Computational Intelligence (CI-MAI) EAs exercise

Lluís Belanche

14/12/2022

Training Neural Networks with Evolutionary Algorithms

The application of evolutionary search procedures to artificial neural networks is a long-standing research topic. Literally hundreds of works have been published in the matter, with different approaches and levels of sophistication. The topic is still active, so I give you (as a zipped folder) several classical and recent references if you want to get more inspiration or delve deeper in the subject.

We can distinguish five main kinds of evolution in artificial neural networks:

- evolution of connection weights (their values);
- evolution of architectures (number of layers, neurons per layer and connectivity pattern);
- evolution of hyper-parameters;
- evolution of activation functions;
- evolution of learning rules.

These topics can be carried out simultaneously, although only the first two or three are combined. Finding a suitable way of using evolutionary algorithms for optimizing artificial neural networks is by no means a trivial matter. However, they bring several clear advantages over traditional, derivative-based methods:

- the objective (error) function need not be continuous nor differentiable
- the objective (error) function can incorporate discrete information (like the number of neurons)
- the objective (error) function can be noisy (as is typically the case in machine learning)

On the other hand, the error landscape is complex and noisy since the mapping from an architecture to its performance is indirect and dependent on the evaluation method used. Moreover,

1. The error landscape is deceptive since similar architectures may have quite different performance.[1]
2. The error landscape is multimodal since different architectures may have similar performance.[2]

All these features make the matter very appealing. Arguably the worse drawback is the computational cost.

The exercise consists in creating a full method for training (i.e., *learning*) an artificial neural network (ANN) as an alternate method to backprop-based techniques. To make the exercise feasible for the amount of time given, the following decisions must be adopted:

1. Use a MLP or a RBF network, with a single layer of hidden neurons
2. Use synthetic data (the precise problem is left to you); this allows to be in control of:

- the sample size of the datasets used for training, validation and testing (I suggest varying only the first of these and keep the others constant, to a large number)
- the true generalization error of a model (approximated by its error on a large test set)
- the amount of noise (controlled by a single hyperparameter)

[1] This is caused by the fact that a single neuron can have an important impact on *predictive* performance.

[2] This is caused by the presence of symmetries: exchanging hidden neurons, input weights, etc, which will not affect performance.

- the problem hardness (controlled by a single hyperparameter) [3]

3. Use only one problem and one kind of ANN
4. You can set a maximum number of neurons and weight values
5. You can create either a classification or a regression problem, and then choose a suitable error function
6. Use a single validation set (no need for cross-validation)

Regarding regularization, you can choose one or both commonly used strategies:

- Avoid the use of a regularization parameter (also called weight decay, in the neural network context) and search among a predefined set of numbers of hidden neurons
- Fix a maximum number of hidden neurons (such that the network overfits[4]), and then search search among a predefined set of values for the regularization parameter

In either case, notice that the validation error of a neural network depends on the initial values for the weights, which are typically initialized randomly. In the evolutionary setting, this is not a problem, since the network comes from a population individual.

The (strongly) suggested R packages to be used are: *GA, cmaesr, nnet*. Other packages are up to you. The goal of the work is to compare:

1. Genetic algorithms against derivative methods
2. Evolution Strategies against derivative methods
3. All three

What to report in the comparison?

- execution time (see the *tic* and *toc* functions)
- estimate of true error of the chosen model
- size (number of neurons or decay parameter) of the chosen model

There is no need to use Rmarkdown to produce the document. If you do, have a look at

https://bookdown.org/yihui/rmarkdown/

Delivery date: December 29, 2022

Suggestions:

- keep the generated problem simple; once the whole thing works, you can complicate it
- a crucial part is how to communicate the EA and the ANN: it must be flexible and efficient; make good use of the given packages

[3] Again, the theoretically optimal generalization error can be estimated as the error on a large test set of the non-noisy data generator.

[4] How do we do this? One simple way is to find the simplest network that is clearly overfitting the data.