Deep Learning of Visual Control Policies

Sascha Lange and Martin Riedmiller

University of Freiburg - Dept of Computer Science - Germany

Abstract. This paper discusses the effectiveness of deep auto-encoding neural nets in visual reinforcement learning (RL) tasks. We describe a new algorithm and give results on succesfully learning policies directly on synthesized and real images without a predefined image processing. Furthermore, we present a thorough evaluation of the learned feature spaces.

1 Introduction

The dimensionality of the state space has always been critical to the success of applying Reinforcement Learning [1] to a given task. Present algorithms are typically restricted to state spaces of low dimensionality, pushing an application directly to visual input—e.g. raw image data as captured by a camera—completely out of scope. Usually, the original task of learning a policy given high-dimensional images is split into two separate processing stages (see fig. 1). The first is for extracting and condensing the relevant information into a low-dimensional feature vector and the second for learning a policy on this particular encoding, keeping the feature space fixed. This is exactly the setting where we see a big potential for deep auto-encoding neural nets [2] replacing hand-crafted preprocessing and more classical learning in the first "sensing" stage.

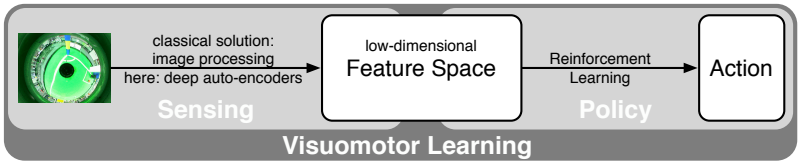

Fig. 1: Classical decomposition of the visual reinforcement learning task.

In this paper, we propose to integrate deep auto-encoders directly into RL-algorithms where they can help by automatically reducing the dimensionality of the visual input using *unsupervised learning*. The combination with memory-based batch RL [3, 4] in the new DFQ-algorithm promises outstanding data-efficiency, making optimal use of observations, introducing only little overhead. The sparse representations constructed by deep learning (DL) form the basis for applying RL and as such, their quality also influences the quality of resulting policies. Hence, this paper offers a thorough empirical evaluation of the properties of the automatically constructed feature spaces as a main contribution.

Related Work Instead of using Restricted Boltzmann Machines during the layer-wise pretraining of the deep auto-encoders [2] our own implementation relies on regular multi-layer perceptrons, as proposed in chapter 9 of [5]. Previous publications have concentrated on applying deep learning to classical face and

letter recognition tasks [2, 5]. The RL-tasks studied here also add the complexity of tracking moving objects and encoding their positions adequately.

[3] was the first attempt of applying model-based batch RL directly to (synthesized) images. Following the classical decompositional approach, in [6] Jodogne tried to further integrate the construction of feature spaces into RL but succeeded only in learning the *selection*—not the *extraction*—of the features. Both [3, 6] lacked realistic images, ignored noise and just learned to memorize a finite set of observations, not testing for generalization at all.

2 RL on image data: Deep Fitted Q-Iteration

In the general reinforcement learning setting [1], an agent interacts with an environment in discrete time steps t, observing some markovian state $s_t \in S$ and reward r_t to then respond with an action $a_t \in A$. The task is to learn a stationary policy $\pi : S \mapsto A$ that maximizes the expectation of the discounted sum of rewards $R_t = \sum_{k=0}^{\infty} \gamma^t r_{t+k+1}$ with discount factor $\gamma \in [0, 1]$. A standard solution is learning the optimal q-function $Q^*(s, a) = E[R_t | s_t = s, a_t = a]$ that specifies the expected reward when always selecting the optimal action starting with $t + 1$ and then deriving the optimal policy π^* by greedy evaluation [1].

In the tasks considered here, the learner does not know anything about the system state but only observes a high-dimensional image $\boldsymbol{o}_t \in [0, 1]^d$. The observations $\boldsymbol{o}_t$ are assumed to be markov[1]. The new idea followed here is to integrate the unsupervised training of deep auto-encoders into Ernst's fitted q-iteration (FQI) [4] in order to obtain an encoding of the images in a low-dimensional feature space. Due to space limitations, we will only briefly discuss the basic version of the new algorithm "deep fitted q-iteration" (DFQ) and may refer the reader to [7] for a more thorough treatment of batch RL in general.

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

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

C. Encoder Training Train an auto-encoder on the p observations in $\mathcal{F}_\mathcal{O}$ using resilient propagation during layer-wise pretraining and finetuning. Derive the encoder $\text{ENC}(\cdot\,; W^{k+1})$ (first half of the auto-encoder, see [2]). Set $k \leftarrow k + 1$

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

E. Inner Loop: FQI Call FQI with $\mathcal{F}_\mathcal{Z}$. Starting with an initial approximation $\hat{Q}^0(\boldsymbol{z}, a) = 0 \quad \forall (\boldsymbol{z}, a) \in Z \times A$ FQI (details in [4]) iterates over a dynamic programming step creating a training set $\mathcal{P}^{i+1} = \{(\mathbf{z}_t, a_t; \bar{q}_t^{i+1}) | t = 1, ..., p\}$ with

[1] This does hold for a system that generates different observations $\boldsymbol{o}$ for exactly the same state s (e.g. due to noise) but never produces the same observation for two different states.

$\bar{q}_t^{i+1} = r_{t+1} + \gamma \max_{\mathbf{a}' \in A} \hat{Q}^i(\mathbf{z}_{t+1}, a')$ and a supervised learning step training a function approximator on P^{i+1}, obtaining the approximated Q-function $\hat{Q}^{i+1}$.

F. Outer loop If satisfied or if running in pure batch mode without recurring exploration, return approximation $\hat{Q}^i$, greedy policy π and encoder $\mathrm{ENC}(\boldsymbol{o}; W^k)$. Otherwise derive an exploration strategy π^k from $\hat{Q}^i$ and continue with step B.

In the outer loop, the learner uses the present approximation of the q-function to derive a policy—e.g. by ϵ-greedy evaluation [1]—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 FQI to improve an approximation of the q-function. Each time a new encoder $\mathrm{ENC}(\cdot; W^{k+1})$ is learned in C, thus the feature space is changed, the approximation of the q-function and the derived policy become invalid. Whereas online-RL would have to start calculating the q-function 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 any further interaction.

When using an averager [3] or kernel-based approximator [8] 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 q-function $\bar{Q}$ that is within a specific bound of the optimal q-function Q^* [4, 8]. Since the non-linear encoding $\mathrm{ENC} : O \mapsto Z$ does not change during the inner loop, these results also cover applying the FQI algorithm to the feature vectors. The weights of the averager or kernel theoretically can be adapted to include the non-linear mapping as well, as the only restriction on the weights is summing up to 1 [3, 8]. The remaining problem is the bound on the distance $||Q^* - \bar{Q}||_\infty$ [8] that—by "hiding" the encoding in the weights—becomes dependent on the particular encoding. Since, there are no helpful formal results on the properties of the feature spaces learned by DL, the best to do is an empirical evaluation.

3 Results

We evaluated the algorithm on a *continuous* grid-world problem using synthesized images (see fig. 2). Instead of the internal system state $(x, y) \in [0, 6)^2$, the agent receives only a rendered image with 30-pixels and added gaussian noise $\mathcal{N}(0, 0.1)$. Due to the discrete nature of the pixels, the number of possible agent positions is limited to 900. Each action moves the agent 1m in the selected direction. The task of reaching the goal has been modeled as a shortest-path problem [1], with a reward of -1 for any transition outside the goal area.

After testing several topologies on a set of 3100 evenly-distributed training images and as many testing images, we have selected a rather huge auto-encoder with 21 layers in total and receptive fields[2] of 9×9 neurons in the four outermost layers, otherwise being fully connected between adjacent layers. This net with more than 350 000 weights achieved the best reconstruction error (RE, mean

[2]The method of connecting only a number of neighbouring neurons to a neuron of the subsequent layer has some motivation in biology and dates back at least to the neocognitron.

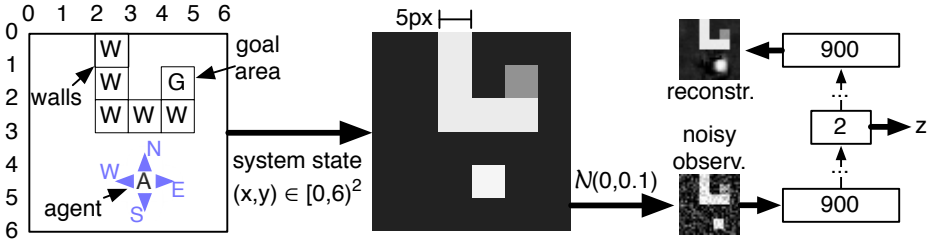

Fig. 2: Continous grid-world experiment with synthesized images.

square error per image) of 7.3 on the testing set producing (visibly) correct reconstructions. Worthy to note, a net with all layers fully connected—as Hinton successully used on the 60000 MNIST images [2]—achieved only a worse RE of 9.8 on the grid-world images.

More interesting than the reconstructions is the quality of the 2-dimensional feature space produced by the encoder part of the auto-encoder, since the feature vectors $\boldsymbol{z}$ will be the basis for learning policies. 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 [2, 5]. We used both methods keeping three criteria in mind that would directly affect the quality of learned policies: A) different agent positions must be distinguishable, B) the robustness to image noise, and C) the support for generalizing among similar states.

The feature space of the encoder is depicted in figure 3. The same color has been used for marking all 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 and are later added for visualtization purposes only. As can be seen, the feature vectors of positions in the same tile reside close to each other (criterion C), forming clearly visible clusters throughout the whole feature space, with sharp boundaries and almost no overlap, giving hint to a low sensitivity to image noise (criterion B). When compared to the results of a PCA (fig. 4), obviously, the PCA produces much more overlap and completely fails to properly encode the agent's position in its first two principal components (PC).

In another experiment, a neural net with two hidden layers was trained using the feature vectors as inputs and class labels corresponding to a numbering of the previously used tiling as targets. The classification error (CE) on the feature vectors of the testing images was 19.90% ("CE fixed" in tab. 1). Less than 1% of the false-classifications were more than 1 cell wrong (criterion A). If the gradient of the classification error was backpropagated in the encoder net ("CE adaptive" in tab. 1), also adapting the encoder's weights slowly (see [2]), the CE of the combined net could be reduced to an impressive 0.54%.

Although the results so far are promising, there still remains the problematic size of the training set. 3100 images corresponds to 4 samples for each possible position of the agent. This is still too much for an efficient integration into RL. Especially in the beginning, there are only few observations available—seldomly evenly distributed. As can be seen in row A of table 1, the RE and

Table 1: Reconstruction errors (RE) and classification errors (CE) for differnt sizes of the training set Row A: receptive fields. Row B: convolutional kernels. For comparison reasons: results of a PCA in the last column (using 2 PCs).

	Samples ↦	155	465	775	1550	3100	3100
A	RE	27.58	18.9	12.0	9.4	7.3	15.80
	CE Fixed	75.48%	62.69%	40.39%	38.58%	19.90%	60.42%
	CE Adaptiv	74.19%	59.14%	19.13%	8.41%	0.54%	–
B	RE	12.38	9.42	7.8	7.6	6.8	–
	CE Fixed	36.13%	28.17%	17.29%	17.03%	11.23%	–
	CE Adaptiv	35.48%	12.25%	7.61%	0.13%	0.0%	–

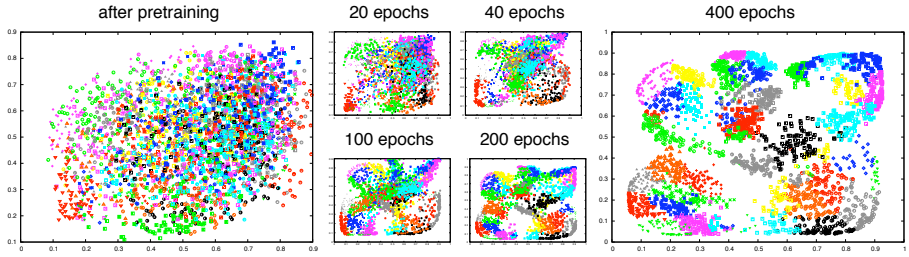

Fig. 3: Evolution of the feature space during finetuning the auto-encoder.

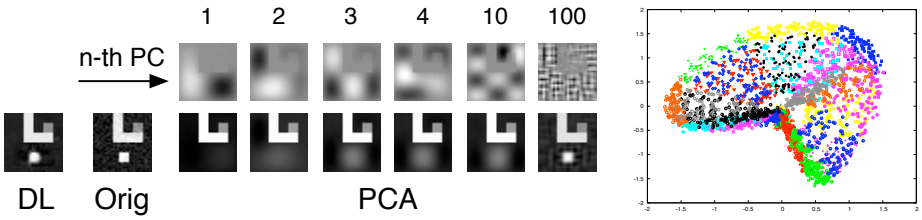

Fig. 4: Eigenimages of some PCs found by PCA (top row) and reconstruction of the original image ("Orig") when using the n first PCs or DL (left of Orig).

CE quickly degrades when the number of samples is further reduced. For those small training sets, the results can be greatly enhanced by using weight sharing and convolutional kernels replacing the (independent) receptive fields [9] (row B). Even in the case of only 155 random distributed training samples (that is a covering of the possible agent positions by only 20%) the CE of $< 40\%$ is better than the result of the receptive fields when training on a 100% covering.

Learning a policy We started the DFQ-algorithm from scratch and let it learn a policy by interacting with the grid-world. Every few episodes a new encoder was trained in the outer loop, using 9×9 convolutional kernels. The q-function was approximated using a regular-grid approximator with 20×20 cells. The best policy within 500 episodes (maximal length of 20 steps, 3795 steps in total)

achieved an average reward (AR) of -5.42, what is within half a step of the optimal policy. During the last 100 episodes the AR was very stable ($\sigma = 0.01$).

Finally, we repeated the same experiment with a real image-formation process: the grid-world was plotted on a computer monitor and captured in real-time (10 Hz) using a video camera. DFQ received a sub-sampled version (40×30 pixels, no smooting) of the captured QVGA-image and had to handle real image noise, slight barrel distortion and non-uniform lighting conditions. After 44 episodes the learned policy reached the goal from 50% of the tested starting states collecting an average reward of -11.53. The best policy (episode 492) had an AR of -5.53, only slightly below the simulation.

4 Discussion

We have presented the new DFQ algorithm for efficiently learning near-optimal policies on visual input. The evaluation of the feature spaces has shown deep auto-encoders being able to adequately extract and encode relevant information in a basic object tracking task, where linear methods like PCA fail. The learned feature spaces form a good basis for applying batch RL. For overcomming the problem of few, unevenly distributed obseravtions during the beginning of the exploration, we introduced the usage of convolutional kernels in the auto-encoders. To our knowledge, this was the first presentation of succesfully applying RL to *real* images without hand-crafted preprocessing or supervision. An open problem that has not yet been discussed is how to handle *dynamic* systems, where velocity—that can not be captured in a single image—is important. One idea is to use not only the present feature vector as basis for learning the q-function, but to also include the previous feature vector or the difference between them. First results on applying this to a real slot car racer have been very promising.

References

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

[2] G.E. Hinton and R.R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[3] G. Gordon. Stable Function Approximation in Dynamic Programming. In *Proc. of the 12th ICML*, pages 261–268, 1995.

[4] D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 6(1):503–556, 2006.

[5] Y. Bengio. Learning deep architectures for AI. Technical Report 1312, Dept. IRO, Universite de Montreal, 2007.

[6] S. Jodogne and J.H. Piater. Closed-Loop Learning of Visual Control Policies. *Journal of Artificial Intelligence Research*, 28:349–391, 2007.

[7] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[8] D. Ormoneit and Ś. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, 2002.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.