Learning action features for structured exploration

Nicolas Heess, Gerhard Neumann, David Silver, Yee Whye Teh

Introduction High-dimensional control problems are a significant challenge for many learning algorithms. They suffer from the curse of dimensionality, and naïve exploration, e.g. by adding a small amount of uncorrelated noise to the action vector, can be highly inefficient (leading to "motor babbling") and even damaging for the plant. Knowing about the dependencies between action dimensions inherent to many control problems can guide our search through the high-dimensional space and greatly speed up learning. Many real world reinforcement learning applications therefore rely on action representations in the form of "action features" or "motor primitives" that capture domain specific knowledge and hence simplify the learning problem. In this work we are interested in how to *learn* such features. One area where there has recently been been much success in feature learning for a diverse sets of perceptual tasks (e.g. in computer vision or natural language processing) is the deep and unsupervised learning community. These works employ general architectures such as neural networks or related probabilistic formulations which are able to learn distributed and hierarchical representations of the data.

Here, we employ such a generic learning architecture in the form of a neural network in order to learn a low-dimensional representation of actions for a high-dimensional control problem, the octopus arm. In order to learn a diverse set of features we consider a multi-task setup with a small set of carefully chosen, relatively simple training tasks to learn a distributed representation of actions. These learned action features capture important correlations between the action dimensions. We propose to use these features in a probabilistic architecture that can take advantage of the learned dependencies performing structured exploration, and we demonstrate that this combination can significantly speed up learning of novel tasks.

Environment To illustrate our approach we consider a simulated octopus arm (Engel et al., NIPS 2005). The goal is to control the arm to move from an initial position to hit a target with its tip while avoiding obstacles. Our arm consist of $C = 8$ segments, each associated with a 8 dim. continuous state vector (i.e. $8C = 64$ state dimensions in total) and a 3 dim. binary action vector (two lateral muscles and a transversal muscles per segment; $3C = 24$ action dimensions in total).

Learning architecture We use a variant of the episodic natural actor critic algorithm (eNAC) proposed by (Peters & Schaal, Neurocomp. 2008). We parameterize the policy using a 2-layer sigmoidal neural network, with (continuous) input layer $\mathbf{s}$, binary hidden layer $\mathbf{h}$ (8 hidden units), and binary output layer $\mathbf{a}$ (see Fig. 1). In order to allow for structured exploration, taking into account correlations between the actions, we use stochastic hidden units so that the overall policy is given by $\pi^\theta(\mathbf{a};\mathbf{s}) = \sum_\mathbf{h} p(\mathbf{a},\mathbf{h}|\mathbf{s};\theta) = \sum_\mathbf{h} p(\mathbf{a}|\mathbf{h};\theta)p(\mathbf{h}|\mathbf{s};\theta)$. The conditional probabilities are given by $p(h_i = 1|\mathbf{s};\theta) = \sigma(W^S\mathbf{s} + \mathbf{b}_i)$ and $p(a_j = 1|\mathbf{h};\theta) = \sigma(W^A\mathbf{a} + c_j)$. Marginalization over the latent variables $\mathbf{h}$ allows the policy to capture correlations between the action dimensions. Note that deterministic hidden units allow for a non-linear dependence on the state but do not induce stochastic dependencies between the action units.

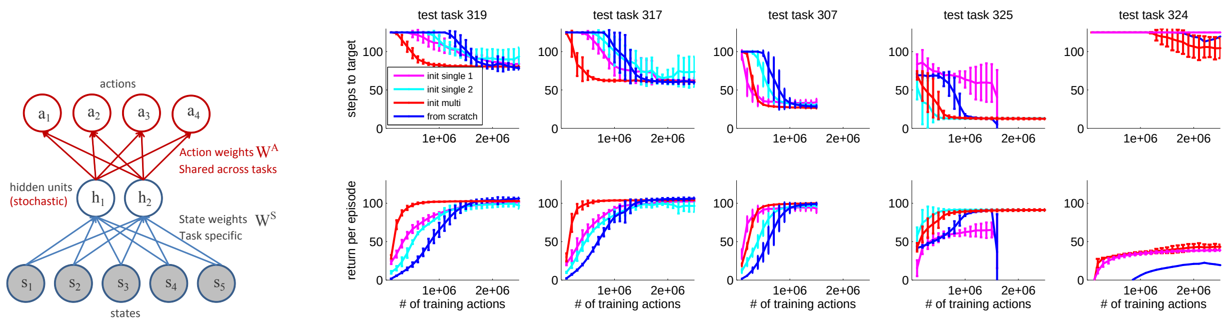

Figure 1: *Left*: Illustration of policy parameterization. *Right*: Convergence of learning for the 5 test tasks. *Top*: steps to target; *bottom*: return per episode. *Blue* and *red* traces show results for policies learned from scratch, and with pre-learned action features respectively. *Cyan* and *magenta* traces show results obtained with action features obtained with single training tasks. Traces show averages over multiple runs, typically 2-4. Error bars show std-dev..

Multi-task feature learning We design a small number of training tasks that activate muscles in different parts of the arm (e.g. muscles close to the tip, close to the base, on the left side of the arm, the right side of the arm etc.) and learn a controller jointly for these tasks: The mapping from hidden units to actions $p(\mathbf{a}|\mathbf{h};\theta)$ is shared across tasks (i.e. we learn a single set of parameters $W^A, \mathbf{c}$), but there is a task-specific, stochastic mapping $p(\mathbf{h}|\mathbf{s};\theta)$ from the states to the hidden units (i.e. we learn a separate set of parameters $W^S_k, \mathbf{c}_k$ for each task $k = 1 \ldots K$). With $M < 3C$ the hidden units $\mathbf{h}$ can be seen as providing a low-dimensional representation of actions, capturing dependencies between actions.

Once learning in the multi-task controller is complete we test the effectiveness of the learned action features for learning novel tasks (differing from the training tasks wrt. the initial configuration of the arm and the positions of targets and obstacles): We attempt to learn controllers for different test tasks, fixing the action features (W^A, $\mathbf{c}$) to those from the multi-task setup and learning only a new mapping from the state to the hidden units $\mathbf{h}$.

Results We use three training tasks and 5 test tasks (some videos of training and test tasks can be seen here[1]). Figure 1 shows convergence of learning for the test tasks. We compare convergence with pre-learned action features (only W^S, $\mathbf{b}$ are learned) to a baseline where the full policy is learned from scratch (i.e. $W^A, \mathbf{c}, W^S, \mathbf{b}$ are all learned anew). In all cases learning is considerably faster when the pre-learned action features are used (compare red/blue traces). For one of the test tasks (324) learning only succeeds with pre-learned action features. We further find that the choice of training tasks matter: action features obtained by learning to solve only a single one of the training tasks are effective for some of the test tasks but generalize less well (compare red traces with magenta / cyan traces). In additional experiments (not shown) we find that pre-learning action features is considerably more effective than pre-learning state features, and that the use of stochastic hidden units (allowing for correlated exploration) are important to exploit the learned features fully.

Discussion Our preliminary results suggest that a multi-task setup in combination with a sufficiently general learning architecture allows to learn useful representations of actions. The proposed approach is not tied to a particular learning task, algorithm (e.g. the particular form of policy gradient that we have chosen here), or policy parameterization, although our results suggest that the ability of the parametric policy to model dependencies between actions is important. One interesting direction for future work is to investigate the possibility to use richer parametric forms than the simple NN considered here to learn more complex, non-linear action features, possibly in a hierarchical manner. An especially appealing approach would be to design a sequence of training tasks of increasing difficulty, akin to a "curriculum", where simple tasks are used to learn simple features first, which subsequently facilitate learning of more complex ones. A second interesting direction would be to extend this work in the temporal domain in order to capture not just "spatial" dependencies between action dimensions but also correlations over time (e.g. , in the case of the octopus arm, to maintain activation of a muscle group over several time steps).

[1]`http://www.gatsby.ucl.ac.uk/~nheess/papers/rssabstract/`