# Project #6: Neural Networks

Paul Fornia | JHU 605.649
November 18, 2018

## Abstract

*I explore training feedforward Artificial Neural Networks (ANNs) using the backpropagation (backprop) algorithm. I compare these to a simple baseline, as well as to the Radial Basis Function Network (RBF). I also extend the basic feedforward network to a multi-layer network with one and two hidden layers. The datasets considered look at breast cancer diagnoses, chemical characteristics of glass, physical characteristics of different species of the Iris, diseases among soybean samples, and vote records of several US house representatives. These all come from the University of California Irvine Machine Learning dataset repository. I test the hypotheses that a) both ANN and RBF networks will outperform a simple baseline, b) the best tuned ANN will outperform RBF, and c) for each dataset, for the ANN, more hidden layers will perform better than or equal to fewer layers (up to two) for approximately optimal numbers of nodes in each layer. I reject the last of these hypotheses, finding exceptions to the rule of more layers outperforming fewer. I find evidence to support a popular rule-of-thumb for using feedforward networks: create a single hidden layer with two-third the number of nodes as the input and output layers. This architecture performs the best or nearly the best for all five datasets.*

## Intro and Problem Statement

Artificial Neural Networks (ANN) are models for which input, outputs, and latent variables are stored as "nodes" in a graph, and the interaction between these are stored as edge weights. These models are very powerful, and have dramatically improved predictive accuracy in fields ranging from text processing to image recognition. However, this accuracy comes at a cost, and ANNs are notoriously difficult to train, interpret, and understand.

One of the simpler and more popular ANN schemes is the fully connected feed-forward network. This network can represent latent features as *hidden nodes*, and the weights in the graph can be learned by a method called *backpropagation,* where weights closer to the outputs are learned from predictive errors, and these errors are then extended to earlier weights in the graph. Another popular scheme is the RBF network, which does not require backpropagation.

I implement and test the feedforward and RBF classification ANNs on five datasets. Each of these datasets lends to a two-class or multi-class classification problem. They have a mixture of binary, categorical, and numerical features. First, I fill in any missing values with the most frequently occurring value for that feature. For the categorical variables, I perform one-hot-encoding (creating one binary variable for each possible value). For the numerical features, I

normalize values to fall between -1 and +1. Multi-class classification problems are handled by including one output node per possible class. The final classification is determined by the corresponding output node with the highest confidence. This is conceptually similar to a one-vs-all classification

I test the following three hypotheses:

### Hypothesis 1

For each of the five datasets, both the feedforward ANN and the RBF networks will outperform a naïve baseline (always predicting the most frequent target class).

### Hypothesis 2

For each of the five datasets, there exists a scheme of hidden nodes organized into two or fewer layers for which the feedforward ANN will outperform the RBF network.

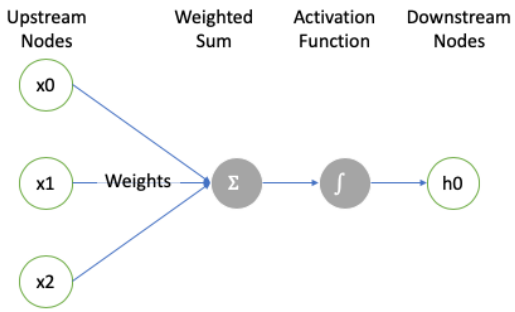
### Hypothesis 3

For each of the five datasets, more layers in the feedforward ANN will outperform fewer (up to two layers). Specifically, there will exist a scheme of nodes in a two-hidden-layer organization that will outperform all one-layer organizations, and there will exist a one-hidden-layer organization that outperforms the model with no hidden layers. For the two-vs-one-layer test, I approximate the "best" organization by trying a reasonable number of diverse schemes, as there are infinite combinations of possible hidden node organizations.

## The Algorithms

### Artificial Neural Nets

Generally speaking, ANNs are any models with nodes and edges describing how information flows between the nodes. Typically, the value of a node contains either a) and input value, b) an output value, or c) a "hidden" latent feature (which may not be fixed). The edge weights between the nodes describe how information flows from one node to the next. In many ANNs, including the feedforward version considered here, a value of a hidden node (or predicted value of an output node) is calculated as the weighted sum of the upstream nodes (where the weights are described by the edge weights), wrapped in an *activation function*.



$$\widehat{h_i} = activation(\sum_j w_{ij} * x_j)$$

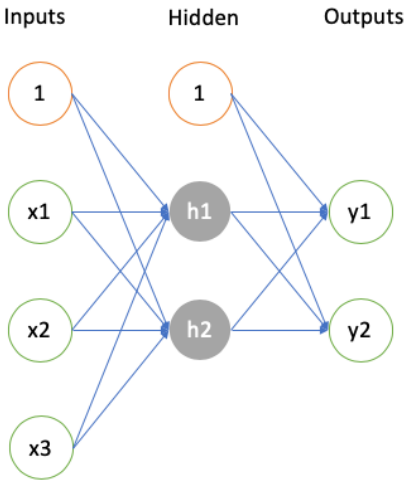
Where $\widehat{h_i}$ is the predicted value of downstream node $h_i$, $x_j$ is an upstream node, and $w_{ij}$ is the directed edge weight from $x_j$ to $h_i$. The variables *h* and *x* are often used to describe hidden and input nodes respectively, but in the above equation, h could also be an output node, and x could also be another upstream hidden node in so-called *deep* networks.

The activation function is any function that converts a real value to a value between 0 and 1 (or sometimes -1 and 1). I use the *logistic* function here:

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

## Feedforward networks and Backpropagation

ANNs can be organized into virtually any shape and form. One common structure is the *fully connected feedforward network*. In this network, all edges are directed "forward" from the input nodes towards the output nodes, and can include intermediary *hidden layers* of nodes, which can be interpreted as latent features. The following shows a basic feedforward network with 3 input features plus fixed *bias* features, one hidden layer, and an output layer with two classes (the summation and activation functions are not shown here for readability):




The use of the weights in forming predictions is relatively straight-forward, but the process of determining the optimal scheme of weight values can be quite complicated! A break-through in the practical use of ANNs (and especially feedforward networks) was the discovery of the *backpropagation* algorithm (Rumelhart, et. al., 1986), in which errors are determined starting at the output layer, and are propagated backwards (left in the diagram) to update weights in layers iteratively towards the input layer.

## Radial Basis Function

Another common ANN architecture is the RBF network. Its layout is similar to a one-layer feedforward network. It can be thought of as 1) K-nearest neighbors, plus 2) gaussian kernel

functions (similar to a gaussian mixture model), plus 3) Logistic regression. First, K-means is used to find k centroids. Then each of these centroids is represented by a hidden node, which is calculated as the sum of the radial basis functions of the inputs. Like the feedforward network, the connection from hidden to output is a weighted average through an activation function (equivalent to logistic regression).

Unlike the feedforward network, training is simpler, faster, and does not require the backpropagation algorithm.

## Experimental Design

For all of these models (and all five datasets), I measure performance based on a simple accuracy measure – number of correct predictions divided by total number of observations. To protect against unbalanced datasets, I compare these metrics to the naïve model of always predicting the most common class.

As always, metrics are calculated on "out-of-sample" test sets, utilizing a 5-fold cross validation approach. For each fold, I use the training data (four-fifths of the data) to train a model. I then use that model to classify the observations of the remaining test data (one-fifth). After all five folds are complete, I calculate an average performance metric across the folds.

### Tuning the size of the hidden layers.

Unlike simpler models, ANNs have the added challenge of a large number of hyper-parameters. When comparing different number of layers, I do *not* want to test different layers with "all else held equal" like a traditional experiment. Instead, I want to find the approximately optimal structure for each experiment. I want to compare the *best one-layer network* against the *best two-layer network*.

I strive to try a diverse range of architectures, without causing the number of experiments to be driven too high. For RBF, I consider 5, 10, and 20 kernels. For the ANN, the following guidance and rules of thumb are considered (Heaton, 2008):

- Input layer contains one node per feature.
- Output layer contains one node per target class.
- Number of hidden nodes should be less than the input nodes, but greater than the output nodes.
- Number of hidden nodes should be 2/3 the input nodes plus the output nodes.

I test the following experiments to search for the optimal architecture.

**No Hidden Layers:**

- No experimentation needed, there is only one traditional structure

**One Hidden Layer:**

- Few nodes: The hidden layer has the same number of nodes as the output layer.
- Moderate: The hidden layer has 2/3 the number of total input plus output nodes, rounded down (hereafter "2/3 i/o nodes").
- Many nodes: The hidden layer has the number of nodes as the input layer.

**Two Hidden Layers*:**

- Very Few Nodes: Both layers have the same number as the output layer.
- Few Nodes: First layer has 2/3 i/o nodes, second layer has the same as outputs.
- Moderate: Both layers have 2/3 i/o nodes.
- Many Nodes: First layer has the same as the inputs, second layer has 2/3 i/o nodes.
- Very Many Nodes: Both layers have the same number of inputs.

*The Glass and Iris datasets have a smaller number of potential combinations since 2/3 i/o nodes is greater than or equal to the number of input nodes.

## Results

| Dataset (inputs nodes/ output nodes) | "Naïve" Baseline | RBF Network (Best K) | ANN no hidden | ANN one hidden (Best Architecture: Hidden Nodes) | ANN two hidden (Best Architecture: Hidden Nodes first, second layers) |
|---|---|---|---|---|---|
| Breast Cancer (9/2) | 65.5% | **96.6% (10)** | **96.6%** | **96.4% (Moderate: 6)** | **96.7% (Very Few: 2/2)** |
| Glass (9/6) | 35.5% | **42.9% (20)** | **49.5%** | **66.4% (Moderate: 9)** | **66.4% (Moderate: 9/9)** |
| Iris (4/3) | 33.3% | **95.3% (10)** | **93.3%** | **96.0% (Few: 3)*** | **96.7% (Very Few: 3/3)*** |
| Soy (72/4) | 36.2% | **67.1% (10)** | **100%** | **100% (Few: 4)*** | **100% (Few: 50/4)** |
| Vote (32/2) | 61.4% | **93.3% (10)** | **96.6%** | **95.9% (Moderate: 22)** | **96.3% (Moderate: 22/22)** |

*Tied with "Moderate" Architecture

- Hypothesis #1 is accepted – There exists RBF and feedforward network architectures that outperform the naïve baseline for all five datasets.
- Hypothesis #2 is confirmed – For all datasets, there exists a feedforward network architecture that outperforms the RBF network.
- Hypothesis #3 is rejected – There exist datasets for which fewer layers are better. For instance, the ANN with no layers outperforms any of the one-layer architectures for the Vote dataset.

## Algorithm Behavior and Conclusions

For all one-layer versions, the "moderate" rule-of-thumb of taking 2/3 the count of input and output nodes either outperformed or tied all other architectures. For the two-layer versions,

there was no such rule-of-thumb; some datasets performed well with this 2/3 rule, while others performed noticeably better with fewer nodes.

Also notable was the performance of different numbers of layers. In theory, all linearly separable decision boundaries can be represented with no hidden layers, all continuous boundaries can be represented with one hidden layer, and all arbitrary decision boundaries can be represented with two layers. In practice this seems consistent with one aspect of the results: Some datasets do much better with one vs zero layers, but there is very little difference between one and two layers. The interpretation here is that most naturally occurring decision boundaries can be very nearly approximated by a continuous function, but some non-linear boundaries (such as the XOR function), cannot be even roughly approximated by a linear one.

Based on the above results, the glass dataset most likely does not exhibit anything resembling a linear decision boundary. All other datasets seem to have decision boundaries that are approximately linear.

In general, while two-layer networks tend to perform best, the accuracy gain over one layer version is generally small, and these are more sensitive to number of hidden nodes. As these are harder to train and slower, it may often be sufficient to train models with only one hidden layer, using the 2/3 rule of thumb.

## Summary

I explore the feedforward ANN as well as the RBF networks by testing classification accuracy on five well-known datasets. Generally speaking, ANN with a single hidden layer containing 2/3 the number of nodes of the input and output layers tend to perform well, thus confirming the guidance put forward by Heaton (2008). ANN tends to outperform RBF, and both tend to outperform the basic naïve baseline model considered.

## References

Alpaydin, Ethem. Introduction to Machine Learning (Adaptive Computation and Machine Learning series). (Ch. 12) The MIT Press. Kindle Edition.

Heaton, J. (2008). Introduction to Neural Networks for Java, 2nd Edition. (Ch. 5) Heaton Research, Inc.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning representations by Back-propagating errors.* Nature 323, 533-536.

University of California Irvine *Machine Learning Repository* https://archive.ics.uci.edu/ml/index.php Accessed 2018-11-04.