Neural Networks 28 (2012) 24–39

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Evolving neural fields for problems with large input and output spaces

Benjamin Inden [a,d,*], Yaochu Jin [b], Robert Haschke [c], Helge Ritter [c]

[a] Research Institute for Cognition and Robotics, Bielefeld University, Universitätsstr. 25, 33615 Bielefeld, Germany
[b] Department of Computing, University of Surrey, United Kingdom
[c] Neuroinformatics Group, Bielefeld University, Germany
[d] Artificial Intelligence Group, Bielefeld University, Germany

ARTICLE INFO

Article history:
Received 12 November 2010
Received in revised form 17 November 2011
Accepted 7 January 2012

Keywords:
Neuroevolution
Indirect encoding
Artificial life
NEAT

ABSTRACT

We have developed an extension of the NEAT neuroevolution method, called NEATfields, to solve problems with large input and output spaces. The NEATfields method is a multilevel neuroevolution method using externally specified design patterns. Its networks have three levels of architecture. The highest level is a NEAT-like network of neural fields. The intermediate level is a field of identical subnetworks, called field elements, with a two-dimensional topology. The lowest level is a NEAT-like subnetwork of neurons. The topology and connection weights of these networks are evolved with methods derived from the NEAT method. Evolution is provided with further design patterns to enable information flow between field elements, to dehomogenize neural fields, and to enable detection of local features. We show that the NEATfields method can solve a number of high dimensional pattern recognition and control problems, provide conceptual and empirical comparison with the state of the art HyperNEAT method, and evaluate the benefits of different design patterns.

1. Introduction

1.1. Evolving artificial neural networks

Artificial neural networks are computational models of animal nervous systems and have found a wide range of successful applications, such as system control and image processing. Due to their nonlinear nature it is often difficult to manually design neural networks for a specific task. To this end, evolutionary algorithms have been widely used for automatic design of neural networks (Floreano, Dürr, & Mattiussi, 2008; Yao, 1999). An important advantage of designing neural networks with evolutionary algorithms is that both weights and topology of the neural networks can be optimized. However, if the network topology is changed by evolution, a number of problems can arise that have to be addressed.

One problem is how to add new neurons or connections to a neural network without fully disrupting the function that it already performs. Of course, new elements that are added to the network should change its function to some degree because if there is no change at all, elements without any function could accumulate over the course of evolution, and the networks would become too large. This problem is known as the "bloat" problem in the genetic programming literature (Poli, Langdon, & McPhee, 2008). Another problem is that most evolutionary algorithms use a recombination operator to obtain the benefits that sexual reproduction provides to evolution. Ideally, recombination could combine the good features of two organisms in their offspring. However, it is not obvious what constitutes a "feature" in a neural network with an arbitrary topology, or how corresponding features in two neural networks with different topologies can be found. Similar problems exist for genomes of variable lengths in general. As discussed in the next section, the already well known NEAT method employs techniques that solve these problems satisfactorily. Our NEATfields method makes use of the same techniques.

Another challenge in evolving neural networks is the scalability issue: the evolution of solutions for tasks of a large dimension. This problem has not been addressed by the NEAT method, which typically evolves neural networks of, say, 1–50 neurons. The problem is particularly serious if, like in NEAT, a direct encoding scheme is used for representing the neural network because if every connection weight is directly encoded in the genome, the length of the genome grows linearly with the number of connections. However, the performance of evolutionary algorithms degrades with increasing genome size.

In contrast, indirect encoding of neural networks (Du & Swamy, 2006; Yao, 1999), in which the weights and topologies are

* Corresponding author at: Research Institute for Cognition and Robotics, Bielefeld University, Universitätsstr. 25, 33615 Bielefeld, Germany. Tel.: +49 0 521 106 12107; fax: +49 0 521 106 6011.
E-mail addresses: binden@techfak.uni-bielefeld.de, binden@cor-lab.uni-bielefeld.de (B. Inden), yaochu.jin@surrey.ac.uk (Y. Jin), rhaschke@techfak.uni-bielefeld.de (R. Haschke), helge@techfak.uni-bielefeld.de (H. Ritter).

0893-6080/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2012.01.001

generated using grammatical rewriting rules, grammar trees, or other methods, can achieve a sublinear relationship between the genome size and the network size. These methods basically use a domain specific decompression algorithm in order to make a large phenotype from a small genotype. Typically, the class of encodable phenotypes is biased towards phenotypes that possess some kind of regularity (Lipson, 2004), i.e., some identically or similarly repeated structures. Indeed many neural networks, whether occurring in nature or in technical applications, possess repeated elements. For example, the cerebral cortex is organized into columns of similar structure (Mountcastle, 1997). Brain areas concerned with visual processing contain many modules, in which similar processing of local features is done for different regions of the field of view in parallel. This occurs in brain regions whose spatial arrangement preserves the topology of the input (Bear, Paradiso, & Connors, 2006).

A particular kind of indirect encoding methods apply artificial embryogeny to neuroevolution (Harding & Banzhaf, 2008; Stanley & Miikkulainen, 2003). These methods are mainly inspired by biological mechanisms in morphological and neural development such as cell growth, cell division, and cell migration under the control of genetic regulatory networks. By using abstractions of these processes, large neural networks can be built from small genomes.

Here we explore whether a very different kind of indirect encoding, i.e., a multilevel neuroevolution method using externally specified design patterns, can solve the scalability problem. The next two subsections discuss the NEAT method, on which our recently introduced NEATfields method (Inden, Jin, Haschke, & Ritter, 2010) is based, and present the general approach of NEATfields. In Section 2, the technical details of the method are explained. Sections 3 and 4 present experiments on using NEATfields for problems with large input and output spaces. Section 5 compares NEATfields to some other indirect encoding methods used for neuroevolution, and explains why the method is a good choice for many problem domains.

1.2. The NEAT neuroevolution method and derivatives

The NEAT method (Stanley, 2004; Stanley & Miikkulainen, 2002) is a well known and competitive neuroevolution method that introduces a number of ideas to successfully deal with the problems discussed in the previous section. One idea is to give genes an unchanging identity. This is achieved by assigning a globally unique reference number to each gene once it is generated by mutation. These numbers are used to make recombination of neural networks effective. Similar to recombination in nature, recombination in NEAT starts by aligning genomes such that corresponding genes on the two genomes match. Two genes correspond to each other if they have the same reference number. After the alignment is done, the offspring gets exactly one copy of each gene that is present in both parents. For genes that are present in one parent only, other rules are specified (not necessarily the same in all NEAT implementations) to ensure that the offspring is viable with a high probability.

Another idea introduced by NEAT is to protect innovation that may arise during evolution through a speciation technique. For example, if a network with a larger topology arises by mutation, initially it may not be able to compete against networks with a smaller topology that are at a local optimum of the fitness landscape. By using the globally unique reference numbers again, a distance measure between two genomes can be defined and used to partition the population into species. The number of offspring assigned to a species is proportional to its mean fitness. This rather weak selection pressure prevents a slightly superior species from taking over the whole population, and enables innovative yet currently inferior solutions to survive. In contrast, the selection pressure between members of the same species is much stronger in NEAT. Recombination is usually only allowed to occur within a species, such that parents look rather similar to each other and the offspring looks similar to its parents.

Another feature of NEAT is that evolution starts with the simplest possible network topology and proceeds by complexification, that is by adding neurons and connections. It makes sense to search for solutions with a small topology first because the size of the search space for connection weights increases with the network size. There is a mutation operator that adds neurons only between two connected neurons, and adjusts the connection weights such that the properties of these connections change as little as possible. This alleviates the problem of disrupting the existing function of a network.

Due to the success of the NEAT method, quite a few derivatives have been developed. The goal of these has often been to evolve larger networks than those evolved by NEAT, and/or combine the power of NEAT, which uses a direct encoding of connection weights, with ideas from artificial embryogeny. For example Reisinger, Stanley, and Miikkulainen (2004) used NEAT networks as modules and co-evolved them with blueprints. Blueprints are lists of modules, together with specifications on how to map module inputs and outputs on network inputs and outputs. In the MBEANN method (Ohkura, Yasuda, Kawamatsu, Matsumura, & Ueda, 2007), explicit modularity is introduced into NEAT networks. In the initial network, all neurons are in a first module m_0. A new module is created every time a neuron is added that connects to at least one element in m_0. New connections are established either within a given module or between a given module and m_0. In yet another approach, sets of rules are evolved with NEAT-like speciation that implicitly define a neural network (Reisinger & Miikkulainen, 2007).

In HyperNEAT (D'Ambrosio & Stanley, 2007; Gauci & Stanley, 2007; Stanley, D'Ambrosio, & Gauci, 2009), neurons are embedded into a substrate that has an externally specified topology. For example, the substrate can be a two-dimensional plane. The placement of neurons is determined by the geometry of the given task, as well as some choices made by the user based on previous experience, while the connection weights are generated by giving neuron coordinates as input to another network, which is termed a "compositional pattern producing network" (Stanley, 2007) and evolved according to a slightly extended NEAT method. HyperNEAT networks can be very large and have shown impressive scaling ability. In the NEON method (Inden, 2008), NEAT mutation operators are used as developmental operators, and a gene can encode arbitrary numbers of operations by referring to an external data pool. However, with the exception of HyperNEAT, these derivative methods have not been used widely, nor have they been used to evolve very large neural networks. We will compare HyperNEAT and a few other recent approaches to evolve large neural networks with NEATfields in Section 5.

1.3. NEATfields: goals and approach

Two considerations provide the motivation for the NEATfields method. The first starts with the notion that in order to make a neural network compute a given function, evolution often needs to change connection weights. In a direct encoding, the problem of changing a connection weight has a simple unimodal fitness landscape as far as the mapping from genotype to phenotype is concerned. In indirect encodings, connection weights are usually calculated using an explicitly or implicitly defined nonlinear function (e.g. the compositional pattern producing network in HyperNEAT). This function will often introduce local optima into the fitness landscape of the weight changing problem, and