Procreation in Genetic Artificial Neural Networks

Todo list

Abstract

Neural networks are often used as individuals within populations of genetic algorithms. Many operators exist to transition a population from one generation to the next. The drawback with many of these methods is that only the offspring of a population are selected to form the next generation, resulting in a fraction of the population being bred on – individuals with highest fitness. This causes the individuals to share the same traits and possibly collect around local minima. We develop a new operator for offspring selection that attempts to maximize diversity in a population and hence be more effective at finding the global maximum. It only selects the offspring if it has a higher fitness than either of its parents. If this is the case, both parents are replaced with the individual and a random immigrant individual is introduced. If its fitness is lower, both parents survive.

To evaluate our offspring selection policy, we also develop a method to evaluate diversity of a population consisting of neural networks with outputs that can be classified into species, or bins. This is done by repeatedly giving all individuals in a population the same, random inputs and classifying them by which of their outputs is largest, then evaluating the diversity using Simpson's Diversity Index known from ecology.

Our experiments show how well our selection policy compares to crowding and other methods. . . (we have only just started running experiments)

Abstract is too long. Shorten it.

1. Introduction

Many problems have non-linear search spaces with multiple local maxima. Genetic algorithms can be used to lead a search for global maxima. A genetic algorithm manages a number of individuals, which consitute a population. Each of these individuals is a possible optimal solution to the problem. Individuals possibly offering a better solution to this problem are formed by two individuals in the population combining their traits by breeding, or *procreating*. The intuition is that by combining two individuals with sought-after traits, their offspring will have a better solution to the problem. How well an individual solves a problem is expressed by a fitness value associated with each individual.

Ideally, offspring should contain the best traits from both parents, hence a diverse set of individuals is crucial to identify the combination of traits that work best. Consider a function for which the population must find an optimal solution for. If diversity is not maintained, only a local optimal solution of those traits available in the population will be explored, and a global optimal solution might ever not be found [8].

Diversity measures can be either genotypic or phenotypic. The former concerns different ways individuals are represented, while the latter is concerned with behavioural differences. Phenotypic diversity is often measured based on how fit each individual is. Two individuals with different traits do not necessarily have different fitness values. Consider two artificial intelligent (AI) players for the cell phone game "Snake". Their fitness can be calculated based on how many pieces of food they collect before they die. One of the AI players may have traits that makes it good at avoiding death by not hitting any walls or its own body. Sometimes, by chance, it hits a piece of food. The other AI player may have traits that makes it good at searching for food. The two AI players can have the same fitness value, that is, they collect the same amount of food before they die, and still have completely different traits. Therefore, fitness-based diversity measures do not always catch the difference in traits among individuals.

We develop a new method for measuring phenotypic diversity, which we claim is more suitable for measuring diversity in genetic algorihtms that use neural networks as individuals, compared to fitness-based diversity measures. With “more suitable”, we mean a diversity function that better reflects different traits among the individuals of a population. We call this measure *Neural Network Trait Diversity* (NNTD). Futhermore, we develop a rule for how offspring are replaced to form the next generation of a population, which forces a higher NNTD. We call this replacement policy *Ancestor Elitism Replacement Rule* (AERR).

1.1. Artificial Neural Network

An artificial neural network is a graph structure that, from the outside, can be seen as a black box, that given the values $x_1, x_2, \dots, x_n$ outputs the values $y_1, y_2, \dots, y_m$. With the right internal structure, a neural network can be used for a variety of purposes, e.g. face recognition, where the intensities of different pixels in an image are used as input, and a single output value y_1 is produced, where $y_1 = 1$ if the image was of a face and 0 if not. We now describe the structure and inner workings of neural networks to understand how these output values are calculated based on input values.

1.1.1. Neurons. Nodes in the graph of a neural network are called neurons. Three classes of neurons exist: input, hidden, and output. Input neurons receive $x_1, x_2, \dots, x_n$ and forward the same value. Hidden and output neurons take a number of values as input from edges exiting other neurons, applies a weight to each value, sums them, then applies a function to produce a single output value. The function applied is called the transfer function, and is the same for all hidden and output neurons in the network. This is recursively expressed by

$$y_i = \begin{cases} x_i & \text{if } i \text{ is an input neuron} \\ \theta\left(\sum_{j=1}^{n} w_{ji} y_j\right) & \text{otherwise} \end{cases}$$

where x_i is the input to input neuron i, n is the amount of neurons, y_i is the output value of neuron i, w_{ji} is the weight of the edge from neuron j to i (0 if no connection exists), y_j is the output of neuron j, and θ is the transfer function, usually defined to be the sigmoid function, taking the form

$$\theta(t) = \frac{1}{1+e^{-t}}$$

In a feedforward network, neurons are placed in one or more layers in an acyclic directed graph structure, where each neuron in layer i is connected to every neuron in layer $i+1$. The value output by the last layer, or the output layer, becomes the output of the neural network. Figure 1 illustrates the generic graph structure of a feedforward neural network with a single hidden layer.

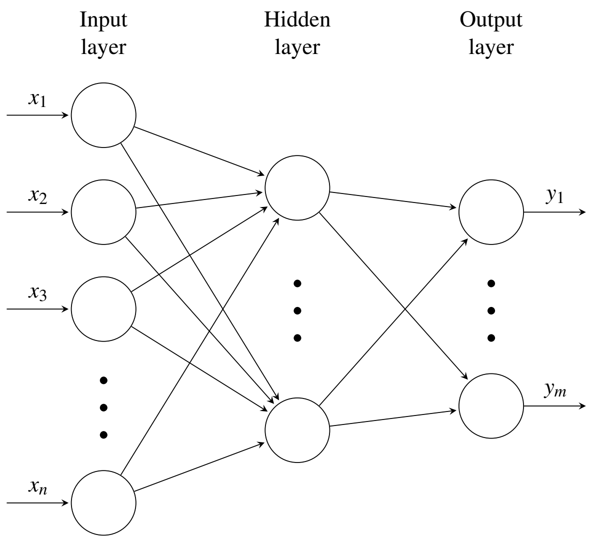


Figure 1: Structure of a neural network.

1.1.2. Training a Neural Network. Any application of a neural network requires a suitable number of layers, neurons, and weights between neurons, to adequately solve a given problem. How many hidden neurons and layers to have is a highly debated subject, see [7].

The weights on edges connecting neurons are decided by a process called training. A well known training algorithm called backpropagation, requires that for each input to the neural network, the correct output is already known [4]. This kind of learning is called supervised learning. For the application of face recognition, this means that the pictures used for training, each has a predicate indicating whether or not it is a picture of a face.

Consider training an artificial intelligent (AI) player for a computer game. Given a state of the game, the desired action for the AI player to take might not be known beforehand, because any action may be either good or bad, depending on the actions that follow. Instead of backpropagation, a genetic algorithm (GA) can be used to construct a desired player. A GA requires that fitness can be measured for each individual. For the application of an AI player, the fitness value of a player indicates how well it performs according to some criteria. This can be measured by simulating the player in the game.

1.2. Genetic algorithms

Genetic algorithms are optimization algorithms which imitate the process of natural selection in search of global maxima.

1.2.1. Individuals and chromosomes. Genetic algorithms (GA) maintain a list of *individuals*, which together form a *population*. Each individual represents a possible solution to the optimization problem and has a fitness value, which denotes how adequately the individual can solve the optimization problem. An individual is encoded by its *chromosome*, which is typically represented by a bit string. Therefore, using a GA requires a way of decoding an individuals chromosome into a solution to the optimization problem.

The population used by a GA typically has a fixed number of individuals, each initialized with a random chromosome when the GA is run. That is, the bit string representing the chromosome is initialized with random bits. As the GA iterates, new individuals are made by combining and modifying chromosomes from existing individuals of the population. Over time, more fit individuals will replace the less fit individuals, using a replacement policy that aims to maximize the fitness of the entire population. After each iteration of the GA, where a new population is formed, we say that we have another *generation* of individuals.

1.2.2. Individuals. Individuals in GAs have a set of traits and behaviors which define each individual. They can take on any form of data structure, as long as they wholly represent a possible solution to the problem.

1.2.3. Neural Networks as Individuals. As we have discussed, neural networks are ideal for decision making and hence are appropriate as individuals in a GA. To support various GA operators, individuals are encoded as bit strings.

We represent each neuron and weight of any given neural network as a single bit string of a fixed length. Each weight is encoded with n bits. Each bit string is concatenated with the next, to form one large bit string containing the entire encoding for the network.

The bit string is constructed in an ordered manner, such that the first n bits represent the weight for the first connection between the first input neuron and the first hidden neuron, the next n bits represent the weight for the connection between the first input neuron to the second hidden neuron, and so forth. It is this large bit string that is manipulated by different GA operators. From now on, we will refer to the bit string encoding of a neural network as its *chromosome*.

If two chromosomes have different bit strings, we say that they have different genotypes. If the neural network they encode produces a different output for some input, we say they have different phenotypes.

1.2.4. Crossovers and mutations. In natural evolution, a pair of individuals come together to produce one or more new child individuals, with genes from both of the parent individuals. The process of procreation is done by performing a *crossover* of the two parent individuals' chromosomes. Parts of each parent's bit strings are used to create the child individual.

Mutations can also occur randomly at any point in time upon creating a child individual. If genes are encoded as bit strings, then a mutation arbitrarily toggles one of the bits. This ensures that the population can evolve if no progress would be made if the chromosomes did not allow it.

1.2.5. Fitness functions. A *fitness function* must be defined to calculate the desirability for each individual. This function is used to define the most fit individuals in a population. The higher the fitness level is of a given individual, the higher the chance it has to be chosen to reproduce with another individual. The intuition behind this is that choosing two fit individuals to crossover will create an even better individual with the best traits of each of its parents.

1.2.6. Crowding. Many methods have been proposed to overcome the problem that diversity decreases through generations – these methods include inserting new random individuals into the population (called random immigrants), using complex population structures to lower the gene flow, and the use of special selection procedures [8]. The latter is known as crowding. How crowding is performed can be described formally by a replacement rule.

1.2.7. Replacement rule. A *replacement rule* determines how a number of individuals that compete against each other are replaced by only a subset of themselves, e.g. when the individuals of the ith generation, G_i, and their offspring, O_i, compete to be selected for the next generation G_{i+1}. A commonly used replacement rule is to make G_i contain the n most fit individuals from $G_{i-1} \cup O_{i-1}$, where $|G_x| = n$ for all x [3]. We will refer to this as the *naive replacement rule*. It is easy to see that this method causes a low diversity over time – since the best individuals are more likely to be selected for procreation, much of the offspring will have similar properties that yields a high fitness value and thus be a part of the next generation just like their parents. Now the chance is even greater that the same properties will be spread out even further in the next generation.

Another well known replacement rule is the probabilistic crowding replacement rule [5]

$$p_x = \frac{f(x)}{f(x)+f(y)}$$

where p_x denotes the chance that individual x replaces the individuals $\{x,y\}$, and $f(i)$ is the fitness of the individual i. This rule can easily be generalized for larger family size.

The probabilistic crowding replacement rule favours the more fit individuals to be selected. A replacement rule can also favour more diverse individuals to be selected, for instance by dictating that for any a individuals that have produced b individuals (their offspring), we only select a of these $a+b$ individuals for the next generation.

1.3. Measuring diversity

In GAs, a high diversity among the individuals is important [1, 10]. It is often argued that the weakness of GAs is the fall in diversity over generations, which can result in premature convergence [2].

Two types of diversity measures are generally used, namely genotypic and phenotypic [6, 9]. *Genotypic diversity* is concerned with how different the encoding of individuals are, while *phenotypic diversity* is concerned with how much the individuals' behavior differ. One drawback of measuring genotypic diversity is that two individuals of different genotypes can still have the same phenotype. For instance, the two different neural networks in fig. 2 will always have identical outputs on the same input, even though their genetic makeup is completely different. Thus a high genotypic diversity does not necessarily imply that individuals with many different properties are represented in the population. When measuring phenotypic diversity, a formula is needed for calculating how different two phenotypes are. Usually, the phenotypic diversity is measured only based on the fitness value of each phenotype [6], which has the advantage of requiring no extra computational power. We see one drawback to this approach however; Two individuals having the same fitness value might have different ways of achieving these, which will not be reflected by the diversity.

In the following, we propose a method for measuring diversity, which tries to overcome the drawbacks of fitness based diversity measures. We also propose a selection policy that aims to increase this diversity measure.

2. Neural Network Trait Diversity

In the following, we will use the term individual and neural network interchangeably, since we represent an individual by a neural network. Let $F = \{f_1, f_2, \ldots, f_n\}$, denote the set of neural networks contained in a population, which all have the same architecture of a input and b output neurons. NNTD is calculated with respect to a number of random inputs $\{R_1, R_2, \ldots, R_m\}$, where each R_i for $1 \leq i \leq m$ is an a-tuple of real values chosen randomly. For each R_i, a Simpson's Diversity Index (SDI) is calculated. SDI is a diversity measure used in ecology to quantify the biodiversity of a habitat. NNTD is calculated as the average SDI for all R_i. To calculate SDI, the total number of neural networks $|F|$ is needed, as well as a distribution of neural networks into a set of species. We classify species of individuals using $S_i(R)$ as the set of neural networks belonging to the ith species with respect to input R.

We distribute the neural networks into species based on which of their output neurons yield the highest value on input R. This means that for any neural network belonging to the species $S_i(R)$, the value of the ith output neuron will be greater than or equal to the value of any other output neuron given the input R. This definition implies that the number of species equals the number of output neurons b. We distribute neural networks into a set of species as follows

$$S_i(R) = \{f_p \mid \forall j \in \{1,2,\ldots,b\}\,(\sigma_{Rpi} \geq \sigma_{Rpj})\}$$

where $\sigma_{xyz} \in \{0,1\}$ is the output of the zth output neuron in neural network f_y on input x.

One disadvantage of NNTD is that it relies on random inputs, which means that fewer random inputs implies less statistical significance.

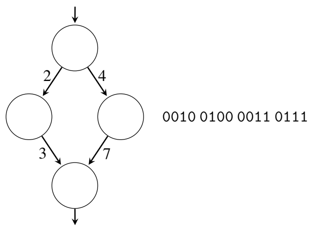
3. A naive replacement rule

In common implementations of GA, the next generation G_i is made by selecting the best next generation by selecting the best individuals from both the existing population and the offspring created. Such a policy leads to homogeneous individuals and thus a low diversity due to the fact that the same, best individuals will with a large probability be selected as parents.

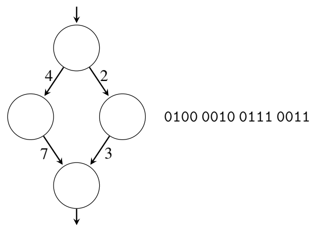
I think we should drop this section title and use the subsection below as the section, then write why we develop this: to combat the problems discussed in 1.3 Measuring diversity –Martin

3.1. Ancestor Elitism Replacement Rule

Given a generation of individuals, G_k, and their offspring, O_k, AERR dictates that the next generation, G_{k+1}, is created as shown in algorithm 1. In the algorithm, $i.\phi$ denotes the fitness of the individual i, $i.p_j$ denotes



(a) An artificial neural network with connections and weights.



(b) An artificial neural network, which is equivalent to fig. 2a.

Figure 2: Networks with same phenotype, but different genotypes. The binary representation assumes that each weight is represented by four bits.

Algorithm 1 Pseudocode for AERR

```
1:  procedure AERR($G_k, O_k$)
2:      $G_{k+1} \leftarrow G_k$
3:      for all $i \in O_k$ do
4:          if $i$.num_parents $= 1$ then
5:              if $i.\phi > i.p_1.\phi$ then
6:                  $G_{k+1} \leftarrow (G_{k+1} \setminus \{i.p_1\}) \cup \{i\}$
7:              end if
8:          else
9:              if $i.\phi > \max(i.p_1.\phi, i.p_2.\phi)$ then
10:                 $G_{k+1} \leftarrow (G_{k+1} \setminus \{i.p_1, i.p_2\}) \cup \{i\} \cup \{\rho\}$
11:             end if
12:         end if
13:     end for
14: end procedure
```

the jth parent of i, and ρ denotes a random immigrant, which is a newly created individual with a random chromosome. Briefly explained, any individual from the offspring that has only a single parent will replace that parent only if it is more fit than its parent. If it has two parents and is more fit than both of them, it will replace them both by itself and a random immigrant.

4. Experiments

In this section, we evaluate our proposed method of a better diversity measurement (NNTD), and compare our Ancestor Elitism Replacement Rule to a standard naive replacement rule as well as the probabilistic crowding replacement rule.

Each of the following experiments use the same configuration of certain variables. Populations consist of 100 individuals, and upon creating a new population, rank-based selection is utilized on the current population p to select individuals for crossover and mutation to create offspring. Half of the offspring is created from crossover between two selected parents in population p, the other half created solely by mutation of the individuals also in p. Of the individuals in p's offspring belonging to the half created by crossover, there is a 10 % chance that the individual also will be selected for mutation. For each bit belonging to the individuals selected for mutation, there is a 5 % chance for that bit to be selected for mutation. Of the bits constructing the individuals selected for mutation, each bit has a 50 % chance it will remain the same, and a 50 % chance to be inverted.

4.1. Diversity measurements

To test our NNTD method, we introduce two variables, both of which only have an effect during the very first generation of a population: initial similarity and initial mutation of a population.

Initial similarity describes how equal the individuals initially are. It can take on a range between 0 and 1, where 0 means that all individuals are random, and 1 denotes that 100 % all individuals have the exact same genotype. With a value of 0.5, one individual is cloned so that half of the population has the same genetic makeup and the other half is initialized to be completely random. A population with the value of 1 for initial similarity should have the lowest diversity, and a completely randomly initialize population (initial similarity of 0) should have the highest diversity, since every individual is randomly chosen.

Initial mutation requires an initial similarity set to 1, e.g. the initial population consists of a single, cloned individual. This is due to the fact that mutating individuals already randomly initialized will not yield a different diversity of individuals. The initial mutation rate simply signifies the mutation percentage of all bits in every individual of the population. A population consisting of the same cloned individual with an initial mutation rate of 1 should have the highest diversity, since every bit in every, initially similar individual, will have a 100 % chance to be selected for mutation, resulting in a randomly initialized population.

Experiments on NNTD using both of these two variables are shown in fig. 3. Each point represents the average diversity measure of 100 runs at a variable intervals of 0.05. As we can see by the chart, diversity increases as expected for both variables. Diversity is maximum at a value of 7.2 for a completely random population. To reach this maximum, it requires an initial mutation rate of a mere 2 % and an initial similarity of 0 %.

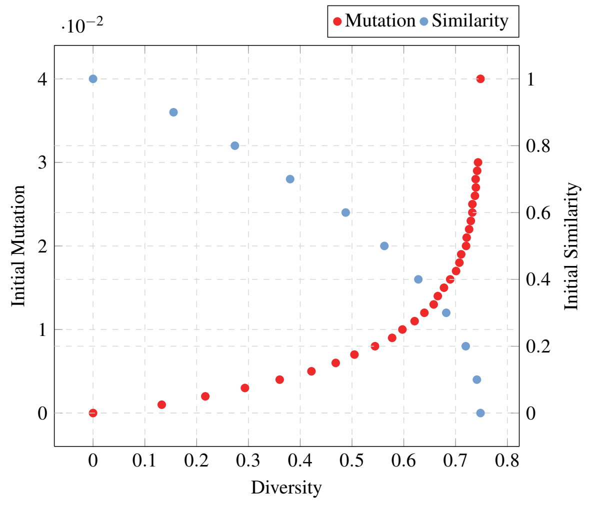


Figure 3: Diversity measurements with NNTD, given increased ranges of initial similarity and initial mutation rates. Each point is the average diversity over 100 runs.

References

[1] E. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: an analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, Feb 2004.

[2] P. A. Diaz-Gomez and D. F. Hougen. Empirical study: Initial population diversity and genetic algorithm performance. In *Artificial Intelligence and Pattern Recognition*, pages 334–341, 2007.

[3] P. Koehn. Combining Genetic Algorithms and Neural Networks: The Encoding Problem. Master's thesis, The University of Tennessee, Knoxville, December 1994.

[4] Y. LeCun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop. In G. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer Berlin Heidelberg, 1998.

[5] O. J. Mengshoel and D. E. Goldberg. Probabilistic crowding: Deterministic crowding with probabilisitic replacement. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, volume I, pages 409–416, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

[6] T. H. Nguyen and X. H. Nguyen. A brief overview of population diversity measures in genetic programming. In T. L. Pham, H. K. Le, and X. H. Nguyen, editors, *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, pages 128–139, Military Technical Academy, Hanoi, VietNam, 2006.

[7] W. S. Sarle. Neural network FAQ. `ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu`, May 2012. Accessed: 2014-03-21.

[8] R. K. Ursem. Diversity-guided evolutionary algorithms. In *Parallel Problem Solving from Nature—PPSN VII*, pages 462–471. Springer, 2002.

[9] K. Zhu. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 176–183, Nov 2003.

[10] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173–195, 2000.