# A Feed-Forward Neural Net for Parity Checking

Dennis Ideler

December 4th, 2009

Computer Science Department, Brock University
COSC 3P71: Introduction to Artificial Intelligence
di07ty@brocku.ca, 4134466

## Contents



## 1 Introduction

Neural networks are not a new concept; research on them has been going on for some time. For instance, the McCulloch and Pitts neural model dates back

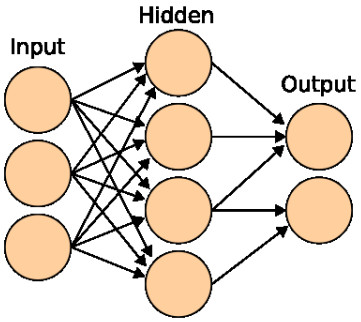


Figure 1: Example of an ANN with one hidden layer.[1]

to the early 1940s and the perceptron[1] was built back in the late 1950s. Unfortunately, interest declined from the late 1960s until the 1980s, due to proofs and demonstrations of the perceptron's limitations and computational weakness. Fortunately for the sake of science, interest was raised again[2] and research in this field continues at a healthy pace. Nowadays neural networks have many uses and are used in many fields, such as physics, psychology, statistics, engineering, econometrics, and computer science to name a few.

The rest of the introduction includes a basic intro to the learning system used and the problem at hand. If you already have a decent understanding of both, you can skip the introduction.

## 1.1 Artificial Neural Networks

### 1.1.1 What Are Artificial Neural Networks?

Artificial neural networks (ANN), or simply neural networks (NN), are adaptive statistical models based on an analogy with the structure of the brain[3]. Artificial neural networks are basically built from simple units called *(artificial) neurons*[4]. These units are interlinked by a set of weighted connections. Learning is usually accomplished by adjusting the weights. The units are organized into layers. A network will usually have several layers, where the first layer is called the *input* layer, and the last one is called the *output* layer. Any intermediate layers are called *hidden* layers (see Fig. 1). The information to be analyzed is fed to the first layer and then proceeds to the next layer until the last layer. The goal of the network is to learn some association between input and output patterns.

[1] A single layer feed-forward neural network invented by Frank Rosenblatt.
[2] See section 1.1.5 on page 4 to find out why.
[3] Biological Neural Nets (BNN) are the natural "equivalent" of the ANN.
[4] Also called *nodes*, *neurodes*, *processing elements (PEs)*, *units*, among many others.

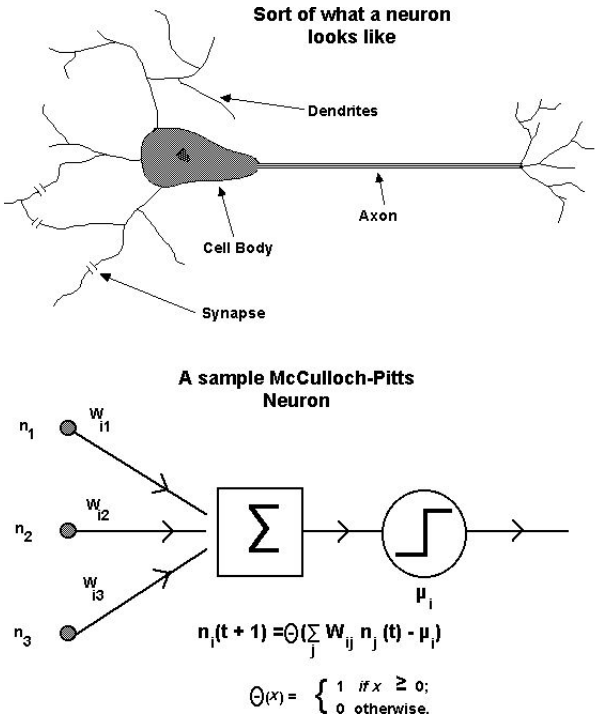


Figure 2: A biological and mathematical model of a neuron.[2]

### 1.1.2 The Artificial Neuron

McCulloch and Pitts (1943) introduced the basic model of a neuron. The neuron basically receives one or more inputs (which are usually weighted) and sums them to produce an output. The sum is then passed through a non-linear transfer function (also known as an activation function).

### 1.1.3 Feed-Forward Neural Networks

Feed-forward networks are the most basic form of ANN. A feed-forward network only contains forward paths[5]. A feed-forward network can be either single-layer (no hidden layers), or multi-layer (there exists at least one hidden layer). This project consists of a multi-layer feed-forward network.

### 1.1.4 Error-Correction Learning

Error-correction learning is used with supervised learning systems. It is the technique of comparing the actual output to the desired or expected output, and using that error to train ANNs by adjusting the weights with help of the

[5] Data only flows in one direction; there are no cycles.

error values. Error-correction learning attempts to minimize this error signal with each training iteration. The most popular learning algorithm (and the one used in this project) is the backpropagation algorithm (see section 1.1.5).

### 1.1.5 Error Backpropagation

The main problem with early neural nets was that they could only deal with linear problems. Researchers knew however that they could overcome this limitation by adding one or more hidden layers. The problem was that there was no way to automatically adjust the weights in the hidden layer in case of errors; there was no learning algorithm. This caused the decline in interest in neural nets. The revival was due to error backpropagation[6].

In short, backpropagation networks consist of:

1. multiple layers of non-linear units
2. computation of an error signal[7] using the rate of change[8] of the non-linear function
3. backpropagation of an error signal
4. estimation of an error signal by the hidden units

Just like a linear unit, the non-linear unit computes its activation by summing all the weighted activations it receives. However, unlike the linear unit, it will create a response by putting this sum through a non-linear transfer function (also known as a sigmoid function or activation function) which is mentioned in step 2 above. If threshold is used, then the threshold value is added to the sum before going through the transfer function. Several transfer functions exist. For an output range of [0, 1], the *logistic function* is used: $f(x) = logist(x) = \frac{1}{1+e^{-x}}$ This is also the function used in this project.

The algorithm for backpropagation can be broken down into the following high-level steps:

1. Initialize the network with small random weights[9]
2. Present an input pattern to the input layer
3. Feed the input pattern forward through the network to calculate its activation value (i.e. generate a forward flow of activation)
4. Take the difference between desired output and the activation value to calculate the network's activation error (i.e. compute the error term)

[6] Backpropagation was around since the 1950s but only gained acceptance in the mid 1980s

[7] A backpropagation network uses supervised learning to calculate the difference between the output and expected output.

[8] That is the derivative.

[9] Note that if *all* the weights are zero, then the error term is zero and that means no learning because the correction of the weights will then be equal to 0.

5. Adjust the weights feeding the output neuron to reduce its activation error for this input pattern
6. Propagate an error value back to each hidden neuron that is proportional to their contribution of the network's activation error
7. Adjust the weights feeding each hidden neuron to reduce their contribution of error for this input pattern
8. Repeat steps 2 to 7 for each input pattern in the input collection
9. Repeat step 8 until desired epochs reached or until the network is trained to satisfaction

### 1.2 Parity Checking

Parity checking is a simple form of error detection. A parity bit is an extra bit that is added to a given set of bits. There are two varients of parity bits: odd parity & even parity. In odd parity, if the number of ones in a given set of bits is *even*, the parity bit is set to 1, which results in the total amount of ones in the bit data set being *odd*. In even parity, if the number of ones in a given set of bits is *odd*, the parity bit is set to 1, which results in the total amount of ones in the bit data set being *even*.

The standard in computer memory is odd parity, however, this project focuses on an even parity bit.

## 2 Network Parameters

Parameter values play a big role in the success of an ANN, and there are many different parameters that must be decided on when designing a neural network. These parameters include (but not limited to):

- the number of layers
- the number of neurons per layer[10]
- the number of training iterations
- the learning rate
- the momentum

### 2.1 Number of Hidden Layers

For most non-linear problems, selecting one hidden layer is sufficient. It is said that two hidden layers are required for modeling data with discontinuities and that there is no theoretical reason to use more than two hidden layers [DTREG]. Adding a hidden layer enlarges the space of hypotheses that the network can represent.

[10] That is, in the input layer, hidden layer(s) if any, and the output layer.

### 2.2 Number of Neurons per Layer

The number of units in the input layer should be adequate to the number of variables of which a pattern consists of. This is because each variable is presented to an input unit.

The number of hidden units represent the number of representations the network can map for every stimuli's input and output pattern association. If too few hidden units are used, the network will be unable to model (complex) data completely, resulting in poor results. If too many hidden units are used, the training time will become very long and the network may *overfit* the data. This is when the network performs really well on trained data, but very poorly on unseen data (i.e. testing data).

The number of output units depends on the classification that the problem outputs. For boolean classification, a neural net will usually have a single output unit, where a result over 0.5 is one classification, and a result below 0.5 is another. A neural net can also have $k$-way classification (thus $k$ classes). This can be done in two ways: one is to divide a single output unit into $k$ portions; the other way is to have $k$ output units, each representing a different classification.

### 2.3 Number of Iterations & Epochs

(This subsection isn't finished yet)

### 2.4 Learning Rate & Momentum

Learning rate is the scaling factor for which all weight adjustments are made. Momentum ensures a higher probability that a consequent weight adjustment will be in the same direction.

## 3 Training

### 3.1 Types of Training

Training configures the neural network to produce a desired set of outputs given a set of inputs. There exist several learning methods to train a neural net:

**Supervised learning** the system is provided with matching outputs for the inputs

**Unsupervised learning** the system develops its own representation of the input stimuli

**Reinforcement learning** the system learns by feedback (e.g. "good" or "bad")

The set of patterns to be learned is presented to the input units (usually in random order) and each pattern is learned in turn. Learning one pattern is called an *iteration*; learning the full set of patterns is called an *epoch*.

| input pattern | expected output |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| ... | ... |
| 1111 | 0 |

Table 1: Sample of training data for 4-bit parity checking.

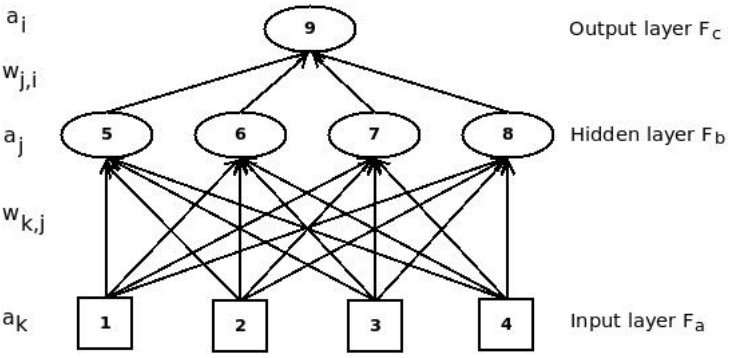


Figure 3: Network architecture used for this problem (supervisor not visible).

### 3.2 A Numerical Example

The network in this problem implements a 4-bit system, checking for even parity (see table 1).

To make the example easy to follow, figures of the network have been included. A three-layer network, made up of $a_k$= 4 input units, $a_j$= 4 hidden units, and $a_i$= 1 output unit (see Fig. 3) will be trained to associate the stimulus $\mathbf{x}$ = [0 0 0 1] to the response $\mathbf{t}$ = [1]. To keep the example simple, no threshold is used. [this section isn't finished yet!]

## 4 Testing

[section not finished]

## 5 Conclusion

[section not finished, but basically, parity bit is a bad problem for NNs because NNs are good for recognizing patterns but a small change in bit data results in a completely different parity bit]

## References


[1] http://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/Artificial_neural_network.svg/Artificial_neural_network.svg.png

[2] http://www.bordalierinstitute.com/images/Neuron.JPG

[DTREG] http://www.dtreg.com/mlfn.htm