A DEAP-based Neuroevolution approach to enhance DNN architectures

1 Description

Neuroevolutionary approaches are used to evolve the architecture and/or parameters of neural networks [1, 2, 3, 4]. Usually, the architecture or weights of the network are represented using a genotype. Crossover and mutation operators are then applied to create new NN configurations. One drawback of the neuroevolutionary approach is that each evaluation requires running the NN configuration and estimating the NN performance (e.g., accuracy or regression quality).

Despite the evaluation cost, neuroevolutionary approaches are increasingly applied for automatic configuration of deep network architectures as an alternative to manually setting the parameters [5, 6, 7]. There are available Python implementation of evolutionary algorithms that allow the combination of deep learning approaches with evolutionary algorithms. DEAP is one of these implementations [8].

2 Objectives

The goal of the project is to design and implement, using DEAP, a neuroevolutionary approach that evolves the network architecture of DNNs for improving their performance on a given classification or regression task.

The student should: 1) Select a ML task and an DNN architecture (MLPs can be used). 2) Estimate the the quality of the network for the architecture/parameters manually set. 3) Use DEAP to search for an architecture and/or set of parameters that improve the performance of the original network.

As in other projects, a report should describe the characteristics of the design, implementation, and results. A Jupyter notebook should include calls to the implemented function that illustrate the way it works.

3 Suggestions

- Read the relevant bibliography about neuroevolution for DNNs [5, 6, 7].
- See the help of the DEAP software `http://deap.readthedocs.io/en/master/api/`.
- Start with one of the architectures and/or tasks implemented in the course labs.
- Create a tensorflow class that can serve to evaluate any network configuration evolved by DEAP.
- Implementations can use any Python library that implements DNNs.

References

[1] C. Cotta, E. Alba, R. Sagarna, and P. Larrañaga. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, chapter Adjusting weights in artificial neural networks using evolutionary algorithms, pages 357–374. Kluwer Academic Publishers, Boston/Dordrecht/London, 2002.

[2] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[3] Gene Sher, Kyle Martin, and Damian Dechev. Preliminary results for neuroevolutionary optimization phase order generation for static compilation. In *Proceedings of the 11th Workshop on Optimizations for DSP and Embedded Systems*, pages 33–40. ACM, 2014.

[4] M. O'Halloran, S. Cawley, B. McGinley, R.C. Conceição, F. Morgan, E. Jones, and M. Glavin. Evolving spiking neural network topologies for breast cancer classification in a dielectrically heterogeneous breast. *Progress In Electromagnetics Research*, 25:153–162, 2011.

[5] Travis Desell. Large scale evolution of convolutional neural networks using volunteer computing. *CoRR*, abs/1703.05422, 2017.

[6] Emmanuel Dufourq and Bruce A Bassett. Eden: Evolutionary deep networks for efficient machine learning. *CoRR*, abs/1709.09161, 2017.

[7] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. *CoRR*, abs/1704.00764, 2017.

[8] Félix-Antoine Fortin, De Rainville, Marc-André Gardner Gardner, Marc Parizeau, Christian Gagné, et al. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012.