Deep Auto-Encoder Neural Networks in Reinforcement Learning

Sascha Lange and Martin Riedmiller

Abstract— **This paper discusses the effectiveness of deep auto-encoder neural networks in visual reinforcement learning (RL) tasks. We propose a framework for combining deep auto-encoder neural networks (for learning compact feature spaces) with recently-proposed batch-mode RL algorithms (for learning policies). An emphasis is put on the data-efficiency of this combination and on studying the properties of the feature spaces automatically constructed by the deep auto-encoders. These feature spaces are empirically shown to adequately resemble existing similarities between observations and allow to learn useful policies. We propose several methods for improving the topology of the feature spaces making use of task-dependent information in order to further facilitate the policy-learning. Finally, we present first results on successfully learning good control policies using synthesized and real images.**

I. Introduction

Recently, there have been reported several impressive successes of appyling reinforcment learning to real-world systems [1], [2], [3]. But present algorithms are still limited to solving tasks with state spaces of rather low dimensionality[1]. Learning policies directly on visual input—e.g. raw images as captured by a camera—is still far from being possible. Usually, when dealing with visual sensory input, the original learning task is split into two separate processing stages (see fig. 1). The first is for extracting and condensing the relevant information into a low-dimensional representation using methods from image processing. The second stage is for learning a policy on this particular encoding.

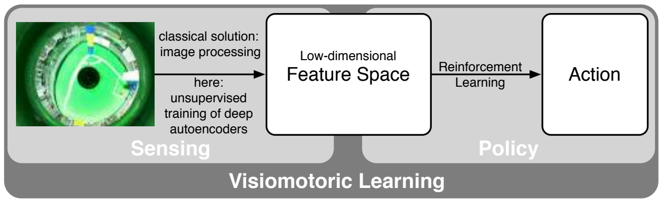
classical solution: image processing
here: unsupervised training of deep autoencoders
Low-dimensional Feature Space
Reinforcement Learning
Action
Sensing
Policy
Visiomotoric Learning

Fig. 1. Classic decomposition of the visual reinforcement learning task.

In oder to increase the autonomy of a learning system, letting it adapt to the environment and find suitable representations by itself, it will be necessary to eliminate the need for manual engineering in the first stage. This is exactly the setting where we see a big opportunity for integrating recently proposed deep auto-encoders replacing hand-crafted preprocessing and more classical learning in the first stage.

Sascha Lange and Martin Riedmiller are with the Department of Computer Science, Technical Faculty, Albert-Ludwigs University of Freiburg, D-79194 Freiburg, Germany (phone: +49 761 203 8006; email: {slange,riedmiller}@informatik.uni-freiburg.de).

[1]over-generalizing: less than 10 intrinsic dimensions for value-function-based methods and less than 100 for policy gradient methods

New methods for unsupervised training of very deep architectures with up to millions of weights have opened up completely new application areas for neural networks [4], [5], [6]. We now propose another application area, reporting on first results of applying Deep Learning (DL) to visual navigation tasks in RL. Whereas most experiments conducted so far have concentrated on distinguishing different objects, more or less perfectly centred in small image patches, in the task studied here, the position of one object of interest wandering around an image has to be extracted and encoded in a very low-dimensional feature vector that is suitable for the later application of reinforcement learning.

We will mainly concentrate on two open topics. The first is about how to integrate unsupervised training of deep auto-encoders into RL in a data-efficient way, without introducing much additional overhead. In this respect, a new framework for integrating the deep learning approach into recently proposed memory-based batch-RL methods [7] will be discussed in section III. We will show the auto-encoders in this framework to produce good reconstructions of the input images in a simple navigation task after passing the high-dimensional data through a bottle neck of only two neurons in their innermost hidden layer.

Nevertheless, the question remains whether or not the encoding implemented by these two inner-most neurons is useful only in the original task, that is reconstructing the input, or can also be used for learning a policy. Whereas properties of deep neural networks have been thoroughly studied in object classification tasks, their applicability to unsupervised learning a useful preprocessing layer in visual reinforcement learning tasks remains rather unclear. The answer to this question—the second main topic—mainly depends on whether the feature space allows for abstracting from particular images and for generalizing what has been learned so far, to newly seen similar observations. In section V, we will name four evaluation criteria and do a thorough examination of the feature space in this respect and finally give a positive answer to this question. Moreover, we will present some ideas on how to further optimize the topology in the feature space using task specific information. Finally, we present first successes of learning control policies directly on synthesized images and—for the very first time—using a real, noisy image formation process in section VI.

II. Related Work

[8] was the first attempt of applying model-based batch-RL directly to (synthesized) images. Ernst did a similar experiment using model-free batch-RL algorithms [9]. The interesting work of [10] can be seen at the verge of fully integrating the learning of the image processing into RL.

Nevertheless, the extraction of the descriptive local features was still implemented by hand, learning just the task-dependent *selection* of the most discriminative features. All thre [8], [9], [10] lacked realistic images, ignored noise and just learned to memorize a finite set of observations, not testing for generalization at all.

Instead of using Restricted Boltzman Machines during the layer-wise pretraining of the deep auto-encoders [4] our own implementation completely relies on regular multi-layer perceptrons, as proposed in chapter 9 of [11]. Previous publications have concentrated on applying deep learning to classical image recognition tasks like face and letter recognition [4], [5], [11]. The RL-tasks studied here also add the complexity of tracking moving objects and encoding their positions adequately in very low-dimensional feature vectors.

III. Deep Fitted Q-Iterations

In this section, we will present the new deep fitted q-iteration algorithm (DFQ) that integrates unsupervised training of deep auto-encoders into memory-based batch-RL.

A. General Framework

In the general reinforcement learning setting [12], an agent interacts with an environment in discrete time steps t, observing some state $s \in S$ and some reward signal r to then respond with an action $a \in A$. We're interested in tasks that can be modelled as markov decision processes [12] with continous state spaces and finite action sets. The task is to learn a strategy $\pi : S \mapsto A$ maximizing the expectation of the sum $R_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1}$ of future rewards r_t with discount factor $\gamma \in [0, 1]$. In the visual RL tasks considered here, the present state of the system is not directly observable by the agent. Instead, the agent receives a high-dimensional, continuous observation $o \in O$ (image) in each time step.

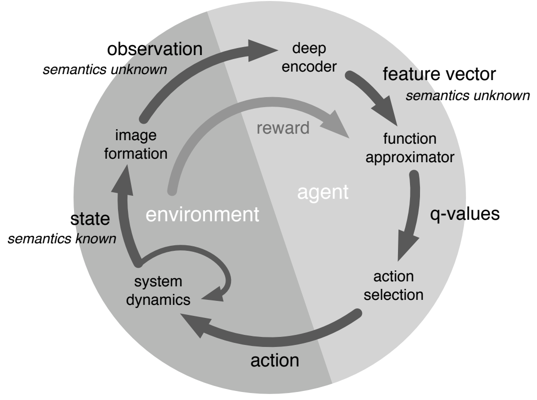

Fig. 2. Extended agent-environment loop in visual RL tasks. In the deep-RL framework proposed here, a deep-encoder network is used to transfer the high-dimensional observations into a lower-dimensional feature vector which can be handled by available RL-algorithms.

We will insert the training of deep auto-encoders on the agent's side of the processing loop (fig. 2) in order to learn an encoding of the images in a low-dimensional feature space. We advocate a combination with recently proposed model-free batch-RL algorithms, such as Fitted Q-Iteration (FQI) [13], LSPI [14] and NFQ [15], because these methods have been successful on real-world continuous problems and, as these sample-based batch-algorithms [7] already store and reuse state transitions $(s_t, a_t, r_{r+1}, s_{t+1})$, the training of the auto-encoders integrates very well (see fig. 3) into the batch-RL-framework with episodic exploration as presented in [3].

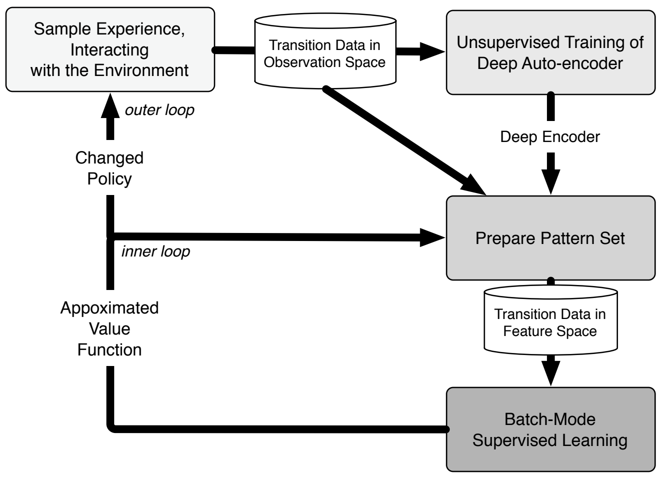
Sample Experience, Interacting with the Environment
Transition Data in Observation Space
Unsupervised Training of Deep Auto-encoder
outer loop
Deep Encoder
Changed Policy
Prepare Pattern Set
inner loop
Appoximated Value Function
Transition Data in Feature Space
Batch-Mode Supervised Learning

Fig. 3. Graphical sketch of he proposed framework for deep batch-RL with episodic exploration.

In the outer loop, the agent uses the present approximation of the Q-function [12] to derive a policy—e.g. by ϵ-greedy exploration—for collecting further experience. In the inner loop, the agent uses the present encoder to translate all collected observations to the feature space and then applies some batch-RL algorithm to improve an approximation of the Q-function. From time to time, the agent may retrain a new auto-encoder. The details of the processing steps will be discussed in the following subsections.

B. Training Deep Auto-Encoders with RProp

Training the weights of deep auto-encoder neural networks to encode image data has been thoroughly treated in the literature [4], [11]. In our implementation, we use several shallow auto-encoders for the layer-wise pre-training of the deep network, starting with the first hidden layer, always training on reconstructing the output of the previous layer. After this pre-training, the whole network is unfolded and fine-tuned for several epochs by training on reconstructing the the inputs. Differing from other implementations, we make no use of RBMs but use multi-layer perceptrons (MLP) and standard gradient descent on units with sigmoidal activations in both phases, as proposed in chapter 9 of [11]. Weights are updated using the RProp learning rule [16]. As RProp only considers the direction of the gradient and not its length, this update-rule is not as vulnerable to vanishing gradients as standard back-propagation. Furthermore, results do not depend on extensive tuning of parameters [16].

C. DFQ: Integrating deep auto-encoders into batch-RL

We combined the deep auto-encoders with Ernst's Fitted Q-Iteration for learning a policy. The new DFQ algorithm consists of the following steps realizing the inner and outer loop of figure 3:

A. **Initialization** Set episode counter $k \leftarrow 0$. Set sample counter $p \leftarrow 0$. Create an initial (random) exploration strategy $\pi^0 : z \mapsto a$ and an inital encoder $\text{ENC} : o \mapsto_{W^0} z$ with (random) weight vector W^0. Start with an empty set $\mathcal{F}_{\mathcal{O}} = \oslash$ of transitions $(o_t, a_t, r_{t+1}, o_{t+1})$

B. **Episodic Exploration** In each time step t calculate the feature vector z_t from the observed image o_t by using the present encoder $z_t = \text{ENC}(o_t; W^k)$. Select an action $a_t \leftarrow \pi^k(z_t)$ and store the completed transition $\mathcal{F}_{\mathcal{O}} \leftarrow \mathcal{F}_{\mathcal{O}} \cup (o_p, a_p, r_{p+1}, o_{p+1})$ incrementing p with each observed transition.

C. **Encoder Training** Train an auto-encoder (see [4]) on the p observations in $\mathcal{F}_{\mathcal{O}}$ using Rprop during layer-wise pretraining and finetuning. Derive the encoder $\text{ENC}(\,\cdot\,; W^{k+1})$ (first half of the auto-encoder). Set $k \leftarrow k + 1$.

D. **Encoding** Apply the encoder $\text{ENC}(o; W^k)$ to all transitions $(o_t, a_t, r_{t+1}, o_{t+1}) \in \mathcal{F}_{\mathcal{O}}$, transfering them into the feature space $\mathcal{Z}$, constructing a set $\mathcal{F}_{\mathcal{Z}} = \{(z_t, a_t, r_{t+1}, z_{t+1}) | t = 1, \ldots, p\}$ with $z_t = \text{ENC}(o_t; W^k)$.

E. **Inner Loop: FQI** Call FQI with $\mathcal{F}_{\mathcal{Z}}$. Starting with an initial approximation $\hat{Q}^0(z, a) = 0 \quad \forall (z, a) \in Z \times A$ FQI (details in [13]) iterates over a dynamic programming (DP) step creating a training set $\mathcal{P}^{i+1} = \{(z_t, a_t; \bar{q}_t^{i+1}) | t = 1, ..., p\}$ with $\bar{q}_t^{i+1} = r_{t+1} + \gamma \max_{a' \in A} \hat{Q}^i(z_{t+1}, a')$ and a supervised learning step training a function approximator on $\mathcal{P}^{i+1}$, obtaining the approximated Q-function $\hat{Q}^{i+1}$. After convergence, the algorithm returns the unique fix-point $\bar{Q}^k$.

F. **Outer loop** If satisfied return approximation $\bar{Q}^k$, greedy policy π and encoder $\text{ENC}(\,\cdot\,; W^k)$. Otherwise derive an exploration strategy π^k from $\bar{Q}^k$ and continue with step B.

Each time a new encoder $\text{ENC}(\,\cdot\,; W^{k+1})$ is learned in C, thus the feature space and its semantic are changed, the present approximation of the Q-function becomes invalid. Whereas online-RL would have to start over completely from scratch, in the batch-approach the stored transitions can be used to immediately calculate a new Q-function in the new feature space, without a single interaction with the system.

When using an averager [8] or kernel-based approximator [7] for approximating the q-function, the series of approximations $\{\hat{Q}^i\}$ produced by the FQI algorithm—under some assumptions—is guaranteed to converge to a unique fix-point $\bar{Q}$ that is within a specific bound of the optimal q-function Q^* [13], [7]. Since the non-linear encoding $\text{ENC} : O \mapsto_{W^k} Z$ does not change during the inner loop, these results also cover applying the FQI algorithm to the feature vectors. The weights of the averager can be adapted easily to include the non-linear mapping as well, as the only restriction on the weights are non-negativity and summing up to 1 [8], [7].

D. Variations and Optimizations

The following variations have been found useful, improving the data efficiency and speeding up the learning process.

Sparse networks: Using receptive fields in the n-outermost layers instead of a full connection structure can help to greatly reduce the total number of weights in the encoder, thus significantly decreasing the number of samples and time needed for training. Furthermore, receptive fields are an effective method of using the neighbourhood-structure between individual image dimensions. This idea has some motivation in biology and its usage in artificial neural networks dates back to at least the neocognitron.

Transfering information: If, after learning a new encoder $\text{ENC}(o; W^k)$ in step C of DFQ, recalculating the approximation $\hat{Q}^k$ from scratch by iterating over all the available transitions is too expensive, the q-values can be transferred from the old approximation $\bar{Q}^{k-1} : (z^{k-1}, a) \mapsto \mathcal{R}$ in the old feature space $\mathcal{Z}^{k-1}$ to the new space $\mathcal{Z}^k$. The trick is to do one DP-update by looking up the Q-values of z_{t+1} in the old approximation $\bar{Q}^{k-1}$ but then storing the resulting target value $\bar{q}$ within the new feature space. We simply prepare a training set $\mathcal{P}_{\mathcal{Z}^k} = \{(z_t^k, a_t; \bar{q}_t^{i+1}) | t = 1, \ldots, p\}$ with one sample $(z_t^k, a_t; \bar{q}_t^{i+1})$ for every transition in $\mathcal{F}_{\mathcal{O}} = \{(o_t, a_t, r_{t+1}, o_{t+1}) | t = 1, \ldots, p\}$, where $z_t^k = \text{ENC}(o_t; W^k)$. In this case $\bar{q}_t$ is calculated using the old feature vector $z_{t+1}^{k-1} = \text{ENC}(o_{t+1}; W^{k-1})$ and old approximation $\bar{Q}^{k-1}$ as $\bar{q}_t = r_{t+1} + \gamma \max_{a' \in A} \bar{Q}^{k-1}(z_{t+1}^{k-1}, a')$. The patterns are then used to train a new, initial approximation $\hat{Q}^k$ in the new feature space. In practice, the number of necessary iterations until convergence of the FQI algorithm in step E is often reduced, when starting from this initial approximation.

Re-training: For an optimal learning curve, the auto-encoder must be re-trained whenever new data is available. But since the expected improvement of the feature space given just a few more observations is rather limited—as a fair compromise between optimal feature spaces and computing time—the re-training of the auto-encoder is only triggered with every doubling of the number of collected observations.

IV. Proof of Concept

In a very first proof-of-concept experiment and throughout the evaluation of the feature spaces we will use a continuous grid-world-like [12] problem using synthesized images, thus having complete control on the image formation process (see figure 4). After very careful consideration we have chosen to use such a task with a rather simple optimal policy for our analysis, as the focus of this paper is on handling the complex, high-dimensional observations and not on learning very difficult policies. At the same time, the task allows for a thorough analysis of the proposed methods, as the decision-boundaries and optimal costs can be derived precisely.

A. Continuous grid-world with noisy image formation

In this problem, the agent observes a rendered image $o \in [0, 1]^n$ with $n = 30 \times 30 = 900$ pixels and added gaussian noise $\mathcal{N}(0, 0.1)$ instead of the two-dimensional system state $s = (x, y) \in [0, 6)^2$. Due to the discrete nature of the pixels, the number of possible agent positions in the images is limited to 900 minus 125 positions blocked by walls. Each action moves the agent exactly 1m in one of four directions. The task of reaching the $1m \times 1m$ goal area has been modeled as a shortest-path problem [12], with a reward of -1 for any transition not ending in the absorbing goal area.

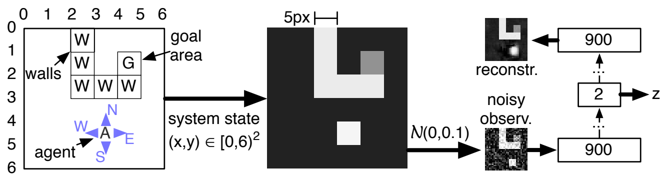

Fig. 4. Continous grid-world with noisy image formation. Left: $6m \times 6m$ world, walls (W), $1m^2$ goal area (G) and agent (A). Middle: rendered image. Right: an auto-encoder calculates feature vectors z from noisy observations.

B. Simplified proof-of-concept experiment

In order to make a point about neglecting noise as done in previous publications [8], [10], the described system has been further simplified for this very first experiment. The synthesized images were used without adding any noise and the agent was always starting in one of only 30 different starting positions corresponding to the center of $1m \times 1m$ cells of a regular partition of the grid-world's state space. In this version of the task there are only 31 different observations, thus making it comparable to rather simple problems with a small, finite set of states.

C. Results

In analogy to the task's internal state (x, y), we have chosen to learn a two-dimensional encoding of the observations. After several preliminary tests, the auto-encoder's topology was fixed to 21 layers with 900-900-484-225-121-113-57-29-15-8-2-8-15-29-57-113-121-225-484-900-900 neurons and 9×9-receptive fields between the 5 outermost layers. This auto-encoder had more than 350 000 connections in total. The average reconstruction error (total sum of squares / number of images) of the second generation auto-encoder was 10.9 after 190 epochs of fine-tuning. Activations of output-neurons representing immobile parts (walls, goal state), have been observed to 'attach' to the bias weights within the very first episodes of the training. Representing the learned Q-function with a large 2D grid approximator (500×500 cells) that assigns a constant Q-value to all state-action pairs residing in the same hyper-rectangle, the DFQ algorithm found the *optimal* policy within 65 episodes.

D. Discussion

Although the agent learned the optimal policy, the relevance of this result as well as the earlier results of Jodogne and Ernst [9], [10] is rather limited, when it comes to answering the central question of whether or not the automatically constructed feature spaces will be useful for learning policies on *real* images. The simplified image-formation process in this experiment being completely deterministic, without any noise, each of the non-observable states is represented by exactly one observation. Even a random-initialized encoder would produce different outputs for all of them, thereby identifying the 31 system states uniqely. Q-values for that few observations can easily be learned by heart storing them in a hash-table, without the need for ever generalizing to previously unseen observations. Hence, we will examine the feature spaces under more realistic conditions in the next section.

V. Evaluating the Feature Space

When targeting a realistic image-formation process that involves noise and many different observations, learning Q-values by heart is not possible (see fig. 6 column c). Abstraction from the pure-pixel values and generalization among similar observations in this case is a necessity for learning good policies and at the same time being data-efficient, needing as few interactions as possible. Robustness to noise and not confusing the underlying, non-observable system states is another challenge. In this respect, there are 4 different criteria for evaluating the feature spaces constructed by the deep auto-encoders.

a) Identification: There should be a clear correspondence between feature vectors and underlying system states, allowing a reliable identification and introducing as little performance-limiting state-aliasing as possible.
b) Robustness: Encodings of several noisy images of the agent at the exact same position should be close to each other, forming narrow clusters in the feature space.
c) Generalization: Two images of the agent at only slightly differing positions should not result in vastly differing feature vectors but should have a tendency of being close to each other.
d) Topology: Somehow preserving the general relation among the underlying system states (e.g. x-y-coordinates) would provide additional benefit for approximators being able to generalize globally.

Whereas the second and third properties would allow to relate new observations to what has been learned before, thus—as the first criterion—are a prerequisit for any succesful

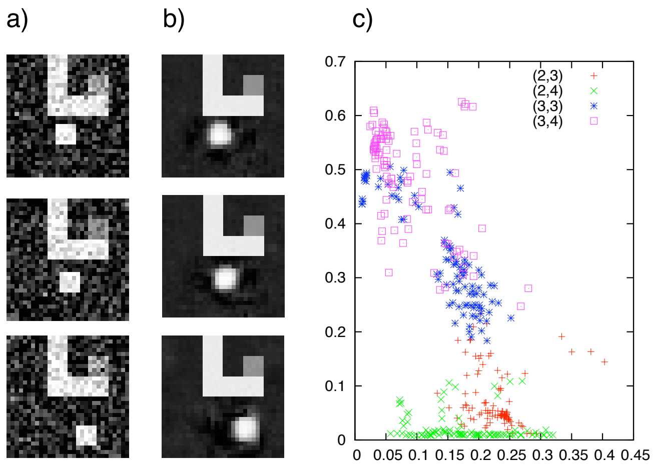

Fig. 6. Effects of noisy image-formation in the continous grid-world task. *a)* three sample-observations of the agent from the testing set. *b)* Reconstructions of these images after training on noisy training images (s. sec. V-A). *c)* Feeding several hundred noisy-samples of four neigbhoring agent positions to an encoder that was trained only on the noise-free observations produces feature vectors that are spread out in about a quarter of the feature space, forming fraying, overlapping clusters.

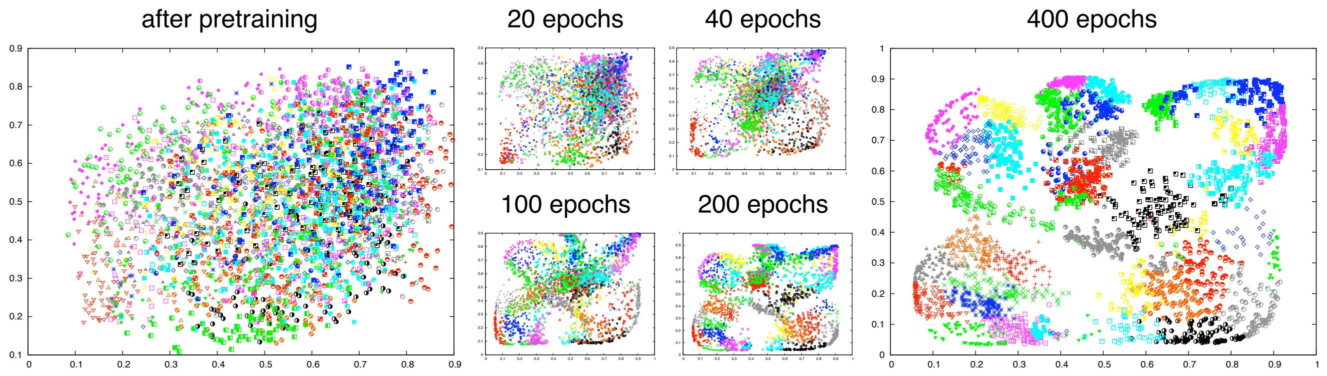

Fig. 5. Evolution of the the feature space during finetuning the deep auto-encoder. Feature vectors of testing images depicting the agent in the same tile of an arbitrarily 6×6 tiling of the grid world are marked using the same color and symbol (used only for visualization-purposes). During the unsupervised training, the feature space is 'unfolded', gradually improving the ordering of neighboring observations towards a near-perfect result after 400 epochs.

learning, the fourth property is not an absolute requirement but could facilitate the learning process.

A. Training of the auto-encoder

Before letting the agent explore the world on its own, we examined the resulting feature spaces in several experiments under controlled, perfect sampling conditions. We sampled a total of 6200 noisy images, the agent's position evenly distributed throughout the maze (3100 images for training, 3100 for testing). An auto-encoder of the same topology as in section IV-C achieved an average reconstruction error (RE) of 7.3 per testing image (down from 17 after pre-training). Some exemplary observations and their reconstructions can be seen in figure 6 columns a and b.

More interesting than the reconstruction error is the quality of the learned feature space. Within DL, there seem to be two established methods for assessing feature spaces; first, a visual analysis and second, training classifiers on the feature vectors [4], [11]. We used both methods keeping the four criteria mentioned above in mind.

B. Visual analysis of the feature space

The encoder-part of the auto-encoders has been used to map all testing images from the observation space to the two-dimensional feature space spanned by the encoder network's output neurons (fig. 5). We have used the same color and symbol to mark feature vectors residing in the same tile, arbitrarily superimposing a 6×6 tiling on the grid-world. Of course, these labels are not available during training but added later for visualization purposes only. As can be seen in the first plot, the pre-training realized a good spreading of the data in the feature-space but did not achieve a good separation. With the progress of the finetuning, structure becomes more and more visible. After 400 epochs, images that show the agent in the same cell of the maze form clearly visible clusters in the feature space (criteria b, c).

C. Classification of feature vectors

In order to further test the usefulness of the derived feature space, we did a supervised-learning experiment. A second neural net ("task net") with two hidden layers was trained on class labels corresponding to the previously introduced 6×6 tiling using the feature vectors produced by the deep encoder as inputs. The optimal network size was determined in a series of preliminary experiments.

A reasonable number of hidden-layer neurons was needed, to achieve good results. Using 2-25-25-2 neurons in the task net, and training the output layer on the coordinates of the center of the corresponding tile the trained net classified 80.10% of them correctly, whereas a classification is counted as correct, when the output of the task net is closer to the center of the correct tile than to the center of any other tile.

As can be seen in table I (column CR Fixed) the quality of the feature space depends heavily on the number and distribution of samples. Experiments with 1550 and 3100 samples did use 'perfect sampling', evenly distributing the observations among the possible agent-positions, experiments with less samples did select samples by random.

TABLE I
AVERAGE SQUARED RECONSTRUCTION ERROR (REC) AND CLASSIFICATION RATES (CR) ON TRAINING SETS OF DIFFERENT SIZE.

SPACE	SAMLES	REC	CR FIXED	CR ADAPTIV
DL	465	18.9	27.31%	40.86%
	775	12.0	59.61%	80.87%
	1550	9.4	61.42%	91.59%
	3100	7.3	80.10%	99.46%
PCA 02	3100	15.8	39.58%	–
PCA 04	3100	14.9	70.80%	–
PCA 10	3100	12.6	88.29%	–

D. Comparison with Principle Component Analysis (PCA)

For comparison reasons, we did the same experiments using a PCA on the exact same set of images. The first n principal components (PC) and a scatter plot of the first two components are depicted in fig 7. Basic PCA clearly fails to construct a useful, compact representation, as the first two principle components only explain 6% of the overall variance. Training a classifier on supervised learning task using the first 2 (PCA 2) and 4 (PCA 4) principal components led to unsatisfying results (see tab. 1). Even more PCs are needed to allow for a better classification than the two dimensions found by DL (PCA 10).

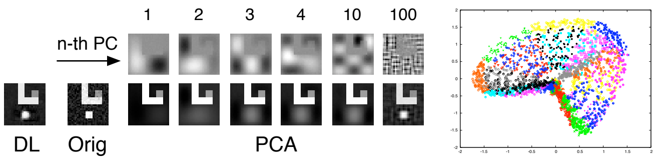

Fig. 7. Eigenimages of some of the principal components found by a PCA (top row) and reconstruction of the original (orig) image when using the n first PCs. DL needs only 2 dimensions for producing a reconstruction that is clearly more accurate than those produced by the first 10 PCs.

E. Backpropagating error terms

In order to test whether the feature space constructed by the auto-encoders could be further improved by letting the encoding adapt to a specific task, we attached a small 'task-network' with only three hidden layer neurons directly to the output-neurons of the original encoder, as can be seen in the top row of figure 9. The key was to use only three hidden-layer neurons, giving the task-net some expressive power, but by far not enough to learn a good mapping from the initial feature space. During supervised training, the error was back-propagated through the whole net, including the encoder, and was used to also slowly adapt the weights of the encoder, thus adapting the 'preprocessing' itself in respect to the needs of the classification task at hand. This combined net achieved an impressive classification rate of 99.46% on the testing images (left column of fig. 9).

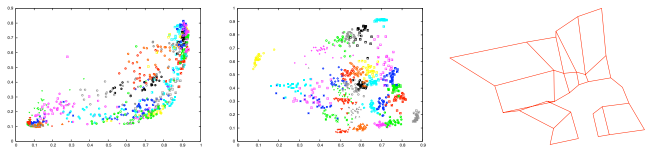

Fig. 8. Improvement of an encoder that has been derived from an imperfectly trained auto-encoder, after training the supervised x-y-coordinates task. Distribution in the feature space at the beginning of the training (left) and after training (middle). Topology after training (right).

Even more astonnishing is the quality of the improvement of the topology in the encoder-part of the combined net. Whereas with the fixed weights we got mixed results regarding the preservation of the topology—there is some preservation of the neighborhood-relations between the cells of the maze, but there are several foldings and deformations in the feature space—letting adapt the encoder's weights to the task at hand, helped to significantly improve the overal organization of the feature space. The right column of 9 depicts the gradual improvements of the topology in the third layer counted from the back of the combined network that is the encoder's output layer. The final result is a near-perfect preservation of the original topology with only few distortions and no crossings. Moreover, this technique could be used to improve a really weak initial encoding to be useful in the RL-task. For example, an encoder that was only shortly trained on 775 non-evenly distributed samples (RE: 12.0) could be improved from a classification rate (CR) of 59.61% with fixed-encoder weights to a CR of 80.87% (fig. 8).

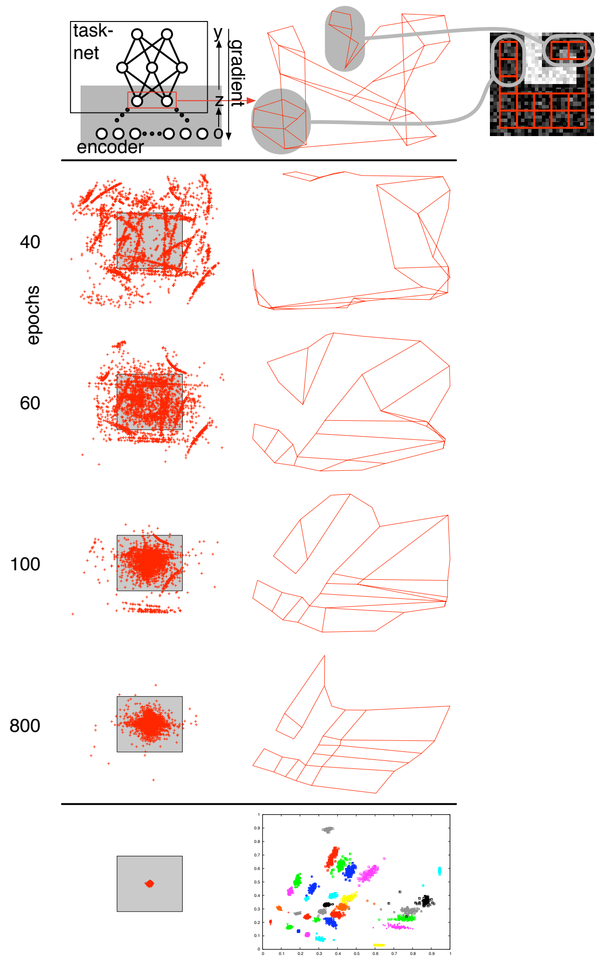

Fig. 9. Improvement of the topology during supervised learning. Top: Arbitrary grid structure and its initial mapping to the feature space. Middle: Topology of the feature space (right column) and absolute differences between targets and outputs on the testing patterns (left column). The gray rectangular region marks the error that would not lead to a misclassifcation. Bottom: Errors on the training images (left) and the final distribution of the testing images in the feature space (right).

F. Training on the optimal value function.

In analogy to what the encoder will be used within DFQ, we trained a combined encoder-task-net on the optimal state-value function (easily derived by hand) of the maze-task.With fixed encoder weights, the best classification rate we could achieve on the testing set was only 78.65% with a relatively large net of two hidden layers with 17 neurons each. Letting adapt the encoder weights, the classification rate was improved to 99.45% for a task-net with 12 neurons in a single hidden-layer. In this case, the topology of the feature space completely adapted to the new task, contracting to a narrow band in the feature space, having at one end all the observations with a state value of 0, at the same time having those with a value of -10 at the other end. For example, the positions (3.5,0.5) and (1.5,0.5) that are close to each other in the original state-space but due to the wall have largely differing optimal state-values (-1 and -9), have been moved towards opposite ends of the band, grouped together with other observations with the same state value (fig. 10).

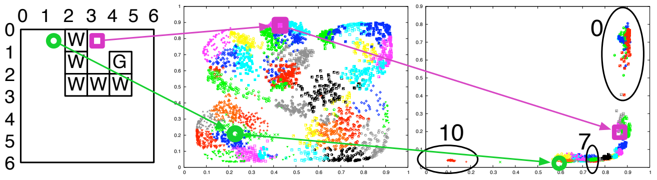

Fig. 10. Position of the agent in two observations (left), original feature space (middle) and after training on the optimal state-value function.

VI. Learning policies

We did two RL-experiments using synthesized and real images. First, DFQ was used to learn a policy in the grid-world, directly on basis of the synthesized images ($\sigma = 0.1$). The agent was started in random positions outside the goal room. An episode was ended either when the agent reached the $1m \times 1m$ goal area or didn't arrive within 20 steps. After every episode the greedy policy was derived from the present approximation of the Q-function and was evaluated, starting from 30 fixed, evenly distributed states. The optimal policy would collect an average reward of -5.1 from these states. The DFQ algorithm was used with a constant regular grid approximator partitioning the feature space into 20×20 cells. This resolution turned out to produce the best results in a preliminary evaluation of different quadratic grid sizes. The agent began with a random encoder using ϵ-greedy to direct the exploration. The first auto-encoder was trained after collecting 100 observations, triggering re-training after every doubling of the number of observations from there on.

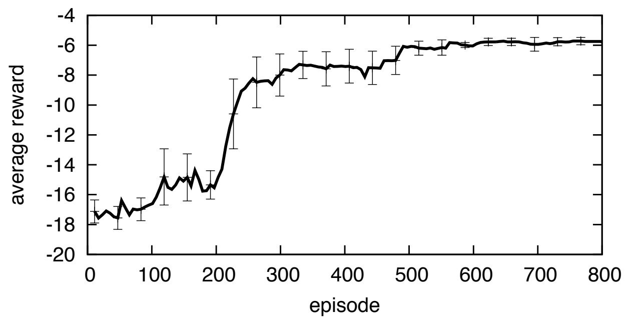

Fig. 11. Performance of DFQ in the continuous grid-world task with synthesized images. Average of five independent runs, with standard deviation.

In this experiment, the policy constantly improves from the random policy (average reward of -17.3) until collecting an average reward of -5.60 after 732 episodes, not improving thereafter. This policy is within half a step of the optimal policy. The final, 7th generation feature space and the learned value function are shown in figure 12. The averaged results of five independent repetitions of the whole experiment are depicted in figure 11. The best final policy achieved an average reward of -5.4, the worst still collecting -5.97.

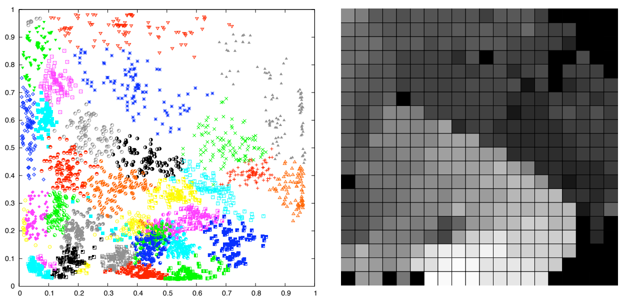

Fig. 12. Learned feature space (left) and state-value function (right). Brightness is proportional to expected rewards; completely black cells (upper right and lower right corner) haven't been hit by any transition.

Finally, we repeated the same experiment, but this time capturing observations of the grid-world from a computer screen using a digital video camera instead of synthesizing them (see fig. 13 with some example images on the left). The agent received a timing signal whenever the labyrinth server had drawn the new state on screen and expected an action to be chosen. During each trial, the agent had to operate in near-realtime, passing 15 images per second through the deep encoder and evaluating the feature vectors.

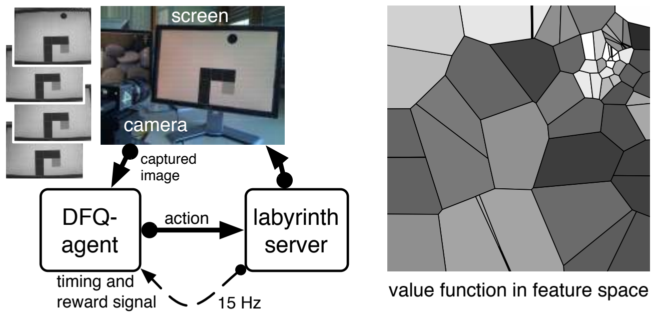

Fig. 13. Grid-world experiment with *real* image formation. Experimental setup (left) and value-function learned by DFQ using an irregular grid approximator. The brightness of the cells is proportional to expected rewards.

In this experiment, we used an *irregular* grid appoximator within DFQ. This type of approximator is capable of adapting its grid to the structure of the automatically constructed feature space using clustering techniques. Thus, it is not that dependent on selecting the optimal resolution as are the regular grid approximators. Dealing with a slight barrel distortion, pixel-aliasing, real image noise, non-uniform lighting conditions and an increased size of the real images (80×60 pixels), the DFQ-agent still managed to learn a good policy collecting an average reward of -6.13 after only 635 episodes.

VII. Discussion

The empirical results presented in section V give rise to positive answers to the four evaluation criteria. First, the feature vectors allow to identify the underlying system states and to learn good classifications, second, the feature representation is robust under image noise, third feature-vectors of observations of similar agent-positions also tend to be close together in the feature-space, forming clearly visible clusters, and fourth, the topology is preserved to a certain degree during unsupervised training.

Back-propagating error-terms, thus letting the encoder adapt its encoding to a specific task after initial unsupervised training on the reconstructions, can further improve the results regarding a classification or regression. At the same time the topology of the feature-space can be further improved and adapted to a specific task. If labelled data is available, using multi-task-learning [17], would be an option for benefitting from domain-specific knowledge.

The quality of the feature spaces was found to depend on the number of samples used during training (see tab. 1). One problem with too few samples is a bad covering of the receptive fields. Fields that were not presented an agent at least once during the training, will not be trained on its detection at all and thus will fail to detect it during the testing phase. Hence, collecting enough observations and using an appropriate exploration strategy is crucial to the success of a reinforcement learner relying on the automatically constructed feature spaces. Adapting from convolutionary neural networks [18] the shared weights between the receptive fields could help reduce this dependence in the future.

In section 3 we have presented a framework for efficiently integrating deep learning with RL. Data-efficiency comes with the usage of recently introduced memory based batch-RL-methods and with storing the transitions in observation space. Changing the semantics of the feature space is not a problem anymore as a new Q-function can be easily calculated using the stored transitions, whereas traditional online-RL would have to start from scratch.

Concerning the original motivation of replacing other methods in the pre-processing stage of visual RL, we can give several arguments in favor of the deep auto-encoders. First, Deep Learning was found to really be able to produce a very condensed, useful and robust representation in the navigation experiment. We have been able to learn near-optimal policies on synthesized and real images using the learned feature vectors. Second, the compact representation is much better than representations that can be found using more traditional techniques. Due to the similarity of images being based on the implicit proximity of orthogonal dimensions of the observation, linear PCA did fail to construct as useful *compact* representations. This result is completely in line with similar results of Hinton, who has given further examples of DL beating classic techniques like PCA and Locally Linear Embedding in image recognition tasks [4].

VIII. Summary and Outlook

In this paper, we have proposed a new approach for closing the existing gap between the high dimensionality of visual observations and the low dimensionality of state spaces that can be handled by existing batch-RL methods. With DFQ we have introduced an efficient method for integrating the unsupervised training of deep auto-encoder networks into batch-RL. The autonomously learned feature spaces have been demonstrated to be useful for learning near-optimal policies in a grid-world like task. To our knowledge, this was the first time RL was successfully applied to *real* images without hand-crafted preprocessing or supervision. The next step will be to apply DFQ to real-world systems needing more complex policies like e.g. dribbling a ball with a soccer robot [3] or driving a slot car [19] just on basis of the visual feedback. An open problem that has not yet been discussed is how to handle the *dynamics* of systems like these, in which velocity—that can not be captured in a single image—is important. A promising idea we plan to examine next is to enrich the state representation by the difference to the previous feature vector.

References

[1] A. Ng, H. Kim, M. Jordan, S. Sastry, and S. Ballianda, “Autonomous helicopter flight via reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 16, 2004.

[2] J. Peters and S. Schaal, “Learning to Control in Operational Space,” *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, 2008.

[3] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, “Reinforcement learning for robot soccer,” *Autonomous Robots*, vol. 27, no. 1, 2009.

[4] G. Hinton and R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] G. Hinton, S. Osindero, and Y. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[6] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and Q. Montreal, “Greedy Layer-Wise Training of Deep Networks,” in *Proc. of the NIPS 2006*. MIT Press, 2007.

[7] D. Ormoneit and Ś. Sen, “Kernel-based reinforcement learning,” *Machine Learning*, vol. 49, no. 2, pp. 161–178, 2002.

[8] G. Gordon, “Stable Function Approximation in Dynamic Programming,” in *Proc. of the 12th ICML*, 1995, pp. 261–268.

[9] D. Ernst, R. Marée, and L. Wehenkel, “Reinforcement learning with raw pixels as input states,” in *Int. Workshop on Intelligent Computing in Pattern Analysis/Synthesis (IWICPAS)*, 2006, pp. 446–454.

[10] S. Jodogne and J. Piater, “Closed-Loop Learning of Visual Control Policies,” *Journal of Artificial Intelligence Research*, vol. 28, pp. 349–391, 2007.

[11] Y. Bengio, “Learning deep architectures for AI,” Dept. IRO, Universite de Montreal, Tech. Rep. 1312, 2007.

[12] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[13] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-Based Batch Mode Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 6, no. 1, pp. 503–556, 2006.

[14] M. Lagoudakis and R. Parr, “Least-Squares Policy Iteration,” *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[15] M. Riedmiller, “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method,” in *Proc. of ECML*, 2005.

[16] M. Riedmiller and H. Braun, “A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm,” in *Proc. of the ICNN*, 1993, pp. 586–591.

[17] R. A. Caruana, “Multitask learning: A knowledge-based source of inductive bias,” in *Proc. of the 10th International Conference on Machine Learning*, 1993, pp. 41–48.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] T. Kietzmann and M. Riedmiller, “The Neuro Slot Car Racer: Reinforcement Learning in a Real World Setting,” in *Proc. of the 8th Int. Conf. on Machine Learning and Applications 09*, 2009.