Procreation in Genetic Artificial Neural Networks

Todo list

Abstract

Lorem ipsum...

1. Introduction

Genetic algorithms rely on a number of individuals in a population to procreate to create better, more fit individuals. Offspring should ideally contain the best traits from both parents, hence a diverse set of individuals is crucial to identify the combination of traits that works best. If diversity is not maintained, only the local optimal solution of those traits available in the population will be explored, and a global optimal solution might not be found. [4]

Many existing methods aim to increase diversity require much more computational power [1]. We wish to develop a lightweight method that is computationally inexpensive, yet maintains a high diversity in a population. Common methods for measuring the phenotypic diversity of a population only take into account the fitness of each individual [2]. We develop a method for measuring phenotypic diversity that takes into account the different traits among individuals.

[1] Citation needed here.
[2] Citation needed here.

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 $x_1, x_2, \ldots, x_n$ outputs the values $y_1, y_2, \ldots, y_m$. 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 $x_1, x_2, \ldots, x_n$ and output 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

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

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 f is the transfer function, usually defined to be the sigmoid function taking the form

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

In a feedforward network, neurons are placed in one or more layers in an acyclic 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.

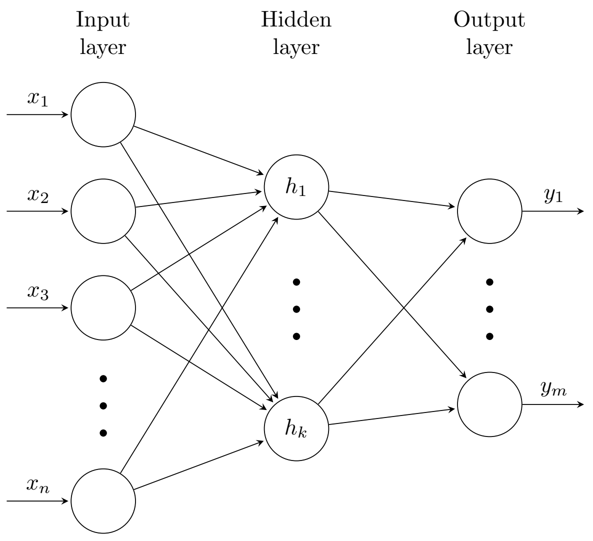

Figure 1. Structure of a neural network.

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 [3].

The weights on edges connecting neurons are decided by a process called training. Well known training algorithms, such as backpropagation, typically require that for each input to the neural network, the correct output is already known. 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.

For some purposes, the desired output of a neural network is not known beforehand. Consider for instance a computer game, where a player is controlled by a neural network. The neural network takes values indicating the state of the game as input, such as the observed position of the players and enemies. For each input, the neural network returns a single value indicating which action the player should take, e.g. move left, move right, or jump. Given a state of the game, we might not be able to say whether an action output by the neural network is right or wrong. It might be wrong to move closer to an enemy if he eliminates you, but it might be right if the next action is successfully to eliminate the enemy.

There are other reasons not to use backpropagation. They might be used as a stronger argument?

From this, it is clear that a backpropagation algorithm is not appropriate for training a neural network to control the actions of an artificial intelligent player in a computer game. For this purpose we propose another approach. We can tell how good a neural network performs, or how fit it is, by simulating a game being played using the neural network to control the player and determining the score achieved in the game, or any other function indicating how well the artificial player performed according to some criteria using that particular neural network. By being able to measure the fitness of a particular neural network, we can use genetic algorithms to create and search for the best performing neural network.

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 replace 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.

We need a subsubsection about different selection policies. –Martin

1.3. Measuring diversity

In GAs, a high diversity among the individuals is important. It is often argued that the weakness of GAs is the fall in diversity over generations, which causes the GA to do a simple local hill climbing [3].

Two types of diversity measures are generally used, namely genotypic and phenotypic [2]. 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 **??** will always have identical outputs on the same input. Thus a high genotypic diversity does not necessarily imply that individuals with many different properties are represented in the population.

Include a picture showing two different neural networks that always produces the same output

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 [2], 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 achieve 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

We propose a new method for measuring phenotypic diversity, which we claim is more suitable for measuring diversity in GAs using neural networks with binary output, 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 will refer to diversity, calculated using our method, by *Neural Network Trait Diversity* (NNTD).

NNTD is calculated based on the neural networks contained by the individuals of a population. In the following, we will use the term individual and neural network interchangeable, since we represent an individual by a neural network. Let $F = \{f_1, f_2, \dots, f_n\}$, denote the neural networks contained in a population, which all have the same architecture of a input and b output neurons. NNTD is based on the inputs $\{R_1, R_2, \dots, R_m\}$, where each R_i is an a-tuple of real values chosen ran-

[3]Citation needed here.

domly. NNTD calculates SDI[4] of all neural networks with respect to R_i, and returns the average SDI. 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. By $S_i(R)$, we denote the set of neural networks belonging to the i'th specie with respect to the input R.

We distribute the neural networks into species based on their output on R. This means, that every neural network in $S_i(R)$ produces the same output on R, which is distinct from the output of any other neural network $f \in S_j(R)$, where $i \neq j$. Since we are only concerned with boolean output values, we have the number of species $k = 2^b$. We distribute neural networks into a set of species as follows:

$$S_i(R) = \{f_j \mid i = 2^0\sigma_{R,j,1} + 2^1\sigma_{R,j,2} + \ldots + 2^{b-1}\sigma_{R,j,b}\}$$

where $\sigma_{x,y,z} \in \{0,1\}$ is the output of the z'th output neuron in neural network f_y on input x.

Radu: Interested readers can easily find the formula for SDI on Google. Should we include the formula here?

One disadvantage of NNTD is that it relies on random inputs, which means that the less inputs we have the more statistical uncertainty we end up with. The more inputs we have the more computational power is required. Consider three neural networks, which given the same input produce the output $\{0,0,0\}$, $\{0,0,1\}$, and $\{1,1,1\}$. All three neural networks belong to indifferent species even though the first two seem to be more similar than the third one, because their outputs are more similar. This kind of similarity is not caught by NNTD. NNTD is also not suitable for neural networks with real valued outputs, because of the infinite number of species this would result in. A possibility is to distribute real valued outputs into a number of buckets, where each bucket represents a range. We will however not look into this and only focus on neural networks with binary outputs.

3. Selection policy

Offspring are produced in each iteration of a GA from a generation of individuals, which combined with the existing individuals, must form the next generation according to a selection policy. In some GA implementations, only the offspring is used for the next generation and all individuals of previous generations are discarded[5]. Other common implementations form the next generation by selecting the best individuals from both the existing population and the offspring created [1]. 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.

3.1. AESP

We propose a computationally inexpensive selection policy that forces a higher diversity, we call this selection policy *Ancestor Elitism Selection Policy* (AESP). Given a generation of individuals G_k and their offspring O_k, AESP creates the next generation, G_{k+1}, as shown in algorithm 1. In the algorithm, $\beta(o)$ denotes the fitness of the individual o, $\pi_i(o)$ denotes the i'th parent of o, and ρ denotes a random immigrant, which is a newly created individual with a random chromosome.

Algorithm 1 Procedure AESP

```
1:  procedure AESP($G_k, O_k$)
2:    $G_{k+1} \leftarrow G_k$
3:    for all $o \in O_k$ do
4:      if $o.\mathtt{num\_parents} = 1$ then
5:        if $\beta(o) > \beta(\pi_1(o))$ then
6:          $G_{k+1} \leftarrow (G_{k+1} \setminus \{\pi_1(o)\}) \cup \{o\}$
7:        end if
8:      else
9:        if $\beta(o) > \mathtt{max}(\beta(\pi_1(o)), \beta(\pi_2(o)))$ then
10:         $G_{k+1} \leftarrow (G_{k+1} \setminus \{\pi_1(o), \pi_2(o)\})$
                $\cup \{o\} \cup \{\rho\}$
11:       end if
12:     end if
13:   end for
14: end procedure
```

Radu: We introduce two new methods. Should we first explain the first method, then test it, explain the last method and test it - or should we introduce both methods and test both methods after?

References

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

[2] 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 Pro-*

[4]Simpson's Diversity Index (SDI) is used in ecology to quantify the biodiversity of a habitat.

[5] Give an example of such a selection policy

gramming, pages 128–139, Military Technical Academy, Hanoi, VietNam, 2006.

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

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