A GPU based distributed algorithm for HyperNEAT

Emad Hosseini *   Ali Kamandi

September 23, 2019

1   Introduction

The advancements in the field AI in the past few decades has led to it being one of the tools of our everyday life. Yet there are some great challenges in developing and training AI systems. There are some tasks that today's AI systems are particularly more capable of doing. That includes image classification [?], natural language processing [?], motor control [?] and many other fields that previously seemed impossible for machines to do and were considered specific to humans and animals.

But what made these traits possible is the advancements in computers and hardware that made faster processors and larger memories and with the help of more data scientists could build working AI software. Yet training each model that is capable of a narrow field of tasks takes thousands of hours of CPU time in huge clusters and on sometimes petabytes of data [?]

Having more data and enough time to process that data is not something to come by easily and in many cases like autonomous navigation in unknown environments or critical decision making for self driving vehicles lack either the data or the time to train conventional neural network models. There are also other challenges including vanishing and exploding gradients [?], optimum network structure and other known issues of back propagation that are known to scientists.

One group of models that tackles these challenges are neuroevolution models. That is defined by Gomez and Miikkulainen in 1999 as "systems that evolve neural networks using genetic algorithms" [?] and genetic algorithm has some features that makes it ideal for such tasks as training a neural network. These features include no assumptions about the search space and its derivatives, high capability of parallel processing and incremental complexity.

Therefor many research is done in actual models that implement neuroevolution including early works that we mentioned before ([?]), probabilistic models like [?], NEAT [?] and many others.

Full utilization of any GA means using its distributable capacities and that is the target of this paper.

In this paper we will introduce a computation method for a specific type of NE model i.e. NEAT using general purpose computers. In the next part we see what has already been done in this field and after that in section 3 our computational method is presented. Some benchmarks are done and results are shown in section 4 with conclusions that follows.

2   Background

Neuroevolution of augmented topologies (NEAT) is one of the most successful models of neuroevolution introduced in 2002 by Stanley and Miikkulainen [?] in this model there is an encoding of the neural network in a genotype that considers an innovation number for each newly formed weight. Also different complexity networks are kept separate using a mechanism of speciation to allow each of them excel in their own rival group and avoid new and unfit individuals get consumed by the older and more mature ones.

A population of such genomes and their respective phenotypes is then created. Using genetic algorithm

---

*University of Tehran, Department of Algorithms and computation.

this population is then directs towards better networks that work better to solve the problem at hand.

NEAT is really successful in finding minimal networks for many tasks that are simple enough but when the requirement of the task is more than that, the search space gets big enough that NEAT is inefficient in finding the best networks. This was addressed in another model called HyperNEAT [?] that uses underlying symmetries in tasks through a substrate that is essentially a raw initial network.

This substrate is then filled with the connections that are themselves products of another smaller network trained through NEAT algorithm. For example an object detection task would consider the rotation of the target object as a symmetry therefore having a circular substrate leads to automatic consideration of the required symmetry in the task.

The nature of GA involves many simple individuals controlled by an environment that produces the survival of the fittest mechanism for a certain goal. This seems an ideal task for a distributed system. And some researches have already exploited this feature.

for example Such et al. compared using a parallel GPU based and distributed CPU based neuroevolution against other methods of training the network like Q-learning (DQN) and policy gradients (A3C). [?]

Also the fact that GA can utilize GPU and run more efficient on a distributed systems is not new and many existing research in this field is gathered by Cheng and Gen in their recent review of the field. [?]

Using the same methods for getting better results in the HyperNEAT is the target of this paper. There are two main steps in distributing the task of any GA based algorithm the first and easy part is distributing the individuals (which is very important in the case of HyperNEAT as explained in section 3) and the next step is to distribute the control unit that is often called the environment. This part involves each of the separate individuals of the population in the task of finding the fittest and crossover the parents to create child genome replacing them with the less fit individuals.

3 Distributed HyperNEAT

There are a few words that are used in this paper that we should clarify their meaning before we delve into the algorithm. Every neural network problem has vector of real numbers as it's *input* and another vector is generated as *output*. In HyperNEAT there is an empty network that consists only of nodes with no connections which is called *substrate*. The nodes in the substrate have coordinates and it is defined by the expert considering the symmetries and other features of the problem space as defined by Stanley et al. [?]

Generally in the literature the genetic algorithm of NEAT is applied on small neural networks with the task of creating best connections for the substrate but in this paper, for the reason which we will discuss later, an *individual* of the genetic algorithm consists of its own copy of the substrate plus, a NEAT genome.

Each individual in this model is capable of generating output based on an input by creating the genotype of NEAT network filling its substrate and running the input vector through the newly generated ANN.

A collection of individuals create a *population* that runs for many *generation*s. Each generation is done by calculating the performance measure for each individual and replacing the unfit ones with the new offsprings of the fitter ones.

*Offspring*s of a generation is the result of *crossover* and *mutation* of existing individuals, the details of which will be addressed later on.

We can summarize the HyperNEAT algorithm in layers of components. This way each execution of the algorithm is one population that contains multiple species, each of them consisting of many individuals. Every individual in turn has a network built up from pre-configured substrate that is filled by connection and weights that are the product of a NEAT phenotype built from its genotype.

Each individual in each cycle has a serries of operations that needs to be addressed.

1: Create NEAT phenotype from its genome

Algorithm 1 Test

Euclid a, b ▷ The g.c.d. of a and b
$r \leftarrow a \bmod b$ $r \neq 0$ ▷ We have the answer if r
is 0 $a \leftarrow b$ $b \leftarrow r$ $r \leftarrow a \bmod b$
return b ▷ The gcd is b

2. Fill substrate with connections based on the output of the phenotype

3. Calculate the output of the network with respective input

4. Compare to expected output, if in training phase

3.1 Thread per Individual

3.2 Environment Distribution

REMEMBER Random instead of innovation number

4 Comparison of Results

5 Conclusion