Variable Activation Functions and Spawning in Neuroevolution

Derek Smith
ECCS Department
Ohio Northern University
Ada, Ohio 45810
Email: d-smith.34@onu.edu

Heath LeBlanc
ECCS Department
Ohio Northern University
Ada, Ohio 45810
Email: h-leblanc@onu.edu

Abstract—Neural networks and the Artificial Intelligence (AI) driven by them are beginning to see widespread deployment in applications such as autonomous driving, voice-driven user interfaces, and control of complex systems. With the recent advancements in convolutional neural networks and deep learning (often called deep neural networks due to the many layers present in the convolutional neural network architecture), neural networks research has seen a resurgence. In traditional neural networks, a single activation function – which sets the triggering behavior of the artificial neuron – is fixed for each layer of neurons in the design of the neural network, prior to training. Most commonly, supervised learning is used, where a training algorithm is selected to train the network to a specific data set using a collection of input-output pairings, such as images and their labels. All training algorithms attempt to optimize the neural network to the training set by adjusting the connection weights between the neurons. If no optimum point is reached, a close approximation is selected. This paper proposes the Variable Activation Function Neural Network (VAFNN), an architecture where activation functions are varied on a per-neuron basis. This method may have the potential to model similar behavior as deep neural networks with fewer layers, therefore making the network more efficient. In addition, the proposed architecture enables the possibility of using activation functions that need not be monotonic, continuous, or differentiable. Traditional training algorithms typically require smooth activation functions for training and optimization. Instead of traditional training, a form of neuroevolution is used to vary the weights and activation functions simultaneously. The evolution algorithm only mutates a single individual candidate network at a time, as opposed to a population of networks. While the local minima problem is still an issue, this neuroevolutionary approach uses significantly less memory than the population-based neuroevolutionary approach. Finally, the results of VAFNN are compared to the traditional fixed activation function approach on a two-input XOR network and it is shown that the VAFNN approach uncovers a more efficient implementation than has previously been reported.

Introduction

Artificial neural networks have regained popularity in intelligent systems and artificial intelligence research. This resurgence is due to recent advancements in convolutional neural networks and deep neural networks[1]. These architectures are common for image recognition[2], video processing[3], natural language processing[4], and complex system control[5]. Applications driven by these capabilities range from autonomous driving[6] to medical diagnosis and treatment[7].

All neural networks are based on artificial neurons, or simply neurons. These computational models accept a vector of inputs, multiplied by a set of connection weights. An activation function performs an operation on the sum to create an output that may be treated as an input to another layer of neurons or as an output of the network. The most basic form of neural network is the multi-layer perceptron (MLP)[8], shown in Fig. 1. It is named so because a number of layers (3 or more) of neurons are connected by a set of weighted connections. During runtime, each layer feeds into the subsequent layer. In some architectures, a neuron can have a bias, or value that is added to the input sum before applying the activation function to compute its output[7].

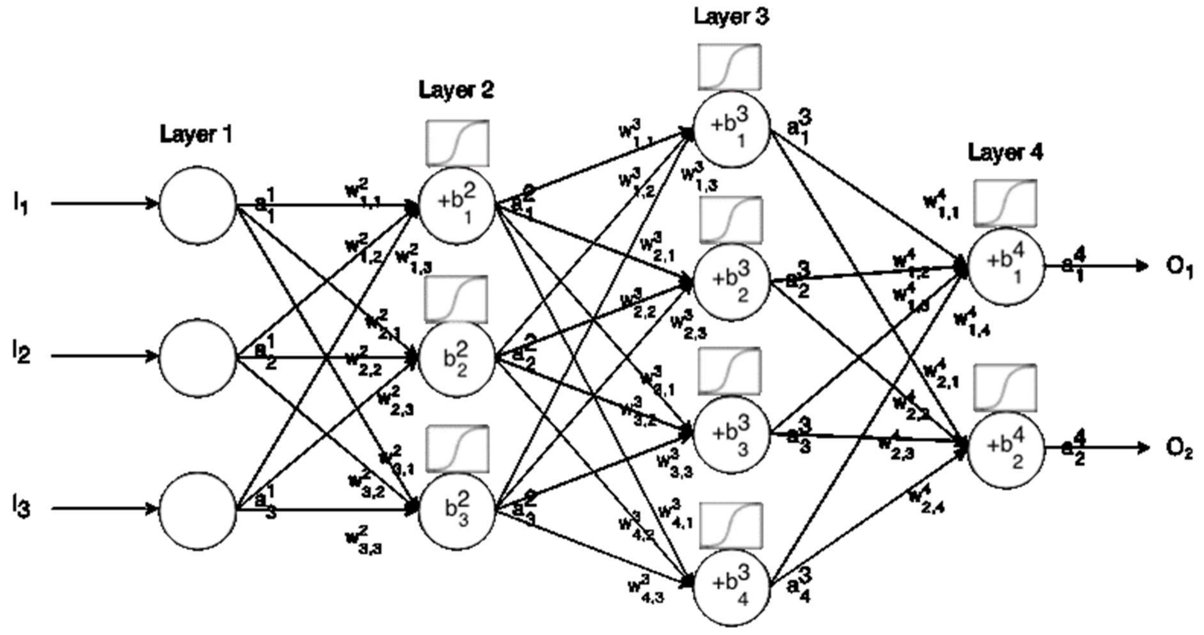

Figure 1: Example Multi-Layer Perceptron Neural Network

Many machine learning problems can be characterized as a form of function approximation problem. An input, such as an image, features of a tumor, or a piece of text, is given as "input". The "target" is produced from that input, and examples of target values may be a classification category, sentiment, and so on. A significant number of these input and target pairings are gathered in a data set that is used for training and testing. The values that define the network are the connection weights and neuron biases, which are often initialized randomly[9]. A training algorithm, such as gradient descent[10], is used to adjust the weights and biases of the network based on how well the inputs are evaluated by the network in comparison to the targets. This is continued until a solution is found, either by the error falling to zero or no improvement halting the process[8]. Some algorithms are susceptible to a problem called local minima, in which a network converges to a sub-optimal solution in the solution space. Various means, such as re-training the weights and biases, using a different algorithm, or applying a different data set, exist to mitigate this problem[11].

Other means of training and optimizing a network exist, such as neuroevolution[12]. In this process, a network is evaluated using the learning data set, and given a fitness value - typically the accuracy of reproducing the target set. In all evolutionary configurations, one network is compared to another, and the most fit of them (according the fitness metric) are used to create new networks for evaluation. Population-based approaches operate on a population of networks bearing the same structure, but have different values for the weights and biases. Networks are evaluated, the least fit are deleted, and new networks are created from the remaining individuals.

This is done, first, by randomly selecting genes (weights and biases) from one of the two parents using a fair Bernoulli process (i.e., a flip of a fair coin). Then secondly, genes of the child are mutated based on a probability of mutation. The process is iteratively repeated in a loop, until a stopping condition is met[13]. The stopping conditions used in neuroevolutionary approaches are like those used in training.

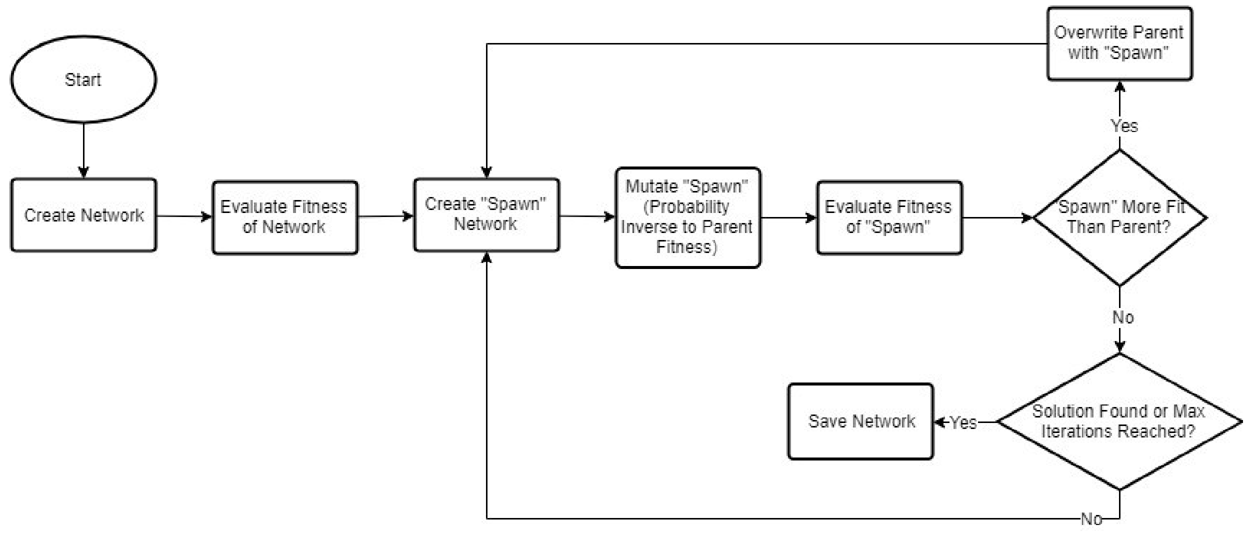
Start
Create Network
Evaluate Fitness of Network
Create "Spawn" Network
Mutate "Spawn" (Probability Inverse to Parent Fitness)
Evaluate Fitness of "Spawn"
"Spawn" More Fit Than Parent?
Yes
Overwrite Parent with "Spawn"
No
Solution Found or Max Iterations Reached?
Yes
Save Network
No

Figure 2: Spawning Neuroevolution Process

Another to neuroevolution is to mutate a single network at a rate inversely related to its fitness, is compared to the most fit "parent," and then replaces the parent only if it is more fit. Such a process is shown in Fig. 2, and is inspired by the Dawkins' weasel[21]. This approach is also susceptible to the local minima problem.

Humans have imposed a limit on the runtime variability of neural networks. A number of transfer functions, or activation functions, exist for neurons. Each has its own advantages and drawbacks[14]. Some networks have different activation functions for each layer[8], and even neuron[15]. However, the different activation functions are declared at design-time and remain static during training or breeding. In contrast, evolution has allowed the altering of the network structure by selectively pruning links[16]. Traditional training algorithms require activation functions to be monotonic and "smooth," (continuous and differentiable)[17].

This paper proposes the Variable Activation Function Neural Network (VAFNN), an architecture where activation functions are varied during evolution on a per-neuron basis. The proposed architecture enables the possibility of using activation functions that need not be monotonic, continuous, or differentiable. Although the number of neurons per layer is fixed at design time, a nullifying function is included in the selection of activation functions to allow for evolution to discover a mathematically equivalent, more efficient, network structure that still solves the given problem.

The VAFNN approach is applied to the approximation of the exclusive-or (XOR) logic gate. This is for two reasons. XOR is not linearly-separable, meaning a graph of its inputs and outputs cannot be separated by a line[8]. Secondly, XOR is a small, standardized and well-explored problem to verify the working order of any neural network architecture. Traditional solutions to the 2-input XOR gate use a network with 2 inputs, 2 hidden neurons, and an output to achieve ideal performance[18]. The VAFNN, in contrast, achieves the ideal solution with a 1 hidden neuron, 1 output solution.

VAFNN and Spawning Neuroevolution Implementation

The implementation of the VAFFN and spawning neuroevolution approach is coded as a neural network evolver implemented in C++. It follows the standard design principals of a multi-layer perceptron network. A vector of input values is given to a first layer of neurons after being multiplied by a vector of weight values. Each per-neuron input is added to a neuron-specific bias value. A node-specific transfer function, or activation function, then produces an output of the neuron. This process is repeated, layer-by-layer, to produce an output vector. However, two design decisions differentiate the resulting process from traditional neural networks.

In traditional neural networks, a training algorithm, such as gradient descent, measures the error produced by the network in comparison to a target, given an input vector. The error is used to mathematically adjust the weight vectors and biases, given the activation function of each layer of neurons. In the proposed approach, a narrow form of neuroevolution, "spawning," is utilized to train the network. Similar to other methods and frameworks, values defining the network (weights and biases) are randomly generated. However, this approach focuses on the fitness evaluated over an entire dataset. The fitness is given below, where $size$ is the number of input-target pairings used in training:

$$fitness = 1 - \frac{\sum_{i=0}^{size} |output_i - target_i|}{size}$$

As the sum of all errors between the desired value and the evaluated value of the network approaches zero, the fitness increases and approaches 1. The spawning algorithm makes a copy of the network and mutates random values of the network. Mutation is a probability of the rate, given by:

$$rate = (1 + minRate) - fitness$$

$minRate$ (selected as 0.09) is the minimum probability that a value of the network will be changed. For a parameter, a random value between 0 and 1 is generated. If this is less than the probability, it the parameter will change. By this equation, as fitness rises, the probability of mutation falls. The child network is evaluated against the parent and overrides it if it has a higher fitness. If not, the child is disregarded, and the process is repeated. If no improvement in fitness is reached after a certain number of iterations, (selected as 200,000) the process stops. The full spawning and neuroevolution process is shown Fig. 2.

A specific network implementation was required to allow for per-neuron variation of the activation functions. The network is defined as a DNA structure made of the following vectors: sizes, weights, biases, and activation functions, where the size is the number of neurons per layer. Sizes are constant, defined upon the initialization of the network. When mutating values, the latter 3 vectors are changed according to the mutation rate, including functions (an enumerated type representative of all selectively-represented activation functions, shown in Table 1). During evolution, a case statement is used to calculate a neuron's output according to its designated activation function.

Table 1: Selection of 22 Activation Functions

Function	**Enumeration**	**Formula**
Sigmoid	1	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic Tangent	2	$f(x) = \frac{2}{1+e^{-2x}} - 1$
Sine	3	$f(x) = \sin(x)$
Cosine	4	$f(x) = \cos(x)$
Tangent	5	$f(x) = \tan(x)$
Cosecant	6	$f(x) = \csc(x) = \frac{1}{\sin(x)}$
Secant	7	$f(x) = \sec(x) = \frac{1}{\cos(x)}$
Cotangent	8	$f(x) = \cot(x) = \frac{1}{\tan(x)}$
Exponential	9	$f(x) = e^{x}$
Step	10	$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$
Inverse Tangent	11	$f(x) = \tan^{-1}(x)$
Inverse Sine	12	$f(x) = \sin^{-1}(x)$
Inverse Cosine	13	$f(x) = \cos^{-1}(x)$
SoftSign	14	$f(x) = \frac{x}{1+\|x\|}$
Rectified Linear Unit	15	$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$
Leaky Rectified Linear Unit	16	$f(x) = \begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}$
SoftPlus	17	$f(x) = \ln(1+e^{x})$
Bent Identity	18	$f(x) = \frac{\sqrt{x+1}-1}{2} + x$
Sinc	19	$f(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin(x)}{x} & x \neq 0 \end{cases}$
Scaled Gaussian	20	$f(x) = e^{-x^{2}}$
Zero	21	$f(x) = 0$
Identity	22	$f(x) = x$

Initialization of a network utilizes a function selection flag. It can be set to “variable,” in which case the activation functions can be mutated, meaning they can be changed to any of the functions listed in Table 1. Or, the flag can be set to any of the specific functions in Table 1, in which case every neuron in the network is set to have that activation function and not to change during evolution. These functions include various mathematical functions and common functions for use in neural networks[9,14,17,19,20]. Also included is a "zero" function, used for potentially identifying nodes in a network that are unnecessary.

VAFNN and Spawning Neuroevolution Applied to the Two Input XOR Problem

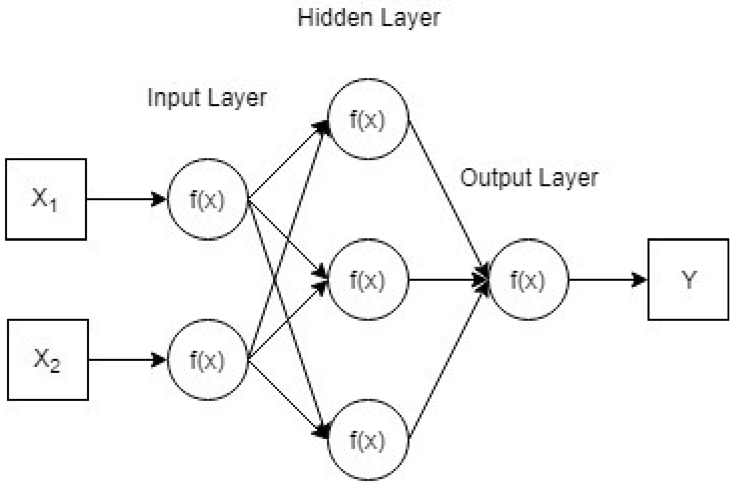

Figure 3: 2-3-1 XOR Architecture

In this section, we study the VAFNN and spawning neuroevolution approach for the 2-input XOR logic gate. The inputs are denoted x_1 and x_2 and may take values in the set $\{0,1\}$, which are the target values t. The data set includes 4 records of the form $(x_1, x_2) = t$: $(0,0) = 0$, $(0,1) = 1$, $(1,0) = 1$ and $(1,1) = 0$. In the first simulation, a standard 2-3-1-layer configuration is chosen, as shown in Fig. 3. Fig. 4 shows fitness statistics over 10 simulations of each function selection flag for the 2-3-1-layer configuration.

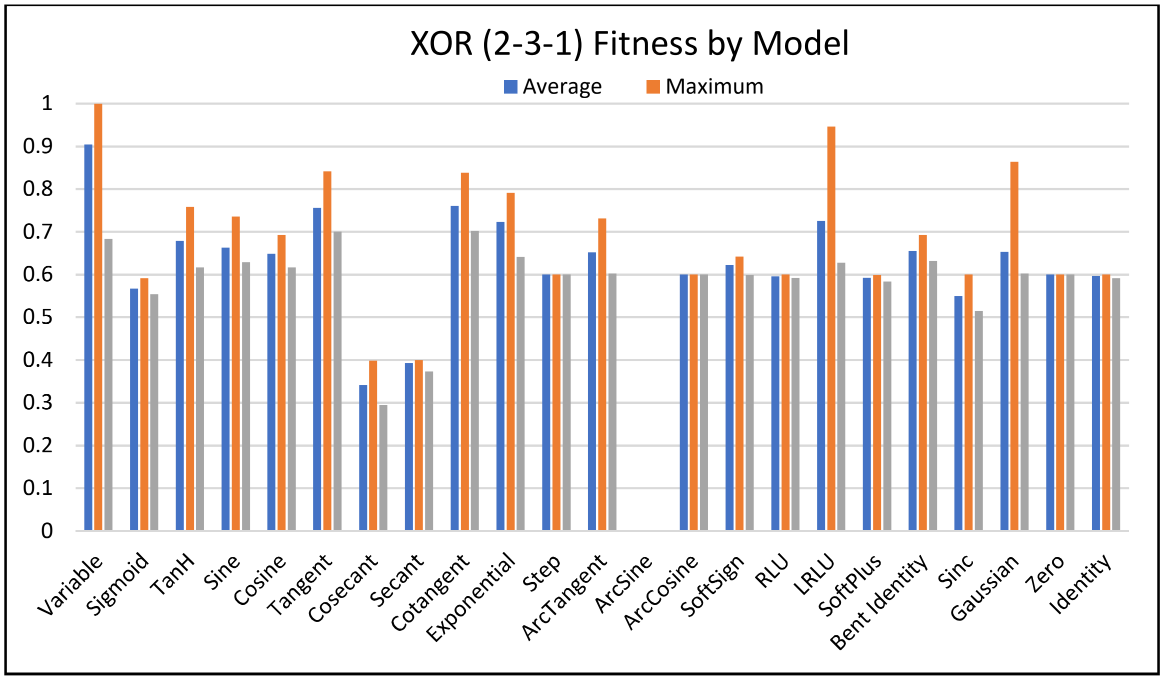

Figure 4: 2-3-1 XOR: Functions and their Fitness

Over the 10 iterations, networks with activation functions varied on a per-neuron basis performed better than all 22 of their fixed-function counterparts. At maximum, variable-function networks produced a fitness of 1, meaning 100% accuracy in function-fitting. This is achieved by utilizing the unit step function in the output neuron. Fig. 5 shows fitness statistics over 10 simulations of each function selection flag for the 2-2-1-layer layer configuration.

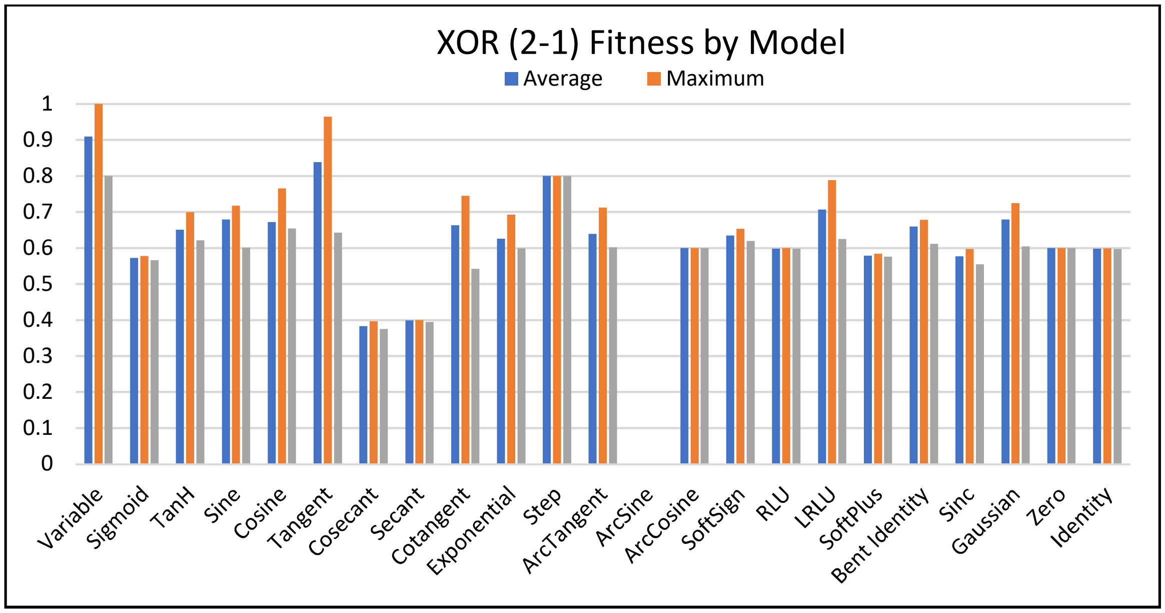

Figure 5: 2-1 XOR: Functions and their Fitness

The 2-2-1-layer configuration clarifies the capabilities of the VAFNN architecture. A few activation functions demonstrate improved fitness, despite the reduced size of the network.

Efficiency considerations notwithstanding, the variable-function network architecture also achieves another demonstrative result. The network with the optimal solution is shown in Fig. 6.

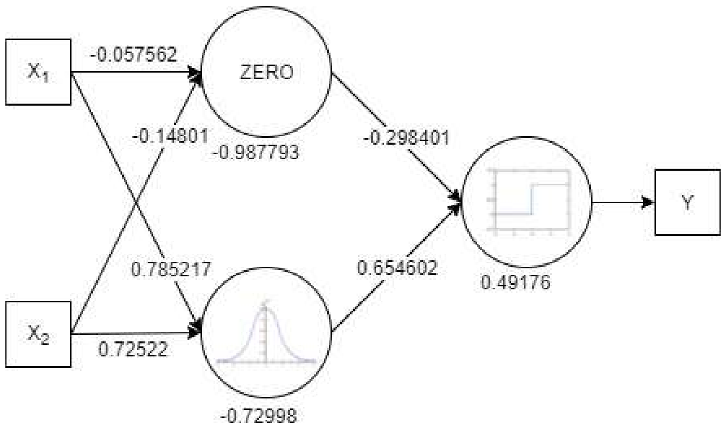

Figure 6: XOR: 100% Fitness Network

The top neuron of the first layer utilizes the “zero” function. This neuron may be removed from the network entirely, resulting in the network in Fig. 7. This network achieves a 100-percent accuracy in implementing the 2-input XOR gate with just 2 neurons: 1 scaled Gaussian, and 1 step function. To the author's knowledge, no traditional neural network architecture has been shown to achieve 100% accuracy with 2 neurons.

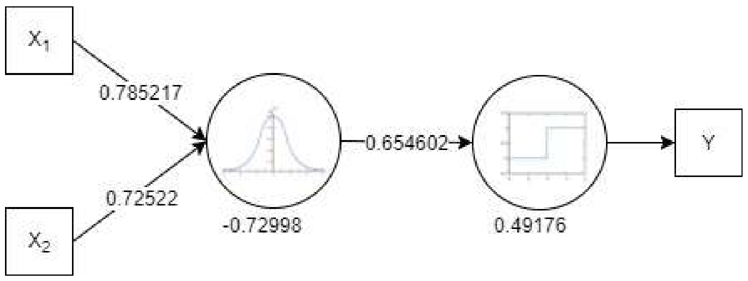

Figure 7: XOR: Simplified 100% Fitness Network

Conclusions

In this paper, a Variable Activation Function Neural Network (VAFNN) architecture is proposed that is trained using a spawning neuroevolutionary approach. Neuroevolution permits the variation of activation functions similar to how connection weights and biases are varied and optimized. This opens the possibility of exploring the use of activation functions that are not continuous, differentiable, or monotonic. VAFNNs can be evolved in a manner that identifies unnecessary nodes and weights that can be removed, reducing the size of networks and increasing efficiency. In addition, the non-linearly separable XOR problem is found to have a 2-node solution consisting of a scaled Gaussian and step function as activation functions. Future work will include the incorporation of new and potentially un-tested activation functions, exploration of larger problems and data sets, and use of richer evolutionary processes.

Bibliography

1. Oh, Kyoung-Su, and Keechul Jung. "GPU implementation of neural networks." *Pattern Recognition,* Vol. 37, No. 6 (2004): 1311-1314.
2. Kheradpisheh, Saeed Reza, Masoud Ghodrati, Mohammad Ganjtabesh, and Timothée Masquelier. "Deep networks can resemble human feed-forward vision in invariant object recognition." *Scientific reports,* Vol. 6 (2016): 32672.
3. Wang, Xuanhan, Lianli Gao, Jingkuan Song, Xiantong Zhen, Nicu Sebe, and Heng Tao Shen. "Deep appearance and motion learning for egocentric activity recognition." *Neurocomputing,* Vol. 275 (2018): 438-447.
4. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature,* Vol. 521, No. 7553 (2015): 436.
5. Wang, Jinjiang, Yulin Ma, Laibin Zhang, Robert X. Gao, and Dazhong Wu. "Deep learning for smart manufacturing: Methods and applications." *Journal of Manufacturing Systems* (2018).
6. Tai, Lei, and Ming Liu. "Deep-learning in mobile robotics-from perception to control systems: A survey on why and why not." *arXiv preprint arXiv:1612.07139* (2016).
7. Yao, Xin, and Yong Liu. "A new evolutionary system for evolving artificial neural networks." *IEEE transactions on neural networks,* Vol. 8, No. 3 (1997): 694-713.
8. Demuth, Howard B., Mark H. Beale, Orlando De Jess, and Martin T. Hagan. *Neural network design*. Martin Hagan, 2014.
9. Sussillo, David, and L. F. Abbott. "Random walk initialization for training very deep feedforward networks." *arXiv preprint arXiv:1412.6558* (2014).
10. Haykin, Simon. "Neural networks: a comprehensive foundation, 1999." *Mc Millan, New Jersey* (2010).
11. Zaki, Mohammed J., Wagner Meira Jr, and Wagner Meira. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.
12. Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." *Evolutionary computation,* Vol. 10, No. 2 (2002): 99-127.
13. Sher, Gene I. *Handbook of neuroevolution through Erlang*. Springer Science & Business Media, 2012.
14. Wu, Huaiqin. "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions." *Information Sciences,* Vol. 179, no. 19 (2009): 3432-3441.
15. Vargas, Danilo Vasconcellos, and Junichi Murata. "Spectrum-diverse neuroevolution with unified neural models." *IEEE transactions on neural networks and learning systems,* Vol. 28, no. 8 (2017): 1759-1773.
16. Ling, S. H., H. K. Lam, Frank HF Leung, and Y. S. Lee. "A genetic algorithm based variable structure Neural Network." In *Industrial Electronics Society, 2003. IECON'03. The 29th Annual Conference of the IEEE*, Vol. 1, pp. 436-441. IEEE, 2003.
17. Snyman, Jan. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Vol. 97. Springer Science & Business Media, 2005.
18. Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." In *Neural networks for perception*, pp. 65-93. 1992.
19. Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of Control, Signals, and Systems (MCSS),* Vol. 5, no. 4 (1992): 455-455.
20. Gashler, Michael S., and Stephen C. Ashmore. "Training deep Fourier neural networks to fit time-series data." In *International Conference on Intelligent Computing*, pp. 48-55. Springer, Cham, 2014.
21. Dawkins, Richard. *The blind watchmaker: Why the evidence of evolution reveals a universe without design*. WW Norton & Company, 1986.