Implementing a Neural Network on an FPGA

Jason Gunthorpe
Darren O'Reilly
Ryan Lewis

April 13, 2000

1 An Introduction to Neural Networks (NN)

1.1 What is a neural network?

As a basic definition, a neural network is an assembly of interconnected processing elements used to represent a real-world system. The processing ability of the neural network is determined by the interconnection weights as "learned" from a set of training patterns. In simpler terms, a neural network is a mathematical model of neurons in a brain that is used to approximate a system output based on a specific input.

Ex #1. A very simple NN could be used to approximate the single variable function: $y = x^2$. The neural network would be trained with a large data set containing specific x and y coordinates. The interconnection weights of the network would be determined from this training. Then, when other values of x are inputted, the network would approximate the output y. This would be within a reasonable amount of accuracy.

Ex #2. Another possible example of a real world application of a neural network is shown in Figure 1. A simplified NN could be used to estimate the purchase price of an automobile based on a variety of inputs describing an automobile's features. Training of this network would be accomplished using previously known values for the inputs and the output price of the automobile.

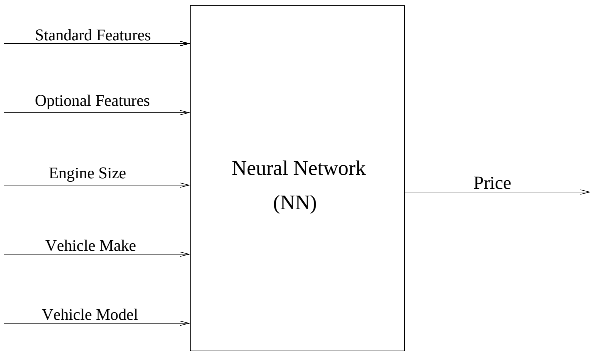


Figure 1: Example of Neural Network

1.2 Why use a neural network?

There are many reasons why a person would choose to implement a system using a neural network rather than traditional control methods.

1. Information storage in a neural network simply requires storing the different values of the weights.
2. A neural network responds well in the presence of noise (i.e. small changes in input won't drastically affect the output).
3. A neural network responds well to hardware failure (i.e. a change in the value of a certain weight will only affect certain outputs, not all of them).

4. A NN has the ability to extrapolate upon what it has 'learned' to predict outputs for unforseen input patterns.

2 Neural Network Design

2.1 Neuron Theory

A neural network is an interconnected web of individual neurons. Each neuron is equipped with a specific activation function which is then used to evaluate the data coming in to it. The data coming into the neuron can either be a specific input, or an output from another neuron. There are many different functions that neurons can be equipped with. Two typical functions will be discussed below. Their purpose is to mimic the average firing of a biological neuron. Figure 2 provides an example of a basic neuron module.

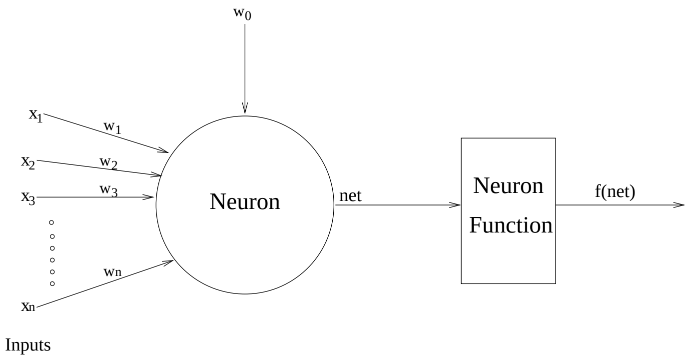


Figure 2: Basic Neuron Module

As mentioned above, each neuron is equipped with a specific function to determine its ouput. Two typical neuron functions are the sigmoid function and the sign function:

1. Sign (sgn) Function:

$$sign(x) = \begin{cases} +1 & if \quad x \geq 0 \\ -1 & if \quad x < 0 \end{cases}$$

2. Sigmoid Function:

$$sigmoid(x) = \frac{1}{1 + e^{-\lambda x}}$$

Graphical representations of the sigmoid function and the sign function can be seen in Figure 3(a) and Figure 3(b) respectively. The sigmoid function will be used in future examples. This function is often used because it has a very convenient derivative property when $\lambda = 1$. That is, if

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

and $\lambda = 1$, the following property is observed:

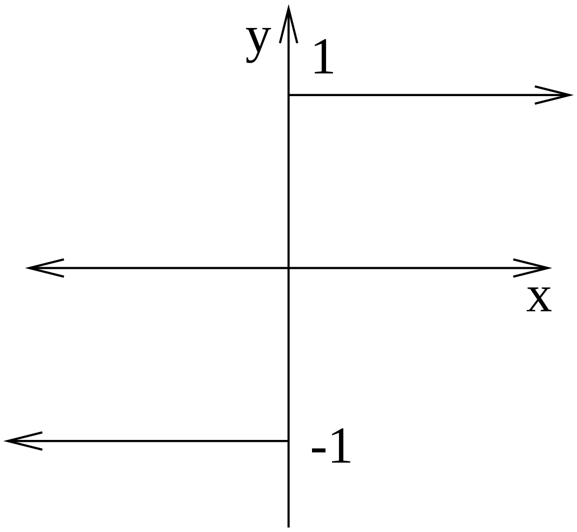
$$\frac{d}{dx}(f(x)) = f(x)[1 - f(x)]$$

For each neuron in the network, the input can be defined as:

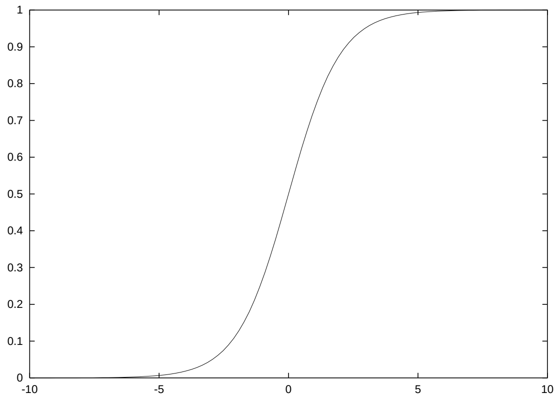
$$input = w_0 + \sum_{i=1}^{n} x_i w_i$$

and the neuron output equals:

$$output = NeuronFunction(input)$$



(a) Sign Function



(b) Sigmoid Function

Figure 3: Plots of Functions

or, with a neuron equipped with a sigmoid function

$$output = sigmoid(input)$$

where the value of *output* will be determined by evaluating the sigmoid function at the input value.

2.2 Network Overlay

A neural network consists of an arrangement of neurons and interconnections between the neurons. Figure 4 shows an example of a neural network (note that the connections for the offsets to each neuron are not shown here). It can be seen in this figure that the layer of neurons whose inputs are the direct network inputs is called the input layer. This layer is sometimes omitted in notation as it merely acts as a buffer stage. Correspondingly, the layer of neurons whose outputs provide the network outputs is called the output layer. All other layers that are not directly connected to the input or output layer are called hidden layers. The notation of hidden layer should not be interpreted to mean that these layers are actually hidden; this is simply a notation used to describe the neuron layers that are not directly connected to the network inputs or outputs. To run a neural network such as this one, simply feed data through the network by calculating neuron outputs for a specific input array until you have calculated all possible neuron outputs.

2.3 How do you design a Neural Network?

Unfortunately, there is no given set of rules governing neural network design. Designing a network which best represents a system is a skill that can only be acquired through practice and experience. However, there are a couple of simple guidelines that one can follow in designing a network to represent a system:

1. Keep the network as simple as possible. Adding unnecessary neurons to the network will not only make things more difficult, but can also reduce the accuracy of the network. The best idea is to start with a very simple network and increase the complexity of the network as it is required. Unfortunately, finding the best representation of a given system will often required multiple network designs.

2. Unless the system you are representing is extremely complex, usually only one hidden layer is required for the network. However, it will probably be necessary to experiment with the number of neurons in the hidden layer to determine which design is the most suitable for the system that is being represented. The number of hidden layers in the network will usually never exceed two.

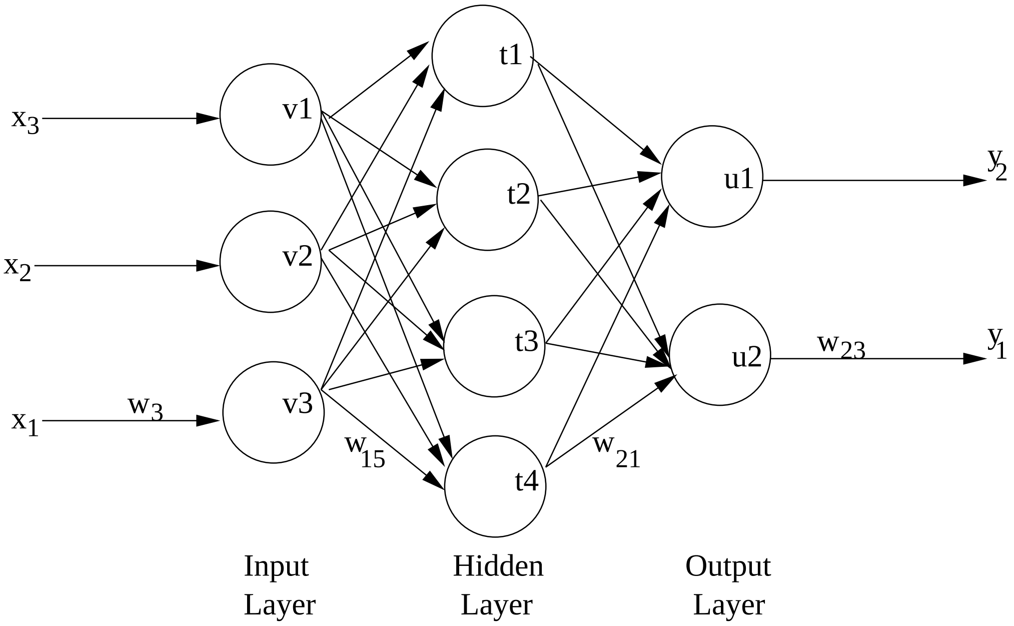


Figure 4: Sample Neural Network

2.4 How do you train a neural network?

The training of a neural network can be executed using a given set of known inputs with the corresponding known outputs. These known outputs are often referred to as target values. There are many different ways in which one can train a neural network, but one of the most common methods is standard gradient based learning using back propagation, which will be discussed below. This method involves using partial derivatives, which is why the simple derivative of the sigmoid function is very convenient for this type of training. It is a iterative process in which each weight is adjusted based on its RMS error of the output of the network vs. the expected network output for a specific input point. For this learning, you will need to adjust the weights after feeding each point into the network. It is common to exceed 10000 epochs or entire data set iterations when training a small network. In standard gradient based learning the following formulas are used:

$$Q = \sum_i \frac{1}{2}(target_i - y_i)^2$$

Q: Quality Index / RMS error (indicates how well the network performs where a lower value of Q indicates a better performance)

target: desired value for the output from the training data

y: output as calculated by the neural network

$$\Delta w_i = -\alpha \frac{\partial Q}{\partial w_i}$$

α: learning rate (indicates how fast the network will converge, and is usually less than one)

$$w_i(new) = w_i(old) + \Delta w_i$$

By iterating through these formulas you can determine the ideal weight configuration for the system. The number of iterations that the training will require will depend upon how fast the network weights converge to some value. The rate of convergence is dependent upon the value set for the learning rate (α), where a high αwill lead to a faster convergence, and a low αwill lead to a slower convergence. While a fast convergence is desirable, the value for αcannot be set too high or the network will diverge instead. On the other hand, setting the αvalue too low will result in a network that takes too long to converge. Usually, it is necessary to experiment with the value of $\alpha$to determine what is optimal for the designed network. Typical values for αrange from about 0.01 to 0.5. The main reason for experimenting with α is due to possible minima trapping which is a prevalent problem when dealing with gradient based learning.

2.4.1 A Simple Sample Network with Calculations

Seen in Figure 4 is an example of a very simple network. To illustrate the idea of network training, sample calculations for the network shown are provided.

$$N_1 = Sigmoid(x_1 \cdot w_3)$$

and

$$N_2 = Sigmoid(x_2 w_2 + N_1 w_3)$$

The neurons have a bias = 0.

$$\Delta w_1 = \frac{\partial Q}{\partial w_1} = \alpha(target - y)\frac{\partial y}{\partial w_1}$$

$$\frac{\partial y}{\partial w_1} = y(1-y)\frac{\partial N_2}{\partial w_1}$$

$$\frac{\partial N_2}{\partial w_1} = N_2(1-N_2)N_1$$

Therefore,

$$\Delta w_1 = \alpha(target - y)y(1-y)N_2(1-N_2)N_1$$

Also (calculations not shown),

$$\Delta w_2 = \alpha(target - y)y(1-y)N_2(1-N_2)x_2$$

$$\Delta w_3 = \alpha(target - y)y(1-y)N_2(1-N_2)w_1 N_1(1-N_1)x1$$

Iterate with these formulas on a given training set to train to a desired error or quality.

2.5 Tips and tricks for Neural Network Implementation

While there is no definite approach to creating a successful neural network, there are a few tips and tricks that can be uses to increase the likelihood of succeeding:

1. A major problem encountered in the training of neural networks occurs when the values for the weights don't converge. Many times, this problem can be overcome by simply experimenting with α(learning rate) until a value is found that works for the network.
2. Another common problem that often occurs is when the network performs very well for the training data, but then performs poorly for the testing data. This could possibly be due to memorization. If the network design is too large for the system that it implements, the network will "memorize" the training data, rather than "learn" from it. This will lead to a poor network performance for the testing data.
3. It is important that the network is not overtrained. An overtrained network could lead to eratic network performance.
4. Remember not to expect perfect error results from the network during testing. While having a perfect error is ideal, it is not practical. In practical situations, strive for an acceptable error.
5. When assigning initial weights to the network before training, randomize them with values between -1 and 1. This will usually result in faster convergence of the network.
6. Since the derivative term of a neuron equipped with a sigmoid function is small, another trick that can improve network performance is to add an offset value of 0.1 to each derivative term in the network.

3 Neural Network Implementation on an FPGA

Implementing a neural network onto an FPGA is a relatively simple process. However, due to the limited space available on an FPGA, some restrictions have to be made.

1. It is not possible to train the neural network on the FPGA. The training will have to be done independently of the FPGA, where the FPGA is used in "dummy" mode. That is, training data is collected on the FPGA but then output to some interface so that the network training can be executed.

2. Once the training is completed and the correct network weights determined (using back propogation), these weights will have to be hard coded onto the FPGA. The accuracy in which these weights can be coded will depend upon the number of bits available to implement the weights. The weights will then have to be scaled to values that can be coded within this restriction.

3. Remember, on an FPGA, each neuron can only effectively be implemented as a "digital" or step function based neuron. You will have to treat it as a unit step function neuron. It will be the same as your standard sigmoid function with its λ set to infinity. That is, the outputs of the neurons will be translated into either a "1" or a "0". The trick is to use BP training to get an ideal weight configuration for a digital style NN and then translate the sigmoid neurons directly into step function neurons.

4. To calculate the neuron outputs on an FPGA, it will be necessary to implement some combination of adders and multipliers. These will be used according to the equations discussed above to determine the neuron outputs. Two possible implementations exist, one using purely combinational logic to compute the entire network in parallel, or one using a small number of adders and multipliers along with a state machine to clock the values through them - similar in design to a CPU.

5. The neural network that is to be implemented on the FPGA will be restricted in complexity depending upon the availability of space on the FPGA. Also, the system that is implemented should have a relatively small number of low resolution inputs and weights with a few digital outputs.

References

[1] ARVP Neural Network Tutorial `http://ugweb.cs.ualberta.ca/~cbarton/arvp/tutorial.ps`

[2] Neural Networks FAQ `ftp://ftp.sas.com/pub/neural`

[3] Pedrycz, W. EE 563 Lecture notes fall 1999, University of Alberta