Programming Project 5: Feed-Forward Neural Networks and Backpropagation

Ricca D. Callis RCALLIS1@JH.EDU
Whiting School of Engineering
Engineering for Professionals, Data Science
Johns Hopkins University

Abstract

This project provided students enrolled in an Introduction to Machine Learning course (605.649.83.SU20), at Johns Hopkins University, the opportunity to implement a feed-forward neural network, a technique used for learning non-linear functions. Here, the Backpropagation algorithm was used in conjunction with gradient descent. Neural networks of 0, 1, and 2 hidden layers were applied to 5 data sets obtained from the UCI Machine Learning Repository. The size of the hidden layer was tuned using a validation set. A Sigmoid Activation Function was used to introduce nonlinearities into the network. A softmax classifier was used to train multi-net classifiers.
Keywords: feed-forward, neural network, backpropagation, sigmoid activation, softmax

1 Problem Statement & Hypothesis

Feed-forward neural networks are estimate non-linear relationships between a set of inputs and the densities of a corresponding class label or regression target variable. The goal of a feed-forward neural network is to approximate some function f^*. For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A feed-forward neural network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

Thus, these models are "fed-forward" because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. Typical intermediate computations include an affine transformation conducted on an input, at which point a non-linear function (i.e., activation function) is applied.

The neural network architecture includes multiple neurons, or Perceptrons. Groups of perceptrons are arranged in layers, with the first later taking in inputs and the last layer producing outputs. The middle layers are referred to as hidden layers because they have no connection with the external world. Although there are no connections among Perceptrons in the same layer, each perceptron in one layer is connected to every perceptron in the next layer. Hence, information can always be "fed forward" from one layer to the next. Since the Perceptrons within the hidden layer are independent of each other, input data fed into the network can become linearly separable.

1.1 This Project

This paper introduces a feed-forward neural multi-network trained using the Backpropagation algorithm. Here, 3 network configurations were used. For each dataset, networks with 0, 1, and 2 hidden layers were utilized.

1.2 Expectations

Multi-net vs Baseline. As has been demonstrated in previous projects, a simple linear classifier will outperform baseline (as measured by classification accuracy). Therefore, it is hypothesized that all neural networks (i.e., 0, 1, and 2 hidden layers) will outperform naïve baselines across all datasets used in this project.

Large Datasets. Due to the fact that neural networks are sensitive to learning rates, number of iterations, and size of each unit, it is hypothesized that neural networks will perform much slower on large datasets, especially those which are largely linearly separable.

Zero, One, or Two Hidden Layers. It is hypothesized that zero hidden layer networks will out-perform networks with one or two hidden layers on datasets which are largely linearly separable. For non-linearly separated classes, however, it is expected that neural networks with one or two hidden layers will out-perform those with zero hidden layers.

2 Description of Algorithm

2.1 Neuron: A Single Unit

In neural networks, neurons are represented as Perceptrons, or computational units which take as input x_1, x_2, x_3 (and a +1 intercept term), and outputs (o).
$h_{wb}(x) = f(w^T x) = f(\sum_{i=1}^{3} w_i x_i + b)$, where $f: \Re \mapsto \Re$ is called the activation function. Here, we use the Sigmoid Activation Function

$$\sigma(x) = \frac{1}{1 + exp^{-x}}$$

$$o = \sigma(XW)$$

Where,

W is a matrix with the same number of rows as X has columns.
o is the output

We note that the sigmoid unit's output is a nonlinear function of its inputs, but whose output is also a differentiable function of its inputs (e.g., Mitchell, 1997; see Figure 1). This output ranges between 0 and 1, increasing monotonically with its input. This is important because when training a classifier, the output must be interpreted as binary. Therefore, the output of the sigmoid function can be interpreted as the probability of a positive prediction.

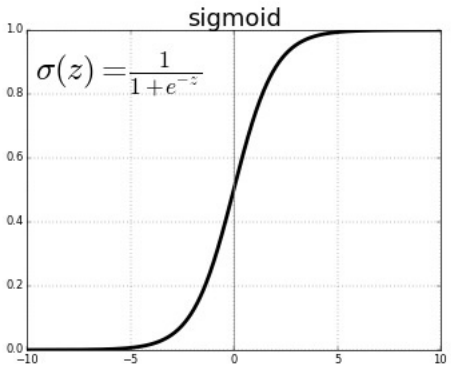

Figure 1. Sigmoid Function.

The sigmoid function also has the useful property that its derivative is easily expressed in terms of its output, which is utilized by the gradient descent learning rule.

$$\frac{d\sigma(x)}{dx} = \sigma(x) * \big(1 - \sigma(x)\big)$$

This makes it easy to train because the gradient is continuous and differentiable across the real number line. It is steepest around the decision boundary (where $x = 0$). If $x > 0$, then $\sigma(x)$ can be interpreted as indicating a positive class label.

2.2 Network Architecture

A neuronal network is constructed by linking multiple neurons such that the output of a neuron can be the input of another (see Figure 2). Let n_l denote the number of layers in the network and L_1 denote the layer. The first layer, L_1, indicates the input layer. The input layer contains input neurons x_1, x_2, x_3, and a +1 bias unit (which corresponds to the intercept term). The middle layer, L_2, indicates the hidden layer, named as such because its values are not observed in the training set. The final layer, L_3, is the output layer (note only one output in Figure 2). The neural network has parameters $(W, b) = (W^1, b^1, W^2, b^2, W^3, b^3)$, where W_{ij}^l denotes the parameter (or weight) associated with the connection between unit *j* in layer *l* and unit *i* in layer *l + 1*. Also, b_i^l is the bias associated with unit *i* in layer *l + 1*. Note that bias units have no connections going to them. We also further denote s_l as the number of nodes in layer *l* (excluding the bias unit).

We denote the activation (meaning output value) of unit i in layer l as a_i^l, where. $a_i^1 = x_i$ indicates the *i*th input. Given a fixed setting of the parameters *W, b,* the neural network defines a hypothesis $h_{wb}(x)$ that outputs a real number. Specifically,

$$a_1^2 = f(W_{11}^1 x_1 + W_{12}^1 x_2 + W_{13}^1 x_3 + b_1^1)$$
$$a_2^2 = f(W_{21}^1 x_1 + W_{22}^1 x_2 + W_{23}^1 x_3 + b_2^1)$$
$$a_3^2 = f(W_{31}^1 x_1 + W_{32}^1 x_2 + W_{33}^1 x_3 + b_3^1)$$
$$h_{wb}(x) = a_1^3 = f(W_{11}^2 a_1^2 + W_{12}^2 a_2^2 + W_{13}^2 a_3^2 + b_1^2)$$

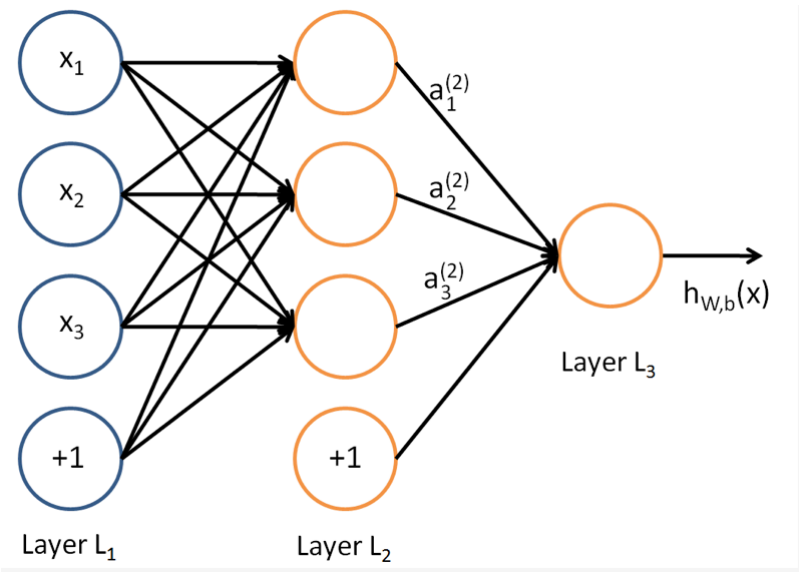

Figure 2. Multi-layer Neural Network with One Hidden Layer.

2.3 Forward Propagation

We also let z_i^l denote the total weighted sum of inputs to unit *i* in layer *l*, including the bias term (e.g., $z_i^2 = \sum_{j=1}^{n} W_{ij}^1 + b_i^1$), so that $a_i^l = f(z_i^l)$. If we extend the activation function to apply to vectors in an element-wise fashion (i.e., $f([z_1, z_2, z_3]) = (f(z_1), f(z_2), f(z_3))$, then we can write:

$$z^2 = W^1 x + b^1$$
$$a^2 = f(z^2)$$
$$z^3 = W^2 a^2 + b^2$$
$$h_{W,b}(x) = a^3 = f(z^3)$$

More generally, since we also use $a^1 = x$ to denote the values from the input layer, then given layer *l*'s activations a^l, we can compute layer $l + 1$'s activation a^{l+1}as:

$$z^{l+1} = W^l a^l + b^l$$
$$a^{l+1} = f(z^{l+1})$$

2.4 Multiple Hidden Layers

When we add multiple hidden layers, layer 1 is the input layer, layer n_l is the output layer, and each layer *l* is densely connected to layer $l + 1$ (See Figure 3). In this setting, we calculate the output of the network by successively computing all the activations in layer L_2, then layer L_3, and so on, up to layer L_{n_l} using forward propagation.

The number of additional hidden layers is vital to defining the capacity of the neural network. As the number of hidden layers increases, the capacity of the network increases. As the corresponding search space increases as a result, selection bias plays a larger role in determining whether a network converges to a performant solution.

Beyond the number of hidden layers, the size of each hidden layer must also be chosen. Above, W was defined as $m \times k$ (where X is $n \times m$). This leaves k as a parameter. In the experiments conducted for this paper, k was be tuned over a validation

set. A large value of k will have a similar effect as adding more hidden layers. Specifically, larger values of k will increase the representation capacity, but will create more parameters (and thus a larger search space). For this project, k is restricted to a value between the number of input features and the number of output features. Although selecting $k > m$ allows for a higher-dimensional representation, it is unnecessary because the decision boundary here is non-linear.

Furthermore, the relationship between number of units and number of hidden layers may also influence training. A network might have fewer large hidden layers or many small hidden layers. Fewer layers may be easier to train, but many layers can provide greater capacity. Both parameters should be selected based on the specific experimental problem.

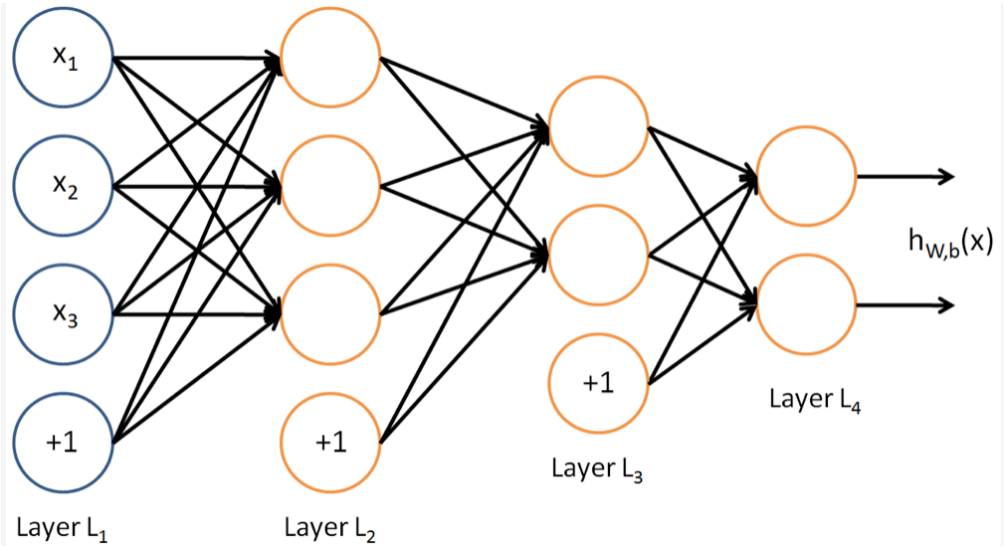

Figure 3. Multi-layer Neural Network with Multiple Hidden Layers.

2.5 Backpropagation

Given the network defined above, the challenge is to tune the weight matrices W_l for each l units in the network. Here, gradient descent was utilized. Given the loss function$E(W, X) = (O - Y)^T(O - Y)$, where $o_i \in O$ is the output of the network given input row X_i and $y_i \in Y$ is a vector of targets corresponding row-wise to X, the derivative of the loss relative to the output of the network is:

$$\frac{dE}{dO} = 2(O - Y)$$

We use the chain rule to find the derivative of the loss, relative to the weights in the final layer. Specifically, the derivative of the sigmoid function is:

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$

Therefore,

$$\frac{dE}{dW_l} = \frac{dE}{dO}\frac{\partial O}{\partial W_l} = (O - Y)\sigma(X)\big(1 - \sigma(X)\big)X$$

Then, the derivative for each of the preceding units must be found. First, δ_i is defined as

$$\delta_i = -\frac{\partial E}{\partial O_i}$$

where, O_i is the output of the i^{th} layer. Again, using the chain rule, this can be written as:

$$\delta_i = -\frac{\partial E}{\partial O_{i+1}}\frac{\partial O_{i+1}}{\partial O_i}$$

$$= \delta_{i+1}\frac{\partial O_{i+1}}{\partial O_i}$$

$$= \delta_{i+1}W_{i+1}\frac{\partial O_i}{\partial X_i}$$

$$= \delta_{i+1}W_{i+1}O_i(1 - O_i)$$

Essentially, we leverage the delta rule. Instead of recalculating the derivative for each unit, the derivative is expressed as the cumulative effect of all the units downstream from a given unit. This cumulative effect is denoted δ_j for the j^{th} unit in the network:

$$\delta_j = \delta_{j+1}W_{j+1} \circ O_i\big(1 - O_j\big)$$

Where $A \circ B$ denotes the Hadamard (element-wise) product of A and B, and O_j indicates the output of the j^{th} unit. We can see that the derivative of the error is propogated backwards through each of the preceeding units, all without having to recalculate the individual quantities, by accumulated δ_i throughout the network. This gives the algorithm it's name – backpropagation.

2.6 Gradient Descent

Following from the delta-rule, gradients are updated as follows:

$$\Delta W_l = \eta\delta_l X_l$$

where η is the learning rate, δ_l is defined above as the sum over all downstream δ_i, and X_l is the input to the l^{th} unit (the output of the l – 1 unit or the original set of features if h = 1). This process is iteratively repeated until a convergence criteria is met or the specified number of iterations is reached.

Mini-Batch Gradient Descent. Instead of using all training data items to compute gradients (as in batch training) or using a single training item to compute gradients (as in stochastic training), mini-batch training was utilized. Here, the user specifies the number of training items. These randomly selected batch training samples are then fed to the neural network. The mean gradient of the mini-batch is then calculated and used to update the weights. This step is repeated for each of the mini-batches created. Just like stochastic gradient descent, the average cost over the epochs in mini-batch gradient descent fluctuates because we are averaging a small number of examples at a time.

3 Experimental Approach

This analysis was conducted on 5 data sets, each obtained from the UCI Machine Learning Repository:

(1) Breast Cancer Data Set

(2) Glass Data Set

(3) Iris Data Set

(4) Soybean Data Set

(5) House Votes Data Set

For each data set, pre-processing included the elimination of null-values and transformation of all multi-valued discrete features into one-hot encoded variables. Descriptive statistics were calculated for all features and for each feature grouped by class label.

For each of the 5 experiments conducted, neural networks with 0, 1, and 2 hidden layers were trained. The size of the hidden layers and the learning rate were both tuned using a validation set, and chosen based on which set minimized the out-of-sample classification error.

Five-fold stratified cross-validation was used to estimate the out-of-sample error. All problems were trained using a multi-net approach, where each class label had a corresponding output. A softmax function was used to calculate the error relative to one-hot encoding of the class labels.

4 Experiment Results

4.1 Breast Cancer Dataset

Data Description. Classifies tumors as either malignant or benign, based on 10 feature attributes: id number, clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitosis (Wolberg, 1992; Wolberg & Mangasarian, 1990). This is a multivariate data set with 699 instances, where each attribute's instance is represented as a discrete value integer, ranging from 1 to 10.

Data Cleaning & Transformation: All instances with missing data were dropped from the data set (16 rows out of 699). The attribute id number was also dropped from the data set, as it represented a unique identifier that would not serve to teach class attributes. As a simple two-class problem (benign or malignant), with continuous-valued inputs, all features were mean-centered at zero and scaled by the standard deviation. Cross-validation was utilized to select the size of the hidden layers and the learning rate. Batches of size 48 were used in order to provide some stability in the learning process.

Exploratory Data Analysis & Results: Roughly 65.47% of the actual data instances were classified as benign, where as roughly 34.53% of the actual data instances were classified as malignant. Each feature was also described by class (descriptive statistics included mean, standard deviation, minimum, maximum, range, 1st quartile, median, and 3rd quartile) and were plotted using a box-plot. The results of the neural network classification experiment are shown in Table 1. We see that the model converges to a solution in line with previous linear classifiers. Clearly, the neural network outperforms the Naïve baseline model.

Model	Size of H1	Size of H2	Mean Accuracy	Learning Rate
Naïve Baseline	N/A	N/A	65.01%	N/A
0 Hidden Layers	N/A	N/A	96.91%	0.001
1 Hidden Layer	9	N/A	96.77%	0.005
2 Hidden Layers	12	3	96.62%	0.01

Table 2. Breast cancer feed-forward neural net classification results.

4.2 Glass Dataset

Data Description: Classifies origin of broken glass, based on 10 feature attributes of the broken shards: id number, refractive index, sodium, magnesium, aluminum, silicon, potassium, calcium, barium, and iron (German, 1987). The class attribute has 6 classifications: building windows float processed, building windows non-float processed, vehicle windows float processed, containers, tableware, or headlamp. This is a multivariate data set with 214 instances, where each attribute's instance is represented as a continuous value float.

Data Cleaning & Transformation: All instances with missing data were dropped from the data set (16 rows out of 699). The attribute id number was also dropped from the data set, as it represented a unique identifier that would not serve to teach class attributes. All features were mean-centered at zero and scaled by the standard deviation. Cross-validation was utilized to select the size of the hidden layers and the learning rate. Batches of size 48 were used in order to provide some stability in the learning process.

Exploratory Data Analysis: There were 76 instances classified as 2 (building windows nonfloat processed), 70 instances classified as 1 (building windows float processed), 29 instances classified as 7 (headlamps), 17 instances classified as 3 (vehicle windows float processed), 13 instances classified as 5 (containers), and 9 instances classified as 6 (tableware). Each feature was also described by class (descriptive statistics included mean, standard deviation, minimum, maximum, range, 1st quartile, median, and 3rd quartile) and were plotted using a box-plot. The results of the neural network classification experiment can be found in Table 2. We see that the network did not converge to a performant solution when hidden layers were added. It should be noted that there is not a lot of data for each class, which may in fact prevent convergence. The model with zero hidden layers, however, did converge to a solution.

Model	Size of H1	Size of H2	Mean Accuracy	Learning Rate
Naïve Baseline	N/A	N/A	35.55%	N/A
0 Hidden Layers	N/A	N/A	45.95%	0.025
1 Hidden Layer	7	N/A	34.22%	0.0625
2 Hidden Layers	5	7	38.46%	0.025

Table 2. Glass feed-forward neural net classification results.

4.3 Iris Dataset

Data Description: Classifies Iris species (Iris Setosa, Iris Versicolour, or Iris Virginica) based on 4 feature attributes from leaf measurements: sepal length, sepal width, petal length, and petal width (Fisher, 1988). As mentioned, the class attribute has 3 classifications: Iris Setosa, Iris Versicolour, or Iris Virginica. This is a multivariate data set with 150 instances, where each attribute's instance is represented as a continuous value float.

Data Cleaning & Transformation: All features were mean-centered at zero and scaled by the standard deviation. Cross-validation was utilized to select the size of the hidden layers and the learning rate. Batches of size 48 were used in order to provide some stability in the learning process.

Exploratory Data Analysis: There were 50 instances classified as Iris Versicolor, 50 instances classified as Iris Virginica, and 49 instances classified as Iris Setosa. Each feature was also described by class (descriptive statistics included mean, standard deviation, minimum, maximum, range, 1st quartile, median, and 3rd quartile) and were plotted using a box-plot. The results to the neural network classification experiment can be found in Table 3. We see that the networks had a little difficulty converging to highly performant results, but were all able to outperform the naïve baseline model, on average.

Model	Size of H1	Size of H2	Mean Accuracy	Learning Rate
Naïve Baseline	N/A	N/A	32.87%	N/A
0 Hidden Layers	N/A	N/A	73.20%	0.01
1 Hidden Layer	3	NA	89.95%	0.6025
2 Hidden Layers	4	4	66.44%	0.80

Table 3. Iris feed-forward neural net classification results.

4.4 Soybean Dataset

Data Description: Classifies soybean disease based on 36 feature attributes of the crop: date, plant stand, precipitation, temperature, hail, crop history, area damaged, severity, seed tmt, germination, plant growth, leaves, leafspots-halo, leafspots-marg, leafspot size, leaf shred, leaf malf, leaf mild, stem, lodging, stem cankers, canker-lesion, fruiting bodies, external decay, mucelium, int-discolor, sclerotia, fruit pods, fruit spots, seed, mold growth, seed discolor, seed size, shriveling, and roots (Michalski, 1987). The class attribute has 4 classifications: D0, D1, D2, or D3. This is a multivariate data set with 47 instances, where each attribute's instance is represented as a discrete value integer.

Data Cleaning & Transformation: Some attributes only had a single value and were dropped due to the fact that these algorithms would be unable to learn classifications that way. All remaining attributes were mapped into one-hot encodings. All features were mean-centered at zero and scaled by the standard deviation. Cross-validation was utilized to select the size of the hidden layers and the learning rate. Batches of size 48 were used in order to provide some stability in the learning process.

Exploratory Data Analysis: There were 17 instances classified as D3, 10 instances classified as D2, 10 instances classified as D1, and 9 instances classified as D0. Each feature was also described by class (descriptive statistics included mean, standard deviation, minimum, maximum, range, 1st quartile, median, and 3rd quartile) and were plotted using a box-plot. The results to the neural network classification experiment can

be found in Table 4. We see that the zero hidden layer network performed very well . However, networks with hidden layers (both one and two) performed similarly to the naïve baseline model.

Model	Size of H1	Size of H2	Mean Accuracy	Learning Rate
Naïve Baseline	N/A	N/A	36.83%	N/A
0 Hidden Layers	N/A	N/A	88.78%	0.005
1 Hidden Layer	9	N/A	36.83%	0.005
2 Hidden Layers	5	5	36.83%	0.005

Table 4. Soybean feed-forward neural net classification results.

4.5 House Votes Dataset

Data Description: Classifies party (republican or democrat) based on 16 feature attributes of legislation votes: handicapped infants, water project cost sharing, adoption of the budget resolution, physician fee freeze, el Salvador aid, religious groups in schools, anti-satellite test ban, aid to Nicaraguan contras, mx missile, immigration, synfuels corporation cutback, education spending, superfund right to sue, crime, duty free exports, and export administration act south Africa (Congress Quarterly Almanac, 1984). This is a multivariate data set with 435 instances, where each attribute's instance is represented as a discrete Boolean value, where 1 indicates that the congressperson voted for a measure and 0 indicates that the congressperson voted against a measure.

Data Cleaning & Transformation: All of the features were one-hot encoded. Null values were encoded as their own Boolean values. All features were mean-centered at zero and scaled by the standard deviation. Cross-validation was utilized to select the size of the hidden layers and the learning rate. Batches of size 48 were used in order to provide some stability in the learning process.

Exploratory Data Analysis: There were 124 instances classified as Republican and 108 instances classified as Democrat. The results of the neural network classification experiment are shown in Table 5.

Model	Size of H1	Size of H2	Mean Accuracy	Learning Rate
Naïve Baseline	N/A	N/A	53.45%	N/A
0 Hidden Layers	N/A	N/A	90.94%	0.006
1 Hidden Layer	7	N/A	86.38%	0.01
2 Hidden Layers	4	4	53.45%	0.01

Table 5. House votes feed-forward neural net classification results.

5 Behavior of Algorithms

Overall, we see that the performance of the neural networks utilized here were mostly in line with the hypotheses stated above. The neural networks outperformed the naïve baseline model in 3 out of 5 of the classification experiments conducted for this project. This is likely due to the fact that the class discriminates were mostly linear.

Grid search was utilized to find optimal learning rates, layer sizes, and number of training iterations. While this made tuning easier, it did significantly slow down training times.

6 Summary

This project provided students the opportunity to implement a feed-forward neural network, trained using the Backpropagation algorithm and mini-batch gradient descent. Neural networks with zero, one, and two hidden layers were performed on 5 data sets obtained from the UCI Machine Learning Repository and compared against a naïve baseline model. Overall, the neural networks outperformed the naïve baseline model when the boundary between classes were more non-linear.

References

Alpaydın, E. (2020). *Introduction to Machine Learning*. Cambridge, MA: MIT Press.

Congressional Quarterly Almanac (1984). 98th Congress, 2nd Session, 1984, Volume XL: Congressional Quarterly Inc., Washington, D.C. Retrieved June 8, 2020 from https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

German, B. (1987, September 1). Glass Identification Data Set. Retrieved June 8, 2020, from https://archive.ics.uci.edu/ml/datasets/Glass+Identification

Fisher, R.A. (1988). Iris Data Set. Retrieved June 8, 2020, from https://archive.ics.uci.edu/ml/datasets/Iris

Michalski, R. S. (1987, January 1). Soybean (Small) Data Set. Retrieved June 8, 2020, from https://archive.ics.uci.edu/ml/datasets/Soybean+%28Small%29

Mitchell, T. (1997). *Machine Learning*. New York, NY: McGraw-Hill.

Rao, C.R. (1948). The utilization of multiple measurements in problems of biological classification. Journal of the Royal Statistical Society, B, 10, 2: 159-203.

Wolberg, W. (1992, July 15). Breast Cancer Wisconsin (Original) Data Set. Retrieved June 8, 2020, from https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the National Academy of Sciences, 87, 9193-9196.