Autonomous reinforcement learning on raw visual input data in a real world application

Sascha Lange, Martin Riedmiller
Department of Computer Science
Albert-Ludwigs-Universität Freiburg
D-79110, Freiburg, Germany
Email: [slange,riedmiller]@informatik.uni-freiburg.de

Arne Voigtländer
Shoogee GmbH & Co. KG
Krögerweg 16a
D-48155 Münster, Germany
Email: arne@shoogee.com

***Abstract*—We propose a learning architecture, that is able to do reinforcement learning based on raw visual input data. In contrast to previous approaches, not only the control policy is learned. In order to be successful, the system must also autonomously learn, how to extract relevant information out of a high-dimensional stream of input information, for which the semantics are not provided to the learning system. We give a first proof-of-concept of this novel learning architecture on a challenging benchmark, namely visual control of a racing slot car. The resulting policy, learned only by success or failure, is hardly beaten by an experienced human player.**

I. Introduction

Making learning systems increasingly autonomous in the sense that they can not only autonomously learn to improve from their own experiences [1], [2], but furthermore achieve this by requiring less and less prior knowledge for their setup, is one of the promising future research directions in machine learning. In this paper, we target at a learning system, that gets a potentially broad stream of sensor information from its environment - here in form of visual data - and reacts with a sequence of control decisions to finally reach a desired goal. In the following, we present a first prototypical implementation of such a learning system, that learns to control a real world environment - a slot car racer in this case - by learning to react appropriately to visual input information provided by a digital camera. The crucial point here is, that no knowledge is provided to the learning system about the semantics of the visual input information (e.g. no classical computer vision methods are applied to extract position of the car etc.), but instead, only a stream of raw pixel information is provided. Also, no prior information about the dynamics of the controlled system is provided. Thus, the task of the learning system is two-fold: first to autonomously learn to extract out of the stream of raw pixel data the state information that is needed to control the car, and secondly, to learn a control policy depending on that representation that achieves the learning goal of driving the car as fast as possible without crashing.

We describe a learning control architecture, that learns by experience of success or failure based on raw visual input data. As a proof-of-concept it is applied to a challenging real-world task, namely camera based control of a slot car [3]. Extracting state information out of raw images is done by a deep encoder neural network, whereas the reinforcement learning task is solved within a fitted Q-learning framework (see e.g. [4], [5], [6]).

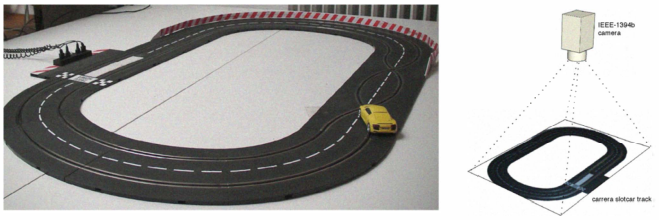

Fig. 1. The visual slot car racer task. The controller has to autonomously learn to steer the racing car by raw visual input of camera images.

II. Related Work

Deep neural networks have been quite successful in unsupervised learning of sparse representations of high-dimensional image data. This includes training with restricted boltzmann machines [7] as well as pre-training deep networks by stacking flat multi-layer perceptrons layer-by-layer on each other [8], [9]. Applications of these deep learning techniques so far include learning interpretable visualizations of high-dimensional data [7], letter recognition [10], face recognition, [11] and object recognition [12] as well as natural language processing [13]. In several of these tasks deep architectures have been found to yield results superior to flat networks or directly trained random initialized deep networks [14], [15], PCA and to more traditional non-linear techniques for learning manifolds [7], [16]. Actually, at present, the most successful techniques in the well-known MNIST letter recognition task are based on deep architectures [17], [18]. All the mentioned papers have in common that they concentrate on learning either representations (e.g. for visualization purposes) or classifications (using supervised learning). One thing that so far hasn't been investigated thoroughly is whether deep learning can also be a basis for learning visual control policies in reinforcement learning—that is solving sequential decision problems with a high-dimensional state space without any supervision, just by means of trial and error. Although the described approaches mainly used small image patches or rather structured environments, there is the reasonable belief that this technique will

scale towards bigger real-world problems [17], [19], [20], [21] and may form a sound basis for automatically learning useful state representations in reinforcement learning.

So far, there have been mainly two different proposals for how to learn policies from high-dimensional image data. Gordon [22] and Ernst [23] applied their "fitted" methods directly to approximating value functions in high-dimensional image space. Having been applied only to small, simulated toy problems and having never been tested for generalizing among similar images, there remain strong doubts this "direct" approach scales to anything beyond simple toy problems. Jodogne and Piater proposed to apply an extra tree algorithm[24] to feature descriptors extracted from images and to then approximate a value function in the constructed feature space [25], [26]. Featuring a two-stages approach build on well-studied image feature descriptors (SIFT), there is reasonable expectation this approach will be able to exhibit some generalization among similar images. This approach has been tested on more realistic simulations than the first approach [27], [25] but still misses a successful demonstration on a real-world problem. By relying on handcrafted feature descriptors and extraction algorithms, this approach only solves half of the feature-learning problem, only learning the *selection* of useful features, not their autonomous extraction. This ambitious goal is pursued in the following work.

III. The Deep-Fitted-Q Framework

The task of the autonomous learning controller is to take raw visual data as input and compute an appropriate control action. Here, this is handled in a reinforcement learning setting, where one seeks an optimal control policy, that maximizes the cumulated reward over an infinite number of time steps, i.e. $\pi^*(s) = \max_\pi \sum_{t=0}^{\infty} \gamma^t r_t$. Here r_t denotes the immediate reward, given in each decision step and γ is a discount parameter, that decays the influence of future rewards over current ones. For a concrete selection of these parameters see below.

One of the crucial points when plugging visual data directly into a learning control system, is that a high-dimensional stream of pixel-data must be reduced to its essential information. A classical approach would typically analyse each image pixelwise and extract relevant information by means of machine vision methods. In this work, the challenge is to design a learning system, that is able to extract relevant information out of the input data completely autonomously.

The deep-fitted-Q (DFQ) framework [28], [29] assumes the following situation: input to the control system is a continuous series of images taken by a digital camera, observing the system to be controlled. Image information thus is given as a vector of pixel-data recorded at discrete time steps, denoted by s_t. Since s_t contains the whole image information, it is typically high-dimensional (typically $d >> 1000$ dimensions).

The next step is to learn a mapping of the raw input information s_t to a condensed information vector $z_t = \phi(s_t)$. In contrast to s_t, z_t is of low dimension (e.g. 2 or 3). This mapping $\phi : R^d \mapsto R^2$ is learned autonomously via a deep-autoencoder approach [7], [8], [28], [30].

The actual control signal a_t is then computed on the basis of the condensed information, i.e. $a_t = \pi^*(z_t)$. The optimal control policy π^* is given by optimizing the cumulated expected sum of rewards, i.e. $J^{\pi^*} = max_\pi \sum_{t=0}^{\infty} \gamma^t r_t$. To find π^*, fitted Q iteration (FQI) is used as the basic learning scheme [4], [31]. The controller, that maps the condensed information vector z_t to a concrete action, is called FQI-controller accordingly. See section V-D for a description of the FQI approach used here.

In dynamical systems, sensor information usually does not contain complete access to state information. A common approach is then to also provide previous sensor and action information as input to the FQI controller. This method is also used in the presented system and described in detail in section V-C.

From the viewpoint of the overall controller, learning information is provided in terms of tuples of camera image, action, reward and next camera image (s_t, a_t, r_t, s_{t+1}). The immediate reward signals reflect the user's notion about what the learning system should achieve. The definition of the $r_t's$ for the visual slot car task is given in section VI.

To summarize, three main steps can be distinguished:

- learn to encode the raw visual information by a deep encoder neural network
- compute state information by using the encoded input information and - if necessary - information from previous time steps
- learn to compute the action based on current (condensed) state information by a Fitted-Q scheme.

Learning can be done in an interwoven mode, where both the deep neural encoder and the inner FQI-controller are learned incrementally while interacting with the real system.

Note, however, that it is also possible to train both parts, neural encoder and inner FQI-controller completely seperately [29]. This has the advantage, that one can check correct functioning of both parts individually. Since training of each individual module takes quite long, this approach is applied here.

IV. The Visual Neuro-Racer

A. The task

The goal is to learn a control policy for a slot-car, that moves the car as fast as possible along a given track without crashing. The learning system shall learn this behaviour in a typical reinforcement learning setting by the only training information of success or failure. As a novelty, input information to the learning system is given by unprocessed, i.e. raw camera sensor information. The learning control system therefore has to learn to filter out the relevant information out of the image data and upon this information learn a control policy. Only if both steps are successful, it can achieve to optimize the cumulated reward and fulfill its learning goal (see figure 1).

Besides its large input dimension, which is far beyond that of typical reinforcement learning tasks, the visual slot

1) **Initialization** Set episode counter $k \leftarrow 0$. Set sample counter $p \leftarrow 0$. Create an initial (random) exploration strategy $\pi^0 : \mathcal{Z} \mapsto A$ and an initial encoder $\phi : \mathcal{S} \mapsto_{W^0} \mathcal{Z}$ with (random) weight vector W^0. Start with an empty set $\mathcal{F}_\mathcal{S} = \oslash$ of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$
2) **Episodic Exploration** In each time step t calculate the feature vector z_t from the observed image s_t by using the present encoder $z_t = \phi(s_t; W^k)$. Select an action $a_t \leftarrow \pi^k(z_t)$ and store the completed transition in image space $\mathcal{S}$: $\mathcal{F}_\mathcal{S} \leftarrow \mathcal{F}_\mathcal{S} \cup (s_p, a_p, r_{p+1}, s_{p+1})$ incrementing p with each observed transition.
3) **Encoder Training** Train an auto-encoder (see [7]) on the p observations in $\mathcal{F}_\mathcal{S}$ using RProp during layer-wise pretraining and finetuning. Derive the encoder $\phi(\cdot; W^{k+1})$ (first half of the auto-encoder). Set $k \leftarrow k + 1$.
4) **Encoding** Apply the encoder $\phi(s; W^k)$ to all transitions $(s_t, a_t, r_{t+1}, s_{t+1}) \in \mathcal{F}_\mathcal{S}$, 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 $z_t = \phi(s_t; W^k)$.
5) **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 [4]) 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 $\bar{q}_t^{i+1} = r_{t+1} + \gamma \max_{a' \in A} \hat{Q}^i(z_{t+1}, a')$ [4] 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 $\bar{Q}^k$.
6) **Outer loop** If satisfied return approximation $\bar{Q}^k$, greedy policy π and encoder $\phi(\cdot; W^k)$. Otherwise derive an ϵ-greedy policy π^k from $\bar{Q}^k$ and continue with step 2.

Fig. 2. General algorithmic scheme of Deep Fitted Q with the two basic building blocks encoder training and fitting the Q values (FQI).

car problem offers another interesting challenge to learning controllers: the optimal (e.g. fastest) control policy is always close to failure.

B. System setup

Control decisions are made in discrete time steps. In the following, we set this value to be four decisions per second (the exact value is $\triangle_t = 0.267s$, resulting from the image frequency of the camera). This choice is a compromise between a small time step that allows fine granularity control and a large time step that allows all relevant information to be captured without information being delayed between different time steps. At each time step, the controller is provided with the current image of the digital camera, s_t, and is expected to respond with a control action a_t, which corresponds to a voltage, applied to the car on the track.

V. A VISUAL-INPUT BASED CONTROLLER FOR THE VISUAL SLOT-CAR RACER

A. Overview

The control system proceeds in three basic steps, which are described in more detail in the following subsections:

- process input data by an autonomously learned deep neural encoder network
- build state information based on encoded image information and temporal information

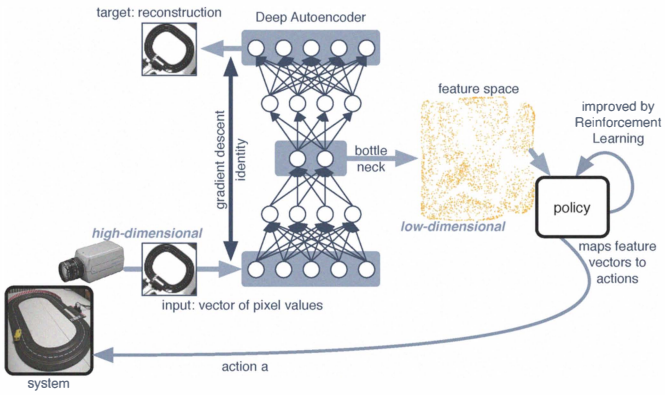

Fig. 3. Overview of the system.

- learn a control policy by a cluster-based Fitted Q learning method called cluster-RL

B. The neural deep encoder

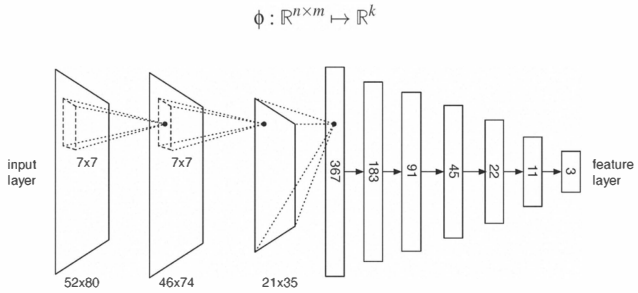

Fig. 4. Encoder part of the deep neural autoencoder.

1) Structure: The neural encoder net is a multilayer perceptron with a special "sparse" wiring structure in the first layers, that is inspired by principles found in the structure of the vertebrate retina and has been widely adapted in the neural networks community [32], [33].

The size of the input layer is 52x80 = 4160 neurons, one for each pixel provided by the digital camera. The input layer is followed by two hidden layers with 7x7 convolutional kernels each. The first convolutional layer has the same size as the input layer, whereas the second reduces each dimension by a factor of two, resulting in 1 fourth of the original size.

The convolutional layers are followed by seven fully connected layers, each reducing the number of its predecessor by a factor of 2. In its basic version the coding layer consists of 2 neurons.

Then the symmetric structure expands the coding layer towards the output layer, which shall reproduce the input and accordingly consists of 4160 neurons.

Althogether, the network consists of about 18000 neurons and more than one million connections. The actual number of free parameters is less, since connections in the convolutional layers are shared weight connections.

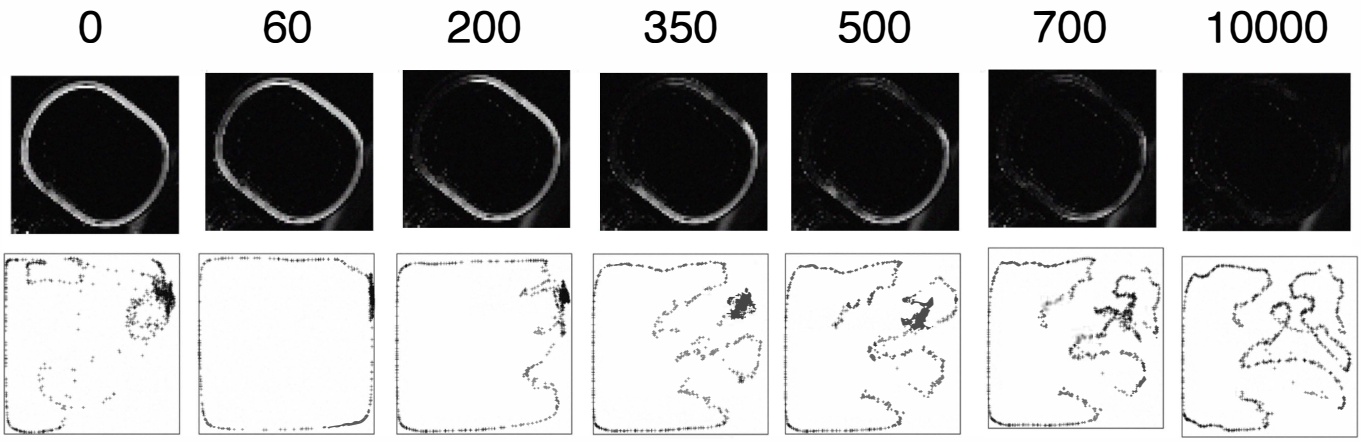

Fig. 5. Error plot of the neural decoder during fine training, i.e. after completion of pretraining. The brightness of each pixel is proportional to the error made at the corresponding output neuron. Upper left corner shows the situation after pretraining is finished: An error is made along bascially all positions of the car along the track, i.e. the network is not able to reproduce the position of the car reliably. Outside the track, no error is observed any more; the reproduction of these pixel values is already correctly learned in the pretraining phase. The lower row shows the activation of the two hidden neurons in the coding layer (neuron 1: x-axis, neuro 2: y-axis). With reduction of the reproduction error, the mapping of the image to the x-y plane nicely 'unfolds', finally yielding a condensed coding, that allows a good distinction between different input situations.

2) Training: Training of deep networks per se is a challenge: the size of the network implies high computational effort; the deep layered architecture causes problems with vanishing gradient information. We use a special two-stage training procedure, layer-wise pretraining [8], [29] followed by a fine-tuning phase of the complete network. As the learning rule for both phases, we use Rprop [34], which has the advantage to be very fast and robust against parameter choice at the same time. This is particularly important since one cannot afford to do a vast search for parameters, since training times of those large networks are pretty long.

The training set of the deep encoder network consists of 7000 images, generated while moving the slot-car by a constant speed. For the layer-wise pretraining, in each stage 200 epochs of Rprop training were performed. The result of reconstruction after the pre-training phase can be seen in the image in the upper left corner of figure 5. The network has learned to reconstruct the still parts of the image (near-zero error outside the track) and encodes some useful information in the code layer (partially unfolded feature space displayed in the lower row of the figure), but has problems with the pixel positions, where the image information varies due to the movement of the car. To reduce this error is the goal of the fine-tuning phase. The error-diagrams in figure 5 show the reduction of the error in the fine-tuning phase. After 10,000 epochs, the error has been significantly reduced and nearly perfect reconstruction of all images is achieved (nearly no bright spots in the error diagram in the upper right corner of figure 5). The remaining white fragment in the lower left corner of the last error image is caused by a sun beam lighting the floor in only part of the images. The lower row of figure 5 shows the activation of the two hidden neurons in the coding layer (neuron 1: x-axis, neuron 2: y-axis). With reduction of the reproduction error, the mapping of the image to the x-y plane plane nicely 'unfolds', finally yielding a condensed coding, that allows a good distinction between different input situations.

Let us emphasis the fundamental importance of having the feature space already partially unfolded after pretraining. A partially unfolded feature space indicates at least some information getting past this bottle-neck layer, although errors in corresponding reconstructions are still large. Only because the autoencoder is able to distinguish at least a few images in its code layer it is possible to calculate meaningful derivatives in the finetuning phase that allow to further "pull" the activations in the right directions to further unfold the feature space.

To reduce training times as much as possible and to allow the application of the trained encoder in a real-time setting, efficient coding of the neural network implementation is required, that additionally exploits parallelism as much as possible. We therefore re-implemented our C/C++ neural network library from scratch and made it publicly available at GitHub[1] as the open-source package n++2. N++2 exploits SIMD[2] architectures by using cBLAS[3] functions for propagating activations and partial derivatives. Furthermore, it makes use of the associative property of the calculated error term as sum of errors of all individual patterns by splitting the k training patterns p into n batches and calculating n parts of the sum on as many parallel threads using identical copies of the neural network's structure:

$$\sum_{p=1}^{k} E_p = \underbrace{\sum_{p_1=0k/n+1}^{k/n} E_{p_1}}_{\text{on 1st copy}} + \underbrace{\sum_{p_2=1k/n+1}^{2k/n} E_{p_2}}_{\text{on 2nd copy}} + \cdots + \underbrace{\sum_{p_n=(n-1)k/n+1}^{k} E_{p_n}}_{\text{on n-th copy}} .$$

[1]https://github.com/salange/NPP2
[2]Single Instruction Mutliple Data
[3]C version of Basic Linear Algebra Subroutines

The same splitting is used for calculating partial derivatives. Altogether, training the deep encoder network takes about 12 hours on an 8-core CPU with 16 parallel threads.

C. Building state information

The output-layer of the deep encoder network delivers an encoding of the high-dimensional, static input image to a condensed low-dimensional representation of the current scene as captured by the camera. For most dynamical systems, however, complete state representation also requires temporal elements. A common way to go is, to add information about past values of sensor and action information to the state description. Two ways to realize this are discussed in the following.

1) The tapped-delay-line approach: In the simplest case, state information is approximated by providing present and previous input information to the control system. Here, this can be done by using encoded information from previous and current images. This is called the 'DFQ-base'-approach in the following. State information for the slot car task consists of the encoding of the current and the previous image, z_t and z_{t-1} respectively, and the previous action a_{t-1}.

2) The Kohonen-Map trick: As the spatial resolution is non-uniform in the feature space spanned by the deep encoder, a difference in the feature space is not necessarily a consistent measure for the dynamics of the system. Hence, another transformation, a Kohonen map ($K : R^k \mapsto R$), is introduced to linearize that space and to capture the a priori known topology, in this case a ring. A sequence of several thousand images of the system is recorded while the slot-car is moving at a fixed speed. The data is mapped into the feature space and the resulting feature vectors $z_t = \phi(s_t)$ are fed into the SOM by using the first round around the track as the starting prototypes for the ring. These points are augmented by introducing additional points in the center between two consecutive points. After this initialization, regular SOM training is performed using the transformed data of all images as input. The neighborhood function is defined as $D(i,j) = min(|i-j|, n-1-|i-j|)$ and while learning the influence on neighboring points is cropped at a distance of 5. The resulting SOM in the experiments was trained for 50 iterations ($\eta = 0.5$) and had 269 prototypes. In the application phase, feature points are projected onto the embedded topology by applying an orthogonal projection (see figure 7) using the algorithm shown in figure 8. The algorithm determines the prototype closest to the feature point and projects it onto the line segment to a neighboring prototype. After the projection, the result is normalized. The one-dimensional space spanned by this projection is continuous and temporally uniform as the slot-car was travelling at a fixed speed during image acquisition, roughly producing a uniform distribution of the training images along the track. Therefore this space is better suited to derive dynamics of the system state than the non-uniform feature space. The state representation, called 'DFQ-SOM'-approach, uses the mapped encoding of the current and a difference with the previous image[4] as well as the previous action, resulting in $(s'_t = K(z_t), \dot{s}'_t = \|K(z_t) - K(z_{t-1})\|, a_{t-1})$.

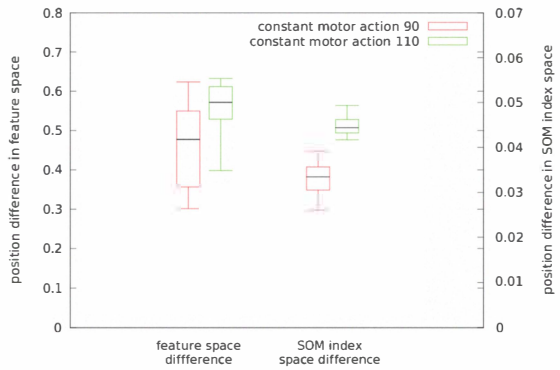

Fig. 6. Speed estimate based on feature point difference (left) as well as SOM index space difference (right) given two fixed action policies (resulting in two different constant real-world speeds). The feature point distance is not able to separate the two velocities reliable as the distributions of estimates of both constant-action policies overlap. The estimates based on differences in the SOM index space are more reliable as distributions do not overlap.

In fig. 6 we can see the benefit for using the SOM-positions for calculating the state differences for estimating velocities. The graph displays the velocity estimates for two different actions at a problematic position in the state space (a jump). Whereas the feature space difference does not allow for a good separation of the slower and faster actions (displayed: N measurements of the velocity-difference of a passing-by car), the difference on the embedded SOM is much more expressive and allows for a good separation. That is possible because the prototypes do the same jump (one prototype before the jump, one after) as the data, but the 'normalized' inter-prototype distances do not have such a high variance and are a much more reliable encoding of the position.

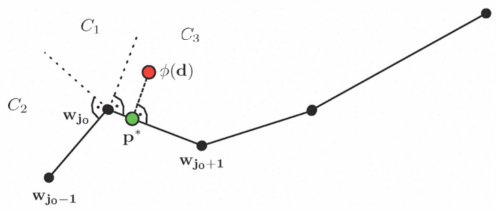

Fig. 7. Orthogonal projection of feature point $\phi(\mathbf{d})$ onto the SOM structure: Depending on which side of the closest prototype $\mathbf{w_{j0}}$ the projection will intersect, the algorithm separates three cases: C2 which intersects the line segment to the previous prototype, C3 intersects the line segment to the following prototype and the special case Cl where the feature point is mapped onto the prototype itself.

D. The FQI Controller - approximating the Q-function with ClusterRL

An important decision is what approximator to use for approximating the value function inside the FQI - algorithm

[4]The difference $\|K(z_t) - K(z_{t-1})\|$ approximates the derivative $\dot{s}'$ of the mapped states s' and could be interpreted as a 'feature-space velocity'.

Input: new feature point $\phi(\mathbf{d})$
Output: projection on SOM index space: $\mathbf{p}^*$
$j_0 \leftarrow \min_j \|\phi(\mathbf{d_i}) - \mathbf{w_j}\|\ (\forall \mathbf{w_j} \in W)$ /* determine winner */
$\mathbf{v} \leftarrow \phi(\mathbf{d}) - \mathbf{w_{j_0}}, \mathbf{v_L} \leftarrow \mathbf{w_{j_0-1}} - \mathbf{w_{j_0}}, \mathbf{v_R} \leftarrow \mathbf{w_{j_0+1}} - \mathbf{w_{j_0}}$
$\beta_L \leftarrow \frac{\langle \mathbf{v},\mathbf{v_L}\rangle}{|\mathbf{v}||\mathbf{v_L}|}, \beta_R \leftarrow \frac{\langle \mathbf{v},\mathbf{v_R}\rangle}{|\mathbf{v}||\mathbf{v_R}|}$ /* determine cosines */
if $(\beta_L \leq 0 \wedge \beta_R \leq 0)$ **then**
  $\mathbf{p}^* \leftarrow \mathbf{w_{j_0}}$ /* case C_1 */
end
else if $\beta_1 < \beta_2$ **then**
  $\mathbf{p}^* \leftarrow \mathbf{w_{j_0}} - \frac{\langle \mathbf{v},\mathbf{v_L}\rangle}{|\mathbf{v}|}$ /* case C_2 */
end
else
  $\mathbf{p}^* \leftarrow \mathbf{w_{j_0}} + \frac{\langle \mathbf{v},\mathbf{v_R}\rangle}{|\mathbf{v}|}$ /* case C_3 */
end

Fig. 8. Orthogonal projection algorithm used to map feature points of arbitrary dimensionality onto the SOM structure defined by the list of prototypes $\mathbf{w_j} \in W$ (see fig. 7 for details on the cases)

(step 5 of DFQ as shown in figure 2). Stable convergence is guaranteed for approximators known as averagers [35], [29]. This includes a specific type of non-parametric kernel-based approximators [35] as well as a whole class of 'standard' but non-expanding (parametric) function approximators [22], [4], [29]. We have decided to use a parametric grid approximator ('ClusterRL' approach, see [29]) in this experiment for two reasons: first, the parametric representation allows us to throw the training data away after training, and second, we expected the sharp cell borders of this approximator to be better suited than a 'smoothing' kernel-based approach for the approximation of the sharp discontinuities ('crash-boundary') in the problem's state space.

The motivation for the ClusterRL approach is the following: the continuous state space is partitioned into a number of distinct cells that are than each assigned a single q-value for each action, locally approximating the Q-function. Thus, all states in the same cell share the same Q-values. This is similar to the extra-trees approach used in [4]. However, the grid approximator used in our ClusterRL approach, needs only a single parameter (number of cluster centers), is completely data-driven and allows for irregular shapes of its cells. This is achieved by doing a cluster analysis of the observed states (here: using k-means, but other methods like SOM [36], [37] and neural gas [38] are also possible) and then constructing a Voronoi-diagram from the cluster centers found (in 2D this looks like an irregular grid). Each cell in the Voronoi-diagram—defining the 'area of influence' or 'receptive field' of the cluster center (prototype neuron)—becomes one cell of the state space's partition.

The details of ClusterRL are shown in the pseudo code below. The algorithm is started with the number k of cluster centers to use, an initial q-value $\bar{q}^0$ (typically but not necessarily $\bar{q}^0 = 0$), and the observed state transitions $\mathcal{F}$. Using only the starting states in the state transitions, k cluster centers are initialized by placing them on the positions of k randomly selected states. Afterwards, this initial distribution $\mathcal{C}_0$ of cluster centers is improved applying the the k-means algorithm.

Input : $k, \bar{q}^0, \mathcal{F}$
Result : Appoximation $\bar{Q}$ of the optimal q-function Q^*
$i \leftarrow 0$;
$X \leftarrow$ `extract observed states(`$\mathcal{F}$`);`
$C_0 \leftarrow$ `initialize prototypes(`X,k`);`
$C_i \leftarrow$ `k-means(`C_{i-1}, X`);`
$\hat{Q}^0 \leftarrow$ `construct grid approximator(`C_i, $\bar{q}^0$`);`
$\bar{Q} \leftarrow$ `Fitted Q-Iteration(`$\hat{Q}^0, \mathcal{F}$`);`

The resulting distribution of cluster centers is then used to approximate the value function. Basically, this is done by storing one q-value for each of the actions at each of the cluster centers (prototypes for their region of influence). Initially, these are set to $\bar{q}^0$. When accessing one particular state in the state space, we look up the cell it is in (this is done by simply finding the closest cluster center) and then using the values stored for this particular cell at the cell's cluster center (or prototype).

During training, the FQI-algorithm alternates between calculating training patterns $\mathcal{P} = \{s_t, a_t; \bar{q}_t |\ t = 1, \ldots, p\}$ in a DP step and training a function approximator on these patterns. In the case of using the grid approximator, 'training' the function approximator simply means setting the q-values at the cluster centers to the average of the training pattern that fall into the center's cell. Technically, for updating the q-value q_a of action a at a prototype c_i, we use the subset $\mathcal{P}_a = \{(s_t, a_t; \bar{q}_t) \in \mathcal{P} |\ a_t = a\}$ of training pattern $(s_t, a_t; \bar{q}_t)$ that used action a and calculate from it the new Q-value q_a as the weighted average

$$q_a = \sum_{(s_t, a_t; \bar{q}_t) \in \mathcal{P}_a} \bar{q}_t \delta_i(s_t) \Big/ \sum_{(s_t, a_t; \bar{q}_t) \in \mathcal{P}_a} \delta_i(s_t)\ , \tag{1}$$

with $\delta_i(s_t)$ being the indicator function

$$\delta_i(s) = \begin{cases} 1, & \text{iff } s_t \text{ is within cell of } c_i \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The value q_a remains unchanged, if there's no training pattern in the cluster center's area of influence.

This supervised training procedure can be done with a single sweep through the pattern and, therefore, is by orders of magnitude faster than training a neural network for several epochs. The most expensive operation is assigning the training pattern to the correct cells in the partition (finding the closest prototype). But, since during the FQI procedure the structure of the grid remains unchanged, the assignement of training pattern to cells in our implementation is calculated only once, and then 'cached' with a linked-list data structure.

VI. Results

To test our approach, the learning controller was applied to the track shown in figure 1. The controller has 4 actions (0,90,120,200) available, corresponding to the voltage that is

TABLE I
RESULTS.

CONTROLLER	AVG. TIME PER ROUND	CRASH-FREE
RANDOM	-	NO
CONSTANT SAFE	6.408S	YES
DFQ-BASE	2.937S	YES
DFQ-SOM	1.869S	YES

fed to the car. When a voltage of 0 is applied, the car is actively decelerated. The higher the voltage, the faster the car goes, but the relationship is nonlinear. The reward given corresponds to the magnitude of the action applied, i.e. the reward is $r_t = 90$ if action $a_t = 90$ is applied. When the car crashes, i.e. falls off the track, a negative reward (punishment) of -1,000,000 is given. By this specification, the controller is forced to learn a control law, that maximizes the cumulated speed of the car, under the constraint of always avoiding crashes. For all experiments a discount rate of $\gamma = 0.1$ was used.

The time step was set to $\triangle_t = 267ms$. Each training episode has a length of 80 steps (corresponds to about 21s). To test the performance, the test episode length was set to 400 steps.

First, two non-learning controllers were being applied to give an idea of the difficulty of the task. Policy 'random' chooses the actions completely at random. Not surprisingly, applying this policy, the car often stops, goes slowly, or accelerates too much and crashes. Policy 'safe' applies the highest possible constant action, that does not crash the car. An average round using the constant policy takes 6.408s. This can be seen as a baseline performance. Going faster than this policy will actually require information of the state of the car and the experience, where one can go faster and where one has to be more careful.

Learning was done by first collecting a number of 'baseline' tuples, which was done by driving 3 rounds with the constant safe action. This was followed by an exploration phase using an ϵ-greedy policy with $\epsilon = 0.1$ for another 50 episodes. Then the exploration rate was set to 0 (pure exploitation). This was done until an overall of 130 episodes was finished. After each episode, the cluster-based Fitted-Q was performed until the values did not change any more. Altogether, the overall interaction time with the real system was a bit less than 30 minutes.

The first learning controller approach, DFQ-base, shows already very good performance. Input to the inner FQI controller is a 4 dimensional state representation $(z_1, z_2, ||\Delta z||, a_{t-1})$, where $z_{1,t}$ and $z_{2,t}$ are the activation values of the neurons of the encoder layer of the deep encoder net, and $||\Delta z||$ is the distance to the activations of the previous time step in euclidean norm. Here, 400 clusters were used to represent the Q-function in the cluster-RL approach.

The average round takes 2.937s which is more than twice as fast as the baseline. This performance corresponds approximately to what a good human player will achieve.

Using the above described 'Kohonen-Map trick', the performance can be boosted even more. The DFQ-SOM appraoch uses 3-dimensional state information $(\tilde{z}_t, |\Delta\tilde{z}_t|, a_{t-1})$, where $\tilde{z}_t$ is the real-valued position of the encoded image information in the SOM and $|\Delta\tilde{z}_t|$ represents the absolute difference of current encoded image information and previous encoded image information within the SOM.

For the DFQ-SOM approach, finally a number of 800 cluster neurons gave the best overall result. The average time achieved for a round is 1.869 s, which is 3 times as fast as the 'constant'-controller and improves upon the already well working DFQ-base approach by another 50%. To achieve this performance, the controller not only has to correctly extract knowledge about position of the car by the deep neural encoder, but also has to learn, when its possible to accelerate and when it must decelerate in order to avoid crashes with very high reliability. The latter is learned by the cluster-RL based FQI-controller. The times achieved by the 'DFQ-SOM' approach are hard to beat by a human. A visualisation of the final control policy is given in figure 9.

Fig. 9. Visualisation of the learned policy (car is driving counter-clockwise): the controller has nicely learned to accelerate on pieces, where it is save (e.g. straight track or boundaries the car could lean its tail on at the top corner) and to decelerate at dangerous positions (e.g. at the track crossing or in the curve without boundaries).

VII. CONCLUSIONS

We described a first prototypcial realisation of an autonomous learning system, that is able to learn control based on raw visual image data. By 'raw' we mean, that no semantics of the image are a priori provided to the learning system. Instead, the system must learn to extract a relevant representation of the situation in the image, in order to fulfill its overall control task. While learning the representation is done with a neural deep encoder approach, learning the control policy is

based on cluster-RL, a Fitted-Q batch reinforcement learning scheme.

We have further demonstrated, that with the use of additional prior knowledge (in this case that the system is moving on a two-dimensional manifold) we can use additional self-organized mappings (the Kohonen-Map trick) to further improve internal representation, which results in improved control performance. In this study, this knowledge came from outside, but future learning systems might be able to extract such meta-knowledge autonomously from experience and use it accordingly.

The resulting system was able to autonomously learn to control a real slot car in a fashion, that is hard to beat by a human player.

Being able to autonomously handle raw, high-dimensional input data and acting reasonable on the autonomously extracted information offers exciting opportunities for the control of complex technical systems, where an increasing amount of sensory input information is available.

REFERENCES

[1] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement Learning for Robot Soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–74, 2009.

[2] T. Gabel and M. Riedmiller, "Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents," *International Journal of Information Technology and Intelligent Computing*, vol. 24, no. 4, 2008.

[3] 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*, 2009.

[4] 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.

[5] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to Drive in 20 Minutes," in *Proc. of the FBIT 2007*, Jeju, Korea, 2007.

[6] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine Learning*, vol. 27, no. 1, pp. 55–74, 2011, 10.1007/s10994-011-5235-x. [Online]. Available: http://dx.doi.org/10.1007/s10994-011-5235-x

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

[8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and Q. Montreal, "Greedy Layer-Wise Training of Deep Networks," in *Advances in Neural Information Processing Systems 19*, 2007, pp. 153–160.

[9] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[10] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, june 2005, pp. 539 – 546 vol. 1.

[11] H. Mobahi, R. Collobert, and J. Weston, "Deep learning from temporal coherence in video," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 737–744.

[12] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proceedings of the 20th international conference on Artificial neural networks: Part III*, ser. ICANN'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 92–101.

[13] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 160–167.

[14] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. of the 24th International Conference on Machine Learning*, 2007, pp. 473–480.

[15] D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Proc. of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS?09)*, 2009, pp. 153–160.

[16] R. R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *AI and Statistics*, 2007.

[17] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Proc. of CVPR '07.*, 2007.

[18] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[19] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun, "Deep belief net learning in a long-range vision system for autonomous off-road driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008*, 2008, pp. 628–633.

[20] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, "A committee of neural networks for traffic sign classification," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 1918–1921.

[21] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 1096–1104.

[22] G. Gordon, "Stable Function Approximation in Dynamic Programming," in *Proc. of the 12th Int. Conf. on Machine Learning (ICML)*, 1995, pp. 261–268.

[23] 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.

[24] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[25] S. Jodogne and J. Piater, "Closed-loop learning of visual control policies," *Journal of Artificial Intelligence Research*, vol. 28, pp. 349–391, 2007.

[26] ——, "Interactive learning of mappings from visual percepts to actions," in *Proc. of the 22nd international conference on Machine learning*, 2005, pp. 393–400.

[27] S. Jodogne, C. Briquet, and J. Piater, "Approximate Policy Iteration for Closed-Loop Learning of Visual Tasks," in *Proc. of the European Conference on Machine Learning*, 2006.

[28] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *Proc. of the International Joint Conference on Neural Networks (IJCNN 2010)*, Barcelona, Spain, 2010.

[29] S. Lange, "Tiefes Reinforcement Lernen auf Basis visueller Wahrnehmungen," Dissertation, Universität Osnabrück, 2010.

[30] S. Lange and M. Riedmiller, "Deep learning of visual control policies," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2010.

[31] S. Lange, T. Gabel, and M. Riedmiller, "Batch Reinforcement Learning," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Springer, in press, 2011, in press.

[32] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[33] 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.

[34] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," in *Proc. of the Int. Conf. on Neural Networks*, 1993, pp. 586–591.

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

[36] T. Kohonen, *Self-Organizing Maps*, zweite edition ed., ser. Springer Series in Information Sciences. Springer, Heidelberg, 1997, vol. 30.

[37] B. Hammer and T. Villmann, "Effizient Klassifizieren und Clustern: Lernparadigmen von Vektorquantisierern," *Künstliche Intelligenz*, vol. 6, no. 3, pp. 5–11, 2006.

[38] B. Fritzke, "A growing neural gas network learns topologies," *Advances in Neural Information Processing Systems 7*, 1995.