Learn to Swing Up and Balance a Real Pole Based on Raw Visual Input Data

Jan Mattner*, Sascha Lange, and Martin Riedmiller

Machine Learning Lab
Department of Computer Science
University of Freiburg
79110, Freiburg, Germany
{mattnerj,slange,riedmiller}@informatik.uni-freiburg.de
http://ml.informatik.uni-freiburg.de

Abstract. For the challenging pole balancing task we propose a system which uses raw visual input data for reinforcement learning to evolve a control strategy. Therefore we use a neural network – a deep autoencoder – to encode the camera images and thus the system states in a low dimensional feature space. The system is compared to controllers that work directly on the motor sensor data. We show that the performances of both systems are settled in the same order of magnitude.

Keywords: neural network, pole balancing, deep autoencoder, reinforcement learning, visual input

1 Introduction

One of the main applications of reinforcement learning (RL) is controlling a dynamic system. In the ideal case the RL controller adapts automatically to a new system without the need to incorporate additional domain specific knowledge. Thus part of the setup has to be learned as well as the controlling task itself. A promising way is to use the raw and high dimensional image data of a camera, which monitors all relevant parts of the dynamic system, in order to learn a low dimensional feature space, that can be used for classical RL methods [1]. However when applying this to a real dynamic system, there appear three major problems. Firstly we have a delay. The camera information is not instantaneously available and the applied actions will not immediately have an effect. This results in observation and action delay, which cannot be handled by standard RL methods. Secondly, an image is just a snapshot of the dynamic system, so we have to add motion information of the moving parts of the system. Therefore a correct velocity estimation is needed. Thirdly, by creating a low dimensional map (feature space) of a manifold some information is always lost or obscured. This noise in the feature space may disturb the actual control task.

We analyze these problems and apply the proposed solutions to the well-known pole balancing task (see figure 1). Here the controller has to apply actions via a motor to a pole with a weight at its end in order to swing up and balance

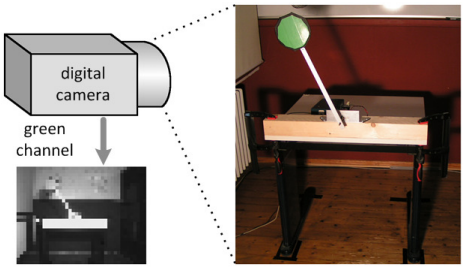

Fig. 1. The pole mounted on a table. The controller has to learn to swing up and balance the pole using only the downsampled visual data from a camera in front of the pole.

the pole in an upright position. The actions are so weak such that from the start position a constant action doesn't bring the pole into the goal area. So the controller has to learn to really swing the pole in both directions to increase it's momentum and beeing able to reach the top. For reducing the dimensionality of the raw image data and creating the feature space a deep encoder neural network is used. The actual controller for the pole balancing task uses a kernel based approximate dynamic programming method [2].

2 Related Work

The pole balancing task and its dynamics are well known [3]. It is also known as 'inverted pendulum', 'rotary pendulum' or 'cart-pole task' with an additional cart which has to be moved instead of direct motor interaction with the pole. It exists as a simulation [4] or real task [5]. In early research [6] a neural network was used to control the cart with small input images based on the simulation data. Visual input was used for a fixed control strategy too [7, 8], however the main issue was to explicitly extract and calculate the position of the cart and the angle of the pole by handcrafted methods. We instead use deep neural networks [9] to autonomously encode the system state and learn a control strategy by RL methods. The deep networks were quite successful [10] and have proven to be able to provide a useful feature space for RL methods [1]. In our work we want to use this approach for the challenging pole balancing task and compare its performance with strategies which are learned directly on motor sensor data instead of the feature space.

3 The Feature Space

In order to apply RL methods to control the system we have to reduce the dimensionality of the input image data. Instead of manually extracting features, which describe the distinct system states, we are using here a deep autoencoder [9]. This is a neural network that can be divided into two separate networks:

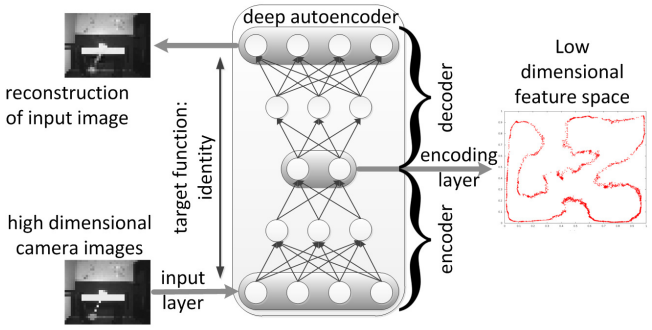

Fig. 2. Overview of an autoencoder.

the encoder and decoder (see figure 2). The input layer neurons of the encoder correspond to the pixel values of the camera images. After multiple hidden layers the encoder reduces the information of the image to a small number of output neurons, which build the encoding layer. The topology of the decoder is mirrored, so these encoding neurons are the input layer of the decoder and its (overall) output layer is again an image, ideally the same image that has been used as the encoder's input. Hence the autoencoder is actually trained to yield the identity function of the camera images and therefore pushing all information through the bottleneck of the encoding layer, which then contains all necessary features to describe the visually perceptible system state. For a detailed description see [1]. The architecture of the used autoencoders is described in 5.1

This feature space has the property that visually close system states are represented by points in the feature space which are close together as well. Thus if there are only small changes, the evolution of the system states over time is reflected as a line of sample points through the feature space. In the case of the pole balancing task the pole can only turn around one axis. This circle topology causes the feature points to form a closed strap (see figure 3). Due to the used gradient descent update method the autoencoder may get stuck in a bad local optimum. For a 2D feature space this could mean dense agglomerations and many crossings of the strap. This can result in two completely distinct system states that cannot be distinguished only by their feature points. This problem is discussed in the next section.

4 Technical Details

In a real dynamic system there are always delays, be it due to sensor information capturing, transport and processing or due to inertia of the physical objects which move within the system. [11] introduces Deterministic Delayed Markov Decision Processes (DDMDP). In the pole balancing task we have deterministic observation and action delay. In [11] is also shown, that a DDMDP can be

reduced to a simple MDP, which is needed for RL methods, by augmenting the state space. Thus for observation delay o and action delay a the state vector has to be augmented by the last $o + a$ actions. So we just have to determine the delays.

The observation delay is quite easy to measure. This is the time from the moment when the camera acquires the image until the point right before the application of the action, for which the image has been used for the first time to choose an action. This of course heavily depends on the synchronisation of sensor and motor.

Determining the action delay is more involved. This is the time from the action application until the action can be uniquely distinguished in the observation data from all other actions. Therefore we design an experiment with two action sequences, which are identical up to time step t, when one sequence applies a different action. Having the velocity data obtained by the motor sensor in several sample runs, we can determine, e.g. with the Welch's t-test (which suits well, as we have continuous data with possibly different variances), for a certain α-level at which time step the velocity sample values are drawn from different distributions.

A correct velocity estimation is crucial for controlling a dynamic system. As the feature space learned by the autoencoder is based on still camera images, we have to manually add velocity information. Adding action history as described above does not suffice unless all past actions are used, which is of course no option since this would drastically increase the dimensionality. Instead we have to estimate the velocity by adding either a past feature point (implicit) or the difference of the current and a past feature point (explicit) to the augmented state representation.

Estimating the velocity in such a way is of course problematic because the distance between the feature points usually does not correspond to the actual changes in the real world. But although the distance and direction in feature space as velocity introduces a systematic error, it works fairly well in practice. This is most likely because in general the velocities in the feature space are locally similar, i.e. states with similar position and velocity in the real world fall into the same area in the feature space and have similar predecessor feature points. Thus the controller can implicitly learn these local similarities. This problem does not inhibit the controller from learning a successful policy and therefore can be neglected.

The automatic mapping of the high dimensional images to the feature space does not work perfectly in all cases. One phenomenon is ambiguities of feature points. Figure 3 shows a two dimensional feature space with a crossing of the strap in horizontal and vertical direction. The reconstruction in (v2) of the vertical point at the crossing is wrong. Apparantly the autoencoder preferred here the horizontal hypothesis since the vertical reconstruction corresponds to the horizontal one in (h2). Only knowing a feature point near the crossing, it is impossible to decide if the pole is pointing up or down. This would be an ambiguous state. A simple solution is to add the information of where the pole was

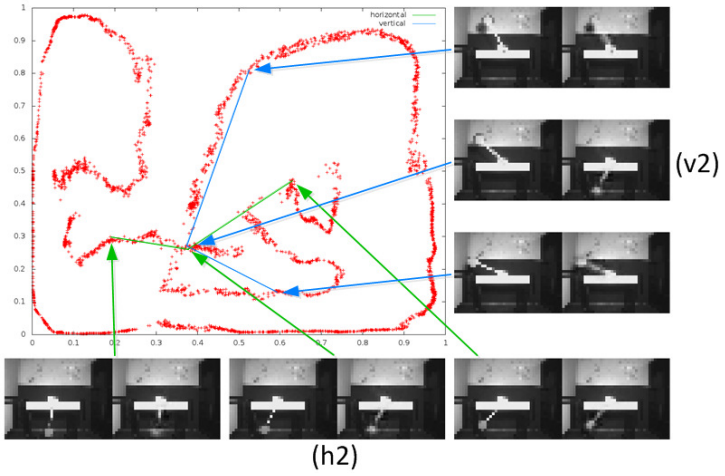

Fig. 3. A feature space with a crossing and each 3 corresponding example images. The example images consist of the original (left) and the reconstruction by the autoencoder (right).

one step before. If we know the previous feature point we can clearly tell that (v2) actually is on the vertical line. However this information is already given if we have added a correct velocity estimation.

5 Results

5.1 Experiment Set-Up

The empirical evaluation was done for the pole balancing task (see figure 1) with camera input (CAM) and direct motor sensor input (SEN). The controller has to learn to swing up and keep the pole upright in the goal area, which is defined to be at angle 0 with a buffer of 0.5 in both directions. The motor can apply actions from -100% (counterclockwise) to $+100\%$ (clockwise). However the controller is restricted to three actions: -40%, 0% (brake), $+40\%$. This way it is not possible to reach the goal area in one go from the start of an episode by just applying constantly the same action. So the controller needs at least one swing to one side and then move down away from the goal and swing up to the other side. The goal is not terminal so it is a continuous time problem. The discounting factor is set to 0.95. For each time step the controller gets a reward of 0 if the pole is in the goal area and -1 for any other state. Each episode consists of 100 steps (or: cycles) but when the camera and motor run out of sync some of these transitions have to be discarded for learning.

The camera runs at $60Hz$, however camera and motor are synchronized such that a full cycle with observation acquisition and action application lasts $100ms$. This comprises 6 images but only the first and fourth image are processed and

encoded to 2-dimensional feature vectors f_1 and f_4. We use explicit velocity estimation, so the state vector at time step t is $s_t = (f_1, f_1 - f_4)$. For a better velocity estimation we use the latest available image instead of the last state. The motor applies the action $83ms$ after the first image of a cycle is taken, which is the observation delay. The Welch's t-test of two action sequences with 30 sample runs each and an α-level of 0.001 has revealed that there is no action delay. This can be explained by the design of one motor cycle. Internally, the motor applies an action at the beginning of its cycle and returns its sensor information at the end of this cycle, after $33ms$. Thus the true action delay is less or equal to $33ms$, however due to the cycle design the actions can be uniquely distinguished within the same cycle of action application. All in all with the total delay of $83ms$ and the overall cycle time of $100ms$ we have to augment the state space by 1 past action. For explicit velocity estimation for the camera experiment the augmented state vector at time t results in the 5-dimensional vector $h_t = (s_t, a_{t-1})$ with $a_i \in \{-40, 0, +40\}$. The sensor experiment uses the same augmented state space, but here the state consists of the two scalars 'pole angle' and 'angular velocity' which leads to a 3-dimensional augmented state vector.

For each camera based controller a separate autoencoder was trained offline on 2000 independent training images of size 40x30, which took about $30min$ on an 8-core CPU. The input layer therefore has 40x30 = 1200 neurons, followed by 2 convolutional hidden layers of size 40x30 and 20x15 with convolutional kernels of size 5x5, which is important for tracking the 3x3 pixel large green marker at the end of the pole. Then fully connected layers of size 150, 75, 37 and 18 up to the encoding layer of 2 neurons complete the encoder. The decoder's architecture is exactly mirrored up to the output layer of size 40x30, which should reconstruct the input image.

5.2 Empirical Evaluation

For the CAM and SEN experiments 5 independent controllers were each trained for 35000 transitions. A test run tests a controller of a given number of transitions for 100 steps. Thus the mean reward per step (MRPS), which serves as performance measurement, of a test run is in the range of the maximum (0) and minimum reward (-1). Therefore an MRPS of e.g. -0.21 means that the pole was in 21 steps not in the goal area. Since a random controller never reached the goal area (MRPS: -1), the beginning of the learning phase was supported by a fixed exploration policy until the controller collected one valid transition into the goal area. Then an ϵ-greedy exploration strategy was applied.

Figure 4 shows the mean learning curves with standard deviation of the 5 independent CAM and SEN controllers with 5 test runs each. Although the CAM controllers worked on higher dimensional input vectors and had to learn the relations of the feature space, they only needed about 2 to 3 times more transitions to come up with a competitive strategy.

The box plot with minimum and maximum in Figure 5 shows the final performance of the best controllers for both experiments and for a CAM controller with a hand-picked autoencoder (well unfolded strap, only one crossing). Each

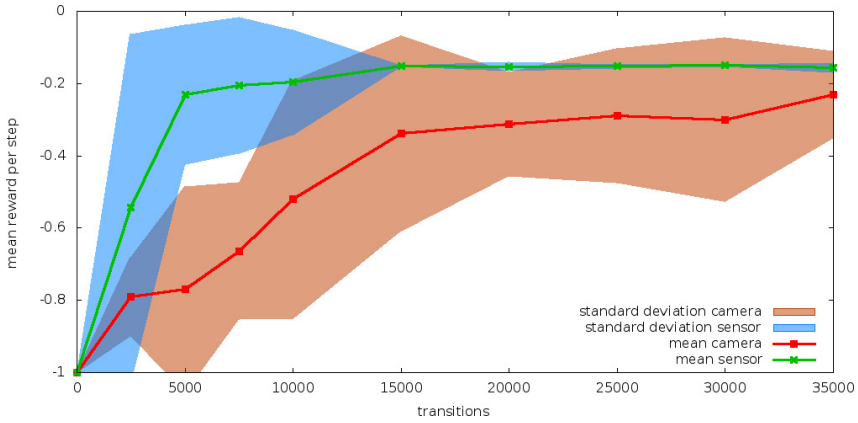

Fig. 4. Mean learning curves with standard deviation.

was tested 20 times. The best SEN controller learned a perfect strategy, so in all tests the pole needed 15 steps into the goal area and stayed there. In 13 out of 20 tests the best CAM controller performs similarly well $(-0.15, -0.16)$. Although the CAM controllers are in general less stable, in practice this means that the pole reaches the goal area just some steps later than in the mean case or drops out of the goal area for one downswing and is then immediately balanced again. The special controller with hand-picked autoencoder is almost as stable as the best SEN controller, only in 1 out of 20 tests the pole drops out of the goal area. Indeed the quality of the feature space is crucial for the performance. Considering the 1200 dimensional input for the overall system, even the lower stability at such a high performance level is a remarkable result. Another CAM controller outside these experiments with a hand-picked autoencoder was able to run in infinite loop for several minutes without any such dropout. An example video is available online at http://youtu.be/E0wZcYcoh-g

6 Conclusions

We have seen that the automatic feature extraction by a deep autoencoder can be successfully applied to the inherently instable pole balancing task. The control strategies can be easily obtained by RL methods and can compete with strategies trained on direct motor sensor data instead of the feature space. Both controller types show performances which are settled in the same order of magnitude. A shortcoming of the used autoencoder approach is that the moving parts have to be clearly visible, which is why a big green marker had to be attached to the pole. However the autoencoder turned out to be robust towards small lighting changes, which resulted in a small shift in the feature space, though not to position or background changes.

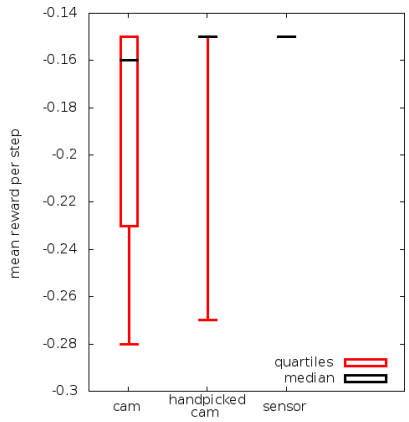

Fig. 5. Best final controllers of the camera and sensor experiments and of a camera controller with hand-picked autoencoder.

References

1. Riedmiller, M., Lange S., Voigtlaender, A.: Autonomous Reinforcement Learning on Raw Visual Input Data in a Real World Application. In: International Joint Conference on Neural Networks. (2012)
2. Ormoneit, D., Sen, Ś.: Kernel-Based Reinforcement Learning. Mach. Learn. 49, 161–178 (2002)
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
4. Riedmiller, M.: Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: 16th ECML, pp. 317–328. Springer (2005)
5. Riedmiller, M.: Neural Reinforcement Learning to Swing-Up and Balance a Real Pole. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 3191–3196. IEEE Press, New York (2005)
6. Tolat, V.V., Widrow, B.: An Adaptive 'Broom Balancer' with Visual Inputs. In: IEEE International Conference on Neural Networks, pp. 641–647. (1988)
7. Wenzel, L., Vazquez, N., Nair, D., Jamal, R.: Computer Vision Based Inverted Pendulum. In: Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference, pp. 1319–1323. (2000)
8. Wang, H., Chamroo, A., Vasseur, C., Koncar, V.: Hybrid Control for Vision Based Cart-Inverted Pendulum System. In: American Control Conference, pp. 3845–3850. (2008)
9. Hinton, G.E., Salakhutdinov, R.R.: Reducing the Dimensionality of Data with Neural Networks. Science 313, 504–507 (2006)
10. Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition. Neural Comput. 22, 3207–3220 (2010)
11. Katsikopoulos, K.V., Engelbrecht, S.E.: Markov Decision Processes with Delays and Asynchronous Cost Collection. IEEE Trans. Autom. Control 48, 568–574 (2003)