2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Stable Reinforcement Learning with Autoencoders for Tactile and Visual Data

Herke van Hoof[1], Nutan Chen[2], Maximilian Karl[2], Patrick van der Smagt[2,3], Jan Peters[1,4]

***Abstract*— For many tasks, tactile or visual feedback is helpful or even crucial. However, designing controllers that take such high-dimensional feedback into account is non-trivial. Therefore, robots should be able to learn tactile skills through trial and error by using reinforcement learning algorithms. The input domain for such tasks, however, might include strongly correlated or non-relevant dimensions, making it hard to specify a suitable metric on such domains. Auto-encoders specialize in finding compact representations, where defining such a metric is likely to be easier. Therefore, we propose a reinforcement learning algorithm that can learn non-linear policies in continuous state spaces, which leverages representations learned using auto-encoders. We first evaluate this method on a simulated toy-task with visual input. Then, we validate our approach on a real-robot tactile stabilization task.**

I. Introduction and Related Work

To minimize the human engineering effort when teaching a robot new skills, robots should be able to learn through trial and error. Such learning can be formalized in the reinforcement learning framework [1], [2]. Reinforcement learning (RL) has been shown to be useful in optimizing, among others, reaching, grasping, and manipulation skills [3]–[5]. These tasks rely on high-dimensional sensor inputs: visual feedback is crucial in locating objects [3], whereas tactile feedback is critical for robustness to perturbations and inaccuracies [3], [6]–[9]. Such high-dimensional sensory inputs pose a major challenge to RL algorithms.

In robotics tasks, reinforcement learning methods have addressed this challenge by relying on human-designed features to represent policies or value functions [1], [2], [10]. For example, [3] used the center of mass extracted from visual images. A popular policy parametrization uses dynamic motor primitives to generate trajectories to be tracked [4], [5], [11]–[14]. Feedback can be integrated using e.g. task-specific controllers [5], anomaly detectors [13], trajectories in sensor space [11], or by adding attractors [14]. Designing such sensory features is non-trivial in complex and high-dimensional sensory domains. These task-specific features are strongly dependent on manual tuning. We propose to instead learn policies in a widely applicable form from high-dimensional sensory inputs.

[1]IAS institute, TU Darmstadt.
{hoof, peters}@ias.tu-darmstadt.de
[2]Faculty for Informatics, TU München.
{nutan, karlma}@in.tum.de
[3]fortiss, TUM Associate Institute.
[4]Max Planck Institute for Intelligent Systems.
This work has been supported in part by the TACMAN project, EC Grant agreement no. 610967, within the FP7 framework programme. Part of this research has been made possible by the provision of computing time on the Lichtenberg cluster of the TU Darmstadt.

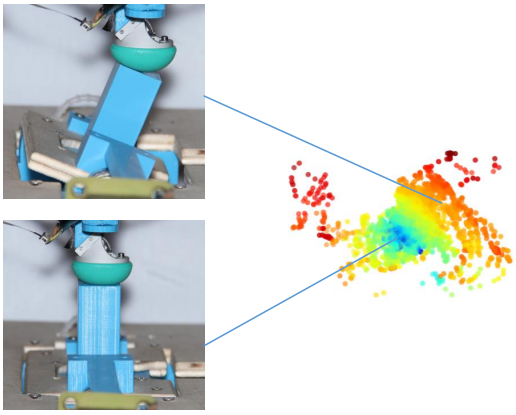


Fig. 1. A 5-DoF robot with 228-dimension sensor data learns to manipulate using a learned three-dimensional feature space. The learned latent representation of the tactile data is plotted on the right. Different contact states, shown on the left, yield different latent-space values. The compact latent space is used as a feature basis for reinforcement learning.

A further challenge in robot reinforcement learning is that datasets obtained with robots are generally quite small. In previous work [15], [16], a reinforcement learning method was proposed that combines non-parametric policies for nonlinear sensor data with limited policy updates to avoid overfitting and optimization bias on small datasets. However, non-parametric methods usually rely on distances between input points. In high dimensional sensor spaces, low-intensity noise or small illumination changes may cause large displacements, making distances less meaningful.

Autoencoders [17]–[19] have shown to be very successful in learning meaningful low-dimensional representations of (robot) movement data [20]–[22]. Therefore, we propose using the representation learned by such autoencoders as input for reinforcement learning of policies of non-task specific form. For example, Fig. 1 shows a robot manipulating an object using the learned latent space.

Many previous researchers have applied neural-network based methods to reinforcement learning problems with high-dimensional sensor data. One possibility is to use a neural network as a forward model for the dynamics and possibly the reward function. Optimal control methods can then be used to obtain optimal action. Recently, this approach was used to find optimal policies when only simulated images of the system are given [23], [24]. The disadvantage of such methods is that gradients have to be propagated through a number of connections that is the product of the planning horizon and the network depth.

Instead, neural networks can also be used as value function. Neural networks as state-action value functions are used in approaches such as neural fitted Q-iteration [25], [26] and deep Q networks [27]. A possible disadvantage of such methods is, that small changes in the Q function can have a big effect of the maximizing policy, which can lead to unstable behavior in on-policy learning.

As an alternative, some methods learned a neural network policy instead of, or in addition to, a neural network value function. Such policy search methods allow the policy to be adapted slowly, avoiding instabilities. For example, [28] used an evolutionary algorithm to search for network coefficients for a visual racing task in a compressed space, [29] learned neural-network policies using an actor-critic algorithm for learning dynamic tasks such as locomotion from pixel images, and [30] used trust-region policy optimization to optimize such policies to learn to play Atari games from pixel images as well as challenging locomotion tasks.

However, these methods tend to require on the order of a million sampled time steps, which might not be easy to obtain in a real-robot setting. For such real-robot tasks, Levine et al. [31] proposed using optimized trajectories in a low-dimensional space to train a neural network policy that directly uses raw image data as input. Convenient low-dimensional spaces are, however, not always provided.

Low-dimensional representations can alternatively be learned using autoencoders [17]–[19]. Such autoencoder are trained to reconstruct sensor inputs, as was done in [21] to learn a pendulum swing-up task directly from visual inputs. However, this approach does not ensure that the learned representations respects the task dynamics. For example, *states that are easily reachable from the current state but have a different visual representation, might be far apart in the learned feature space*. To address this issue, a smoothness penalty can be applied to the feature points, as was done in [22] to learn robot visuomotor tasks. More explicitly, the dynamics can be enforced to be linear in the latent space, as was done in the embed to control method [32].

Similar to these studies, we propose to use deep autoencoders to learn lower-dimensional features for reinforcement learning. Whereas [22] used a separate exploration controller using a simple state and reward function excluding the high-dimensional sensor space, we aim to learn feedback policies directly from the high-dimensional space. Compared to [32], who learned control policies in a single shot based on data under an exploration policy, we aim to learn iteratively on-policy. As the policy starts to generates more relevant samples, the learned feature representation can be improved. We want to train our state encoders in a way that respects the transition dynamics of the controlled system.

In our experiments, we will investigate two tasks: in simulation, we present a visual pendulum swing-up task. The simulated experiments will allow us to compare different variants in quantitative experiments. Thereafter, we show that the proposed method can acquire a real-robot manipulation task based on tactile data.

II. Policy Search with Learned Representations

Our approach consists of two steps: autoencoders are used to learn a suitable representation and non-parametric relative entropy policy search is used to obtain stable policy updates. In this section, we will first explain these two subsystems. Thereafter, we will discuss the set-up of our experiments, including the tasks we test our algorithms on.

A. Learning Representations using Autoencoders

An autoencoder [18] is a deep neural network that transforms the robot states $\mathbf{x} \in \mathbb{R}^d$ to a latent representation $\mathbf{z} \in \mathbb{R}^{d'}$ using an encoder, $\mathbf{z} = f_\theta(\mathbf{x}) = s(\mathbf{w}^T\mathbf{x} + b)$. The parameters θ consist of the weights $\mathbf{w}$ and $\mathbf{b}$, s is a nonlinear transition function, d and d' are the input vector and latent vector dimensions, respectively. Subsequently, the latent representation is mapped back through a decoder, $\mathbf{x}' = g_{\theta'}(\mathbf{z}) = s(\mathbf{w}'^T\mathbf{z} + \mathbf{b}')$ to reconstruction the input $\mathbf{x}'$. Parameters θ' include the weights of the decoder $\mathbf{w}'$ and $\mathbf{b}'$. The weights are restricted by $\mathbf{w}' = \mathbf{w}^T$ to regularize the network. The parameters are updated by gradient descent on the reconstruction error

$$\theta^\star, \theta'^\star = \arg\min_{\theta,\theta'} \frac{1}{n}\sum_{i=1}^{n} L\Big[\mathbf{x}^{(i)}, g_{\theta'}\big(f_\theta(\mathbf{x}^{(i)})\big)\Big], \quad (1)$$

in which L is a mean squared error loss function.

1) Denoising Autoencoder: A denoising autoencoder (DA, [17]) is a basic version of the autoencoder, in which noise is added to regularize learning. In the objective

$$\theta^\star, \theta'^\star = \operatorname*{argmin}_{\theta,\theta'} \frac{1}{n}\sum_{i=1}^{n} L\Big[\mathbf{x}^{(i)}, g_{\theta'}\big(f_\theta(\tilde{\mathbf{x}}^{(i)})\big)\Big], \quad (2)$$

the inputs to the encoder $\tilde{\mathbf{x}}$ are the training inputs $\mathbf{x}$ corrupted by adding random noise. The network is trained to reconstruct the uncorrupted $\mathbf{x}$.

2) Variational Autoencoder: The variational autoencoder (VAE, [19]) efficiently infers the unobserved latent variables of probabilistic generative models. The unobserved latent vectors $\mathbf{z}^{(i)}$ correspond to the observed vectors $\mathbf{x}^{(i)}$ in the dataset. As prior distribution of the latent space variables, an isotropic Gaussian $p^*(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ is used. For non-binary data, a standard choice for the decoder $p(\mathbf{x}|\mathbf{z})$ is a Gaussian, where

$$\begin{aligned}
\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \operatorname{diag}(\boldsymbol{\sigma}^2)),\\
\boldsymbol{\mu} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2,\\
\log \boldsymbol{\sigma}^2 &= \mathbf{W}_3\mathbf{h} + \mathbf{b}_3,\\
\mathbf{h} &= h(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) = \max(0, \mathbf{W}_1\mathbf{z} + \mathbf{b}_1), \quad (3)
\end{aligned}$$

in which the parameters μ, σ are given by a multi-layer perceptron parametrized by $\mathbf{W}$ and $\mathbf{b}$ jointly represented by θ. The activation function h is a rectified linear unit. $\{\boldsymbol{\mu}^{\text{enc}}, \boldsymbol{\sigma}^{\text{enc}}\}$ and $\{\boldsymbol{\mu}^{\text{dec}}, \boldsymbol{\sigma}^{\text{dec}}\}$ represent $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ for the decoder and encoder, respectively.

We would like to find parameters θ that optimize the marginal likelihood $p_\theta(\mathbf{x}^{(i)})$. As this objective is intractable

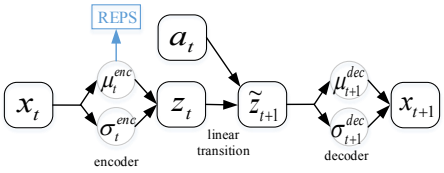


Fig. 2. Illustration of the VAE with dynamics. The network is trained such that the decoder output is close to the next state x_{t+1}. After training, the expected value of $\mathbf{z}_t$, μ_t^{enc}, is used as input to the reinforcement learning algorithms, which generates data used to update the network in turn.

for (3), we re-write the marginal likelihood as

$$\begin{aligned}\log p_\theta(\mathbf{x}^{(i)}) &= \log \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta^*(\mathbf{z}) d\mathbf{z} \\ &= D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}). \quad (4)\end{aligned}$$

In this equation, a parametric approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to $p_\theta(\mathbf{z}|\mathbf{x})$ is used as this term relies on an intractable integral. The Kullback-Leibler divergence D_{KL} is defined as $D_{\mathrm{KL}}(p||q) = \int p(\mathbf{x}) \log(p(\mathbf{x})/q(\mathbf{x}))\mathrm{d}\mathbf{x}$. The encoder $q_\phi(\mathbf{z}|\mathbf{x})$ has a similar structure as (3), but $\mathbf{z}$ and $\mathbf{x}$ are swapped and the weights and biases are in a different set of parameters ϕ. In (4), $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$ is a lower bound on the marginal likelihood

$$\begin{aligned}\mathcal{L}(\theta, \phi; x^{(i)}) =& E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] \\ &- D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z})). \quad (5)\end{aligned}$$

The first term can be interpreted as a reconstruction cost, which is approximated by sampling from $q_\phi(\mathbf{z}|\mathbf{x})$. The KL-divergence term D_{KL} quantifies the loss of information when the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ is used instead of $p_\theta(\mathbf{z})$.

The lower bound is optimized by stochastic backpropagation. As the reconstruction term is estimated through sampling, we compute the gradient through the sampling process of $q_\phi(\mathbf{z}|\mathbf{x})$ with the reparametrization trick $\mathbf{z} = y(\phi, \epsilon)$, where y is a function of ϕ with noise ϵ, as in [19].

3) VAE with Dynamics: We modified the VAE to take the transition dynamics into account, which we expect to yield better representations based on performance of similar networks proposed by [32], [33]. A linear layer is added between the encoder and the decoder (Fig. 2). It predicts the next latent state $\tilde{\mathbf{z}}_{t+1}$ from the latent state $\mathbf{z}_t$ and action $\mathbf{a}_t$

$$\tilde{\mathbf{z}}_{t+1} = [\mathbf{z}_t^T \ \mathbf{a}_t^T]\mathbf{w} + \mathbf{b}, \quad (6)$$

where $\mathbf{w}$ and $\mathbf{b}$ are parameters represented by θ_a. After that, the model reconstructs the next state $\mathbf{x}_{t+1}$ from $\tilde{\mathbf{z}}_{t+1}$. The transition layer is chosen to be linear to enforce control-affine dynamics, which are convenient for control tasks. The modified lower bound is defined as

$$\begin{aligned}\mathcal{L}(\theta, \theta_a, \phi; \mathbf{x}_t, \mathbf{a}, \mathbf{x}_{t+1}) =& E_{q_\phi(\mathbf{z}|\mathbf{x}_t)}[\log p_\theta(\mathbf{x}_{t+1}|\mathbf{z}, \mathbf{a})] \\ &- D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}_t) \| p_\theta(\mathbf{z})), \quad (7)\end{aligned}$$

taking the dynamics into account.

4) DA with Dynamics: Similar to the VAE with dynamics, the denoising autoencoder with dynamics takes transitions into account and has an additional layer as described in (6), written as $\tilde{\mathbf{z}}_{t+1} = h_{\theta_a}(\mathbf{z}_t, \mathbf{a})$, in the latent space. The weights are updated by optimizing

$$[\theta^\star, \theta'^\star, \theta_a^\star] = \underset{[\theta, \theta', \theta_\mathbf{a}]}{\operatorname{argmin}} \sum_{t=1}^{n-1} \frac{L[\mathbf{x}_{t+1}, g_{\theta'}(h_{\theta_a}(f_\theta(\tilde{\mathbf{x}}_t), \mathbf{a}))]}{n-1}.$$

5) Latent Space Updates with On-Policy Samples: To represent data most relevant for the current policy, we consider a variant where we update the autoencoder using samples from the most recent policies. We can either use data from the most recent iterations only, or, when little data is available, we can give recent data a higher weight in the loss function. We will specify the data used for re-training the latent space for each experiment individually.

B. Non-Parametric Policy Updates

In this section, we give a brief overview of non-parametric relative entropy policy search (NP-REPS, introduced in [15], [16]). First, we will introduce the notation that we will use in the rest of this section. Thereafter, we will show how NP-REPS can be used to obtain a re-weighting of sampled state-action pairs to provide stable updates policy updates. Finally, we explain how the policy represented by those weighted samples can be generalized to the entire state space.

1) Notation: To use reinforcement learning (RL) methods, we represent the learning problem as a Markov Decision Process (MDP). In a MDP, an agent in state $\mathbf{s}$ selects an action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$ according to a policy π and receives a reward $\mathcal{R}_\mathbf{s}^\mathbf{a} \in \mathbb{R}$. We assume continuous state-action spaces: $\mathbf{s} \in \mathcal{S} = \mathbb{R}^{D_\mathrm{s}}$, $\mathbf{a} \in \mathcal{A} = \mathbb{R}^{D_\mathrm{a}}$. The stationary distribution $\mu_\pi(\mathbf{s})$ under policy π is the state distribution for which $\int_\mathcal{S}\int_\mathcal{A} \mathcal{P}_{\mathbf{ss}'}^\mathbf{a} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{da}\,\mathbf{ds} = \mu_\pi(\mathbf{s}')$, under transition dynamics $\mathcal{P}_{\mathbf{ss}'}^\mathbf{a} = p(\mathbf{s}'|\mathbf{a}, \mathbf{s})$. The goal of a reinforcement learning agent is to choose a policy such that the joint state-action distribution $p_\pi(\mathbf{s}, \mathbf{a}) = \mu_\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})$ maximizes the average reward $J(\pi) = \int_\mathcal{S}\int_\mathcal{A} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}_\mathbf{s}^\mathbf{a}\mathbf{da}\,\mathbf{ds}$.

2) Non-parametric REPS: Traditional methods for reinforcement learning have no notion of the sampled data, and can thus suffer from optimization bias and overfitting, which can lead to oscillations and divergence [34], [35]. To prevent these problems, the divergence between subsequent policies [35] or state-action distributions [34] might be bounded. This latter method, relative entropy policy search (REPS), works well in practice, and needs relatively few samples such that learning on the real robot is feasible. REPS is defined by the optimization problem

$$\max_{\pi, \mu_\pi} \quad \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}_\mathbf{s}^\mathbf{a}\mathbf{da}\mathbf{ds}, \quad (8)$$

$$\text{s. t.} \quad \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{da}\mathbf{ds} \quad = 1, \quad (9)$$

$$\forall s'. \quad \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{P}_{\mathbf{ss}'}^\mathbf{a}\mathbf{da}\mathbf{ds} = \mu_\pi(\mathbf{s}'), \quad (10)$$

$$D_{\mathrm{KL}}(\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) || q(\mathbf{s}, \mathbf{a})) \ \leq \epsilon. \quad (11)$$

Equation (8) states that the joint state-action distribution should maximize the expected average reward, (9) constrains $\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ to be a probability distribution, and (10) forces the optimizer to respect the system dynamics $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}$. Furthermore, (11) specifies an additional bound on the KL divergence between the proposed state-action distribution and sampling distribution q that ensures smooth policy updates. The solution to the optimization problem obtained through Lagrangian optimization is given by

$$p_\pi(\mathbf{s},\mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) \propto q(\mathbf{s},\mathbf{a}) \exp\left(\frac{\delta(\mathbf{s},\mathbf{a},V)}{\eta}\right), \text{with}$$
$$\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}^{\mathbf{a}}_{\mathbf{s}} + \mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] - V(\mathbf{s}) \quad (12)$$

where $V(\mathbf{s})$ and η denote Lagrangian multipliers [34]. The Lagrangian multiplier $V(\mathbf{s})$ is a function of $\mathbf{s}$ and resembles a value function, so that δ can be interpreted as the Bellman error. Therefore, (12) can be interpreted as a re-weighting of the old policy with the soft-max of the advantage function. Thus, the probability of choosing actions that yield higher expected future rewards will be increased. We assume the function $V = \sum_{\tilde{\mathbf{s}}\in\tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k(\tilde{\mathbf{s}},\cdot)$, where $\tilde{\mathcal{S}}$ is the set of sampled states [15]. That is, V is member of a reproducing kernel Hilbert space specified by a kernel k. For characteristic kernels, such as the squared exponential (Gaussian) kernel, this assumption allows for non-linear and highly flexible V that can adapt to the complexity of the dataset.

The multiplier V is completely determined by the embedding strengths $\boldsymbol{\alpha}$. These embedding strengths and η are obtained through minimization of the dual function

$$g(\eta,\boldsymbol{\alpha}) = \eta\epsilon + \eta \log\left(\sum_{i=1}^{n} \frac{1}{n} \exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,\boldsymbol{\alpha})}{\eta}\right)\right), \quad (13)$$

where the samples $(\mathbf{s}_i,\mathbf{a}_i)$ are drawn from $q(\mathbf{s},\mathbf{a})$. To calculate the Bellman error δ, the transition distribution is required. As this distribution is usually not known, δ needs to be approximated. Earlier work [15] showed that this approximation can be done efficiently using kernel methods.

We set the reference distribution q to the state-action distribution induced by previous policies. Learning starts with samples from an initial wide, uninformed exploration policy $\tilde{\pi}_0$. The variance of policies typically shrinks after every iteration, such that the policy converges to a (locally) optimal policy. Note that this method only learns about the reward function and system dynamics through samples, and therefore does not need an analytical dynamics or kinematics model. The kernel bandwidths for k_{s} are tuned to optimize learning performance. The hyper-parameter λ and the bandwidths of k_{sa} are set to minimize the two-fold cross-validation prediction error of the embedding of $\mathbf{s}'$.

3) Fitting Generalizing Control Policies: Equation (12) can only be computed for the sampled state-action pairs, since the reward $\mathcal{R}^{\mathbf{a}}_{\mathbf{s}}$ is only observed at those samples. We obtain a control policy that generalizes to *all states* by conditioning. We consider policies $\tilde{\pi}(\mathbf{a}|\mathbf{s},\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$ linear in features $\boldsymbol{\phi}(s)$. We choose those features such that their inner product approximates a Gaussian kernel, as suggested by [36].

As the parameters $\boldsymbol{\theta}$ are not known beforehand, we place a Gaussian prior over these parameters. We subsequently condition on the sampled actions to obtain a generalizing policy. We have to consider, however, that the actions were drawn from the previous distribution $q(\mathbf{a},\mathbf{s})$, not the desired distribution $p_\pi(\mathbf{a},\mathbf{s})$. Therefore, we include importance weights w_i, and obtain a posterior distribution over parameters

$$p(\boldsymbol{\theta}|\mathbf{a}_1,\mathbf{s}_1,\ldots,\mathbf{a}_n,\mathbf{s}_n) \propto p(\boldsymbol{\theta}) \prod_{i=1}^{n} \tilde{\pi}(\mathbf{a}|\mathbf{s},\boldsymbol{\theta})^{w_i},$$

with samples $(\mathbf{s}_i,\mathbf{a}_i) \sim q(\mathbf{a},\mathbf{s})$ and importance weights

$$w_i = p_\pi(\mathbf{s}_i,\mathbf{a}_i)/q(\mathbf{s}_i,\mathbf{a}_i) = \exp(\delta(\mathbf{s}_i,\mathbf{a}_i,V^*)/\eta^*),$$

since p_π is of the form given in (12). Hyper-parameters are automatically set by maximizing the marginal likelihood using cross-validation.

III. Experimental Set-Up and Results

We perform two experiments with high-dimensional sensory input. The first experiment considers a simulated pendulum swing-up task with visual input, and the second considers a real-robot manipulation task with tactile input. After describing the set-ups, we will present our results.

A. Experimental Set-Ups

The reinforcement learning agent starts by exploring its environment using its stochastic policy. After every iteration of data gathering, the learned model and the policy of the agent are updated. To bootstrap the model and the policy, the agent is given 30 initial roll-outs using a random exploratory policy (45 roll-outs in the real-robot experiment). To avoid excessive computations, we only keep data from the last three iterations[1].

For the simulated experiment, after each update, the learning progress is evaluated by running the learned policy on 100 roll-outs with a fixed random seed. This data is *not* used for learning. For the real-robot experiment, performance of the training roll-outs is reported. The KL bound ϵ of the REPS algorithm was set to 0.5 in our experiments.

Simulated visual pendulum swing-up: In this experiment, we simulate a pendulum that has to be swung up and balanced at the upright position, based on visual input. We reproduce the set-up of [15], where the pendulum has length $l = 0.5\,\mathrm{m}$, mass $m = 10\,\mathrm{kg}$, and friction coefficient $k = 0.25\,\mathrm{Ns}$. The action a is a torque applied at the pivot, with a maximum of $30\,\mathrm{Nm}$ such that the pendulum cannot be swung up from the downward position directly.

After each time step of $0.05\,\mathrm{s}$, the agent receives a reward according to $r(s,a) = -10\theta^2 - 0.1\dot{\theta}^2 - 10^{-3}a^2$. A rather high level of additive noise is applied to the controls (with a standard deviation of $4.5\,\mathrm{Nm}$). We ended the roll-out after each time-step with a probability of 0.02, resulting in an

[1]As a consequence, the reference distribution q is a mixture of the previous three state-action distributions in our experiments.

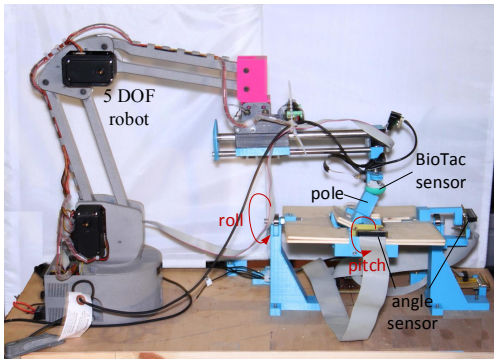


Fig. 3. Set-up of the tactile control experiment. The five-DoF robot touches a blue pole that can rotate about two axes using the green BioTac fingertip sensor. The pitch and roll of the pole are measured to provide a feedback signal, but not used in the control policy.

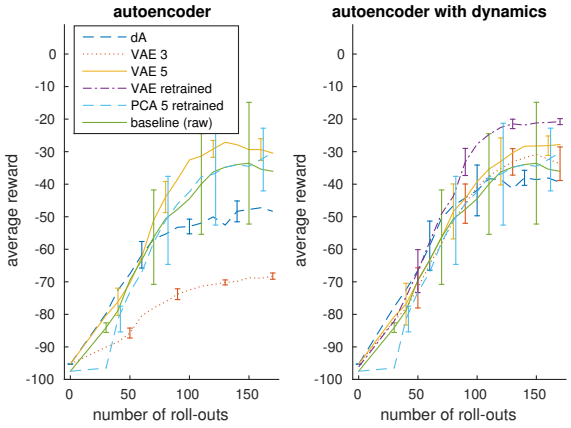


Fig. 4. Comparison of learning curves for the pendulum task using representations learned with different types of autoencoders. Shown are results of denoising autoencoders (DA) with 10 latent variables, and of variational autoencoders (VAE) with three or five latent variables. For the best category of encoders, 'VAE 5', we also compare to a version that is retrained with every policy update. Error bars show the sample standard deviation. Averages are calculated over five independent runs, using a single learned feature space per encoder type. The 'retrained' encoder is retrained independently for each run. Rollouts contained 50 time steps, on average.

effective discount factor $\gamma = 0.98$. For this experiment, we used 10 roll-outs per iteration.

The agent only has access to images of the pendulum. We render an image of the pendulum in its current state, as well as a difference image between visual representations of the pendulum in its current and previous state (to provide a notion of angular velocity). We blur the image and difference image with a Gaussian filter to enhance the spatial generalizability of the kernel. This filter has a standard deviation of 20% of the image width. Both images are resized to 20×20 pixels and concatenated into a vector with 800 features. We choose squared-exponential kernels for all variables.

The denoising autoencoder (DA) has one input layer, five hidden layers and one output layer. The number of neurons per layer are 800, 120, 50, d', 50, 120 and 800 respectively, where d' is the number of latent dimensions. We set the corruption level to 0.2, which means 20% of the image is corrupted in every training image. The DA with dynamics uses the same parameters but uses one more layer, as explained in Sec. II-A.4. For the VAE, we set the number of neurons per layer to 800, 512, d', 512, and 800, respectively. All network parameters where chosen to minimize the reconstruction error. The VAE with dynamics has the same architecture, except the layer from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}$, as shown in Fig. 2. The encoder networks in both VAE and VAE with dynamics used the square as the transfer function for the σ output. We furthermore evaluate a VAE that is retrained on the three most recent iterations with each policy update (note that, since (13) is based on samples only, re-training the VAE does not impact the KL constraint).

With this set-up, we want to answer two questions. First, we want to investigate which type of autoencoder is most suitable to find features for the reinforcement learning task. Secondly, we want to validate whether updating the autoencoder to represent the states visited by the most recent policy improves learning of the task.

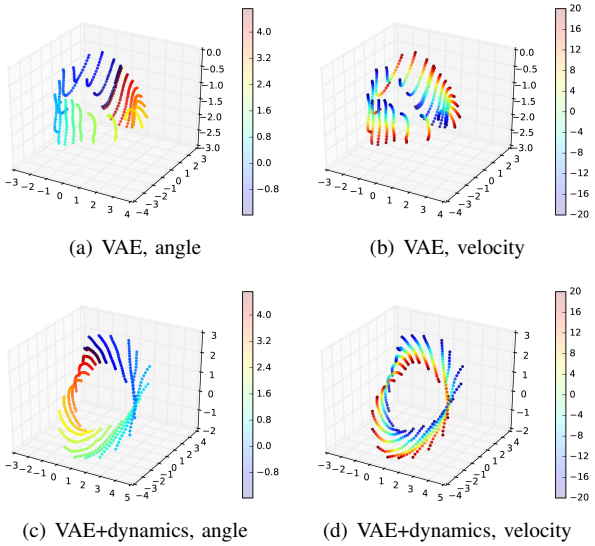
Real-robot tactile control experiment: In this experiment, a 5 degree of freedom robot manipulates a pole through a SynTouch BioTac tactile sensor on the end-effector (see Fig. 3). The pole is on a platform which is able to rotate in roll (α) and pitch (β), measured in degrees. Two angle sensors are mounted on the platform to measure the pole angles. The initial position of the pole is a 5° rotation from the central position with a uniformly random angle. The task for the robot is to move the pole to the stable center position where α and β are zero degree by manipulating the top of the pole. The reward function defining this upright pole position as a goal is $r(s,a) = 60(\exp(-(\alpha^2+\beta^2)/60) - 1)$. The reset probability is set to 0.05 (equivalent to $\gamma = 0.95$). In this experiment, we use 15 roll-outs per iteration.

The input to the neural network is a 12-time step window of the 19 electrodes of the BioTac sensor, yielding 228-dimensional input data. We used a hidden layer with 512 neurons and a feature layer with three neurons, chosen based on reconstruction error. In this experiment, the σ values of the decoder networks are constant, independent of $\mathbf{z}$. Actions consist of an increment in forward-backward and left-right position. The desired position is kept constant during the 33 ms time window. The control frequency is 2.8 kHz.

As the data size recorded on the real robot is relatively small, sampled data from all previous iterations is used to retrain the autoencoder. Recent data is more relevant than older data, and therefore errors on recent data were given a bigger weight in the loss function (triple weight for the most recent iteration, and double for the iteration before that).

B. Results of the Visual Pendulum Swing-up Experiments

In the first evaluation on the simulated visual pendulum swing-up task, we compared different types of autoencoders and inspect the effect of re-training the encoder on the most recently sampled data. For the evaluation, we use five independent trials for each feature representation. The results of this experiment are shown in Fig. 4. Compared to the denoising autoencoder, most variational autoencoders



(a) VAE, angle (b) VAE, velocity

(c) VAE+dynamics, angle (d) VAE+dynamics, velocity

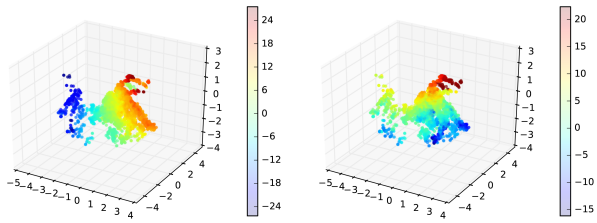
Fig. 5. Visualization of the learned feature spaces by the VAE with 3 latent features. The graph shows the three-dimensional feature space, represented by the x, y and z axis. The color encodes the pendulum angle (left column) or velocity (right column). The latent space was learned using the variational autoencoders (top row), or the modified encoders with system dynamics (bottom row). The angle and velocity are not directly available to the learning algorithm, but are used to calculate the learner's reward. All learned structures reflect the periodic nature of the angle variable.

provide representations that yield higher average rewards when used with reinforcement learning compared to using the raw sensor data, a truncated PCA with five components, or the denoising autoencoder in our tasks. An exception is the variational autoencoder with three latent features without additional dynamics. The learned representation for this type of encoder is presented in Fig. 5. Without the dynamics information, points which are regularly spaced in the original space are not so in the learned feature space. The encoder that forces the dynamics to be linear learned a more regular feature space. In the other cases, the encoders with dynamics also tended to yield slightly better representations.

Figure 4 also shows the performance based on features that are updated as the policy improves. These updates should intuitively ensure that the variational autoencoder focuses on representing the states visited by the current policy. In our experiment, updating the autoencoder this way improves the learning progress. As training the encoder is a computationally expensive procedure, we only performed a this evaluation for the most promising encoder type.

C. Results of the Tactile Manipulation Robot Experiment

As shown in the visual pendulum experiment, retrained VAE with dynamics yielded the best representation. Therefore, this encoder was chosen for the real robot experiment. The learned representation is shown in Fig. 6. The results of reinforcement learning of the tactile manipulation task with or without the encoder are shown in Fig. 7. With the learned representation, RL progress is smooth and stable. The final policy brings the pole within 1° of the desired location in rollouts with at least 10 time steps, on average.



(a) roll (b) pitch

Fig. 6. Latent space for BioTac sensor. The x, y and z axis represent the latent values, the samples are colored according to roll and pitch of the platform. The visualization shows that in the learned feature space, the pitch and roll components are perpendicular to each other.

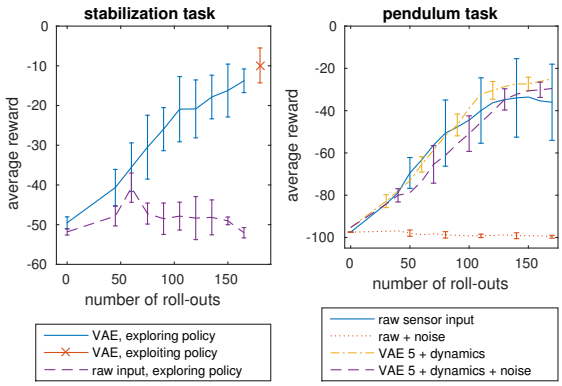


Fig. 7. Left: Learning progress on the real robot tactile manipulation task. During learning, performance of the learned policy including the learned exploration term is shown. After learning, we evaluate the learned mean policy without exploration. Error bars show the sample standard deviation. Averages are calculated over five independent runs, with independently learned feature encoders. Rollouts contained 20 steps, on average. Right: On raw sensor data, the learner is very sensitive to noise. In the simulated experiment, performance collapses for noisy raw sensor inputs, but not when the learned representation is used.

The performance of a baseline policy on the raw data directly is very poor. A possible cause for this poor performance is that the sensor data is too noisy.

To better understand the behavior of the system in the presence of observation noise, we evaluated the performance of reinforcement learners on the simulated visual pendulum swingup with low-intensity per-pixel noise (about 1% of the maximum image value). Figure 7 shows the performance on the noisy pictures, compared to the performance on the original task. When the images are corrupted with a small amount of noise, the learner using raw images seems unable to learn the task.

IV. Discussion and Conclusion

In this paper, we have introduced a reinforcement learning system consisting of two parts: an autoencoder that represents high-dimensional sensor representations in a compact space, and a reinforcement learner that is able to learn stable, non-linear policies. These stable updates allow on-policy learning with relatively small batches of data, making it possible to adapt the neural encoding during learning.

In our experiments, first we compared different types of autoencoder on a simulated visual task. We found that better results were obtained for this task with variational autoencoders compared to the more traditional de-noising autoencoders. Modifying the objective to reproduce the system dynamics, rather than encode individual input patterns, also tended to improve reinforcement learning performance. Re-training the encoders on the state distribution induced by the policy markedly improved performance. In this case, the encoder objective incurs the biggest loss for states that are most relevant to the policy, and therefore, such states are likely to be represented especially well in the learned feature space.

In a second set-up, we considered a real-robot manipulation task based on tactile feedback. In this scenario, we showed that the robot was able to learn a policy that manipulates and stabilizes a platform. Rather then using joint encoders or handcrafted features, our algorithm learned the tasks based on complex and high dimensional tactile representations.

For both tasks, we noticed that learning from the raw data tends to perform especially poorly in the presence of noise. Distances in the learned feature space are not distorted by noise as much as those in the image space, as the learned feature representation tends to average over multiple input channels. Thus, reinforcement learning on the learned feature space is still successful for the noisy task.

We consider this real-robot task to be a first step towards more complex manipulation tasks, where tactile feedback has the potential to improve robustness with respect to perturbances and inaccuracies. We are currently working on learning manipulation policies on multi-fingered robotic hands. In future work, we plan to investigate efficient exploration strategies that are critical for success in this domain.

ACKNOWLEDGMENT

We are very grateful to Justin Bayer, Sebastian Urban and Christian Osendorfer for valuable suggestions concerning this work.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robotics Research*, vol. 11, no. 32, pp. 1238–1274, 2013.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artificial Intell. Research*, vol. 4, 1996.

[3] T. Lampe and M. Riedmiller, "Acquiring visual servoing reaching and grasping skills using neural reinforcement learning," in *IJCNN*, 2013.

[4] O. Kroemer, C. Daniel, G. Neumann, H. van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *ICRA*, 2015.

[5] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *IROS*, 2011.

[6] H. Zhang and N. Chen, "Control of contact via tactile sensing," *Trans. Robotics and Automation*, vol. 16, no. 5, pp. 482–495, 2000.

[7] L. Jentoft, Q. Wan, and R. Howe, "Limits to compliance and the role of tactile sensing in grasping," in *ICRA*, 2014, pp. 6394–6399.

[8] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones, "Contact-reactive grasping of objects with partial shape information," in *IROS*, 2010.

[9] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories," in *Humanoids*, 2012.

[10] P. L. Bartlett, "An introduction to reinforcement learning theory: Value function methods," in *Advanced lectures on Machine Learning*. Springer Berlin Heidelberg, 2003, pp. 184–202.

[11] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," in *IROS*, 2014, pp. 3368–3375.

[12] O. Kroemer, R. Detry, J. Piater, and J. Peters, "Combining active learning and reactive control for robot grasping," *Robotics and Autonomous Syst.*, vol. 58, no. 9, pp. 1105–1116, 2010.

[13] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *ICRA*, 2011.

[14] J. Kober, B. Mohler, and J. Peters, "Learning perceptual coupling for motor primitives," in *IROS*, 2008, pp. 834–839.

[15] H. van Hoof, J. Peters, and G. Neumann, "Learning of non-parametric control policies with high-dimensional state features," in *AISTATS*, 2015.

[16] H. van Hoof, T. Hermans, G. Neumann, and J. Peters, "Learning robot in-hand manipulation with tactile features," in *Humanoids*, 2015.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *ICML*, 2008, pp. 1096–1103.

[18] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *ICLR*, 2014.

[20] N. Chen, J. Bayer, S. Urban, and P. van der Smagt, "Efficient movement representation by embedding dynamic movement primitives in deep autoencoders," in *Humanoids*, 2015.

[21] J. Mattner, S. Lange, and M. Riedmiller, "Learn to swing up and balance a real pole based on raw visual input data," in *ICONIP*, 2012.

[22] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *ICRA*, 2016.

[23] J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth, "Data-efficient learning of feedback policies from image pixels using deep dynamical models," ArXiv, Tech. Rep., 2015.

[24] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," ArXiv, Tech. Rep. 1502.02251, 2015.

[25] S. Lange, M. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *IJCNN*, 2012.

[26] M. Riedmiller, "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *ECML*, 2005.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[28] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Annu. Conf. Genetic and Evolutionary Computation*, 2013.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv, Tech. Rep. 1509.02971, 2015.

[30] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *ICML*, 2015.

[31] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[32] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *NIPS*, 2015, pp. 2728–2736.

[33] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep Kalman filters," in *NIPS Advances in Variational Inference Workshop*, 2015.

[34] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *AAAI*, 2010, pp. 1607–1612.

[35] J. Bagnell and J. Schneider, "Covariant policy search," in *International Joint Conference on Artificial Intelligence*, 2003.

[36] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007, pp. 1177–1184.