

## Appendix 3 - Complexity of Multi-Level Systems

This appendix will illustrate that the time complexity of a system grows exponentially with respect to the number of levels it consists of. A level means a new learning algorithm put on top of another to learn the hyperparameters of the algorithm on the level below. E.g. a neuroevolution system contains two levels, where level 1 is neural networks, and level 2 is genetic algorithms.

With only one level, that learning algorithm will only be run once. In practice, a system is usually run more than one time, but per definition, it is assumed that the system as a whole, and therefore also the top level algorithm, is only run once. The time complexity will be:

$$Comp(d = 1) = Comp(a_1)$$

Here,  $Comp$  is a function for the time complexity,  $d$  is the number of levels such a multi-level algorithm contains, and  $a_i$  is the algorithm used on level  $i$ . E.g. with only one level, and this level being a neural network  $nn$ , the complexity is:

$$Comp(d = 1) = Comp(nn)$$

With two levels, the algorithm on level 2 will only be run once, while the algorithm on level 1 will be run many times to create evaluation data for the level 2 algorithm. Generally, the complexity will therefore be:

$$Comp(d = 2) = Comp(a_1) \cdot n_1 + Comp(a_2)$$

$n_i$  is the number of times the algorithm on level  $i$  is run per run of the algorithm on level  $i + 1$ . For a neuroevolution system, with neural networks on level 1 and genetic algorithms on level 2, the complexity becomes:

$$\begin{aligned} Comp(d = 2) &= Comp(nn) \cdot n_1 + Comp(ga) = Comp(nn) \cdot gens \cdot nets + Comp(ga) \\ &\approx Comp(nn) \cdot gens \cdot nets \end{aligned}$$

Here,  $ga$  is the genetic algorithm,  $gens$  the number of generations, and  $nets$  the number of nets created in each generation.  $n_1 = gens \cdot nets$  because this is the number of neural networks which must be trained when running the genetic algorithm once.  $Comp(ga)$  can be ignored, as the time used to run the genetic algorithm itself is much less than the time it takes to train the neural networks.

In the same manner, the complexity of a system with three levels will be as follows:

$$\begin{aligned} \text{Comp}(d = 3) &= (\text{Comp}(a_1) \cdot n_1 + \text{Comp}(a_2)) \cdot n_2 + \text{Comp}(a_3) \\ &= \text{Comp}(a_1) \cdot n_1 \cdot n_2 + \text{Comp}(a_2) \cdot n_2 + \text{Comp}(a_3) \end{aligned}$$

Generally, with  $d = x$ , the complexity is:

$$\text{Comp}(d = x) = \sum_{i=1}^x \text{Comp}(a_i) \cdot \prod_{j=i}^x n_j$$

$n_x$  is defined as 1 to simplify the formula.

If one assumes all  $n_i$  are approximately equal, the formula can be simplified to:

$$\text{Comp}(d = x) = \sum_{i=1}^x \text{Comp}(a_i) \cdot n^{x-i}$$

Furthermore, by assuming the complexity of the different algorithms are approximately equal, only the algorithm on level 1 will determine the overall complexity, as that one is run the most number of times. Then the formula can be simplified further:

$$\text{Comp}(d = x) = \text{Comp}(a_1) \cdot n^{x-1}$$

This shows what this appendix was trying to illustrate; that the time complexity grows exponentially with respect to the number of levels the algorithm consists of (i.e.  $x$ ).