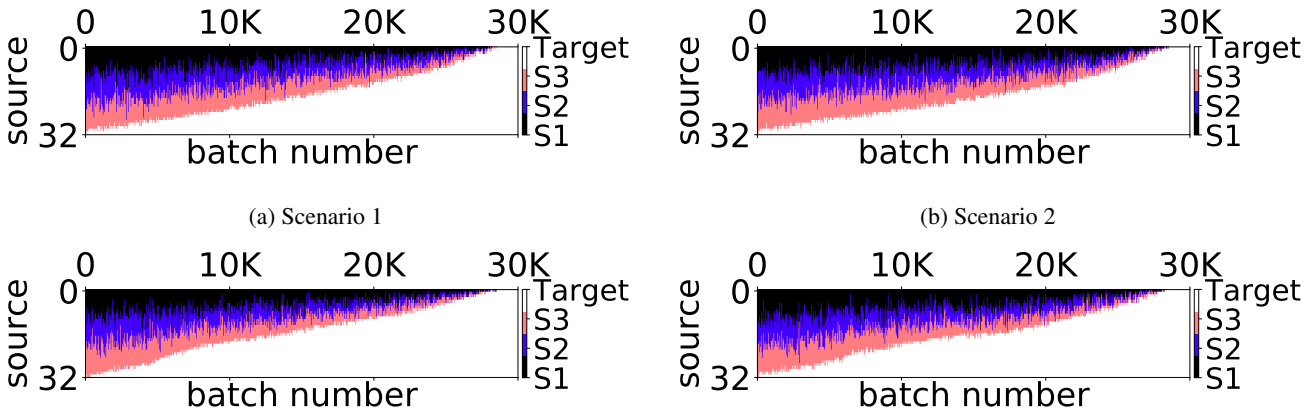
Latent Space Analysis

We include several plots that couldn't go in the main paper due to space limitations. In particular, we analyze the learned latent reward functions $\mathbf{w}$ for source and target tasks for both the function optimization and supply chain problems.

The analysis for the function optimization problem is illustrated in Figure 7 in which we set the latent dimension $d = 2$ for ease of illustration. Similarly, the analysis for the supply chain problem using the original value of $d = 20$ (we tried smaller values of d, but did not obtain satisfactory results due to the complexity of the true reward function for this problem) is illustrated in Figure 8. In both problem settings, BERS is able to learn a meaningful latent representation of the demonstrators' goals.

Additional Experiment for the Reacher Domain

Domain Description: The reacher domain consists of a two-jointed robotic arm that must be controlled to hover above a fixed target location. The state is 4-dimensional and consists of the angle and angular velocities of the two joints. At the beginning of each episode, the angular velocities are set to zero, while the central joint angle is sampled uniformly from $[-\pi, +\pi]$ and the outer joint angle is sampled uniformly from $[-\pi/2, +\pi/2]$. The 2-dimensional continuous action space represents the



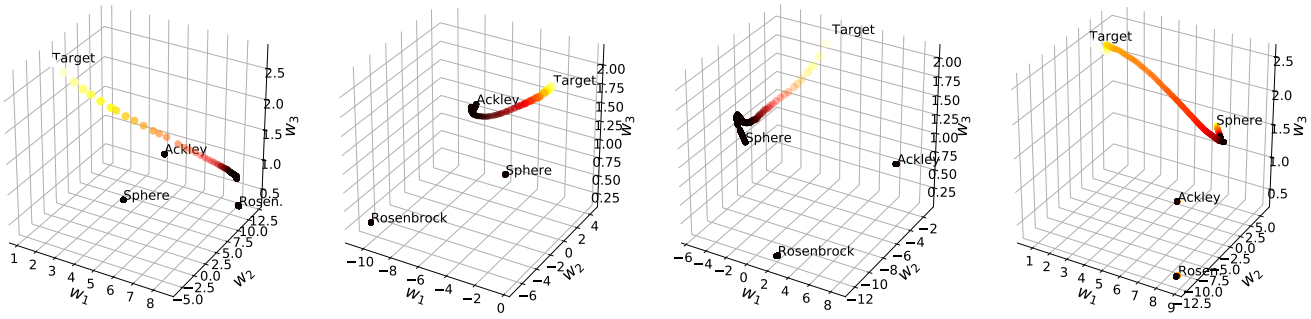
(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 6: The composition of each batch over time sampled from the prioritized replay (PER) according to whether the sample came from the source or target task data, for each ground truth. PER is unable to rank experiences correctly to match the context.



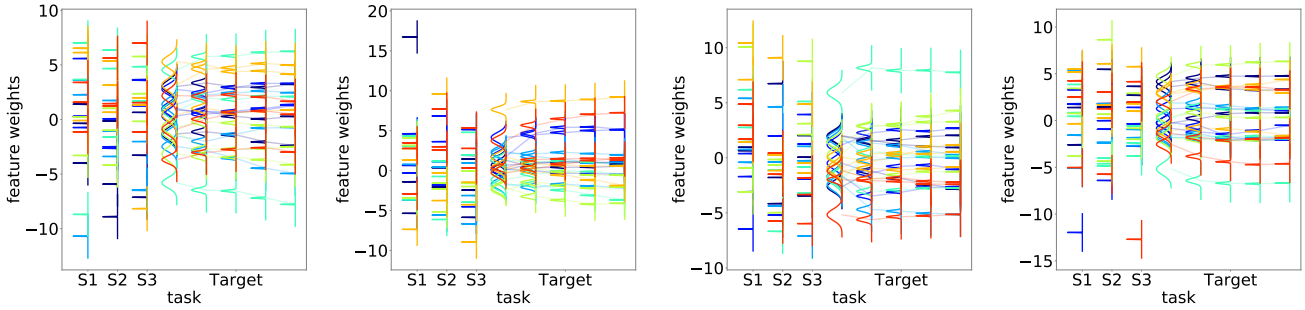
(a) Rosenbrock

(b) Ackley

(c) Sphere

(d) Rastrigin

Figure 7: For function optimization with each source and target task as the ground truth, shows the evolution of the posterior mean of each $\mathbf{w}_i$ and $\mathbf{w}_{target}$ during training for the simplified experiment with latent dimension $d = 2$. As shown here, the target mean eventually converges to the correct source task mean.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Target

Figure 8: For Supply Chain control with each source and target task as the ground truth, shows the posterior marginal distributions of $\mathbf{w}_i$ and $\mathbf{w}_{target}$ during training on the target task (after 0, 10, 50, 100 and 200 episodes). Over time, the target features begin to concentrate on the corresponding values for the correct source task.

possible torques that can be applied to each joint, and are discretized into 3 possible actions per dimension (corresponding to minimum, maximum, and zero torques) for a total of 9 possible actions. Following [Barreto *et al.*, 2017], we initialize 4 source task instances whose target locations are indicated as colored circles in Figure 9, that are used to train each demonstrator. We

define 2 additional task instances as target tasks whose target locations are indicated as gray circles. The target task whose goal position is located close to the top-right corner of the map is similar to the demonstrator associated with the yellow goal position, whereas the target task whose goal position is in the bottom-left is considerably different from any of the source tasks. The reward is defined as $1 - 4\delta$, where δ is the Euclidean distance between the arm tip and the target location.

Solver Settings: All tasks are solved using the DQN algorithm [5]. The Q-value function is parameterized as a feedforward neural network containing two hidden layers of size 256 with $\tanh$ activation functions, followed by a linear output layer of size 9 to represent the Q-value of each action. The Adam optimizer with a learning rate of 10^{-3} is used to train the network, on batches of size 32 sampled at random from a replay buffer of infinite capacity. We use a discount factor $\gamma = 0.9$ and horizon $T = 500$ during training. The epsilon-greedy policy with $\varepsilon = 0.1$ is used for training and $\varepsilon = 0.03$ for testing. The neural-linear model for BERS uses the same parameters indicated above, except that the activation function for hidden units is replaced by $\tanh$ and the learning rate is changed to 5×10^{-4}.

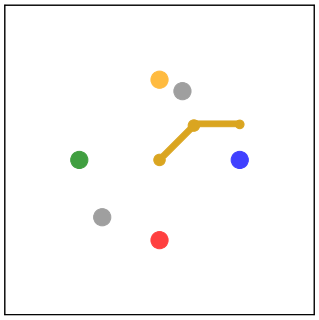
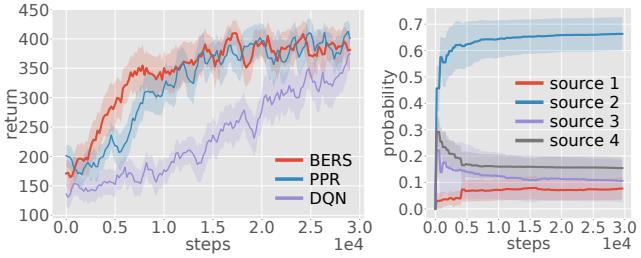


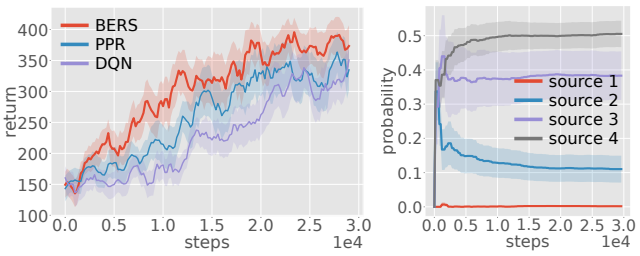
Figure 9: The reacher domain. The 4 colored circles indicate target locations used to train the demonstrators. The target locations for the target tasks are indicated in gray circles.

Transfer Learning: The demonstrators are trained on each of the 4 training target locations for 30000 time steps. The training data is saved, and a random sample of 3200 transitions is used to pre-train BERS and PPR. The neural-linear models are trained on these subsets of demonstrations by sampling 20000 batches of size 32. The policy network for PPR has the same structure as the Q-value network described above, except the output layer uses a $\mathrm{softmax}$ activation function. The Adam optimizer is used to train the policy networks using each of the 4 training task demonstrations data sets, and uses a learning rate of 10^{-3}. These networks are pre-trained on batches of size 32 for a total of 1000 epochs. Similar to the supply chain problem, we set $p_m = 0.95^m$, where m is the current episode number. A budget of 30000 time steps is allocated for training on each of the target task locations for BERS and PPR.

We compare BERS to PPR and standard DQN, and report the results in Figure 10. Here, we see that the performance of BERS is better than PPR on both test tasks. Interestingly, while the performance improvement on the top-right target location is relatively insignificant, the benefits of BERS are quite substantial on the more difficult target task with the bottom-left target. Looking at the plot of the posterior weights $\mathbf{w}$, we see that BERS was able to identify two demonstrators – with target locations located at the bottom and the left of the space – as being the most relevant. As previously demonstrated on the supply chain domain, BERS is once again able to select a *combination* of two relevant demonstrators. By mixing the demonstrations from a subset of the most relevant demonstrators, it is possible to obtain better performance than training on a single policy or demonstrator.



(a) Top-right target.



(b) Bottom-left target.

Figure 10: Total testing reward per episode (left) and weights assigned to source tasks (right) over epochs using DQN for the reacher domain, for the two target positions. Averaged over 5 trials with shaded standard error bars.

References

[1] Andersen, Martin S., Joachim Dahl, and Lieven Vandenberghe. "CVXOPT: A Python package for convex optimization." abel. ee. ucla. edu/cvxopt (2013).

[2] Storn, Rainer, and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces." Journal of global optimization 11.4 (1997): 341-359.

[3] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." ICLR (Poster). 2016.

[4] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." International conference on machine learning. PMLR, 2016.

[5] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.