Compressed Network Complexity Search

Faustino Gomez, Jan Koutník, and Jürgen Schmidhuber

IDSIA
USI-SUPSI
Manno-Lugano, CH
{tino,hkou,juergen}@idsia.ch

Abstract. Indirect encoding schemes for neural network phenotypes can represent large networks compactly. In previous work, we presented a new approach where networks are encoded indirectly as a set of Fourier-type coefficients that decorrelate weight matrices such that they can often be represented by a small number of genes, effectively reducing the search space dimensionality, and speed up search. Up to now, the complexity of networks using this encoding was fixed *a priori*, both in terms of (1) the number of free parameters (topology) and (2) the number of coefficients. In this paper, we introduce a method, called Compressed Network Complexity Search (CNCS), for automatically determining network complexity that favors parsimonious solutions. CNCS maintains a probability distribution over complexity classes that it uses to select which class to optimize. Class probabilities are adapted based on their expected fitness. Starting with a prior biased toward the simplest networks, the distribution grows gradually until a solution is found. Experiments on two benchmark control problems, including a challenging non-linear version of the helicopter hovering task, demonstrate that the method consistently finds simple solutions.

1 Introduction

Indirect or generative encoding schemes for neural network phenotypes [2–4,9,11] offer the potential of allowing very large networks to be represented compactly. In previous work [5,6], we presented a new encoding where network weight matrices are represented indirectly as a set of Fourier-type coefficients that are transformed into weight values via an inverse Fourier transform, so that evolutionary search is conducted in the frequency-domain instead of weight space. If adjacent weights in the matrices are correlated, then this regularity can be encoded using fewer coefficients than weights, effectively reducing the search space dimensionality. For problems exhibiting a high-degree of redundancy, this "compressed" approach can result in an order of magnitude fewer free parameters and significant speedup [5].

Up to now the complexity of networks using this encoding was fixed *a priori*, both in terms of (1) the number of free parameters or topology and (2) the number of coefficients (compression ratio). In this paper, we introduce a

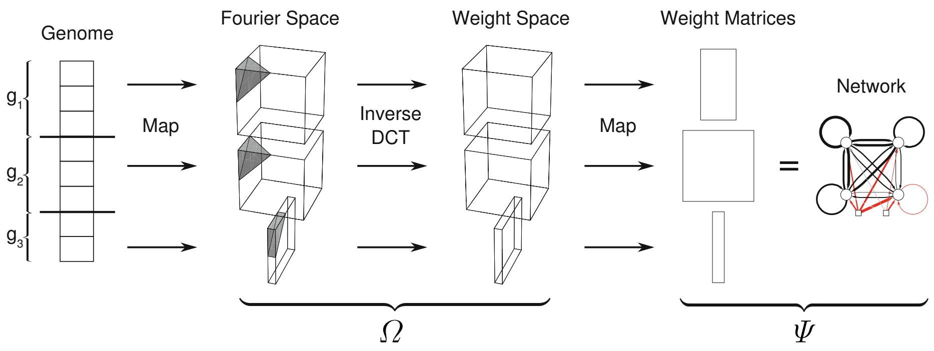


Fig. 1. Decoding the compressed networks. The figure shows the three step process involved in transforming a genome of frequency-domain coefficients into a recurrent neural network. First, the genome (left) is divided into k chromosomes, one for each of the weight matrices specified by the network architecture, Ψ. Each chromosome is mapped, by Algorithm 1, into a coefficient array of a dimensionality specified by Ω. In this example, an RNN with two inputs and four neurons is encoded as 8 coefficients. There are $k = |\Omega| = 3$, chromosomes and $\Omega = \{3, 3, 2\}$. The second step is to apply the inverse DCT to each array to generate the weight values, which are mapped into the weight matrices in the last step.

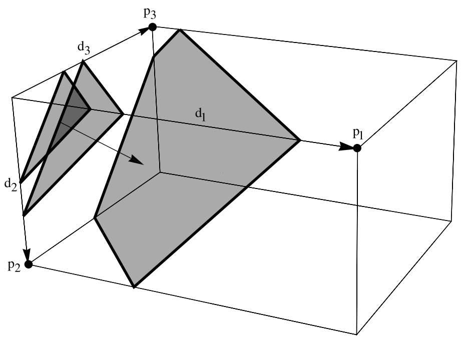
method inspired by universal search [7], called Compressed Network Complexity Search (CNCS), that automatically determines network complexity, favoring parsimonious solutions. CNCS maintains a probability distribution over complexity classes, which it uses to select which class to optimize. The probability of a given class is adapted based on the expected fitness of individuals sampled from it. Starting with a prior biased toward the simplest networks, the distribution adapts gradually until a solution is found.

The idea of enforcing parsimony in neuroevolution has been explored previously [15, 16], usually by adding a regularization term to the fitness function that penalizes complexity. Our approach is more in line with NEAT [10] where simple networks are favored by starting evolution with a population of *directly* encoded networks that have minimal topologies.

The next section describes how networks are encoded in the frequency domain. Section 3, introduces the complexity search method, CNCS. Section 4, presents experiments applying CNCS to the octopus arm task with high-dimen-sional actions to determine the number of coefficient genes used to represent networks; and in section 5 it is used to search for both the number of neurons (topology) and coefficients, for networks controlling a challenging version of the Helicopter Hovering benchmark.

2 DCT Network Representation

Networks are encoded as a string or *genome*, $\boldsymbol{g} = \{g_1, \ldots, g_k\}$, consisting of k substrings or *chromosomes* of real numbers representing Discrete Cosine Transform



Algorithm 1: Coefficient mapping(g, d)

```
$j \leftarrow 0$
$K \leftarrow \mathrm{sort}(\mathrm{diag}(d) - \mathbb{I})$
for $i = 0$ to $|d| - 1 + \sum_{n=1}^{|d|} d_n$ do
    $l \leftarrow 0$
    $s_i \leftarrow \{e | \sum_{k=1}^{|d|} e_{\xi_j} = i\}$
    while $|s_i| > 0$ do
        $ind[j] \leftarrow \underset{e \in s_i}{\mathrm{argmin}} \| e - K[l{+}{+} \bmod |d|] \|$
        $s_i \leftarrow s_i \setminus ind[j{+}{+}]$
for $i = 0$ to $|ind|$ do
    if $i < |g|$ then
        coeff_array[$ind[i]$] $\leftarrow c_i$
    else
        coeff_array[$ind[i]$] $\leftarrow 0$
```

Fig. 2. Mapping the coefficients. The cuboidal array is filled with the coefficients from chromosome g one simplex at a time, according to Algorithm 1, starting at the origin and moving to the opposite corner one simplex at a time.

(DCT) coefficients. The number of chromosomes is determined by the choice of network architecture, Ψ, and data structures used to decode the genome, specified by $\Omega = \{D_1, \ldots, D_k\}$, where D_m, $m = 1..k$, is the dimensionality of the coefficient array for chromosome m. The total number of coefficients, $C = \sum_{m=1}^{k} |g_m| \ll N$ (where N is the number of weights), is user-specified (for a compression ratio of N/C), and the coefficients are distributed evenly over the chromosomes. Which frequencies should be included in the encoding is unknown. The approach taken here restricts the search space to *band-limited* neural networks where the power spectrum of the weight matrices goes to zero above a specified limit frequency, c_ℓ^m, and chromosomes contain all frequencies up to c_ℓ^m, $g_m = (c_0^m, \ldots, c_\ell^m)$.

Figure 1 illustrates the procedure used to decode the genomes. In this example, a fully-recurrent neural network (on the right) is represented by $k = 3$ weight matrices, one for the input layer weights, one for the recurrent weights, and one for the bias weights. The weights in each matrix are generated from a different chromosome which is mapped into its own D_m-dimensional array with the same number of elements as its corresponding weight matrix; in the case shown, $\Omega = \{3, 3, 2\}$: 3D arrays for both the input and recurrent matrices, and a 2D array for the bias weights.

In previous work [5], the coefficient matrices were 2D, where the simplexes are just the secondary diagonals; starting in the top-left corner, each diagonal is filled alternately starting from its corners. However, if the task exhibits inherent structure that cannot be captured by low frequencies in a 2D layout, more compression can potentially be gained by organizing the coefficients in higher-dimensional arrays.

Each chromosome is mapped to its coefficient array according to Algorithm 1 which takes a list of array dimension sizes, $d = (d_1, \ldots, d_{D_m})$ and the chromosome, g_m, to create a total ordering on the array elements, $e_{\xi_1,\ldots,\xi_{D_m}}$. In the first

Algorithm 2. CNCS($\mathcal{D}$,f,s,n,σ_θ)

```
while $\neg$converged do
    for $k = 1$ to $s$ do
        $\mathbf{x}_k \sim \mathcal{D}$                                                   //draw sample
        $(\boldsymbol{\mu}_{\mathbf{x}_k}, \boldsymbol{\sigma}_{\mathbf{x}_k}) \leftarrow \text{SNES}(f, \boldsymbol{\mu}_{\mathbf{x}_k}, \boldsymbol{\sigma}_{\mathbf{x}_k}, \lambda(C_k), n)$
        $\phi_{\mathbf{x}_k} \leftarrow f(\boldsymbol{\mu}_{\mathbf{x}_k})$                                 //store fitness
    foreach $\mathbf{x}_i \in \mathcal{D}$ do
        $g(\mathbf{x}_i) \leftarrow \begin{cases} \sum_{\forall \mathbf{x}_j \in \mathcal{D}} \phi_{\mathbf{x}_j} \frac{1}{h^d} \mathcal{K}\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right) & \max(\boldsymbol{\sigma}_{\mathbf{x}_i}) > \sigma_\theta \\ 0 & \text{otherwise} \end{cases}$
    foreach $\mathbf{x}_i \in \mathcal{D}$ do
        $p(\mathbf{x}_i) \leftarrow \frac{g(\mathbf{x}_i)}{\sum_{\forall \mathbf{x}_j \in \mathcal{D}} g(\mathbf{x}_j)}$                          //normalize
```

loop, the array is partitioned into $(D_m - 1)$-simplexes, where each simplex, s_i, contains only those elements e whose Cartesian coordinates, $(\xi_1, \ldots, \xi_{D_m})$, sum to integer i. The elements of simplex s_i are ordered in the `while` loop according to their distance to the corner points, p_i (i.e. those points having exactly one non-zero coordinate; see example points for a 3D-array in figure 2), which form the rows of matrix $K = [p_1, \ldots, p_m]^T$, sorted in descending order by their sole, non-zero dimension size. In each loop iteration, the coordinates of the element with the smallest Euclidean distance to the selected corner is appended to the list ind, and removed from s_i. The loop terminates when s_i is empty.

After all of the simplexes have been traversed, the vector ind holds the ordered element coordinates. In the final loop, the array is filled with the coefficients from low to high frequency to the positions indicated by ind; the remaining positions are filled with zeroes. Finally, a D_m-dimensional inverse DCT transform is applied to the array to generate the weight values, which are mapped to their position in the corresponding 2D weight matrix. Once the k chromosomes have been transformed, the network is complete.

3 Compressed Network Complexity Search

The basic idea of CNCS is to discover networks with minimal complexity by running multiple independent evolutionary processes in parallel, one for each complexity class, allocating run-time to each according to an adaptive probability mass function, $\mathcal{D}$. Algorithm 2 describes CNCS in pseudocode. The algorithm is initialized with a prior distribution over the complexity classes (C, Ψ), where C is the number of coefficients used to encode the network, and Ψ is the number of neurons (equivalently, the topology). In order to bias the search toward low-complexity solutions, $\mathcal{D}$ should be initialized with a prior that gives high probability to small networks (low Ψ), represented by the fewest number of coefficients (low C).

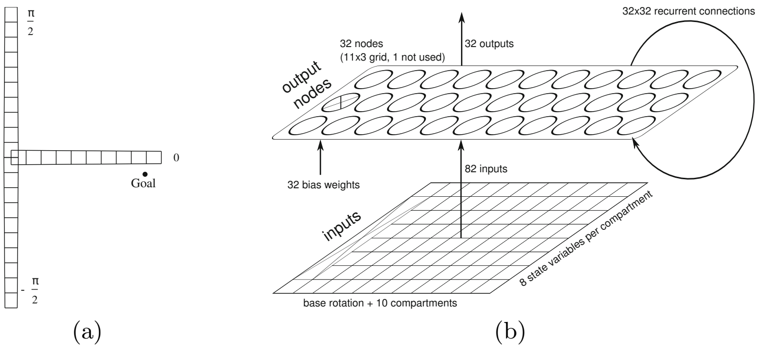


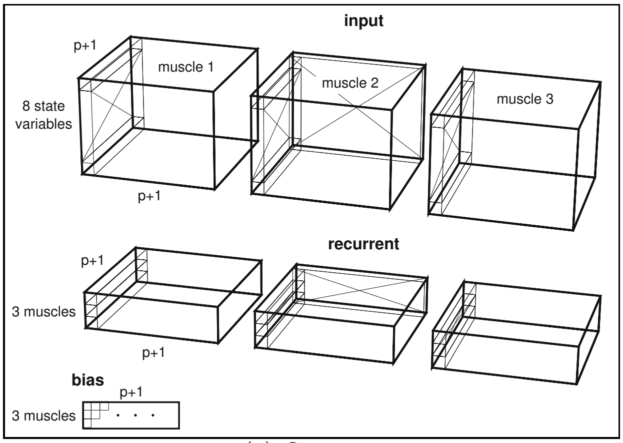
Fig. 3. Octopus arm task. (a) A flexible arm consisting of p compartments, each with 3 muscles, must be controlled to touch a goal location with the arm tip from 3 different initial positions, $-\pi/2$, 0 and $\pi/2$. (b) The arm is controlled by a fully recurrent network with 32 neurons, one for each action (muscle). This topology is fixed, and only the number of coefficients used to represent its weights is determined automatically by CNCS.

Each $\mathbf{x}_\iota = (C_\iota, \Psi_\iota)$ pair in $\mathcal{D}$ has its own dedicated search algorithm used to optimize that particular configuration. In the current implementation we use Separable Natural Evolution Strategies (SNES; [13]), an efficient variant in the NES [12] family of black-box optimization algorithms. In each generation, SNES samples a population of λ individuals, computes a Monte Carlo estimate of the fitness gradient, transforms it to the natural gradient and updates the search distribution parameterized by a mean vector, $\boldsymbol{\mu}$, and *diagonal* covariance matrix, $\boldsymbol{\sigma}$ (see [12] for a full description of NES). The SNES search distribution associated with configuration $\mathbf{x}_\iota$ has mean $\boldsymbol{\mu}_{\mathbf{x}_\iota}$ and covariance $\boldsymbol{\sigma}_{\mathbf{x}_\iota}$.

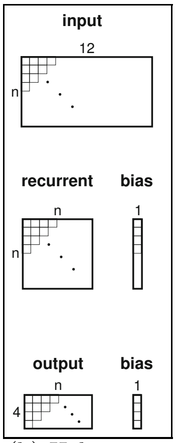
Each iteration, CNCS draws s samples from $\mathcal{D}$, and runs the SNES corresponding to each sample for n generations, after which the search distribution $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ and its expected fitness value ϕ are saved. The distribution $\mathcal{D}$ is then re-estimated using a multivariate Parzen window estimator with radial-symmetric Gaussian kernel $\mathcal{K}$ [8]. First, the values $g(\mathbf{x})$ are computed by applying the kernel weighted by the normalized fitnesses, $\phi_{\mathbf{x}_j}$ (the first `forall` loop), where h is the kernel width, and d is the dimensionality of $\mathcal{D}$, e.g. 2 when estimating C and Ψ (the SNES distributions that have converged, $\max(\boldsymbol{\sigma}) \leq \sigma_\theta$, are assigned a g value of 0). Then the g values are normalized into probabilities, and the cycle repeats. The algorithm terminates when all search distributions (within the bounds of $\mathcal{D}$) have converged, or either the desired fitness or the maximum number of iterations has been reached.

4 Octopus Arm Control

The octopus arm consists of p compartments floating in a 2D water environment (see figure 3a). Each compartment has a constant volume and contains three



(a) Octopus



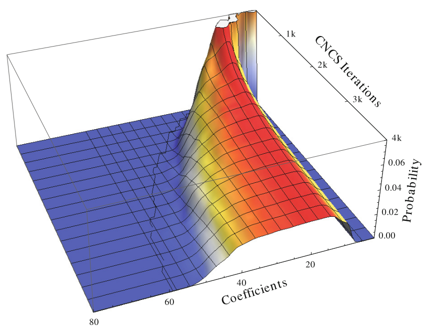
(b) Helicopter

Fig. 4. Network decoding schemes. (a) for the octopus arm networks the genome is split into 3 chromosomes, Ω=$\{4, 4, 2\}$. The coefficients in the chromosome used to generate the input weight matrix are placed in a 4D array, and a 2D array for the bias weights. (b) for the helicopter networks there are 5 chromosomes, Ω=$\{2, 2, 1, 2, 1\}$: a 2D input and recurrent and output arrays, and 1D arrays for the input and output bias weights.

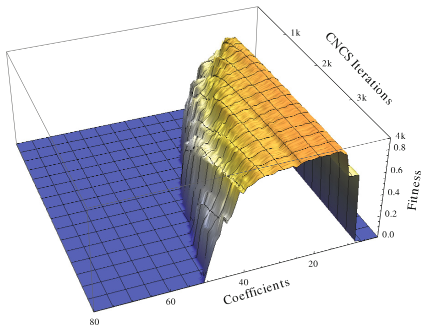
controllable muscles (dorsal, transverse and ventral). The goal of the task to reach a target position with the tip of the arm, starting from three different initial positions, by contracting the appropriate muscles at each 1s step of simulated time. While initial positions $-\pi/2$ and $\pi/2$ look symmetrical, they are actually quite different due to gravity. 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 $8p + 2$ state variables. Though there are $3p + 2$ muscles, the task is normally simplified by aggregating them into 8 “meta”-actions that contract groups of muscles simultaneously (i.e. all dorsal, all transverse, etc.). Here, instead we use the more difficult configuration where the “raw” actions are controlled directly, so that each muscle must be coordinated with the others to move the arm.

4.1 Setup

For the $p = 10$ compartment arm used in these experiments, a network with 32 neurons (one for each raw action) is sufficient to perform well on this task. Therefore, CNCS is used only to search for the number of coefficients, so that the distribution, $\mathcal{D}$, is one-dimensional, with a uniform prior over $C = 1..10$, a sample size of $s = 1$, and number of SNES generations per CNCS update, $n = 1$. The kernel width was $h = 7$, and the convergence condition was set to $\sigma_\theta = 0.01$. Each SNES used a population size of 16. Five experiments were run, for 4 thousand iterations each.



(a)



(b)

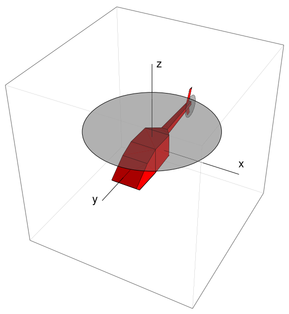
Fig. 5. Octopus arm results. The 3D plots show how (a) the distribution over the number of coefficients, C, adapts over time in CNCS, and (b) configurations with a better fitness are explored. The final distribution after 4,000 iterations is focused between $C = 15$ and $C = 50$, and peaks at $C = 18$, for a compression ratio of 204:1; 3680 weights/18 coefficients and fitness reaching 0.88.

The octopus arm was controlled using the fully-recurrent network architecture shown in figure 3b which is decoded using the following scheme, $\Omega = \{4, 4, 2\}$, depicted in figure 4a: the genome is partitioned into $k = 3$ chromosomes, mapped into three arrays: (1) a 4D $8\times(p{+}1)\times 3\times(p{+}1)$ array that contains input weights for a $3\times(p{+}1)$ grid of neurons, one for each raw action, (2) a $3\times(p{+}1)\times 3\times(p{+}1)$ recurrent weight array, and (3) and a $3\times(p{+}1)$ bias array. The dimension size of 3 in these arrays refers to the number of muscles per compartment.

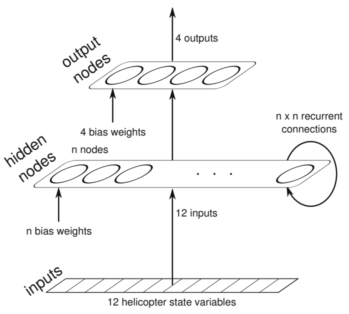
The fitness was computed as the average of the following score over three trials: $\max\left[1 - \frac{t}{T}\frac{d}{D}, 0\right]$, where t is the number of time steps before the arm touches the goal, T is the maximum number of time steps in a trial, d is the final distance of the arm tip to the goal and D is the initial distance of the arm tip to the goal. Each of the three trials starts with the arm in a different configuration (see figure 3a). This fitness measure is different to the one used in [14], because minimizing the integrated distance of the arm tip to the goal causes greedy behaviors. In the viscous fluid environment of the octopus arm, a greedy strategy using the shortest length trajectory does not lead to the fastest movement: the arm has to be compressed first, and then stretched in the appropriate direction. Our fitness function favors behaviors that reach the goal within a small number of time steps.

4.2 Results

Figure 5 shows how the distribution (a) over coefficients, C, and the fitness of each configuration (b) adapts over the course of $4,000$ iterations of CNCS (averaged over 20 runs). The distribution is initialized with a prior that favors networks with low complexity, expressed solely with C (top-right corner of the graphs). The expected fitness forms a ridge with a peak between $C = 15$ and $C = 18$ with a moderate slope on the left, towards higher C. This focuses the search between $C = 15$ and $C = 50$. The expected fitness for $C < 15$



(a)



(b)

Fig. 6. Helicopter hovering task with gusting wind. (a) The helicopter must be controlled to stay within 20m from its initial position at the origin (as shown). Unlike the standard version of the task where wind blows at a constant velocity, the wind here blows in gusts with much higher velocity, forcing the controller to react quickly to sudden changes in wind. (b) The helicopter is controlled by a simple recurrent network (SRN). Both the number of neurons and the number of coefficients are determined automatically by CNCS.

drops significantly lowering the probability of sampling configurations in this area. Reasonable fitness of 0.75 was reached at iteration 350 using less than 15 coefficients. A fitness of 0.88 that is close to optimum was reached at iteration 2560 using 18 coefficients. The weight matrices of such networks were compressed down from 3680 weights (compression ratio of 204:1).

5 Helicopter Hovering with Gusting Wind

The standard Helicopter Hovering benchmark involves maintaining the position of a simulated XCell Tempest [1] as close as possible to origin of a bounded 3D space (see figure 6a). The helicopter model consists of 12 state variables: the coordinates and angular rotations in 3-space and their derivatives; and 4 control variables: longitudinal and latitudinal cyclic pitch and tail, and main rotor collective pitch. The fitness is the sum of squares of all state variables over the course of a flight lasting t time steps. If the helicopter moves more that 20m away from the origin in any direction or its velocity exceeds 5 m/s, then it is considered to have crashed, and the trial is terminated. The fitness is normalized between 0 and 1 and the minimum over 5 trials is used.

The original 2008 RL competition version of this problem featured wind along the x- and y-axes with a drag of up to 5m/s, which is initialized at random in the beginning of each trial. With this setup, it turns out that a linear controller can be easily trained to solve the task. Therefore, in order to make the task more challenging, requiring non-linear control, the original wind model was modified so that instead of constant wind, strong "gusts" buffet the helicopter at random. Wind gusts occur in both x and y directions with probability 0.4 and a velocity of 20m/s, which decays exponentially after striking the helicopter. The gusting

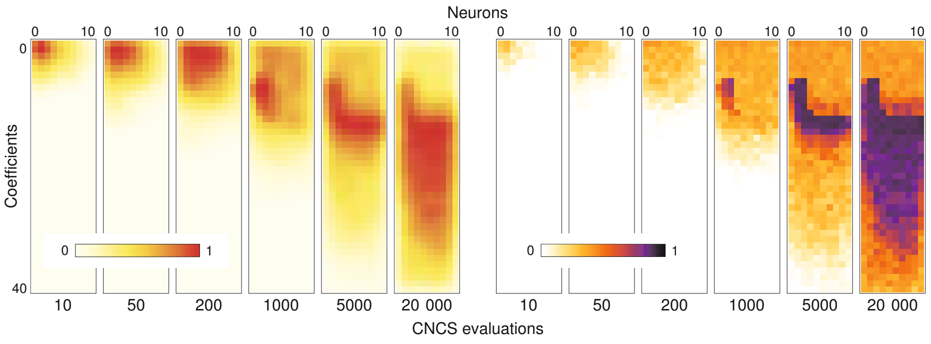


Fig. 7. Complexity search distribution for helicopter hovering task. The figure shows how the distribution $\mathcal{D}$ (left) and fitness (right) evolve over time, averaged across 20 runs. The algorithm first explores simple networks, with only a few neurons, represented by a small number of DCT coefficients (around 500 steps), then gradually spreads the distribution toward more complex configurations, identifying two clusters with high fitness (networks with 3 neurons defined with 8 coefficients and 3-node networks defined with 12 coefficients).

wind makes the task significantly harder—a linear controller cannot cope with the abrupt wind perturbations. Because higher velocities are required to control the helicopter under these conditions, the limit on the velocity was removed.

5.1 Setup

The helicopters were controlled using simple recurrent networks (SRN/Elman; figure 6b). The decoding scheme for the genomes, $\Omega = \{2, 2, 1, 2, 1\}$, is depicted in figure 4b. The original benchmark involves flights of $t = 6000$ time steps (equivalent to 60s flight). For the task with gusting wind used here, this was reduced to 100 both for efficiency, and because, due to the severity of the wind, such a short trial is enough to evaluate relative controller competence.

CNCS was used to search for both the network topology, Ψ (number of neurons), and number of coefficients, C. The complexity distribution, $\mathcal{D}$, was initialized with a uniform prior over the range $\{1, 2\}$ for both Ψ and C, i.e. $p(C, \Psi) = 0.25$, $C, \Psi = 1, 2$, and a sample size $s = 2$. As in the octopus task, $h = 7$, $\sigma_\theta = 0.01$, $n = 1$, and each SNES used a population size of 16. A total of 20 experiments were run, for 20 thousand iterations each.

5.2 Results

Figure 7 shows how the distribution over (Ψ, C) configurations (top row) and the fitness of each configuration (bottom row) adapts over the course of 20k iterations of CNCS (averaged over 20 runs). The distribution is initialized with a prior that concentrates on networks with the lowest complexity (upper-left corner of the graphs). Gradually, as the distribution expands, it finds high fitness individuals with 1 to 3 neurons, using ≈ 8 coefficients, at around iteration

1000. These networks are simple both in terms of their topology (model complexity) and in the regularity of their weight matrices (compression ratio of 8:1, 64 weights/8 coefficients). The distribution then focuses on this area, moving away from configurations with fewer coefficients ($C < 8$) as they cannot express the level of complexity required for nets with more that 3 neurons. At this point, the distribution begins to follow a narrow, high-fitness corridor, adding coefficients to networks with 2 and 3 neurons, until it reaches $C = 12$ (≈ 2000 iterations) and starts to grow the size of networks. The shape of the distribution at 20k iterations emerged consistently for all runs, with a maximum relative entropy between the distributions of any two runs of only 0.038.

6 Discussion and Future Work

CNCS consistently found low-complexity solutions for the two tasks tested. The octopus arm reaching task was successfully solved with networks having 3680 weights that were generated using just 18 DCT coefficients, and networks with 2 neurons (64 weights) represented by 8 DCT coefficients were found in the early stage of the CNCS for the Helicopter Hovering task. Helicopter networks were progressively improved by increasing the number of DCT coefficients beyond 12, and broadening the search to more hidden neurons.

The experimental results show that updating the distribution on complexity classes elegantly addresses the question of how to configure the evolutionary search. Running all configurations in parallel would be prohibitive, whereas CNCS quickly adapts the search distribution towards promising configurations.

A potential drawback of CNCS lies in wide valleys of low fitness that span across complexity space. One has to ensure, that the width of the smoothing kernel used is wider than the potential valley. Otherwise, the distribution will never reach across to sample networks of higher complexity. A possible solution could be to use a variable kernel width for each complexity class based on e.g. the number of samples evaluated from that class.

Future experiments will test the generalization of the evolved controllers to verify whether complexity is correlated with robustness. We expect that the small networks, although they have slightly worse fitness during the training (like those small networks that can control the helicopter to hover), will generalize better.

Acknowledgments. This research was funded by SNF grant 200020-125038/1.

References

1. Abbeel, P., Ganapathi, V., Ng, A.Y.: Learning vehicular dynamics, with application to modeling helicopters. In: NIPS (2005)
2. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with Analog Genetic Encoding. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 671–680. Springer, Heidelberg (2006)

3. Gruau, F.: Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Superieure de Lyon, Institut IMAG, Lyon, France (1992)
4. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Systems 4, 461–476 (1990)
5. Koutník, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2010 (2010)
6. Koutník, J., Gomez, F., Schmidhuber, J.: Searching for minimal neural networks in fourier space. In: Proc. of the 4th Conf. on Artificial General Intelligence (2010)
7. Levin, L.A.: Universal sequential search problems. Problems of Information Transmission 9(3), 265–266 (1973)
8. Parzen, E.: On estimation of a probability density function and mode. The Annals of Mathematical Statistics 33(3), 1065–1076 (1962)
9. Schmidhuber, J.: Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Networks 10(5), 857–873 (1997)
10. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002)
11. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life 9(2), 93–130 (2003)
12. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural Evolution Strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC 2008), Hongkong. IEEE Press (2008)
13. Wierstra, D., Schaul, T., Sun, T.G.Y., Schmidhuber, J.: Natural evolution strategies. Technical report (2011), arXiv:1106.4487v1
14. Woolley, B.G., Stanley, K.O.: Evolving a Single Scalable Controller for an Octopus Arm with a Variable Number of Segments. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 270–279. Springer, Heidelberg (2010)
15. Zhang, B.-T., Muhlenbein, H.: Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex Systems 7, 199–220 (1993)
16. Zhang, B.-T., Muhlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation 3, 17–38 (1995)