Searching for Minimal Neural Networks in Fourier Space

Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber
IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
University of Lugano & SUPSI, Switzerland

Abstract

The principle of minimum description length suggests looking for the *simplest* network that works well on the training examples, where simplicity is measured by network description size based on a reasonable programming language for encoding networks. Previous work used an assembler-like universal network encoding language (NEL) and Speed Prior-based search (related to Levin's Universal Search) to quickly find low-complexity nets with excellent generalization performance. Here we define a more natural and often more practical NEL whose instructions are frequency domain coefficients. Frequency coefficients may get encoded by few bits, hence huge weight matrices may just be low-complexity superpositions of patterns computed by programs with few elementary instructions. On various benchmarks this weight matrix encoding greatly accelerates the search. The scheme was tested on pole-balancing, long-term dependency T-maze, and ball throwing. Some of the solutions turn out to be unexpectedly simple as they are computable by fairly short bit strings.

Introduction

Given some training experience, what is the best way of computing a weight matrix for a neural network such that it will perform well on unseen test data? Let us ignore for the moment the numerous gradient and evolution based training methods (both with or without teachers), and focus on the essential. The principle of minimum description length (MDL) [WB68, Ris78, LV97] suggests one should search for the *simplest* network that works well on the training examples, where simplicity is measured by the description size of the network, in a reasonable (possibly universal) programming language. In theory, the simplest or most compressible weight matrix for a given problem is the one with lowest algorithmic information or Kolmogorov complexity, i.e. the one computable by the shortest program. Unfortunately, there is no general way of finding this program, due to lack of an upper bound on its runtime [Sol64, Kol65, LV97].

However, there is a theoretically "best" way of taking runtime into account [Lev73, Sch02]. This is the basis of previous work on optimal search for simple networks [Sch95, Sch97], which used an assembler-like universal network encoding language (NEL) and Speed Prior-based search [Sch02] (related to Levin's Universal Search [Lev73]), to quickly find low-complexity weight matrices with excellent generalization performance.

In related work, in the context of neuroevolution [Gru92, GS07, BKS09], less general NELs have been used to encode network parameters indirectly in symbol strings which are evolved using a genetic algorithm. Like the early work [Sch95, Sch97], these approaches allow short descriptions to specify networks of arbitrary size.

Here we define a NEL whose instructions are bit representations of Fourier series coefficients, and network weight matrices are computed by applying inverse Fourier-type transforms to the coefficients. This not only yields continuity (a small change to any coefficient changes all weights by a small amount) but also allows the algorithmic complexity of the weight matrix to be controlled by the number of coefficients. As frequency domain representations decorrelate the signal (weight matrix), the search space dimensionality can be reduced in a principled manner by discarding high-frequency coefficients, as is common lossy image coding (note that ignoring high frequencies in the initial phase is encouraged by observations of factorial redundancy in trained weight matrices [Rad93]). Therefore, the search for a good weight matrix can be performed systematically starting with smooth weight matrices containing only low frequencies, and then successively adding higher frequencies.

Encoding in the frequency domain also means that the size of the program is independent of the size of the network it generates, so that networks can be scaled to high-dimensional problems, such as vision, since a very short program consisting of frequency coefficients, each encoded by a few bits, can compute huge weight matrices. While this is the main motivation for most indirect network encoding schemes, here we consider indirect encoding in the opposite direction: given a problem for which a relatively small network solution is known, is there a short encoding that allows the network space to be searched exhaustively?

The next section describes the neural network encoding scheme in detail and the variant of universal search used to find solutions. We then present experimental results in three test domains, showing how some of the solutions turn out to be surprisingly simple, as they are computable by fairly short, network-computing bit strings.

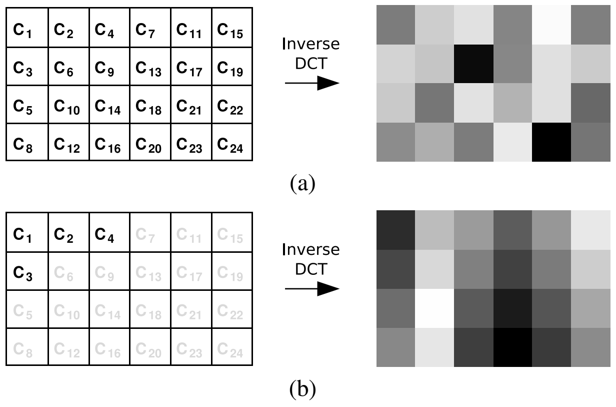


Figure 1: **DCT network representation.** The coefficients are selected according to their order along the second diagonals, going from upper-left corner to the bottom right corner. Each diagonal is filled from the edges to the center starting on the side that corresponds to the longer dimension. (a) Shows and example of the kind of weight matrix (right) that is obtained by transforming the full set of coefficients (left). The grayscale 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).

Algorithm 1: Universal Network Search (r)

```
for $x \leftarrow 1$ to $2^{2^r}$ do
  for $n \leftarrow n_{min}$ to $x$ do
    for $b \leftarrow 1$ to $x$ do
      for $c \leftarrow 1$ to MIN($b,N$) do
        if MAX($n,b,c$)=$x$ then
          for $s \leftarrow 1$ to $2^b$ do
            $D \leftarrow$ DECODE($n,c,$BINARY($s$))
            $network \leftarrow$ INVERSEDCT($D$)
            if SOLVED?(EVALUATE($network$)) then
              return ENCODE($r,n,b,$BINARY($s$))
          end
```

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. This in turn makes it possible to search "universally" from small networks that can be represented by few coefficients, to larger networks requiring more complex weight matrices.

The next two sections describe how the networks are represented in the frequency domain using the Discrete Cosine Transform (DCT), and the version of universal search that is used to systematically find solutions to the experiments that follow.

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 and Jordan/Elman networks, as 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. 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 of the 24 coefficients is 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 only gradual, correlated, changes in value from weight to weight.

Universal Network Search

In order to search the space of networks universally, a strict total ordering must be imposed on the possible DCT encodings. We accomplish this by representing the c coefficients using a total of b bits, and iterating over all possible bit-strings using Universal Network Search (UNS), described in Algorithm 1. The outer-most loop imposes an upper limit, x, for n, b and c. The next three loops examine all combinations of neurons, bits and coefficients, constrained by, n_{min}, the number of output units required by problem in question (second loop), and, N, the total number of weights in the network. Each of the 2^b bit-strings (third loop) is partitioned in b different ways (fourth loop); each partitioning denoting a different number of coefficients. If (b mod c) $\neq 0$ then the modulo is distributed into the coefficients from the beginning. For example, if $b = 3$, each of the $2^3 = 8$ possible bit-strings has three possible partitionings: (1) only one coefficient, C_1, is represented using all three bits, (2) two coefficients, C_1 using two bits, and C_2 using the remaining bit, and (3) three coefficients, C_1, C_2 and C_3, each using one bit. The set a values that a coefficient C can take on is determined by dividing $[-\alpha, \alpha] \in \Re$ into $1/(2^{\ell(C)} - 1)$ intervals, where $\ell(C)$ is the number of bits used to represent C, and α is just a scaling factor. For example, if $\ell(C) = 2$ and

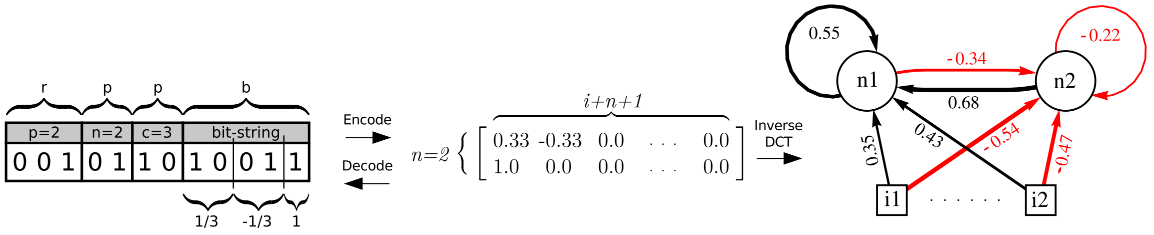


Figure 2: **Network representation and encoding**. A network with two neurons (right) is obtained by applying the inverse DCT of the matrix (middle), where three of the coefficients (C_1, C_2, C_3) are non-zero, in this example. The weight matrix can be encoded (left) by a total of 12 bits, five for the coefficient values, and seven for the "meta" information: three bits for the precision, p, which determines the size of the bit-fields representing n, the number of neurons, and c, the number of coefficients. This is all of the information needed to reconstruct (decode) the complete network.

$\alpha = 1.0$, then the set of values C can take is $\{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$.

Finally, in the inner-most loop, each of the 2^b networks specified by each unique (n, b, c) is decoded into a coefficient matrix D, which is then transformed into a weight matrix via the inverse DCT. The search terminates if either $x > 2^{2^r}$ or a network that solves the problem is found, in which case the successful network is encoded as described in figure 2, and returned.

To completely describe a network, simply storing the bit-string b is not sufficient, the number of neurons, n, and coefficients, c, must be encoded as well. To encode this information in minimal way, we first encode the number of bits that will be used to represent the parameters and then store the parameters with the fixed number of bits. The bit-string that completely describes the network consists of the following fields (see figure 2): r bits represent the bit precision, p, of the n and c fields, p bits each for n and c, and b bits for the actual coefficient values, $b \geq c$, for a total of $r + 2p + b$ bits, where $p \leq 2^r$. For the example 001 01 10 10011, shown in figure 2, the first field has size $r = 3$, and a decimal value of 2, so that n and c are represented by 2 bits, with values of 2 and 3, respectively, meaning that the network has 2 neurons, where 3 coefficients are described with by the last five bits 10011.

A universal search over all possible bit-strings would needlessly examine a large number of invalid bit-strings (having $b < c$). Therefore, we use Algorithm 1 which constrains the search to only valid, decodable strings, and is therefore an instance of Practical Universal Search [SS10].

Experimental Results

Universal Network Search was tested one three tasks: Markovian and non-Markovian pole balancing, the long term dependency T-maze, and the ball throwing task. In all experiments, the scaling factor, α, was set to 20, and the number of bits, r, used to represent the precision of n and c was set to three, which means that the search can continue up to networks with $2^{2^3} = 256$ neurons, more than enough for the tasks in question. In each task, the encoding scheme described in the previous section is used to quantify the complexity of network solutions, and is indicated by the "Total Bits" column in the tables.

Table 1: **Pole balancing results**. Each row describes the minimal (1-neuron) network solution for each task, the number of evaluations that UNS required to find it, and the total number of bits required to encode it. Notice that just 8 evaluations are needed to find a solution to the single pole Markov task.

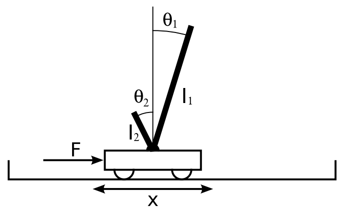
Task	b	c	**Eval.**	**Total Bits**
1 pole Markov	**2**	2	**8**	7
1 pole non-Markov	6	3	290	13
2 poles Markov	16	6	773,070	25
2 poles non-Markov	17	5	1,229,012	26

Pole Balancing

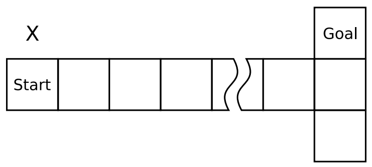
Pole balancing (figure 3a) is a standard benchmark for learning systems. The basic version consists of a single pole hinged to a cart, to which a force must applied in order to balance the pole while keeping the cart within the boundaries of a finite stretch of track. By adding a second pole next to the first, that 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 [Wie91] 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. For the Markov single pole task, a successful network with just one neuron whose weights are represented by 2 DCT coefficients is found after just 8 evaluations. This result shows how simple the single pole balancing is: the single neuron, which solves it has monotonically distributed weights. Non-Markovian single pole balancing increases complexity of the task only slightly.

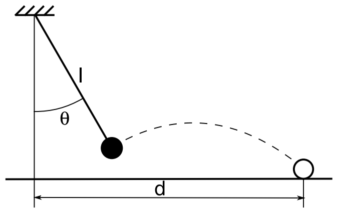
For the two-pole versions, 16 (17 for non-Markovian case) bits are required to solve the problem using a single neuron. Notice that the solution to the Markovian 2-pole task, requiring 8 weights (6 input + 1 recurrent + 1 threshold), has been compressed to 6 parameters, $C_1, ..C_6$. The non-Markovian 2-pole network has 5 weights and 5 coefficients were used, meaning that in this task it does not DCT



(a) Pole balancing



(b) T-maze



(c) Ball throwing

Figure 3: **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. (b) T-maze: the agent must travel down the corridor remembering the signal X which indicates the location of the goal. The length of the corridor is variable. (c) Ball throwing: the ball attached to the end of the arm must be thrown as far as possible by applying a torque to the joint and then releasing the ball.

Table 2: **T-maze results**. The table shows the two networks with the shortest bit descriptions, found by UNS. The check marks in the "T-maze length" column indicate the corridor lengths the network was able to solve. Note that the seven neuron network is found before the four neuron network since it requires fewer bits to encode (19 vs. 21).

				T-maze length			
n	b	c	**Eval.**	5	50	1000	**Total Bits**
7	10	10	$85,838$	$\surd$	$\surd$	$\surd$	19
4	12	11	$306,352$	$\surd$	–	–	21

Table 3: **Ball throwing results**. The table shows the first three network near optimal networks found by UNS (all network have two neurons). The d column indicates the distance, in meters, the ball was thrown by each network using the strategy indicated in the first column (compare to the distance, d_{opt}, of the corresponding optimal controller).

strategy	b	c	**Eval.**	**d [m]**	**d$_{\text{opt}}$[m]**	**Total Bits**
fwd	4	4	70	4.075	5.391	11
bwd-fwd	8	8	2516	5.568	5.391	17
bwd-fwd	9	9	5804	9.302	10.202	20

compress the weight matrix. In the other words, the number of bits per coefficient is the restriction which makes the exhaustive search possible.

Long Term Dependency T-maze

The T-maze task is a discrete non-Markovian problem consisting of a corridor of n rooms with a start state S at one end and a T-junction at the opposite end (figure 3b). Starting in S, the objective is to travel down to the end of the corridor and go either north or south at the T-junction depending on a signal X received in S indicating the location of the goal G. In order to chose the correct direction at the junction, the network must remember X for at least n time-steps.

The agent always sees a binary vector of length three. At the start, the observation is either 011 if the goal is to the north, or 110 if it is to the south. In the corridor, the agent sees 101 and at the junction it sees 010. The agent RNN has three output units, one for each of the possible actions (go east, north or south), where the action corresponding to the unit with the highest activation is taken at each time-step. The agent receives a reward of -0.1 if tries to go north or south in the corridor, or go east or in wrong direction at the T-junction, and a reward of 4.0 if it achieves the goal.

All agents are initially evaluated on a corridor of length 5. If the agents achieve the goal, they are also evaluated in corridors of length 50 and 1000 in order to test the generalization ability.

Table 2 shows the results. A four neuron RNN described with 21 bits can achieve the goal in a corridor of length of 5. A network with seven neurons described with 19 bits can find the goal in a maze of any length (the outputs of the network become stable while in the corridor and the input pattern at the end causes a recall of the goal position stored in the network activation). The 7-neuron network was found before the 4-neuron network because it requires fewer bits and coefficients.

Ball Throwing

In the ball throwing task (figure 3c), the goal is to swing a one-joint artificial arm by applying a torque to the joint, and then releasing the ball such that it is thrown as far as possible. The arm-ball dynamical system is described by:

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

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 ($T_{\text{max}} = [-5\text{Nm}, 5\text{Nm}]$). 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 Algorithm 1 with two optimal control strategies. The first

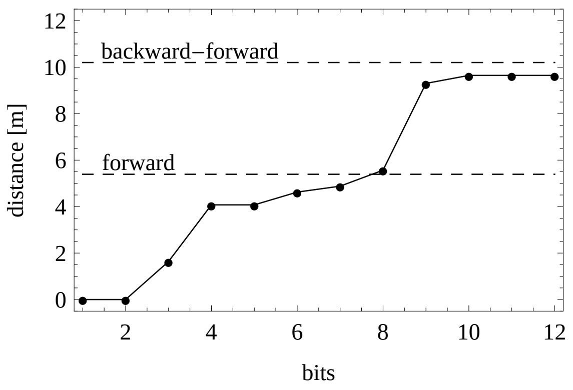


Figure 4: **Ball throwing experiment**. The figure plots the distance reached by the thrown ball against the number of bits, b, used to encode the corresponding two-neuron FRNN weights. Each datapoint denotes the best solution for a given b. The network with weights described with four bits already swings the arm forward and releases the ball with near optimal timing. The network described with eight bits surpasses the optimal forward swing strategy by using a slight backward swing. Nine bits produce a network which swings backward and forward and releases the ball at nearly optimal time. The distances for the two optimal strategies are marked with dashed lines.

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 results are summarized in Table 3. In 70 evaluations, UNS finds a network with four single-bit coefficients, described by a total of 11 bits, that can throw the ball within almost meter of the optimal forward-swing distance. A more complex 17-bit network is found at evaluation 2516 that uses a slight backward-forward strategy to cross the 5.391m boundary. And finally after 5804 evaluations, a 20-bit network is found that implements a nearly optimal backward-forward swing strategy. Figure 4 shows graphically how the performance progresses as the number of bits, b, representing the coefficient values is increased.

Discussion and Future Directions

The experimental results revealed that, using our approach, the solution networks to some widely used control learning benchmarks are actually quite simple, requiring very short descriptions. However, the question remains whether or not the compressed representation improves search efficiency? In order to quantify the advantage gained by searching for weights indirectly in coefficient space, we compared the performance of random search in weight space against random

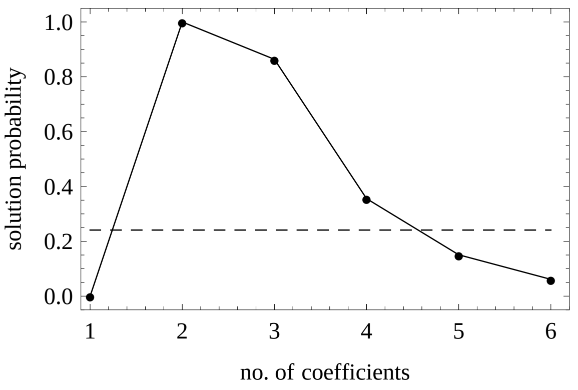


Figure 5: **Coefficient search vs. direct weight search.** The curve shows the probability of finding a solution to the Markov single-pole balancing task within 100 random samples of coefficient space, defined by different numbers of coefficients (calculated of over 1000 runs). The horizontal dashed line at 0.24 indicates the probability of finding a solution by sampling the 6-dimensional weight space directly. Searching for coefficients is more reliable searching weights, for this task, when the number of coefficients is less that five.

search in DCT coefficient space.

Figure 5 shows the results for this comparison in the Markovian single-pole balancing task. Each data-point denotes the probability of finding a successful six-weight neural network (the same architecture that solved the task in Table 1) within 100 random samples, for each number of coefficients. The dashed horizontal line indicates the probability ($p = 0.24$) of finding such a network by randomly sampling the six-dimensional weight space directly. A network represented by just one coefficient is too simple (all weights are equal), and cannot solve the task ($p = 0$). For two, and three coefficients, the task is solved very reliably ($p > 0.9$). As the dimensionality of the coefficient space approaches that of the weights, most of the sampled weight matrices are unnecessarily complex, and, consequently, the probability of finding a solution at random declines rapidly, and falls below the baseline for five and six coefficients (i.e. no compression). This result shows that, on this particular task, just searching the frequency domain without compression only makes the problem harder. It remains to be seen whether compression has a similar profile for all problems, such that there is a *sweet-spot* in a number of coefficients necessary to represent a successful network. However, is seems plausible that, just as with natural signals (e.g. images, video, sound, etc.) most of the energy in useful weight matrices is concentrated in the low frequencies.

In these preliminary experiments, we have focused on benchmark problems for which small network solutions are known to be sufficient. And we have made the implicit assumption that such solutions will have spatially correlated weights. It is possible that, for each task examined here, there exists a permutation in the weight ordering for which the only solutions are those with spatially uncorrelated weights, i.e. requiring the full set of coefficients. However, we have made no attempt to predefine amenable weight orderings, and, ultimately, the potential of this approach lies in providing compact representations for large networks, such as those required for vision, where many thousands of inputs have a natural, highly correlated ordering.

In the current implementation, input, recurrent, and bias weights are all combined in a single matrix. For networks with more layers, it may be desirable to specify a separate set of coefficients for each layer, so that the complexity of each matrix can be controlled independently. Also, the way that bits are currently allocated to each coefficient may be too restrictive. A better approach might be to search all partitionings of the bit-string b, instead of roughly according to (b mod c), such that the precision of each coefficient is less uniform. For example, fewer bits could be assigned to the lowest frequencies, thereby freeing up more bits for the higher frequencies where more resolution may be needed.

The Universal Network Search algorithm was motivated, in part, by the goal of measuring the complexity of well-known test problems by finding minimal solutions, and made possible because of the small number of bits required to encode the DCT representation. While there is a practical limit on number of bits that can be searched exhaustively (e.g. 32), any, more scalable, optimization method can be applied to search larger numbers of coefficients. Immediate future work will use the indirect DCT network representation in conjunction with evolutionary methods to grow large-scale networks vision-capable robots.

Acknowledgments

The research was supported by the STIFF EU Project (FP7-ICT-231576) and partially by the Humanobs EU Project (FP7-ICT-231453). The authors would like to thank Tom Schaul for extensive consultations.

References

[BKS09] Zdeněk Buk, Jan Koutník, and Miroslav Šnorek. NEAT in HyperNEAT substituted with genetic programming. In *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, 2009.

[DDWA91] S. Dominic, R. Das, D. Whitley, and C. Anderson. Genetic reinforcement learning for neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), pages 71–76. Piscataway, NJ: IEEE, 1991.

[Gru92] Frederic Gruau. Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Superieure de Lyon, Institut IMAG, Lyon, France, 1992.

[GS07] Jason Gauci and Kenneth 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.

[Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.

[Lev73] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[LV97] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition)*. Springer, 1997.

[Rad93] Nicholas J. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1(1):67–90, 1993.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Sch95] J. Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML)*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[Sch97] J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

[Sch02] J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, Sydney, Australia, 2002.

[Sol64] R. J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.

[SS10] Tom Schaul and Jürgen Schmidhuber. Towards a practical universal search. In *Submitted to the Third Conference on Artificial General Intelligence*, 2010.

[WB68] C. S. Wallace and D. M. Boulton. An information theoretic measure for classification. *Computer Journal*, 11(2):185–194, 1968.

[Wie91] Alexis 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.