# A Look at Artificial Neural Networks

Aaron Cahn
University of Wisconsin-Madison
cahn@cs.wisc.edu

## Abstract

In this study we implement a general artificial neural network framework. Our framework allows us to construct neural networks with arbitrary input, hidden, and output units as well as an arbitrary number of hidden layers. The user is also able to specify the learn rate, number of epochs to train the neural network, and whether to do cross validated training with n folds. This framework allows us to explore the performance and accuracy trade offs of varying the above parameters on different data sets. For this study, we utilize two data sets to motivate the necessity of parameter space exploration. We also compare the performance and accuracy of training a single neural network to that of training an ensemble of neural networks. We show <list some results/generalizations>.

## 1 Introduction

Artificial neural networks were originally inspired by the complex web of neurons that create the human brain. The core of a neural network is the perceptron, which is a simple representation of a neuron consisting of an activation function. The activation function is responsible for producing the output of the perceptron. Perceptrons are linked together in layers to form what is generally known as a multi-layer neural network. Historically, neural networks consist of an input layer, a single hidden layer, and an output layer. All layers are completely connected to the next layer and edge weights from layer $i$ to layer $i+1$ are trained by an algorithm known as back propagation. Because of this, a neural network with a hidden layers can represent arbitrary functions.

When a neural network contains more than a single layer, it is often referred to as a deep network. However, there is debate over whether this criteria alone constitutes a deep network. Generally, training a deep network through backpropagation alone is insufficient because of the large number of local minimum that exist on the complex error surface. Therefore, different techniques, which incrementally train a layer of weights at a time, are needed for deep networks.

Neural networks, and more specifically the edge weights from layer to layer, can easily be represented by matrices. This representation lends itself to fast, parallel computations. As a result, neural networks are extremely popular for tasks that require a complex representation space. Many groups have used neural networks for modeling complex systems, performing image recognition tasks, and performing autonomous tasks such as flying a helicopter. **citation**

In this paper we focus on neural networks with one and two hidden layers. Attempting to utilize more than two hidden layers trained using standard backpropagation does not improve accuracy over the data sets we use. Extending the neural network framework to include a deep training function is left to future work as we were unable to implement it before the deadline.

Our paper attempts to shed light on the trade offs a researcher can make between performance and computation cost. We motivate this by exploring the parameter space of our neural network framework and demonstrating the accuracy under different network configurations. We also explore building an ensemble of neural networks on top of our framework. We make the following contributions: <list contributions>

The rest of this paper is organized as follows. Section 2 explains the design and implementation choices of our framework. Section 3 explains the data sets we use for experimentation and the experiments we ran. Section 4 reports the results from the experiments we ran on our framework. Section 5 concludes.

## 2 Design and Implementation

The artificial neural network was implemented in Python. It consists of a single class which takes arguments for *(i)* the number of input units *(ii)* a list containing the number hidden units to create at each hidden layer (the length of the list defines the number of hidden layers) *(iii)* the number of output units *(iv)* the learn rate and *(v)* the number of epochs to train the neural network for. The neural network edge weights from layer $i$ to layer $i+1$ are represented by a $m-by-n$ matrix, where $m$ is the number of units in layer $i$ and $n$ is the number of units in layer $i+1$. The network activations and errors are represented as single row matrices for computational convenience. As a result, all computations are fast and efficient. Additionally, this design choice makes the neural network implementation very comprehensible. The neural network framework exports an API which allows the user to *(i)* feed a single instance through the network *(ii)* run backpropagation *(iii)* train on a set of data *(iv)* test a set of data and *(v)* a set of functions to print weights, errors, and activations. The latter being useful for debugging from the command line.

## 3 Methodology

TODO: Add the methodology

## 4 Results

Plots we have data for:
1. average accuracy for 1 hidden layer hidden units vary from 1-10 using 10 fold cross validation, epoch = 10
2. average accuracy for 2 hidden layer hidden units vary from 1-10 using 10 fold cross validation, epoch = 100

Figure 1: add caption

Figure 2: add caption

(1 and 2 will be combined into a single table, wbcd data)
3. average accuracy for 1 hidden layer hidden units vary from 1-120, epoch = 1000
4. average accraucy for 2 hidden layer hidden units vary from 1-120, epoch = 1000
(3 and 4 will combine into a single table, face data)

TODO: Add the results
5. Ensemble of neural networks using the best combination of parameters from 1 and 2 (plot varying the number of neural nets)
6. Ensemble of neural networks using the best combination of parameters from 3 and 4 (plot varying the number of neural nets)

## 5 Summary and Conclusions

TODO: Add the summary

## References