Neural Random Access Machines Optimized by Differential Evolution

M.Baioletti, V.Belli, G. Di Bari, V.Poggioni

Universit di Perugia, Dip. Matematica e Informatica, Italy
{marco.baioletti,valentina.poggioni}@unipg.it,
{belli.valerio,dbgabri}@gmail,

Abstract. Keywords: NRAM, differential evolution, neural network, TROVARE ALTRO

1 Introduction

Recently a research trend of learning algorithms [3, 4, **?**] has started. The basic idea is to learn regularities in sequences of symbols generated by simple algorithms which can only be learned by models having the capacity to count and to memorize. Most of these are different implementations of the controller-interface abstraction: they use a neural controller as a "processor" and provide different interfaces for input, output and memory. In order to make these models trainable with gradient descent, sometimes the authors made them "artificiously" fully differentiable modifying the continuous nature of their interfaces [3, 5].

In this paper we present a different version of the Neural Random-Access Machines, called NRAM and proposed in [5], where the core neural controller is trained with Differential Evolution meta-heuristic instead of the usual backpropagation algorithm.

The NRAM model is very interesting because it is the first able to solve problems with pointers: it can learn to solve problems that require explicit manipulation and dereferencing of pointers and can learn to solve a number of algorithmic problems. Moreover the authors showed that the solutions can generalize well to inputs longer than ones seen during the training. In particular, for some problems they generalize to inputs of arbitrary length.

However, as the authors themselves said, the optimization problem resulting from the backpropagating through the execution trace of the program is very challenging for standard optimization techniques.

One of the aspect adding complexity to the optimization problem is the continuity demand of optimization function and the fuzzyfication process**verificare** **********

It seems likely that a method that can search in an easier abstract space would be more effective at solving such problems.

***** parte inserita anche nella NRAM section *********

In particular the NRAM model can learn to solve problems that require explicit manipulation and dereferencing of pointers. The controller, the core

part of the NRAM model can be a feedforward neural network or an LSTM, and it is the only trainable part of the model.

****** pezzi che userei qua e là **********

Our work puts the differential evolution meta-heuristic to another test, where the neural networks are deep and with a lot of parameters.

2 Neural Random Access Machines

In this section we briefly recall the NRAM model presented in [5]. This model can be included in the recent research trend of learning algorithms [3, 4, **?**]. Most of these are different implementations of the controller-interface abstraction: they use a neural controller as a "processor" and provide different interfaces for input, output and memory. In order make these models trainable with gradient descent, the authors made them "artificiously" fully differentiable modifying the continuous nature of their interfaces.

In particular the NRAM model can learn to solve problems that require explicit manipulation and dereferencing of pointers. The controller, the core part of the NRAM model can be a feedforward neural network or an LSTM, and it is the only trainable part of the model.

******* A fundamental operation of modern computers is pointer manipulation and dereferencing. In this work, we investigate a model class that we name the Neural Random-Access Machine (NRAM), which is a neural network that has, as primitive operations, the ability to manipulate, store in mem- ory, and dereference pointers into its working memory. By providing our model with dereferencing as a primitive, it becomes possible to train models on problems whose solutions require pointer manipulation and chasing. Although all computationally universal neural networks are equivalent, which means that the NRAM model does not have a representational advantage over other models if they are given a sufficient number of computational steps, in practice, the number of timesteps that a given model has is highly limited, as extremely deep models are very difficult to train. As a result, the models core primitives have a strong effect on the set of functions that can be feasibly learned in practice, similarly to the way in which the choice of a programming language strongly affects the functions that can be implemented with an extremely small amount of code. Finally, the usefulness of computationally-universal neural networks depends entirely on the ability of backpropagation to find good settings of their parameters. Indeed, it is trivial to define the op- timal hypothesis class (Solomonoff, 1964), but the problem of finding the best (or even a good) function in that class is intractable. Our work puts the backpropagation algorithm to another test, where the model is extremely deep and intricate. In our experiments, we evaluate our model on several algorithmic problems whose solutions required pointer manipulation and chasing. These problems include algorithms on a linked-list and a binary tree. While we were able to achieve encouraging results on these problems, we found that standard optimization algorithms struggle with these extremely deep and

nonlinear models. We believe that advances in optimization methods will likely lead to better results.

3 Differential Evolution Neural Networks

We have already presented an algorithm that optimizes artificial neural networks using Differential Evolution in [REF TO MODE]. The evolutionary algorithm is applied according the conventional neuroevolution approach, i.e. to evolve the network weights instead of backpropagation or other optimization methods based on backpropagation. A batch system, similar to that one used in stochastic gradient descent, is adopted to reduce the computation time.

3.1 Differential Evolution

Differential evolution (DE) is a metaheuristics that solves an optimization of a given fitness function f by iteratively improving a population of NP candidate numerical solutions with dimension D. The population evolution proceeds for a certain number of generations or terminates after a given criterion is met.

The initial population can be generated with some strategies, the most used approach is to randomly generate each vector. In each generation, for every population element, a new vector is generated by means of a mutation and a crossover operators. Then, a selection operator is used to choose the vectors in the population for the next generation.

The fist operator used in DE is the *differential mutation*. For each vector x_i in the current generation, called *target vector*, a vector $\bar{y}_i$, called *donor vector*, is obtained as linear combination of some vectors in the population selected according to a given strategy. There exist many variants of the mutation operator (see for instance [2, 1]). The common mutation (called DE/rand/1) is defined as follows:

$$\bar{y}_i = x_a + F(x_b - x_c)$$

where a, b, c are mutually exclusive indexes. The crossover operator creates a new vector y_i, called *trial vector*, by recombining the donor with the corresponding target vector by means of a given procedure. The crossover operator used in this paper is the binomial crossover regulated by a real parameter CR.

Finally, the usual selection operator compares each trial vector y_i with the corresponding target vector x_i and keeps the better of them in the population of the next generation.

3.2 DENN

Since the DE works with continuous values, we can use a straightforward representation based on a one-to-one mapping between the weights of the neural network and individuals in DE population.

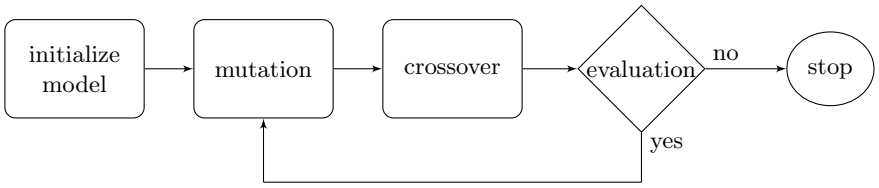

Fig. 1. The evolution of a individual.

In details, suppose we have a feed-forward neural network with k levels, numbered from 0 to $k-1$. Each network level l is defined by a real valued matrix $\mathbf{W}^{(l)}$ representing the connection weights and by the bias vector $\mathbf{b}^{(l)}$.

Then, each population element x_i is described by a sequence

$$\langle(\hat{\mathbf{W}}^{(i,0)}, \mathbf{b}^{(i,0)}), \ldots, (\hat{\mathbf{W}}^{(i,k-1)}, \mathbf{b}^{(i,k-1)})\rangle,$$

where $\hat{\mathbf{W}}^{(i,l)}$ is the vector obtained by linearization of the matrix $\mathbf{W}^{(i,l)}$, for $l = 0, \ldots, k-1$. For a given population element x_i, we denote by $x_i^{(h)}$ its h–th component, for $h = 0, \ldots, 2k-1$, i.e. $x_i^{(h)} = \hat{\mathbf{W}}^{(i,h/2)}$, if h is even, while $x_i^{(h)} = \mathbf{b}^{(i,(h-1)/2)}$ if h is odd. Note that each component $x_i^{(h)}$ of a solution x_i is a vector whose size depends on the number of neurons of the associated levels.

********* TODO *********

4 Experimental Results

5 Conclusions and Future Works

In this paper we have presented a Differential Evolution algorithm for the problem of optimizing neural networks.

References

1. Swagatam Das, Sankha Subhra Mullick, and P.N. Suganthan. Recent advances in differential evolution an updated survey. *Swarm and Evolutionary Computation*, 27:1 – 30, 2016.
2. Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31, 2011.
3. Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.
4. Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*, 2015.
5. K. Kurach, M. Andrychowicz, and I. Sutskever. Neural random-access machines. *CoRR*, abs/1511.06392, 2015.