# Dimensionality Reduced Reinforcement Learning for Assistive Robots

**William Curran**
Oregon State University
Corvallis, Oregon
curranw@onid.oregonstate.edu

**Tim Brys**
Vrije Universiteit Brussel
Belgium
timbrys@vub.ac.be

**David Aha**
Navy Center for Applied Research in AI
david.aha@nrl.navy.mil

**Matthew Taylor**
Washington State University
Pullman, Washington
taylorm@eecs.wsu.edu

**William D. Smart**
Oregon State University
Corvallis, Oregon
bill.smart@oregonstate.edu

**Abstract**

State-of-the-art personal robots need to perform complex manipulation tasks to be viable in assistive scenarios. However, many of these robots, like the PR2, use manipulators with high degrees-of-freedom, and the problem is made worse in bimanual manipulation tasks. The complexity of these robots lead to large dimensional state spaces, which are difficult to learn in. We reduce the state space by using demonstrations to discover a representative low-dimensional hyperplane in which to learn. This allows the agent to converge quickly to a good policy. We call this Dimensionality Reduced Reinforcement Learning (DRRL). However, when performing dimensionality reduction, not all dimensions can be fully represented. We extend this work by first learning in a single dimension, and then transferring that knowledge to a higher-dimensional hyperplane. By using our Iterative DRRL (IDRRL) framework with an existing learning algorithm, the agent converges quickly to a better policy by iterating to increasingly higher dimensions. IDRRL is robust to demonstration quality and can learn efficiently using few demonstrations. We show that adding IDRRL to the Q-Learning algorithm leads to faster learning on a set of mountain car tasks and the robot swimmers problem.

## 1 Introduction

Our ultimate goal is to deploy personal robots into the world, and have members of the general public retask them without having to resort to explicitly programming them. Learning from demonstration (LfD) methods learns a policy using examples or demonstrations given by a human to speed up learning a custom task (Argall et al. 2009). However, these demonstrations must be consistent and accurately represent solving the task. These methods also solve for a specific complex task, rather than solve for general control (Argall et al. 2009). In this paper we present an approach to directly address the problem of learning good policies with RL in high-dimensional state spaces.

The first step in our research goals is to develop an efficient method for teaching the robot. Reinforcement learning is an ideal approach in our application. It can be used to teach a robot new skills that the human teacher cannot demonstrate, find novel ways to reach human-defined goals, and can be used to find solutions to difficult problems with no analytic formulation (Kormushev, Calinon, and Caldwell 2013). However, reinforcement learning does not scale well, and real-world robotics problems are high-dimensional (Kober and Peters 2012). Learning in high-dimensional spaces not only significantly increases the time and memory requirements of many algorithms, but also degenerates performance due to the curse of dimensionality (Kaelbling, Littman, and Moore 1996). Personal robots need to perform complex manipulation tasks to be viable in many scenarios. Complex manipulations require high degree-of-freedom arms and manipulators. For example, the PR2 robot has two 7 degree-of-freedom arms. When learning position and velocity control, this leads to a 14 dimensional state space per arm.

In this work, we focus on the core problem of high-dimensional state spaces. We introduce two algorithms Dimensionality Reduced Reinforcement Learning (DRRL) and Iterative DRRL. In DRRL we use demonstrations to compute a projection to a low-dimensional hyperplane. In each learning iteration, we project the current state onto this hyperplane, compute and execute an action, project the new state onto the hyperplane, and perform a reinforcement learning update. This general approach has been shown to reduce the exploration needed and accelerates the learning rate of reinforcement learning algorithms (Bitzer, Howard, and Vijayakumar 2010; Colome et al. 2014).

The robot can learn more quickly in the low-dimensional hyperplane. However, this leads to a critical trade-off. By projecting onto a low-dimensional hyperplane, we are discarding potentially important data. By adding DRRL to an existing algorithm, we show that the robot can quickly converge to a good policy much faster. However, since DRRL does not represent all dimensions, it could converge to a poor policy.

In many learning domains, poor policies are undesirable. In robotics in particular, bad controllers can damage the robot. We propose a novel framework, IDRRL, combining learning from demonstration techniques, dimensionality reduction, and transfer learning. Instead of learning entirely in one hyperplane, we iteratively learn in all hyperplanes by using transfer learning. The robot can quickly learn in a low-dimensional space $d$, and transfer that knowledge from $d$ dimensions to the $d + 1$ dimensional space using the known

mapping between the spaces.

Our novel approach is a framework to improve other learning algorithms when working in high-dimensional spaces. It combines the speed of low-dimensional learning and the expressiveness of the full state space. We show in a set of mountain car tasks and the robot swimmers problem that reinforcement learning algorithms modified with DRRL or IDRRL can converge quickly to a better policy than learning entirely in the full dimensional space.

## 2 Background

To motivate our approach, we outline previous work performed in the field of reinforcement learning, dimensionality reduction, transfer learning, and learning from demonstration.

### 2.1 Reinforcement Learning

In our reinforcement learning (RL) approach, we use the standard formulation of MDPs (Kaelbling, Littman, and Moore 1996). An MDP is a 4-tuple $\langle S, A, T, R\rangle$, where $S$ is a set of states, $A$ is a set of actions, $T$ is a probabilistic state transition function $T(s, a, s')$, and $R$ is the reward function $R(s, a)$.

In this work, we use function approximation for generalization. CMACs (Albus 1981) partition a state space into a set of overlapping tiles, and maintain the weights ($\theta$) of each tile. The accuracy of the generalization is improved as the number of tilings ($n$) increases. Each tile has an associated binary value ($\phi$) to indicate whether that tile is present in the current state.

The estimate of the value function is:

$$Q_t(s, a) = \sum_{i}^{n} \theta(i)\phi(i) \tag{1}$$

where $Q_t(s, a)$ is the estimated value function, $\theta$ is the weight vector and $\phi$ is a component vector. Given a learning example, we adjust the weights of the involved tiles by the same amount to reduce the error. We use standard model-free Q-Learning to update our function approximation:

$$\begin{aligned} Q_{t+1}(s_t, a_t) =& Q_t(s_t, a_t) + \alpha(R_{t+1}(s_t, a_t) \\ &+ \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \end{aligned} \tag{2}$$

where $\alpha$ is the learning rate, and $\gamma$ is a discount factor between 0 and 1 that represents the importance of future rewards.

A significant amount of research in RL focuses on increasing the speed of learning by taking advantage of domain knowledge. Some techniques include agent partitioning, which focuses mainly on how to divide the problem by the state space, actions, or goals (Curran, Agogino, and Tumer 2013; Jordan and Jacobs 1993; Reddy and Tadepalli 1997); generalizing over the state space with techniques such as tile coding (Whiteson, Taylor, and Stone 2007), neural networks (Haykin 1998), or k-nearest neighbors (Martin H., de Lope, and Maravall 2009); and learning with temporally defined actions, such as options (Sutton, Precup, and Singh 1999). Our framework is designed to be used with these approaches. In this paper, we use the title coding generalization.

### 2.2 Dimensionality Reduction for Learning

Previous work in dimensionality reduction focuses on reducing the space for classification or function approximation. Principal Component Analysis (PCA) (Jolliffe 2002) is effective in many machine learning and data mining applications at extracting features from large data sets (Pechenizkiy, Puuronen, and Tsymbal 2003; Turk and Pentland 1991). Feature selection, which aims at reducing the dimensionality by selecting a subset of most relevant features, has also been proven to be an effective and efficient way to handle high-dimensional data (Cobo et al. 2011).

In our work, we use PCA to discover the low-dimensional representation of the state space during learning. It does this by computing a transform to convert correlated data to linearly uncorrelated data. This transformation ensures that the first principal component captures the largest possible variance. Each additional component captures the largest possible variance uncorrelated with all previous components. Essentially, PCA represents as much of the demonstrated state space as possible in a lower dimension.

Rather than transferring knowledge from a simple representation to a complex one, Taylor, Kulis, and Sha (2011) learn directly which states are irrelevant. They collect data while the agent explores the environment, calculate a state similarity metric, and ignore state variables that do not add additional information and scale those that do. Similarly, Thomas et al. (2011) use demonstrations to develop a subset of features from the original space. The learning algorithm then used this subspace to predict the action that a human expert would take. Both approaches find state variables to ignore or emphasize. This works well if there are unimportant state variables, but would have issues where there are infrequent or non-demonstrated state variables with critically important data. In our IDRRL framework, we iteratively add additional state information until it learns in the full state space, essentially combining the speed of low-dimensional learning and the expressiveness of the full state space.

Similar to our approach, Colomé et al. (2014) apply dimensionality reduction techniques to exploit underlying manifolds in the state space. They learn probabilistic motor primitives in a low-dimensional space discovered by probabilistic dimensionality reduction. Bitzer et al. (2010) also applies dimensionality reduction to solve reinforcement learning planning problems in a reduced space that automatically satisfies their task constraints. Although these approaches greatly accelerate learning, they learn entirely in the low-dimensional space and could discard potentially important data.

### 2.3 Transfer Learning

The core idea of transfer learning is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task. If the relationship between the first (source) task and the second (target) task is not trivial, there must be a mapping between the two tasks so that a learner can apply the older knowledge to the new task (Taylor and Stone 2009). An inter-task mapping is a general structure that defines how two tasks are related.

The mappings $\chi_S$ and $\chi_A$ are defined as a mapping between state variables and actions in two tasks, respectively.

In this work, we transfer from a low-dimensional representation to a higher-dimensional representation. Therefore, the states between representations are not the same. Additionally, the number of state variables we represent change as we change the number of dimensions in the problem. To perform the state projection, we compute a mapping using dimensionality reduction. This mapping is $\chi_S$, which is what we use during transfer. Our action representation remains the same, and therefore we do not need to compute $\chi_A$.

### 2.4 Learning from Demonstration

Learning a policy using traditional reinforcement learning is difficult in real-world applications. Initializing, or bootstrapping, the policy close to the desired robot behavior makes finding an optimal or near-optimal solution easier. LfD learns a policy using examples or demonstrations provided by a human. These examples are typically state-action pairs that are recorded during the teacher's demonstration. These state-action samples are used to initialize a policy that can then either be directly utilized, or improved using reinforcement learning (Argall et al. 2009).

There are a variety of techniques that LfD algorithms use when deriving a policy. The demonstrated data can be used to initialize a policy (Peters and Schaal 2006), develop a reward function (Thomaz and Breazeal 2006), or build a state transition function (Argall et al. 2009). Our approach takes the demonstrated data and uses it to discover a representative lower-dimensional hyperplane using dimensionality reduction.

## 3 Dimensionality Reduced Reinforcement Learning (DRRL)

To learn in high-dimensional state spaces, our algorithm first computes a mapping between the high-dimensional space and a lower-dimensional space. To perform this computation, we need trajectories across a representative set of the agent's state space. We can then use any dimensionality reduction technique to learn the transform. In this work, we use PCA. Note that we use PCA due to its popularity and ease of access. DRRL and IDRRL can use alternative dimensionality reduction techniques.

First, we project the state down onto a low-dimensional hyperplane. We then compute the action using the chosen RL algorithm, and execute that action in simulation. The simulation calculates the new state given the executed action, which we then project that state down to the same lower-dimensional space. We can then perform a learning update.

By learning in a smaller space, reinforcement learning algorithms should converge faster. However, in most cases one principal component cannot represent all of the variance in all of the demonstrations. Therefore, even given infinite time, the converged learning performance in a non-trivial case will always be strictly worse than learning in the full space. This leads to a critical trade-off. By projecting onto a low-dimensional hyperplane, we are throwing out low variance, yet possibly critically important data. However, we experimentally validate that with our DRRL framework, learning can still converge to a good policy much faster than the reinforcement learning algorithm alone.

### 3.1 Iterative DRRL (IDRRL)

The key aspect of performing iterative learning is transferring what was learned in the low-dimensional space to the high-dimensional space. To do this, we borrow from the field of transfer learning. In this work, we want to transfer all of the knowledge from the source to the target task. Additionally, the mapping from the source to the target task ($\chi_S(s)$) is given by the dimensionality reduction mapping. This eases the transfer problem greatly. We only need to choose when to transfer. If we transfer too early, the value function in the source domain is far from optimal. This bad knowledge can be spread throughout the value function in the target task.

To choose when to transfer, we borrow from the definition of convergence in Policy Iteration (Sutton and Barto 1998). In Policy Iteration, the value function is updated at each iteration until the policy does not change between updates. We make this policy comparison and ensure that the most recently executed policy is at least 95% converged (unchanged before and after an update) before transferring to the higher space.

Since IDRRL is a general framework, we can use any transfer learning approach within the constraints of the learning algorithm. We use Q-Value Reuse (Taylor, Whiteson, and Stone 2007). Q-Value Reuse is a simple technique applicable when the source and target task both use TD learning, as in our case. In Q-Value Reuse, a copy of the source task's value function is retained and used to calculate the target task's Q-Value. This computed Q-Value is a combination of the source task's saved value function and the target task's value function:

$$Q(s,a) = Q_{source}(\chi_S(s), a) + Q_{target}(s,a) \qquad (3)$$

where $\chi_S$ is the transfer function between the source and target tasks' states and actions. This transfer function is the PCA projection. We then compute the Q-Learning update step as normal, but only the target's value function is updated.

### 3.2 Convergence Guarantees

Q-Value Reuse can be seen as a heuristic for action-value initialization. Strehl, Li, and Littman (Strehl, Li, and Littman 2009) demonstrated that if the action-values are admissible, then this initialization can decrease the sample complexity while maintaining PAC-MDP (i.e. bounded convergence) guarantees. Admissible heuristics provide valuable prior knowledge to PAC-MDP RL algorithms, but the specified prior knowledge does not need to be exact. An initialization heuristic ($H$) is said to be admissible if:

$$V^*(s) \leq Q^*(s,a) \leq H^*(s,a) \leq \frac{R_{max}}{1-\gamma} \qquad (4)$$

where $V^*(s)$ is the optimal value function, $Q^*(s,a)$ is the optimal action-value function, $H^*(s,a)$ is the initialization heuristic, and $\frac{R_{max}}{1-\gamma}$ is the maximum possible value.

Mann and Choe (Mann and Choe 2012) extend PAC-MDP theory to intertask transfer learning. They introduce the concept of weakly admissible heuristics and show that they can still maintain PAC-MDP guarantees. To be weakly admissible, a heuristic needs to be admissible for only one action in each state. They combine weakly admissible heuristics and intertask mappings and prove they are also PAC-MDP if for each state $s$ there is an action $\widetilde{a}$ such that:

$$V^*_{trg}(s) - \alpha \leq Q^*_{trg}(s, \widetilde{a}) \leq Q^*_{src}(\chi_S(s), \chi_A(\widetilde{a})) \quad (5)$$

where $\alpha$ is the smallest non-negative value satisfying this inequality.

Q-Learning does not provide PAC-MDP bounds. However, we still enforce an admissible heuristic to make use of the *optimism in the face of uncertainty* bias (Brafman and Tennenholtz 2003). This bias has been shown to reduce the chance of converging to a locally optimal policy. As it stands, Q-Value Reuse is not admissible. It is guaranteed to be less than $\frac{R_{max}}{1-\gamma}$, but is not greater than $Q^*(s, a)$. We remove this issue by simply adding $R_{max}$ to each $Q_{source}(\chi_S(s), a)$ function.

Although this paper uses Q-learning, we remind the reader that IDRRL is a general framework. Thus, this proof is relevant when IDRRL is combined with PAC-MDP algorithms like R-Max (Jong and Stone 2007).

# 4 Experimental Setup

In our experiments, we combine both DRRL and IDRRL with Q-Learning (Sutton and Barto 1998). We show that DRRL combined with Q-Learning converges quickly, but could converge to a locally optimal solution due to the less informative state representation. Alternatively, we also show IDRRL with Q-Learning converges quickly without reducing performance. We also show that our IDRRL framework scales well with the size of the state and action space.

## 4.1 Mountain Car

To test the efficacy of DRRL, we first consider the Mountain Car 3D domain. Mountain Car is a standard reinforcement learning domain (Dutech et al. 2005). In this problem an underpowered car must drive up a steep hill. The problem is engineered such that the car cannot overcome the effects of gravity, and cannot simply drive up the hill. Since the car starts in a valley, the agent must learn to build up enough inertia by driving partially up the opposite hill before it is able to make it to the goal.

In 2D Mountain Car, there are two states defined as the continuous position ($-1.2 \leq x \leq 0.6$) and velocity ($-.007 \leq v \leq .007$) of the car. There are three actions: Accelerate left, accelerate right, and neutral. The starting state is a random position at a random velocity. Lastly, the reward is -1 at each time step, and 100 at the goal. The 3D and 4D variant of Mountain Car are similar. There are two new states (position and velocity) and two new actions (accelerate/decelerate in that direction) in the 3D variant and four new states and actions in the 4D variant. Note that the 4D variant is not physically possible and is used to show DRRL and IDRRL scaling to a more high-dimensional state space.

## 4.2 Swimmers

The Swimmers domain (Coulom 2002) is a more complex system than Mountain Car. It includes complex physics with a large state and action space. In the Swimmers domain, there is a simple swimmer (Figure 1) connected by joints that moves in a two dimensional pool. The action space is a torque applied at each joint. The goal of the Swimmers domain is to swim as fast as possible to the right, by using the friction of the water.

We define the state of the swimmer as the angular position and velocity at each joint. Therefore a $n$-link swimmer has $2n$ states. The action space consists of the $n-1$ control torques at each joint. At each control step the learner chooses between a $-3Nm$, $0Nm$ or $3Nm$ torque, making $3^{(n-1)}$ actions. We reward the swimmer for moving as fast as possible to the right ($\Delta x$). In this work we use a 3-link and 6-link swimmer.

To move the swimmer, the reinforcement learner must learn to control an n-link object using the torques at each joint. It must learn to leverage the viscous friction and learn the nonlinear dynamics of the system. This is reminiscent of robot arms. The state and action spaces are identical and involve passive dynamics acting on the agent. In future work we intend to apply the IDRRL framework to learning the control of a robot arm.

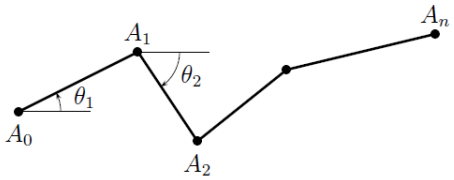


Figure 1: N-link swimmer. The swimmer must learn to leverage the viscous friction of the water to swim.

# 5 Results and Analysis

We applied DRRL and IDRRL with Q-Learning to three domains: Mountain Car 3D, Mountain Car 4D and Swimmers. For each experiment we use the following parameter settings for 20 statistical runs: $\alpha = 0.1$ and $\gamma = 0.99$. Error bars are shown in each graph and represent error in the mean. If an error bar is not visible, the error was negligible.

## 5.1 Mountain Car

In our formulation of Mountain Car we used 16 tiles and a $10^n$ tiling, where $n$ is the number of state variables. There are 4 state variables and 5 actions in the 3D variant, and 6 state variables and 7 actions in the 4D variant. We first learn using good demonstrations and single dimensions to test the efficacy of DRRL. We also show that if the hyperplane does not represent enough variance in the data, the learning algorithm will converge to a poor policy. We then demonstrate that this issue is alleviated by IDRRL. Lastly, we test the robustness of the demonstrations with respect to quality and performance. We then use the Mountain Car 4D domain to

show scalability. To gather the demonstrations we learned good and bad policies with Q-Learning and computed random policies. We define good and bad policies by the reward they received during learning. Good demonstrations reached the goal within 300 time steps, and bad demonstrations within 500–1000 steps. Bad demonstrations reached the goal state, just less efficiently.

In 3D and 4D Mountain Car there was no single state variable more important than all other state variables. Our PCA analysis showed that the first two principal components weighed all of the state variables equally, independent of demonstration quality. Since the demonstrations explored much of the configuration space of the agent, this showed us which states were important to the agents general movement.

Learning in only one $d$-dimensional hyperplane converged faster than learning in the full state space (Figure 2). DRRL converged to the optimal solution using only 2 or 3 dimensional hyperplane, rather than the full dimensional space of 4. This tells us that the Mountain Car domain is simple enough to be learned in a 2 dimensional space. It also converged very quickly to a poor solution in a 1 dimensional hyperplane. However, by using only a single dimension, DRRL does not have a rich enough state space to learn optimally.

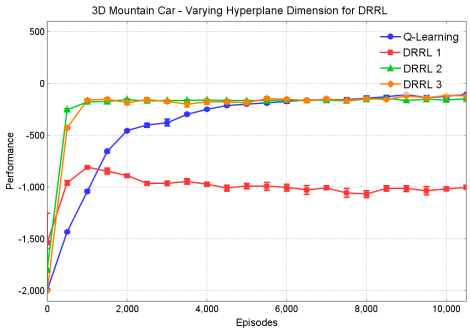


Figure 2: We compare Q-Learning to Q-Learning when combined with DRRL using good quality demonstrations. Q-Learning combined with DRRL converged faster to an optimal solution when learning in the 2 and 3 dimensional hyperplanes. The single dimensional hyperplane did not contain enough information to learn effectively.

Combining standard reinforcement learning with IDRRL led the agent to converge faster with the same converged performance (Figure 3). DRRL converged slightly faster than IDRRL, but IDRRL benefits by eventually learning in the entire state space. This means IDRRL does not lose any state information to speed up learning. Moreover, it does not require the algorithm designer to know beforehand which is the best hyperplane to learn in.

Since IDRRL represents the state space in low-dimensional and sparse hyperplanes, it converges very quickly. With each additional dimension, it starts with a richer state space and the experience gained from all previous dimensions. By episode 1,000, IDRRL bootstrapped learning in the full dimensional space, and was near an optimal solution. This results in much faster convergence than learning entirely in the full dimensional space (Figure 3).

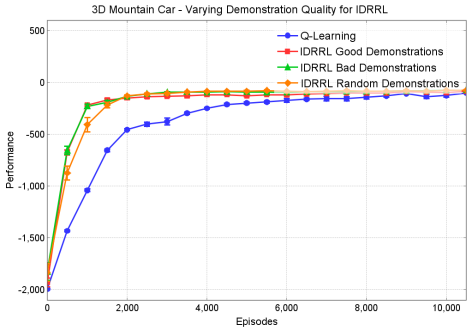


Figure 3: We compare Q-Learning to Q-Learning when combined with IDRRL with demonstrations of varying quality. IDRRL converged at the same speed to the optimal solution when given good, bad or random demonstrations. In Mountain Car 3D IDRRL is robust to suboptimal demonstrations.

To analyze the robustness of the approach, we varied the quality of demonstration data as well as the amount. Figure 3 shows the relationship between demonstration quality and performance. There is no significant difference between good, bad, and random demonstrations. This is due to the equal weighing of all state variables by PCA.

To test the robustness of IDRRL we also varied the amount of demonstration data. For this analysis, we used random demonstrations and varied the amount of demonstration data used between 1,000 and 25,000 demonstration states. We only test with random demonstrations, since demonstration quality was not a factor in performance for Mountain Car 3D. None of the random demonstrations reached the goal state, and each demonstration trajectory was approximately 2,000 samples. The experiment with 1,000 demonstration points converged slightly slower, and there were no significant difference between 10,000 and 25,000 states.

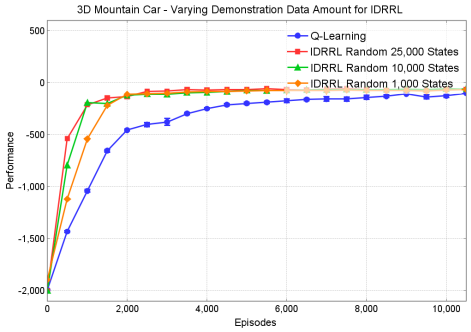


Figure 4: In Mountain Car 3D, there is no significant difference in the performance of IDRRL when using 10,000 or 25,000 demonstration states.

IDRRL scales well with the size of the state space. We modified Mountain Car 3D to add an additional fourth dimension. Mountain Car 4D has 6 continuous states and 7 actions. There are position and velocity states and acceleration/deceleration actions for each of the $x$, $y$ and $z$ dimensions. The trends seen previously in Mountain Car 3D are emphasized with additional states (Figure 5). By using IDRRL, the agent converges much faster to a good solution.

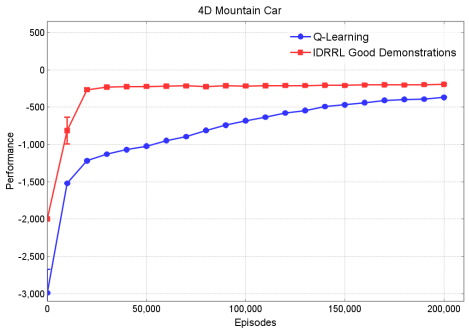


Figure 5: In Mountain Car 4D, Q-Learning with IDRRL scales well with the size of the state space.

## 5.2 Swimmers

In our formulation of Swimmers we used 32 tiles and a $10^n$ tiling, where $n$ is the number of state variables. There are 6 state variables and 9 actions in the 3-link swimmer and 12 state variables and 243 actions per state in the 6-link swimmer. Similar to our Mountain Car experiments, we gather demonstrations by learning in the domain with Q-Learning and collecting demonstrations. These demonstrations represent the best policies found with Q-Learning. In swimmers, the performance is measured by how far the swimmer has moved to the right ($\Delta x$). In the 3-link swimmer problem, the best policy performed well, but in the 6-link problem the policies were highly suboptimal due to the large state and action space.

The Swimmers Domain is a more complex system than Mountain Car. It includes complex physics with a large state and action space. This is where IDRRL can greatly increase learning performance when added to an existing algorithm. When adding IDRRL to Q-Learning, the new learning algorithm can learn a controller for a 3-link Swimmer faster (Figure 6).

IDRRL scales effectively to a state space of 12 dimensions with 243 possible actions at each state (Figure 7). By initially projecting the state space onto a hyperplane, IDRRL samples many of the actions in a smaller space. It then generalizes what was learned in the low-dimensional space to the higher-dimensions. This generalization causes IDRRL to scale well with the size of both the state and action space.

# 6 Conclusion and Future Work

The DRRL and IDRRL frameworks improve the performance of an existing algorithm by combining the speed of low-dimensional learning and the expressiveness of the full state space. By projecting the state space onto a low-dimensional hyperplane, our methods are able to represent a complex state with only a few state variables. Then, by incrementally transferring the knowledge from low-dimensional spaces into higher-dimensional ones, IDRRL learns good policies faster than the reinforcement learning algorithm alone.

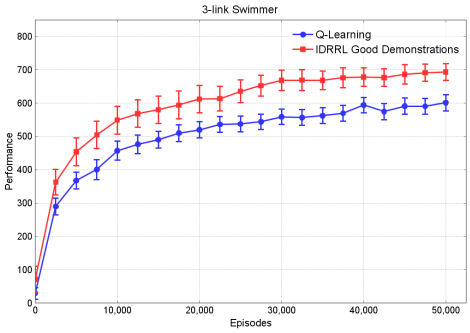


Figure 6: Swimmers is a high-dimensional problem with many state dimensions. Q-Learning with IDRRL converges faster than standard Q-Learning when learning a control policy for a 3-link swimmer with 6 state dimensions and 9 actions.

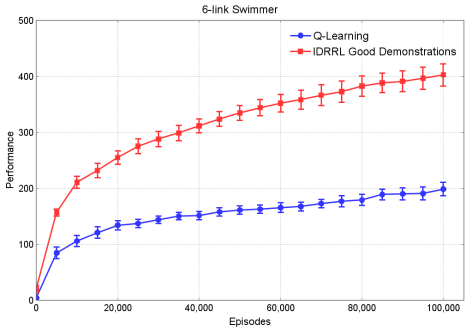


Figure 7: A 6-link swimmer is difficult to control due to an extremely large state and action space. Q-Learning with IDRRL learns a swimmers control policy quickly in 12 state dimensions and 243 actions per state.

In future work we would like to analyze task demonstrations for learning. We hypothesize that by using IDRRL, the robot will learn efficiently and be robust to demonstration quality, a classic issue in learning from demonstration literature (Argall et al. 2009). Therefore, we would like to test this approach on a robot platform. We also want to give the robot agent additional state information and see if it can efficiently leverage the new states and learn an effective policy. However, as it stands, too many iterations are needed to be practical on robots. This is a function of the reinforcement learning algorithm known. In future work, we will use R-Max (Jong and Stone 2007) to greatly speed up learning. By combining R-MAX with our IDRRL approach, we believe we can greatly improve performance in robot applications.

## References


Albus, J. S. 1981. Brains, behavior, and robotics. Byte Books.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Bitzer, S.; Howard, M.; and Vijayakumar, S. 2010. Using dimensionality reduction to exploit constraints in reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 3219–3225.

Brafman, R. I., and Tennenholtz, M. 2003. R-MAX - a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning. *Journal of Machine Learning Research* 3:213–231.

Cobo, L. C.; Zang, P.; Isbell, C. L.; and Thomaz, A. L. 2011. Automatic state abstraction from demonstration. In *Proceedings of the 22nd Second International Joint Conference on Articial Intelligence*.

Colome, A.; Neumann, G.; Peters, J.; and Torras, C. 2014. Dimensionality reduction for probabilistic movement primitives. In *2014 IEEE-RAS International Conference on Humanoid Robots*, 794–800.

Coulom, R. 2002. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Ph.D. Dissertation, Institut National Polytechnique de Grenoble.

Curran, W. J.; Agogino, A.; and Tumer, K. 2013. Addressing hard constraints in the air traffic problem through partitioning and difference rewards. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems*, 1281–1282.

Dutech, A.; Edmunds, T.; J. Kok, M. L.; Littman, M.; Riedmiller, M.; Russell, B.; Scherrer, B.; Sutton, R.; Timmer, S.; Vlassis, N.; White, A.; ; and Whiteson, S. 2005. NIPS workshop: Reinforcement Learning Benchmarks and Bake-offs II.

Haykin, S. 1998. *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd edition.

Jolliffe, I. 2002. *Principal Component Analysis*. Springer Series in Statistics. Springer.

Jong, N. K., and Stone, P. 2007. Model-based exploration in continuous state spaces. In *The Seventh Symposium on Abstraction, Reformulation, and Approximation*.

Jordan, M., and Jacobs, R. A. 1993. Hierarchical mixtures of experts and the EM algorithm. In *Proceedings of 1993 International Joint Conference on Neural Networks*, volume 2, 1339–1344 vol.2.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4(1):237–285.

Kober, J., and Peters, J. 2012. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*. Springer Berlin Heidelberg. 579–610.

Kormushev, P.; Calinon, S.; and Caldwell, D. G. 2013. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics* 2(3):122.

Mann, T. A., and Choe, Y. 2012. Directed exploration in reinforcement learning with transferred knowledge. *JMLR Workshop and Conference Proceedings: EWRL* 24:59–76.

Martin H., J.; de Lope, J.; and Maravall, D. 2009. The kNN-TD Reinforcement Learning Algorithm. In *Methods and Models in Artificial and Natural Computation. A Homage to Professor Miras Scientific Legacy*, volume 5601 of *Lecture Notes in Computer Science*. 305–314.

Pechenizkiy, M.; Puuronen, S.; and Tsymbal, A. 2003. Feature extraction for classification in knowledge discovery systems. In *Knowledge-Based Intelligent Information and Engineering Systems*, volume 2773 of *Lecture Notes in Computer Science*. Springer. 526–532.

Peters, J., and Schaal, S. 2006. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225.

Reddy, C., and Tadepalli, P. 1997. Learning goal-decomposition rules using exercises. In *Proceedings of the 14th International Conference on Machine Learning*, 278–286.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement Learning in Finite MDPs: PAC Analysis. *Journal of Machine Learning Research* 10:2413–2444.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Journal of Artificial Intelligence* 112(1-2):181–211.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10:1633–1685.

Taylor, M. E.; Kulis, B.; and Sha, F. 2011. Metric Learning for Reinforcement Learning Agents. In *Proceedings of the 2011 International Conference on Autonomous Agents and Multiagent Systems*.

Taylor, M. E.; Whiteson, S.; and Stone, P. 2007. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems*.

Thomaz, A. L., and Breazeal, C. 2006. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 1000–1005.

Turk, M., and Pentland, A. 1991. Face recognition using eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 586–591.

Whiteson, S.; Taylor, M. E.; and Stone, P. 2007. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin.