

Evolving Deep Unsupervised Convolutional Networks for Vision-Based Reinforcement Learning

Jan Koutník Jürgen Schmidhuber Faustino Gomez
IDSIA, USI-SUPSI
Galleria 2
Manno-Lugano, CH 6928
{hkou, juergen, tino}@idsia.ch

ABSTRACT

Dealing with high-dimensional input spaces, like visual input, is a challenging task for reinforcement learning (RL). Neuroevolution (NE), used for continuous RL problems, has to either reduce the problem dimensionality by (1) compressing the representation of the neural network controllers or (2) employing a pre-processor (*compressor*) that transforms the high-dimensional raw inputs into low-dimensional features. In this paper, we are able to evolve extremely small *recurrent neural network* (RNN) controllers for a task that previously required networks with over a million weights. The high-dimensional visual input, which the controller would normally receive, is first transformed into a compact feature vector through a deep, max-pooling convolutional neural network (MPCNN). Both the MPCNN preprocessor and the RNN controller are evolved successfully to control a car in the TORCS racing simulator using only visual input. This is the first use of deep learning in the context evolutionary RL.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning—*Connectionism and neural nets*

Keywords

deep learning; neuroevolution; vision-based TORCS; reinforcement learning; games

1. INTRODUCTION

Most approaches to scaling neuroevolution to tasks that require large networks, such as those processing video input, have focused on indirect encodings where relatively small neural network descriptions are transformed via a complex mapping into networks of arbitrary size [4, 7, 10, 12, 14, 21].

A different approach to dealing with high-dimensional input which has been studied in the context of single-agent

RL (i.e. TD [23], policy gradients [22], etc.), is to combine action learning with an unsupervised learning (UL) preprocessor or “compressor” which provides a lower-dimensional feature vector that the agent receives as input instead of the raw observation [5, 8, 11, 15, 17, 18, 19]. The UL compressor is trained on the high-dimensional observations generated by the learning agent’s actions, that the agent then uses as a state representation to learn a value function.

In [3], the first combination of UL and evolutionary reinforcement learning was introduced where a single UL module is trained on data generated by entire population as it interacts with environment (which would normally be discarded) to build a representation that allows evolution to search in a relatively low-dimensional feature space. This approach attacks the problem of high-dimensionality from the opposite direction compared to indirect encoding: instead of compressing large networks into short genomes, the inputs are compressed so that smaller networks can be used.

In this paper, this Unsupervised Learning – Evolutionary Reinforcement Learning (UL-ERL) approach is scaled up to the much more challenging reinforcement learning problem of driving a car in the TORCS simulator using vision from the driver’s perspective as input. The only work to successfully solve this task to date [13] did so using an indirect encoding that searches for networks in a low-dimensional frequency domain representation that effectively compresses over 1 million weights down to just a few hundred Fourier coefficients.

Here, a Max-Pooling Convolutional Neural Network (MPCNN; [2, 20]) is used as an unsupervised sensory preprocessor that reduces the input dimensionality of images to the point where an extremely small (only 33 weights) recurrent neural network controller can be evolved to drive the car successfully. MPCNNs are normally trained using a classifier output layer and backpropagation, meaning that a training set of class-labeled examples is required. Here, instead, the MPCNN without the classifier layer is evolved to maximize variance in the output representations of a small set of images collected at random from the environment. This quickly provides a small number of useful features, without the need for supervision (constructing a training set), that are used as inputs to the evolving recurrent neural network controllers. To our knowledge, this is the first use of a deep learning feedforward architecture for evolutionary reinforcement learning.

The next section describes the MPCNN architecture that is used to compress the high-dimensional vision inputs. Sec-

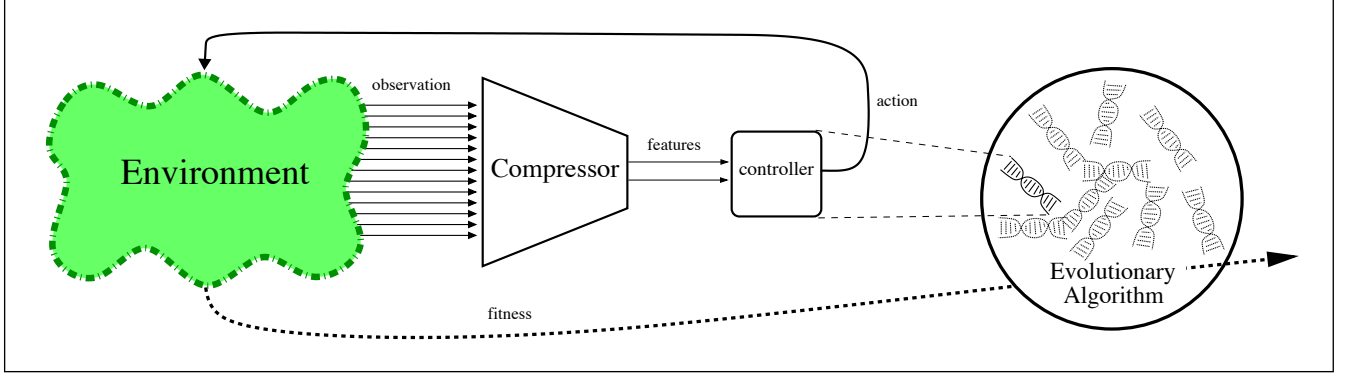


Figure 1: The Unsupervised Learning – Evolutionary Reinforcement Learning (UL-ERL) framework. Controllers (e.g. neural networks) are evolved as in conventional ERL except that the controller receives input from an unsupervised “compressor” that reduces the dimensionality of the raw observations into a compact feature vector. The compressor is trained using images generated by the interactions of candidate controllers with the environment.

tion 3 covers the overall UL-ERL framework. Section 4 presents the experiments in the TORCS race car driving domain, which are discussed in section 5.

2. MAX-POOLING CONVOLUTIONAL NEURAL NETWORKS

Convolution Neural Networks [6, 16] are deep hierarchical networks that have recently become the state of the art in pattern recognition and image processing due to the advent of fast implementations on graphics card multiprocessors (GPU) [1]. CNNs have two parts: (1) a deep feature detector consisting of alternating *convolutional* and *down-sampling* layers, and (2) a classifier that receives the output of the final layer of the feature detector.

Each convolutional layer ℓ , has a bank of $m^\ell \times n^\ell$ filters, F^ℓ , where m^ℓ is the number input maps (images), I^ℓ , to the layer, and n^ℓ is the number of output maps (inputs to the next layer). Each output map is computed by:

$$I_i^{\ell+1} = \sigma \left(\sum_{j=1}^{m^\ell} I_j^\ell * F_{ij}^\ell \right), \quad i = 1..n^\ell, \ell = 1, 3, 5, \dots,$$

where $*$ is the convolution operator, F_{ij}^ℓ is the i -th filter for the j -th map and σ is a non-linear squashing function (e.g. sigmoid). Note that ℓ is always odd because of the subsampling layers between each of the convolutional layers.

The downsampling layers reduce resolution of each map. Each map is partitioned into non-overlapping blocks and a value of each block is used in the output map. The max-pooling operation used here subsamples the map by simply taking the maximum value in the block as the output, making these networks Max-Pooling CNNs [2, 20].

The stacked alternating layers transform the input into progressively lower dimensional abstract representations that are then classified typically using a standard feed-forward neural network that has an output neuron for each class. The entire network is usually trained using a large training set of class-labeled images via backpropagation [16]. Figure 6 illustrates the particular MPCNN architecture used

in this paper. It should be clear from the figure that each convolution layer is just a matrix product where the matrix entries are filters and the convolution is used instead of multiplication.

3. METHOD: MPC-RNN

Figure 1 shows the general framework used in this paper to evolve small neural network controllers for tasks with high-dimensional inputs such as vision. The controller itself is evolved in the usual way (i.e. neuroevolution; [24]) except that during evaluation the candidate networks do not receive observations directly from the environment. Instead they receive a feature vector of much lower dimensionality provided by the unsupervised compressor. The compressor is trained on observations (images) generated by the actions taken by candidate controllers as they interact with the environment. Training can either be conducted on-line using images generated from the evolving controllers, or off-line, collected by some other means and trained in batch mode.

Deep neural processors such as MPCNNs are normally trained to perform image classification through supervised learning using enormous training sets. Of course, this requires *a priori* knowledge of what constitutes a class. In a general RL setting, we may not know how the space of images should be partitioned. For example, how many classes should there be for images perceived from the first-person perspective while driving the TORCS car? We could study the domain in detail to construct a training set that could then be used to train the MPCNN with backpropagation. However, we do not want the learning system to rely on task-specific domain knowledge, so, instead, the MPCNN is *evolved* without using a labeled training set. A set of k images is collected from the environment, and then MPCNNs are evolved to maximize the fitness:

$$f_{\text{kernel}} = \min(D) + \text{mean}(D), \quad (1)$$

where D is a list of all Euclidean distances,

$$d_{i,j} = \|\mathbf{f}_i - \mathbf{f}_j\|, \quad \forall i > j,$$

between k normalized feature vectors $\{\mathbf{f}_1 \dots \mathbf{f}_k\}$ generated

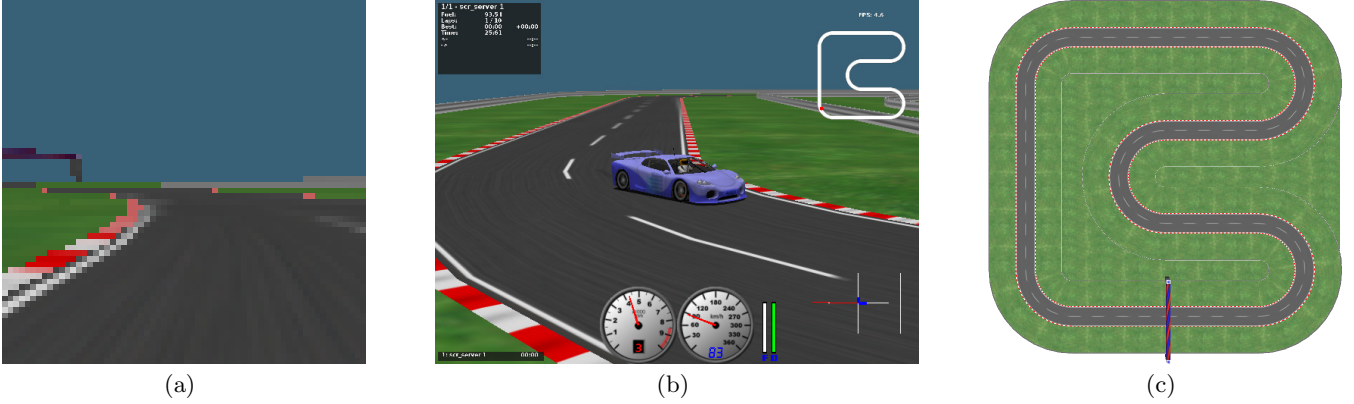


Figure 2: Visual TORCS environment. (a) The 1st-person perspective used as input to the RNN controllers (figure 3) to drive the car around the track. (b), a 3rd-person perspective of car. The controllers were evolved using a track (c) of length of 714.16 m and road width of 10 m, that consists of straight segments of length 50 and 100 m and curves with radius of 25 m. The car starts at the bottom (start line) and has to drive counter-clockwise. The track boundary has a width of 14 m.

from k images in the training set by the MPCNN encoded in the genome.

This fitness function forces the evolving MPCNNs to output feature vectors that are spread out in feature space, so that when the final, evolved MPCNN processes images for the evolving controllers, it will provide enough discriminative power to allow them to take correct actions.

4. VISUAL TORCS EXPERIMENTS

The goal of the task is to evolve a recurrent neural network controller that can drive the car around a race track using the compressed representation provided by the MPCNN which is evolved separately.

The visual TORCS environment is based on TORCS version 1.3.1. The simulator had to be modified to provide images as input to the controllers. At each time-step during a network evaluation, an image rendered in OpenGL is captured in the car code (C++), and passed via UDP to the client (Java), that contains the RNN controller. The client is wrapped into a Java class that provides methods for setting up the RNN weights, executing the evaluation, and returning the fitness score. These methods are called from Mathematica which is used to implement the evolutionary search.

The Java wrapper allows multiple controllers to be evaluated in parallel in different instances of the simulator via different UDP ports. This feature is critical for the experiments presented below since, unlike the non-vision-based TORCS, the costly image rendering, required for vision, cannot be disabled. The main drawback of the current implementation is that the images are captured from the screen buffer and, therefore, have to actually be rendered to the screen.

Other tweaks to the original TORCS include changing the control frequency from 50 Hz to 5 Hz, and removing the “3-2-1-GO” waiting sequence from the beginning of each race. The image passed in the UDP is encoded as a message chunk

with *image* prefix, followed by *unsigned byte* values of the image pixels.

4.1 Setup

In each fitness evaluation, the candidate controller is tested in two trials, one on the track shown in figure 2(c), and one on its mirror image. A trial consists of placing the car at the starting line and driving it for 25 s of simulated time, resulting in a maximum of 125 time-steps at the 5 Hz control frequency. At each control step (see figure 3), a raw 64×64 pixel image, taken from the driver’s perspective is split into three color planes (hue, saturation and brightness). The saturation plane is passed through the MPCNN compressor which generates a 3-dimensional feature vector that is fed into a simple recurrent neural network (SRN) with 3 hidden neurons, and 3 output neurons. The first two outputs, o_1, o_2 , are averaged, $(o_1 + o_2)/2$, to provide the steering signal (-1 = full left lock, 1 = full right lock), and the third neuron, o_3 , controls the brake and throttle (-1 = full brake, 1 = full throttle). All neurons use sigmoidal activation functions.

The 33 network weights were directly encoded in real-valued genomes and evolved using CoSyNE [9] with population size of 100, a mutation rate of 0.8, and each of the two trials is scored using the following function:

$$f = d - \frac{3m}{1000} + \frac{v_{max}}{5} - 100c, \quad (2)$$

where d is the distance along the track axis measured from the starting line, v_{max} is maximum speed, m is the cumulative damage, and c is the sum of squares of the control signal differences, divided by the number of control variables, 3, and the number simulation control steps, T :

$$c = \frac{1}{3T} \sum_i \sum_t^T [o_i(t) - o_i(t-1)]^2. \quad (3)$$

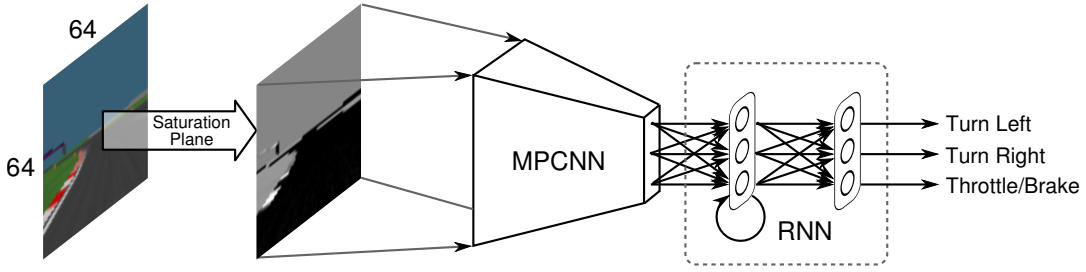


Figure 3: Visual TORCS network controller pipeline. At each time-step a raw 64×64 pixel image, taken from the driver’s perspective, is split into three planes (hue, saturation and brightness). The saturation plane is fed into the max-pooling convolutional network (MPCNN), that generates features for the recurrent neural network (RNN) controller, that drives the car by controlling the steering, brakes, and accelerator. See Figure 6 for the details of the MPCNN network.

Layer	Type	m	n	#maps	map size
1	C	1	10	10	63×63
2	MP	-	-	10	21×21
3	C	10	10	10	20×20
4	MP	-	-	10	10×10
5	C	10	10	10	9×9
6	MP	-	-	10	3×3
7	C	10	10	3	2×2
8	MP	-	-	3	1×1

Table 1: MPCNN topology. The table summarizes the MPCNN architecture used; type of a layer, where C is for convolutional, MP for max-pooling, dimensions of the filter bank m and n , number of output maps and their resolution.

The maximum speed component in equation (2) forces the controllers to accelerate and brake efficiently, while the damage component favors controllers that drive safely, and c encourages smoother driving. Fitness scores roughly correspond to the distance traveled along the race track axis.

Each individual is evaluated both on the track and its mirror image to prevent the RNN from blindly memorizing the track without using the visual input. The original track starts with a left turn, while the mirrored track starts with a right turn, forcing the network to use the visual input to distinguish between tracks. The final fitness score is the minimum of the two track scores (equation 2).

Before the RNN controllers can be evolved, the MPCNN that will provide them with inputs while driving, is first itself evolved off-line at the beginning of each run. A set of 20 images was collected by taking snapshots from the driver’s perspective (i.e. figure 2a) at regular intervals while manually driving the car once around the track. These images were then mirrored horizontally to form a training set of 40 images that were used to evolve MPCNNs that maximized the fitness function in equation (1).

The MPCNNs had 8 layers alternating between convolution and max-pooling operations, as shown in figure 6 and Table 1. The first step in processing a 64×64 pixel image from TORCS is to convolve it with each of the 10 filters in layer 1, to produce 10 63×63 feature maps, each of which is reduced down to 21×21 by the first max-pooling

layer (2). These 10 features are convolved again by layer 3, max-pooled in layer 4, and so on, until the image is reduced down to just 3 1-dimensional features which are fed to the controller. This architecture has a total of 993 weights that were directly encoded in a real-valued genome, and also evolved using CoSyNE [9] with, a population size of 100, and mutation rate of 0.8.

4.2 Results

The average fitness of 3 runs after 300 CoSyNE generations reached 531. Evolving one controller for 300 generations (which involves 45,000 fitness evaluations) takes almost 40 hours on an 8-core machine¹ (running 8 evaluations in parallel).

Table 2 compares the distance travelled and maximum speed of the best MPC-RNN controller with the published results in [13] (a large RNN controller evolved in frequency domain and hand-coded controllers that come with the TORCS package).

The performance of MPC-RNN is not as good as the other controllers, but the car can complete a lap and continue driving without crashing. The hand-coded controllers drive better since they have access to the car state variables such as velocities, accelerations along axes, and some high-level features such as the distance from the car to the edge of the track. The Visual RNN also drives better but requires networks with over 1 million weights and image preprocessing—edge detection, dilation. The Robert’s edge detector it uses convolves the input image with two 2×2 kernels ($\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ and $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$), whereas MPCNN is capable of performing edge detection as well, using the evolved convolutional layers.

In a typical run, 20 generations is required for evolution to overcome a local optima (fitness around 150) where controllers just drive straight and crash to the barrier at maximum speed. Later, the controllers start to distinguish between the left and right turns and the fitness quickly jumps above 300. After reaching fitness of 500, the controllers start to drive more smoothly, accelerate on straight sections and optimize the driving path to complete a lap and keep from crashing.

Figure 4 shows the evolution of the MPCNN feature vectors for each of the 40 images in the training set, in one of the three runs. As the features evolve they very quickly move

¹AMD FX 8120 8-core, 16 GB RAM, nVidia GTX-570

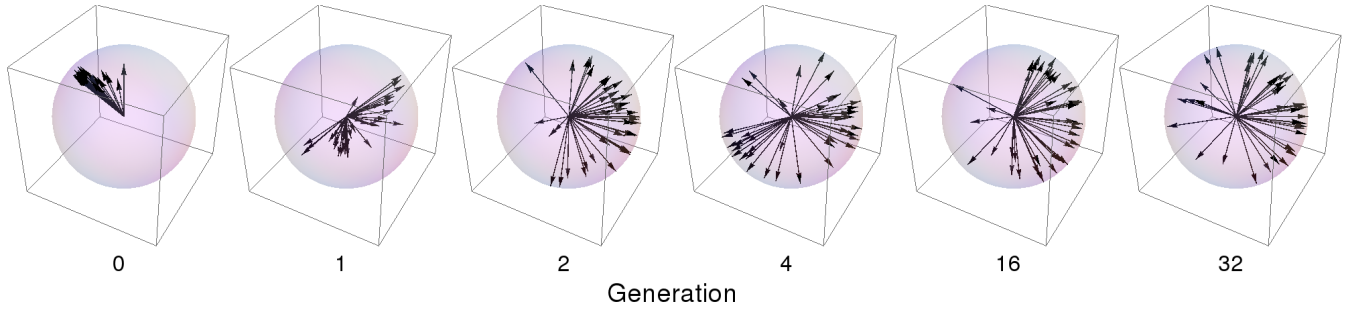


Figure 4: Evolving MPCNN features. Each plot shows the feature vectors for each of the 40 training images on the unit sphere. Initially (generation 0), the features are clustered together. After just a few generations spread out so that the MPCNN discriminates more clearly between the images.

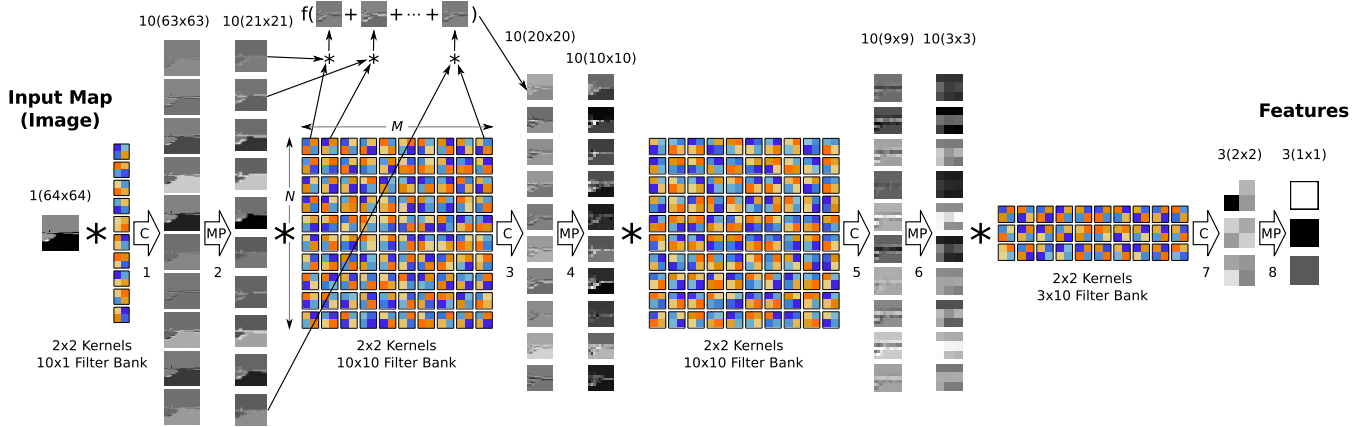


Figure 6: Max-Pooling Convolutional Neural Network (MPCNN) with 8 layers alternating between convolution (C) and downsampling (MP; using max-pooling). The first layer convolves the input 64×64 pixel image with a bank of 10×10 filters producing 10 maps of size 63×63 , that are down-sampled to 21×21 by MP layer 2. Layer 3 convolves each of these 10 maps with a filter, sums the results and passes them through the nonlinear function f , producing 10 maps of 20×20 pixels each, and so on until the input image is transformed to just 3 features that are passed to the RNN controller, see Figure 3.

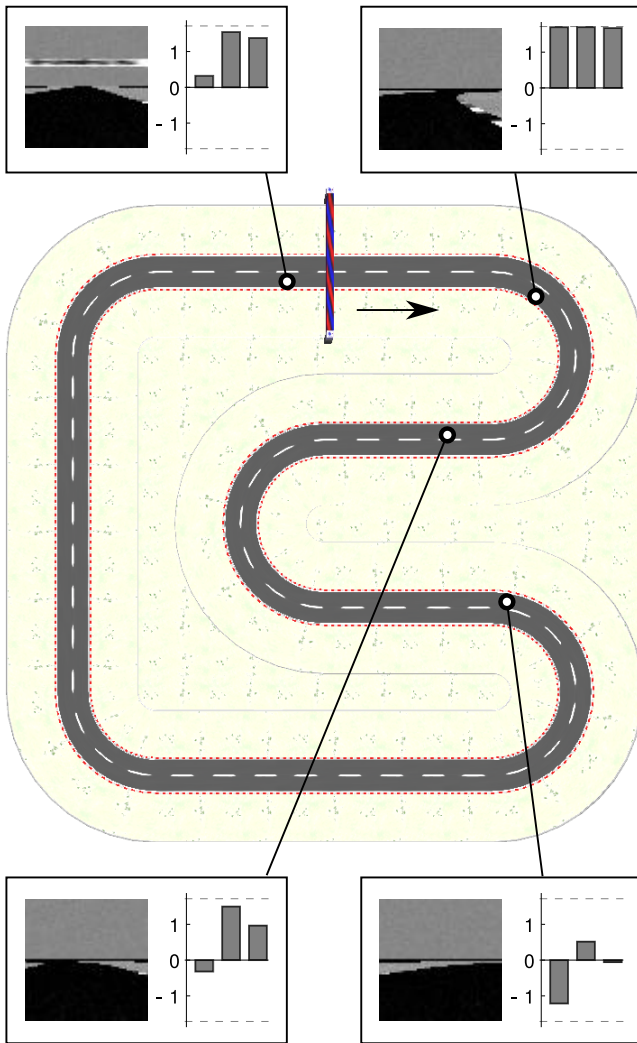


Figure 5: Evolved visual features. Each inset shows the image at a particular point on the track and the 3-dimensional feature vector produced by the evolved MPCNN.

controller	d [m]	v_{max} [km/h]
olethros	570	147
bt	613	141
berniw	624	149
tita	657	150
inferno	682	150
visual RNN[13]	625	144
MPC-RNN	547	97

Table 2: Maximum distance, d , in meters and maximum speed, v_{max} , in kilometers per hour achieved by hand-coded controllers that come with TORCS which enjoy access to the state variables (the five upper table entries), a million-weight RNN controller that drives using pre-processed 64×64 pixel images as input, evolved indirectly in the Fourier domain, and the MPC-RNN agent with just 33 weights in its RNN controller.

away from each other in feature space. While simply pushing the feature vectors apart is no guarantee of achieving maximally informative compressed representations, this simple, unsupervised training procedure provides enough discriminative power in practice to get the car safely across the finish line.

4.3 Generalization

The generalization ability of the controllers was tested on three different tracks constructed from components (e.g. curve with the same radius) from the track used to evolve them, see table 3. Track 1 is a very long oval that tests whether the network is able to accelerate beyond the maximum speed observed during evolution. The two other tracks are shorter and test whether the controllers can cope with different combinations of corners.

The maximum and average distance, and top speed (over 20 tests) achieved by the controllers in 500 seconds of driving is shown in table 3. For track 1, the best controller achieved a maximum speed above 150 km/h and was able to brake soon enough to slow down in front of the hairpin, in order to complete a lap within the allotted time. For tracks 2 and 3, the best controller was only able to complete about half a lap on average.

5. DISCUSSION

The results show that it is possible to circumvent the indirect encoding approach by shifting the computational burden to a deep image processor that removes the redundancy in the environment so that almost trivially small networks can be evolved efficiently to solve a complex control task. The driving performance is not yet as good as the only other method to tackle this problem [13], and the generalization ability of the driver networks needs to be more thoroughly tested (e.g. using more realistic, irregular tracks).

So far the feature generating MPCNN has been evolved off-line. A better approach might be to interleave the compressor evolution with controller evolution so that the compressor is trained continually using images generated by the controller as they become more and more proficient in the task as in [3].

As general approach MPC-RNN harnesses the power of deep learning for RL without having to train a deep network in a supervised manner. Future work will extend the approach to higher-dimensional action spaces like the 41-DOF iCub humanoid, to perform manipulation tasks using vision.

Acknowledgments

This research was supported by Swiss National Science Foundation grant #138219: “Theory and Practice of Reinforcement Learning 2”, and EU FP7 project: “NAoScale Engineering for Novel Computation using Evolution” (NASCENCE), grant agreement #317662.

References

- [1] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.


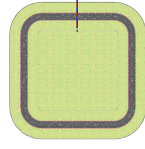
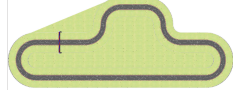
									
	Track 1 (4514 m)			Track 2 (557 m)			Track 3 (980 m)		
	<i>d</i> [m]	laps	v_{max} [km/h]	<i>d</i> [m]	laps	v_{max} [km/h]	<i>d</i> [m]	laps	v_{max} [km/h]
Maximum	5214.2	1.16	154.2	391.2	0.70	106.0	906.0	0.92	122.0
Average	2711.4	0.60	150.7	274.4	0.49	98.0	547.7	0.56	102.2
Std. dev.	1831.5	0.41	2.13	112.6	0.20	9.5	320.8	0.33	16.1

Table 3: Generalization results. The best overall driver from the three runs was tested on three other tracks, different from the one used during evolution.

- [2] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1237–1242, 2011.
- [3] G. Cuccu, M. Luciw, J. Schmidhuber, and F. Gomez. Intrinsically motivated evolutionary search for vision-based reinforcement learning. In *Proceedings of the IEEE Conference on Development and Learning, and Epigenetic Robotics*, 2011.
- [4] D. B. D’Ambrosio and K. O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the 9th Conference on Genetic and Evolutionary Computation*, (GECCO), pages 974–981, New York, NY, USA, 2007. ACM.
- [5] F. Fernández and D. Borrajo. Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245, 2008.
- [6] K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.
- [7] J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, (GECCO), pages 997–1004, New York, NY, USA, 2007. ACM.
- [8] L. Gisslén, M. Luciw, V. Graziano, and J. Schmidhuber. Sequential Constant Size Compressors and Reinforcement Learning. In *Proceedings of the Fourth Conference on Artificial General Intelligence*, 2011.
- [9] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.
- [10] F. Gruau. Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Supérieure de Lyon, Institut IMAG, Lyon, France, 1992.
- [11] S. R. Jodogne and J. H. Piater. Closed-loop learning of visual control policies. *Journal of Artificial Intelligence Research*, 28:349–391, 2007.
- [12] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.
- [13] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Amsterdam, 2013.
- [14] J. Koutník, F. Gomez, and J. Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-10)*, 2010.
- [15] S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN 2010)*, Barcelona, Spain, 2010.
- [16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.
- [17] R. Legenstein, N. Wilbert, and L. Wiskott. Reinforcement Learning on Slow Features of High-Dimensional Input Streams. *PLoS Computational Biology*, 6(8), 2010.
- [18] D. Pierce and B. Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–229, 1997.
- [19] M. Riedmiller, S. Lange, and A. Voigtlaender. Autonomous reinforcement learning on raw visual input data in a real world application. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Brisbane, Australia, 2012.
- [20] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the International Conference on Artificial Neural Networks, ICANN*, 2010.

- [21] J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.
- [22] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 1057–1063, 1999.
- [23] G. Tesauro. Practical issues in temporal difference learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 4 (NIPS)*, pages 259–266. Morgan Kaufmann, 1992.
- [24] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.