Complexity Search for Compressed Neural Networks

Faustino Gomez
IDSIA
USI-SUPSI
Manno-Lugano, CH
tino@idsia.ch

Jan Koutník
IDSIA
USI-SUPSI
Manno-Lugano, CH
hkou@idsia.ch

Jürgen Schmidhuber
IDSIA
USI-SUPSI
Manno-Lugano, CH
juergen@idsia.ch

ABSTRACT

In this paper, we introduce a method, called Compressed Network Complexity Search (CNCS), for automatically determining the complexity of compressed networks (neural networks encoded indirectly by Fourier-type coefficients) 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. Experiments on a challenging non-linear version of the helicopter hovering task, show that the method consistently finds simple solutions.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Neuroevolution, complexity, indirect encodings, recurrent neural networks

1. INTRODUCTION

In previous work [2], 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 with a high-degree of redundancy, this "compressed" approach can result in an order of magnitude fewer free parameters and significant speedup.

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. In this paper, we introduce a method, 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.

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.
ACM 978-1-4503-1178-6/12/07.

Algorithm 1: Coefficient mapping(g, d)

$j \leftarrow 0$, $K \leftarrow \text{sort}(\text{diag}(d) - \mathbb{I})$
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**
    $ind[j] \leftarrow \underset{e \in s_i}{\text{argmin}} \left\| e - K[l\text{++} \bmod |d|] \right\|$
    $s_i \leftarrow s_i \setminus ind[j\text{++}]$
for $i = 0$ **to** $|ind|$ **do**
  if $i < |g|$ **then**
    coeff_array$[ind[i]] \leftarrow c_i$
  else
    coeff_array$[ind[i]] \leftarrow 0$

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 (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, $C = \sum_{m=1}^{k} |g_m| \ll N$ (N is the number of weights), is user-specified, and the coefficients are distributed evenly over the chromosomes. 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, $g_m = (c_0^m, \ldots, c_\ell^m)$.

Each chromosome is mapped to its coefficient array, according to Algorithm 1, which is then transformed using a D_m-dimensional inverse DCT to generate the weight values that are mapped to their position in the corresponding 2D weight matrix.

3. CNCS

Algorithm 2 describes CNCS in pseudocode. The algorithm is initialized with a prior distribution, $\mathcal{D}$, over the complexity classes (C, Ψ), where C is the number of coefficients used to encode the net, and Ψ is the number of hidden neurons (equivalently, the topology). In order to bias the search toward low-complexity solutions, $\mathcal{D}$ is initialized with a prior that gives high probability to small nets (low Ψ), represented by the fewest number of coefficients (low C).

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

Algorithm 2: CNCS($\mathcal{D}$,s,n,σ_θ, h)

while $\neg$*converged* **do**
  for $k = 1$ **to** s **do**
    $\mathbf{x}_k \backsim \mathcal{D}$ `//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`
  forall the $\mathbf{x}_i \in \mathcal{D}$ **do**
    $$g(\mathbf{x}_i) \leftarrow \begin{cases} \sum_{\forall \mathbf{x}_j \in \mathcal{D}} \phi_{\mathbf{x}_j} \frac{1}{h^d} \mathcal{K}\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right) & \max(\boldsymbol{\sigma}_{\mathbf{x}_i}) > \sigma_\theta \\ 0 & \text{otherwise} \end{cases}$$
  forall the $\mathbf{x}_i \in \mathcal{D}$ **do**
    $p(\mathbf{x}_i) \leftarrow \frac{g(\mathbf{x}_i)}{\sum_{\forall \mathbf{x}_j \in \mathcal{D}} g(\mathbf{x}_j)}$ `//normalize`

the current implementation we use Separable Natural Evolution Strategies (SNES; [4]), an efficient variant in the NES family of black-box optimization algorithms.

Each iteration, CNCS draws s samples from $\mathcal{D}$, and runs the SNES corresponding to each sample for n generations, and then saves its state. The distribution $\mathcal{D}$ is then re-estimated using a multivariate Parzen window estimator with radial-symmetric Gaussian kernel $\mathcal{K}$ [3]. 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 SNES search distributions have converged, or either the desired fitness or the maximum number of iterations has been reached.

4. HELI HOVERING W/ 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. 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 is easy to solve due to the simple wind model. To make the task more challenging, requiring non-linear control, strong "gusts" (20m/s decaying exponentially after striking the heli) were added that buffet the helicopter at random in both x and y directions with probability 0.4.

The helicopters were controlled using simple recurrent networks (SRN/Elman). The decoding scheme for the genomes was $\Omega = \{2, 2, 1, 2, 1\}$, i.e. 5 chromosomes: a 2D input, recurrent and output arrays, and 1D arrays for the input and output bias weights. Each flight lasted 100 time-steps. CNCS was used to search for both the network topology, Ψ (number of neurons), and number of coefficients, C, with $\mathcal{D}$ initialized with a uniform prior over $\{1, 2\}$ for both Ψ and C, and a sample size $s = 2$, $h = 7$, $\sigma_\theta = 0.01, n = 1$, and a SNES population size of 16. A total of 20 experiments were run, for 20 thousand iterations each.

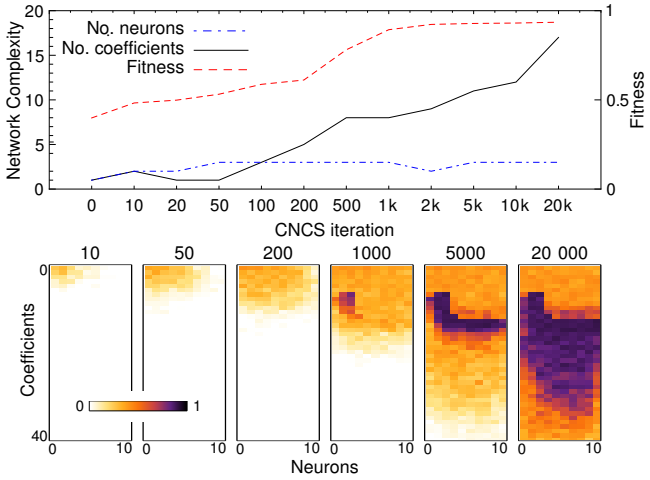

Figure 1: (top) The median fitness, no. of neurons and no. of coefficients of the best network from each run. (bottom) Evolution of fitness over time across complexity classes, averaged across 20 runs.

Figure 1 shows how the fitness of each configuration adapts over the course of 20k iterations of CNCS (average of 20 runs). The prior concentrates on networks with the lowest complexity (upper-left corner of the graphs), gradually expanding to find high fitness individuals with 1 to 3 neurons, using $\approx$8 coefficients, at around iteration 1000 (figure 1). The distribution then focuses on this area, moving away from configurations with fewer coefficients ($C < 8$) as they cannot express the level of complexity required for nets with more that 3 neurons. Then the distribution begins to follow a narrow, high-fitness corridor, adding coefficients to networks with 2 and 3 neurons, until it reaches $C = 12$ ($\approx$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.

CNCS consistently found low-complexity solutions: networks with 2 neurons (64 weights) encoded by 8 DCT coefficients. 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.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

[1] P. Abbeel, V. Ganapathi, and A. Y. Ng. Learning vehicular dynamics, with application to modeling helicopters. In *NIPS*, 2005.

[2] 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.

[3] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):pp. 1065–1076, 1962.

[4] D. Wierstra, T. Schaul, T. G. Y. Sun, and J. Schmidhuber. Natural evolution strategies. Technical report, arXiv:1106.4487v1, 2011.