Evolving Neural Networks in Compressed Weight Space

Jan Koutník
IDSIA
University of Lugano
Manno-Lugano, CH
hkou@idsia.ch

Faustino Gomez
IDSIA
University of Lugano
Manno-Lugano, CH
tino@idsia.ch

Jürgen Schmidhuber
IDSIA
University of Lugano
Manno-Lugano, CH
juergen@idsia.ch

ABSTRACT

We propose a new indirect encoding scheme for neural networks in which the weight matrices are represented in the frequency domain by sets of Fourier coefficients. This scheme exploits spatial regularities in the matrix to reduce the dimensionality of the representation by ignoring high-frequency coefficients, as is done in lossy image compression. We compare the efficiency of searching in this "compressed" network space to searching in the space of directly encoded networks, using the CoSyNE neuroevolution algorithm on three benchmark problems: pole-balancing, ball throwing and octopus-arm control. The results show that this encoding can dramatically reduce the search space dimensionality such that solutions can be found in significantly fewer evaluations.

Categories and Subject Descriptors

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

General Terms

Algorithms

1. INTRODUCTION

Training neural networks for reinforcement learning tasks is problematic because the non-stationarity of the error gradient can lead to poor convergence, especially if the network is recurrent. An alternative approach is to search the space of neural networks directly via evolutionary computation. In this *neuroevolutionary* framework, networks are encoded either directly or indirectly in strings of values or *genes*, called *chromosomes*, and then evolved in the standard way (genetic algorithm, evolutionary strategies, etc.). Direct encoding schemes employ a one-to-one mapping from genes to network parameters (e.g. connectivity pattern, synaptic weights), so that the size of the evolved networks is proportional to the length of the chromosomes.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

In indirect schemes, the mapping from chromosome to network can in principle be any computable function, allowing chromosomes of fixed size to represent networks of arbitrary complexity. The underlying motivation for this approach is to scale neuroevolution to problems requiring large networks such as vision [3], since search can be conducted in relatively low-dimensional gene space. Theoretically, the optimal or most *compressed* encoding is the one in which each possible network is represented by the shortest program that generates it, i.e. the one with the lowest Kolmogorov complexity [8].

Unfortunately, this encoding is not computable, and existing, practical encodings [1–3, 6] often lack *continuity in the genotype-phenotype mapping*, such that small changes to a genotype can cause large changes in its phenotype. For example, using cellular automata [1] or graph-based encodings [6] to generate connection patterns can produce large networks but violates this continuity condition. HyperNEAT [3], which evolves weight-generating networks using Neuro-Evolution of Augmenting Topologies (NEAT; [9]) provides continuity while changing weights, but adding a node or a connection to the weight-generating network causes a discontinuity in the phenotype space. These discontinuities occur frequently when e.g. replacing NEAT in HyperNEAT with genetic programming-constructed expressions [2]. Furthermore, these representations do not provide an *importance ordering* on the constituent genes. For example, in the case of graph encodings, one cannot not gradually cut of less important parts of the graph (GP expression, NEAT network) that constructs the phenotype.

Here we define an indirect encoding scheme in which genes represent Fourier series coefficients that are mapped to network weight matrices using an inverse Fourier-type transform. This representation not only yields continuity but also allows the complexity of the weight matrix to be controlled by the number of coefficients. Because frequency domain representations decorrelate the spatial signal (weight matrix), the search space dimensionality can be reduced in a principled manner by discarding high-frequency coefficients, just as is done in lossy image coding. Encoding in the frequency domain also means that the size of the genome is independent of the size of the network it generates. Therefore, networks can be scaled to high-dimensional problems, such as vision, since relatively few coefficients can encode complex weight matrices of arbitrary size.

This indirect encoding was used in previous work [7] to discover minimal solutions to well-known RL benchmarks using a version of practical universal search where the max-

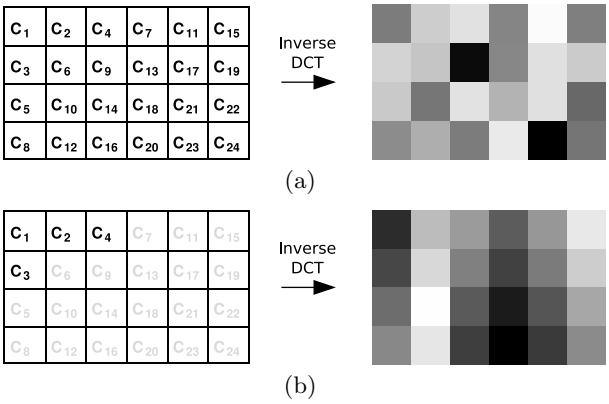

Figure 1: DCT network representation. The coefficients are selected according to their order along the second diagonals, going from upper-left corner to the bottom right corner. Each diagonal is filled from the edges to the center starting on the side that corresponds to the longer dimension. (a) Shows an example of the kind of weight matrix (right) that is obtained by transforming the full set of coefficients (left). The gray-scale levels denote the weight values (black = low, white = high). (b) Shows the weight matrix when only the first four coefficients from (a) are used. The weights in (b) are more spatially correlated than those in (a).

imum number of searchable coefficients was limited. In this paper, it is used with evolution to search larger numbers of coefficients, and potentially extend its applicability to networks of greater complexity.

The next section describes in detail both the network representation and the evolutionary algorithm, CoSyNE, that are used in our method for evolving networks in Fourier space. We then present experimental results in three test domains, showing how evolution in compressed network space can accelerate the discovery of good solutions. The last section discusses our findings and directions for future work.

2. SEARCHING IN COMPRESSED NETWORK SPACE

The motivation for representing weight matrices as frequency coefficients is that by spatially decorrelating the weights in the frequency domain it might be possible to discard the least significant frequencies, and thereby reduce the number of search dimensions.

The next two sections describe how the networks are represented in the frequency domain using the Discrete Cosine Transform (DCT), and the neuroevolution method, Cooperative Synapse NeuroEvolution (CoSyNE[1]), used to search the space of DCT coefficients.

2.1 DCT Network Representation

All networks are fully connected recurrent neural networks (FRNNs) with i inputs and single layer of n neurons where some of the neurons are treated as output neurons. This architecture is general enough to represent e.g. feed-forward (e.g. as used in section 3.1) and Jordan/Elman networks, since they are just sub-graphs of the FRNN.

An FRNN consists of three weight matrices: an $n \times i$ input matrix, $\mathbf{I}$, an $n \times n$ recurrent matrix, $\mathbf{R}$, and a bias vector $\mathbf{t}$ of length n. These three matrices are combined into one $n \times (n + i + 1)$ matrix, and encoded indirectly using $c \leq N$ DCT coefficients, where N is the total number of weights in the network. Figure 1 illustrates the relationship between the coefficients and weights for a hypothetical 4×6 weight matrix (e.g. a network with four neurons each with six weights). The left side of the figure shows two weight matrix encodings that use different numbers of coefficients $\{C_1, C_2, \ldots, C_c\}$. Generally speaking, coefficient C_i is considered to be more significant (associated with a lower frequency) than C_j, if $i < j$. The right side of the figure shows the weight matrices that are generated by applying the inverse DCT transform to the coefficients. In the first case (figure 1a), all 24 coefficients are used, so that any possible 4×6 weight matrix can be represented. The particular weight matrix shown was generated from random coefficients in $[-20, 20]$. In the second case (figure 1b), each C_i has the same value as in figure 1a, but the full set has been truncated to only the four most significant coefficients.

The more coefficients, the more high frequency information that is potentially expressed in the weight matrix, so that the weight values become less spatially correlated—large changes can occur from one weight to its neighbors. As c approaches one, the matrix becomes more regular, with progressively more correlated changes in value from weight to weight, until all the weights become equal at $c = 1$.

Figure 2 shows graphically how DCT genotypes are mapped to FRNN phenotypes. Each row shows three network architectures instantiated with weights generated by applying the inverse DCT transform to the chromosome in the left column. In the first row (a), the chromosome consists of just one gene, the DC coefficient, so that all of the weights in each of the three networks are equal because one coefficient corresponds to a constant function. In the second row (b), one more coefficient is added, and the weights start to vary within each network, but they are still highly correlated because the weight matrix is generated by a 2D sinusoid. In the bottom row (c), the distribution of weights is much less regular because the five coefficients offer more degrees of freedom to the spatial function describing the weight matrix.

2.2 Cooperative Synapse NeuroEvolution (CoSyNE)

Cooperative Synapse Neuroevolution (CoSyNE; [5]) is cooperative coevolutionary method, but unlike other cooperative neuroevolutionary methods (e.g. ESP, SANE) which evolve at the level of neurons, it searches at the level of individual network weights. While CoSyNE has been shown to be efficient in searching for weight values directly [4,5], here it is used *indirectly* to search for sets of DCT coefficients that produce good weight matrices.

Algorithm 1 describes the CoSyNE procedure in pseudocode. First (line 1), a population $\mathcal{P}$ consisting of c subpopulations $P_i, i = 1..c$, is created, where c is the number of DCT coefficients to be evolved, and Ψ is a user-specified network architecture. (Note that while figure 2 shows each chromosome being mapped to multiple architectures, this is merely to convey the independence between the size of the

[1]We apologize for the potential confusion between Cosine and CoSyNE.

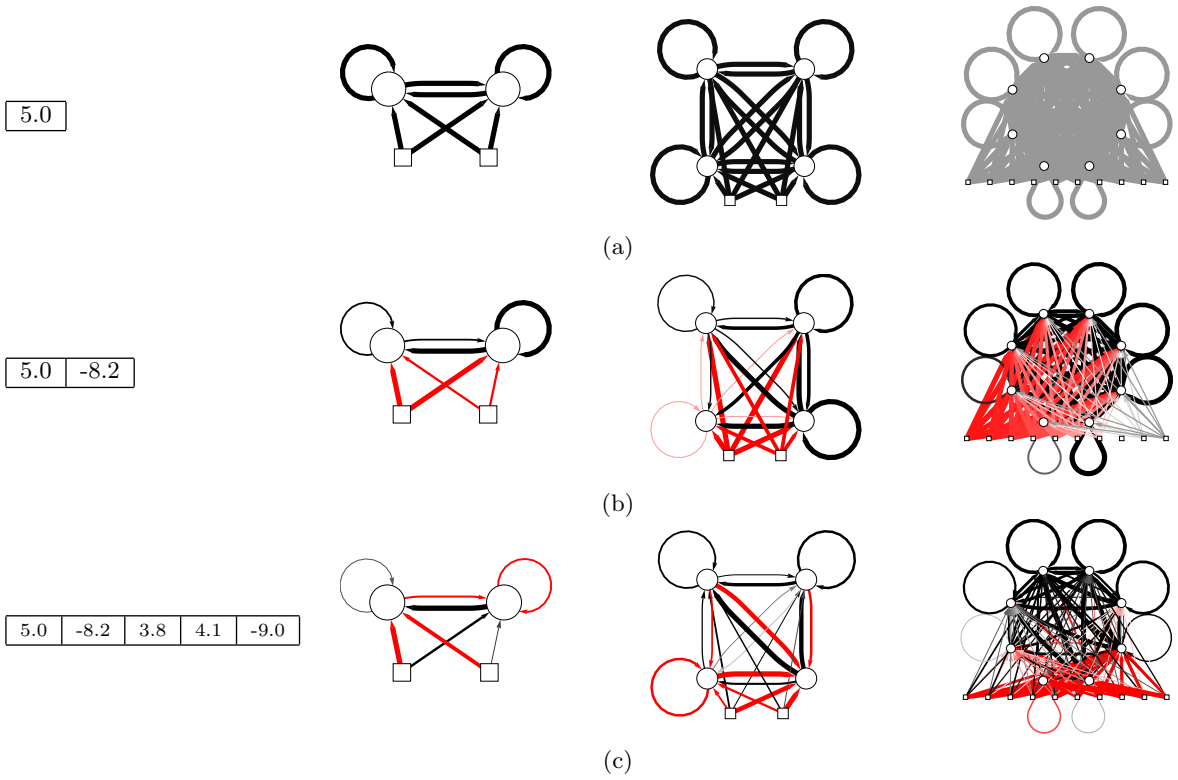

Figure 2: Mapping from DCT genotype to FRNN phenotypes. For each chromosome in the left column, three different networks with different architectures are shown, instantiated with weights generated by applying the inverse DCT. The potential complexity of the weight matrices increases with the number of DCT coefficient genes. The small squares in the networks denote input units, the circles are neurons. The thickness of a connection (arrow) corresponds to the relative magnitude of its weight, and the weight is positive if the arrow is dark (black) and negative if light (red).

Algorithm 1: CoSyNE(c,m,Ψ)

```
1  Initialize P = {P_1, ..., P_c}
2  repeat
3      for j=1 to m do
4          x_j ← (x_1j, ..., x_cj)
5          network← INVERSEDCT(x_j, Ψ)
6          EVALUATE(network)
7      end
8      O← REPRODUCE(P)
9      for k=1 to l do
10         x_{i,m-k} ← o_ik
11     end
12     for i=1 to c do
13         PERMUTE(P_i)
14     end
15 until solution is found
```

genotype and that of the phenotype.) Each subpopulation is initialized to contain m real numbers, $x_{ij} = \mathcal{P}_{ij} \in P_i, j = 1..m$, chosen from a uniform probability distribution in the interval $[-\alpha, \alpha]$. The population is thereby represented by an $c \times m$ matrix.

CoSyNE then loops through a sequence of generations until a sufficiently good network is found (lines 2-15). Each generation starts by constructing a complete chromosome $\mathbf{x}_j = (x_{1j}, x_{2j}, \ldots, x_{cj})$ from each row in $\mathcal{P}$. Each of the m resulting chromosomes is transformed into a network by first plugging its alleles into to the corresponding positions in a coefficient matrix (figure 3), and then applying the inverse Discrete Cosine Transform (line 5) to this matrix to produce a weight matrix of size determined by the network architecture, Ψ.

After all of the networks have been evaluated (line 6) and assigned a fitness, the top quarter with the highest fitness (i.e. the parents) are copied (line 8) into a pool of offspring $\mathcal{O}$ consisting of l new chromosomes $\mathbf{o}_k$, where $o_{ik} = \mathcal{O}_{ik} \in O_i, k = 1..l$, and the offspring are then mutated. The weights in each of the offspring chromosomes are then added to $\mathcal{P}$ by replacing the least fit weights in their corresponding subpopulation (lines 9-11).

At this point the algorithm functions as a conventional neuroevolution system that evolves complete network chromosomes. In order to *coevolve* the coefficients, the subpopulations are permuted (lines 12-15) so that each coefficient forms part of a potentially different network in the next generation. Permutation detaches the subpopulations from each other by assigning new "collaborators" for each weight.

Permuting the subpopulations increases diversity by allowing CoSyNE to sample networks that would not be generated through recombination alone. This means that which weights are retained in the population from one generation to the next is not determined only by which networks scored

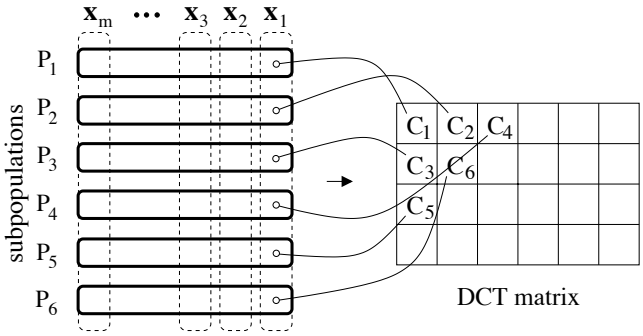

Figure 3: The CoSyNE neuroevolution method. On the left, the figure shows an example population consisting of six subpopulations, $P_1..P_6$, each containing m DCT coefficient values. To create a network, first the coefficients at a given index in each subpopulation are collected into a chromosome x, then the they are mapped to their corresponding position in a DCT coefficient matrix which is then transformed into a network weight matrix via the inverse DCT (see figure 2).

well in the previous generation, but rather by a broader sampling of the possible m^c networks that can be formed by selecting a weight from each subpopulation.

The basic CoSyNE framework does not specify how the weights are grouped in the chromosomes (i.e. which entry in the chromosome corresponds to which synapse) or which genetic operators are used.

3. EXPERIMENTAL RESULTS

The evolutionary search in compressed weight space was tested on three tasks: pole balancing, ball throwing, and Octopus-arm control. In all experiments, the scaling factor, α, was set to 20. The CoSyNE algorithm used mutation only (no crossover), where the probability of a gene (coefficient) being mutated by adding Cauchy distributed noise (mean=0, $\gamma = 0.3$) was 0.8. The population sizes were 20 individuals for easier problems (single-pole balancing and forward ball throwing) and 40 individuals for the harder problems (double-pole balancing, backward-swing ball throwing and the octopus arm).

3.1 Pole Balancing

Pole balancing (figure 4a) is a standard benchmark for learning systems. The basic version consists of a single pole hinged to a cart, to which a force must applied in order to balance the pole while keeping the cart within the boundaries of a finite stretch of track. By adding a second pole next to the first, the task becomes much more non-linear and challenging. A further extension is to limit the controller to only have access to the position of the cart, and the angle of the pole(s), and not the velocity information, making the problem non-Markovian (see [10] for setup and equations of motion). The task is considered solved if the pole(s) can be balanced for 100,000 time steps.

Table 1 summarizes the results for the four most commonly used versions of the task. As in [5], single-layer feedforward networks were used for the Markov versions (no memory is needed for the task), and FRNNs for the non-

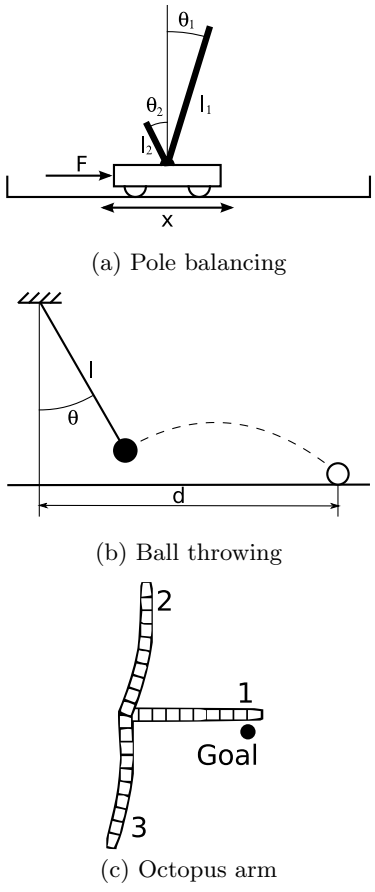

Figure 4: Evaluation tasks. (a) Pole balancing: the goal is to apply a force F to the cart such that the pole(s) do not fall down, and the cart stays within the track boundaries. (b) Ball throwing: a ball attached to the end of an arm must be thrown as far as possible by applying a torque to the joint and then releasing the ball. (c) Octopus arm: a flexible arm consisting of n compartments, each with 3 muscles, must be controlled to touch a goal location with the arm tip from three different initial positions.

Markov versions. We only report "compressed" results for numbers of coefficients, c, that reliably solved a given task.

In turns out that for the simplest, classic Markov one-pole task, searching the space of one coefficient finds a successful network in just two evaluations, on average. This is because all that is required is that the weights be equal and positive, which is exactly what is generated by a single positive (DC) coefficient; and it takes two evaluations on average to find a positive coefficient. As the number of coefficients is increased, more evaluations are required to find more complex solutions where the weights are different.

For the Markovian two-pole task, evolving five coefficients (one less that the number of weights required) cuts the number of evaluations in half, and even evolving with as many coefficients as there are weights produces a slight improvement over the baseline, direct evolution. For the non-Markov tasks, evolving in Fourier space provides no advantage (the

Table 1: Pole balancing results. The table compares the number of evaluations required to evolve weights directly versus evolving an equal or fewer number of DCT coefficients, to solve each task. All networks had one neuron and w weights (depending on the number of inputs). Values are averages from 100 runs.

	Direct		Compressed	
Task	w	eval.	c	eval.
1 pole Markov	4	39	1	2
			2	16
			3	54
			4	88
2 pole Markov	6	464	5	258
			6	358
1 pole non-Markov	4	195	4	151
2 pole non-Markov	5	1422	5	3421

Table 2: Ball throwing results. The table compares the number of evaluations require to evolve weights directly versus evolving a fewer number of DCT coefficients for the forward-swing and backward-swing strategies. The columns contain number of coefficients, number of evaluations of compressed and directly encoded weight matrices, reached and optimal distances.

	Direct		Compressed	
Strategy	w	eval.	c	eval.
forward	10	267	5	126
backward-fwd	10	10224	9	8220

difference in the one-pole results are not statistically significant).

3.2 Ball Throwing

In the ball throwing task (figure 4b), the goal is to swing a one-joint artificial arm by applying a torque to the joint, and then releasing the ball at precisely the right time such that it is thrown as far as possible. The arm-ball dynamical system is described by:

$$(\dot{\theta}, \dot{\omega}) = \left(\omega, \underbrace{-\frac{c \cdot \omega}{}}_{\text{friction}} \underbrace{- \frac{g \cdot \sin(\theta)}{l}}_{\text{gravity}} + \underbrace{\frac{T}{m \cdot l^2}}_{\text{torque}}\right)$$

where θ is the arm angle, ω its angular speed, $c = 2.5\text{s}^{-1}$ the friction constant, $l = 2\text{m}$ the arm length, $g = 9.81\text{ms}^{-2}$, $m = 0.1\text{kg}$ the mass of the ball, and T the torque applied ($T_{\max} = [-5\text{Nm}, 5\text{Nm}]$). In the initial state, the arm hangs straight down ($\theta = 0$) with the ball attached to the end. The controller sees (θ, ω) at each time-step and outputs a torque. When the arm reaches the limit $\theta = \pm\pi/2$, all energy is absorbed ($\omega = 0$). Euler integration was used with a time-step of 0.01s.

In the experiments, we compare the networks found by CoSyNE with two optimal control strategies. The first applies the highest torque to swing the arm forward, and releases the ball at the optimal angle (which is slightly below 45 degrees, because the ball is always released above the ground). The second, more sophisticated, strategy first applies a negative torque to swing the arm backwards up to the maximum angle, and then applies a positive torque to swing the arm forward, and release the ball at the optimal angle of 43.03 degrees. The optimal distances are 5.391m for the forward swing strategy, and 10.202m for the backward-forward swing strategy. The task is considered solved once a network controller throws the ball a distance of at least 5.3m for the forward strategy, and 9.5m for the backward-swing controller).

Table 3: Octopus-arm results. The table compares the best fitness of networks encoded with different compression ratios versus directly encoded weights, average of 20 runs.

Direct	**Compressed**						
728	320	160	80	**40**	20	10	5
0.17	0.23	0.17	0.23	**0.38**	0.27	0.15	0.04

The results are summarized in Table 2. CoSyNE was able to find solutions to the forward swing task using only 5 coefficients in about the half the number of evaluations required by search the 10-dimensional weight space directly. The average distance was higher than optimal (5.453m) for the forward-swing controller because the networks already utilize a slight backward swing. For the backward-forward swing task, evolving 9 coefficients instead of 10 weights produced a 20% performance improvement.

3.3 Octopus-Arm Task

The octopus arm (figure 4c) consists of n compartments floating in a 2D water environment [11][2]. Each compartment has a constant volume and contains three controllable muscles (dorsal, transverse and ventral). The state of a compartment is described by the x, y-coordinates of two of its corners plus their corresponding x and y velocities. Together with the arm base rotation, the arm has $8n + 2$ state variables and $3n + 2$ control variables. The goal of the task to reach a goal position with the tip of the arm, starting from three different initial positions, by contracting the appropriate muscles at each 1s step of simulated time. While initial positions 2 and 3 look symmetrical, they are actually quite different due to gravity.

Direct and indirect encodings were compared for an octopus arm with $n = 10$ compartments, for a total of $8n+2 = 82$ state variables. The $3n + 2 = 32$ control variables were aggregated into 8 "meta" actions: contraction of all dorsal, all transverse, and all ventral muscles in first (actions 1,2,3) or second half of the arm (actions 4,5,6) plus rotation of the base in either direction (actions 7,8). All of the evolved FRNNs had 8 neurons, one corresponding to each of the actions, for a total or 8 (neurons) $\times$ 82 (inputs+bias) + 64 (recurrent weights) = 728 synaptic weights. For indirect encoding, the weight matrices were compressed down to 5, 10, 20, 80, 160 and 320 DCT coefficients. Each set of experiments consisted of 20 runs each lasting 100 generations, using a population size of 40 networks. The fitness of each network was the minimum, over the three trials (i.e. starting in the three initial states shown in figure 4c), of the value

[2]This task has been used in past reinforcement learning competitions, `http://rl-competition.org`

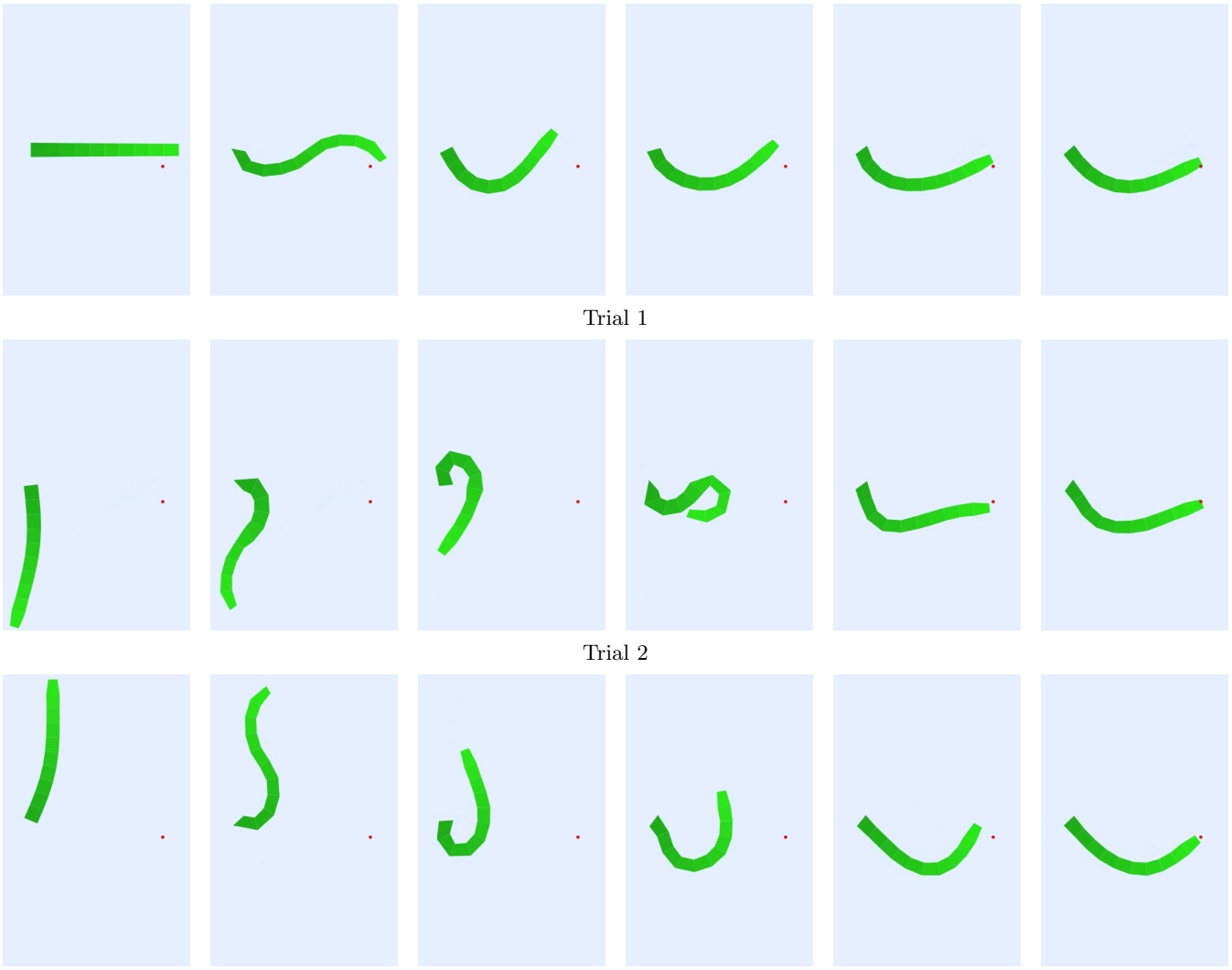

Figure 5: Octopus arm visualization. Visualization of the behavior of one of the successful controllers. The controller uses a *whip*-like motion to overcome the environment friction. This sequence of snapshots was captured from the video available at `http://www.idsia.ch/~koutnik/images/octopus.mp4`.

$1/(t \cdot d)$, where t is the number of transpired 10ms simulation steps, and d is the distance of the arm tip to the target.

The results are summarized in Table 3 showing the average of the best fitness found in each run for both direct and indirect encoding schemes. Searching directly in the 728-dimensional weight space yields an average fitness of 0.173, very close to the fitness achieved by searching in the relatively small 10-dimensional Fourier space. At 5 coefficients performance deteriorates because the weight matrices that can be expressed by so few coefficients are too simple given the number of weights in the network. At 40 coefficients, the fitness compared to direct encoding doubles, and there is a 18-fold reduction in the number of genes being evolved. As the number of coefficient is increased further, the to 80, 160 and 320 the performance trends back down.

The evolved controllers exhibit quite natural looking behavior. For example, when starting with the arm hanging down (initial state 3), the controller employs a *whip*-like motion to stretch the arm tip toward the goal, and overcome gravity and the resistance from the fluid environment (figure 5).

4. DISCUSSION AND FUTURE DIRECTIONS

The experimental results revealed that searching in the "compressed" space of Fourier coefficients can improve search efficiency over the standard, direct search in weight space. The frequency domain representation exploits the structure of the weight matrix which reduces the number of coefficients required to encode successful networks.

Of course, in these preliminary experiments we have made the implicit assumption that solutions will have spatially correlated weights. It is possible that, for the tasks examined here, there exists a permutation in the weight ordering for which the only solutions are those with spatially uncor-

related weights, i.e. requiring the full set of coefficients (no attempt was made to predefine favorable weight orderings). For example, in the non-Markov pole balancing benchmarks, the presence of a single recurrent connection which may need to be set very differently from the rest of what are just a few weights, could be the reason that the frequency domain encoding was not successful. However, ultimately, the potential of this approach lies in providing compact representations for very large networks, such as those required for vision, where many thousands of inputs have a natural, highly correlated ordering.

A case in point is the octopus arm, where the input space has a correlated structure: each compartment adds another 8 variables whose values should be correlated with those of the adjacent compartments given the physical constraints on the arm. However, even in this case, a more amenable ordering of the weights might be to organize them by compartment rather than by neuron, so that a simple, periodic signal (i.e. using a small number of coefficients) can encode the correlation between compartments. Or, the weights could by organized in a 3D cuboid rather than 2D matrix where each 2D slice corresponds to a compartment. As this research moves toward applications in vision, we will explore these ideas further to fully exploit problem-specific regularities.

Another approach would be to use a different basis altogether such as wavelets, which can cope with spatial locality, thereby making the encoding less sensitive to the particular weight ordering. Wavelets could help in the case of networks with multiple layers where it might be desirable for each layer sub-matrix to exhibit different regularities. With the current DCT basis, this could be handled less compactly by using a different set of coefficients for each layer.

In current implementation the number coefficients is simply specified by the user, without fully exploiting the importance ordering property of the representation. It should be straightforward to search incrementally, starting with low-complexity networks represented by just a few coefficients and then adding more, one at a time, when performance stagnates, or some other criteria is met. Future work will also look to extend the underlying concept more radically so that the same set of coefficients can be used to encode both the weights and the topology simultaneously.

ACKNOWLEDGMENTS

The research was supported by the STIFF EU Project (FP7-ICT-231576) and partially by the Humanobs EU Project (FP7-ICT-231453).

5. REFERENCES

[1] Z. Buk. High-dimensional cellular automata for neural network representation. In *International Mathematica User Conference 2009*, Champaign, Illinois, USA, 2009.

[2] Z. Buk, J. Koutník, and M. Šnorek. NEAT in HyperNEAT substituted with genetic programming. In *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, 2009.

[3] J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 997–1004, New York, NY, USA, 2007. ACM.

[4] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceeding of the European Conference on Machine Learning*, number 4212 in LNAI, pages 654–662. Springer, 2006.

[5] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

[6] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, l'Universite Claude Bernard-Lyon 1, France, 1994.

[7] J. Koutník, F. Gomez, and J. Schmidhuber. Searching for minimal neural networks in Fourier space. In *Proceedings of the 4th Annual Conference on Artificial General Intelligence*, 2010.

[8] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition).* Springer, 1997.

[9] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[10] A. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks (Seattle, WA)*, pages 667–673. Piscataway, NJ: IEEE, 1991.

[11] Y. Yekutieli, R. Sagiv-Zohar, R. Aharonov, Y. Engel, B. Hochner, and T. Flash. A dynamic model of the octopus arm. I. biomechanics of the octopus reaching movement. *Journal of Neurophysiology*, 94(2):1443–1458, 2005.