Classification and Regression with a Feed-Forward Neural Network

Turner Haugen TURNER.HAUGEN712@GMAIL.COM
Machine Learning Student
Montana State University
Bozeman, MT 59718, USA

Jackson Thorn JTHORN2448@GMAIL.COM
Machine Learning Student
Montana State University
Bozeman, MT 59718, USA

Abstract

This paper aims to provide a flexible neural network to use on a variety of datasets, provided by the UCI Machine Learning Repository. The neural network can have a wide variety of inputs, outputs, and hidden layers, as well as any number of nodes in each hidden layer. Inputs are fed forward through the network, while backpropagation is used to train/update the weights on each node. Node updating happens with momentum, which takes prior node weights into account when updating the current node. The network can take in both classification and regression datasets, handling each with different loss functions and methods of dealing with the output layers.

1 Introduction

Neural networks have been widely popular for years due to their flexibility and ability to model a variety of complex problems. The idea behind a neural network is to build a highly connected set of simple nodes into a network that can process far more than the individual nodes can do. This is intended to mirror neurons in the brain, where a neuron is very simple on its own, but can do powerful things in massive interconnected networks of neurons. Artificial neural networks aren't exactly one-to-one with biological neural networks, but the inspiration between neurons and basic neural network nodes still stands.

Feedforward neural networks (the type of neural network we are working with) consist of nodes perceptrons that store an activation value. Perceptrons are stored in layers that provide a template for how each perceptron gets arranged and used. The input layer contains all information about whatever data is being input. The number of input nodes for a given dataset usually corresponds to the number of features we want to input to the network. The output layer is where predictions are made. How the output layer is arranged will depend on whether we are looking for classification or regression outputs. In classification, the output layer will consist of the same number of perceptrons as possible unique outputs, where each perceptron corresponds to an outcome. Whichever perceptron has the highest activation will determine which outcome is predicted to be true. In regression, the

output layer will consist of a single perceptron with a linear activation function instead. Hidden layers lie between the input and output layers and provide additional connections for more model flexibility.

The way these layers are connected in a feedforward neural network is done with the feedforward process. Every node in one layer has a uniquely weighted connection to every node in the layer in front of and behind it. These weights control how connected two nodes should be to each other, with higher weights signifying more important connections. Information is fed forward via these weights. For a given node in a layer ahead of the current layer, its activation is determined by the sum of all activations in the previous layer times their corresponding weights. This process is repeated until we reach the activations at the final layer. A bias constant is added to add another unit independent of the activations and weights of previous layers that helps provide more properly fine-tuned outputs.

Backpropagation is used to fine-tune the massive amount of individual weights and biases contained within a network. Backpropagation relies on the gradient of the loss function, which tells us how we should change each parameter to improve the loss of the algorithm. Backpropagation starts after feeding forward some data and finding how much loss we have from that data. This loss is propagated backward through each weight and activation to find how much each weight needs to be adjusted. This process of feeding forward and propagating backward is repeated until our weights converge.

We hypothesize that networks made with more layers will result in generally more accurate predictions, but will result in overfitting more often than with fewer layers. Networks with no hidden layers result in a fairly simple network that isn't very flexible to different situations, but networks with multiple hidden layers may have too much flexibility and overfit depending on what is defined as the model converging. We also hypothesize that the network will perform better on larger datasets (like abalone) and worse on smaller datasets (like soybean_small) due to neural networks typically requiring much more fine-tuning because of the massive number of unique weights and biases.

2 Design

2.1 Initialization

To initialize our neural network there are several crucial pre processing steps that must be completed. We reused our data handler script from previous projects and made slight modifications to ensure compatibility with the network. Any features containing missing values were manually imputed; categorical values with the mode and continuous values with the mean. In addition, categorical features were one-hot encoded by converting each feature into a new binary column represented by a binary vector. Continuous features have a special consideration - to combat the vanishing gradient problem during backpropogation, we must normalize values to lie between 0 and 1. Normalization is represented by the following computation:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Where by subtracting the minimum attribute value from the original attribute value, and dividing that computation by the difference between the maximum attribute value and the minimum attribute value, the feature is normalized. This is all data-related preprocessing required for our network.

Building the network architecture begins with initializing weights, biases and velocity. Weights are initialized, at random, at very small values between -0.01 and 0.01 and connect the layers in the network. Biases are initialized at zero for each layer in the network, starting from the first hidden layer up to the output layer. Velocity is also initialized at zero and corresponds to each weight matrix in the network.

The input layer contains the same amount of nodes as there are features. However, the amount of nodes in the output layer is task-specific. For classification, there are n output nodes where n is the number of classes. For regression there is only a single output node as the goal is to predict a continuous value. Hidden nodes are a tunable hyperparameter, while we tested our network with zero, one or two hidden layers.

2.2 Forward Pass

Starting with the input layer, each feature is fed into its corresponding input node. At the first hidden layer, for each hidden node we compute the weighted sum:

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b$$

Where z is the weighted sum for each node, w_i is the weight of the connection between the input and the node, x_i is the input feature value to the node, and b is the bias for the node.To this sum, the logistic activation function is applied:

$$a = \frac{1}{1 + e^{-z}}$$

This should produce a non-linear output between 0 and 1, which is the input for the next hidden layer. This process continues for subsequent hidden layers. The output layer in the network depends on the problem. For classification, softmax is applied to the final layer's outputs to convert them into probabilities across classes present in the dataset:

$$softmax(z_i) = \frac{e^{z_1}}{\sum_j e^{z_j}}$$

For regression, the output is just the weighted sum. After this forward pass completes, the loss function is applied and represents the total error for the network on the current batch of data, and is problem dependent. For classification, we use cross-entropy loss to measure the difference between the predicted probability distribution and the actual class label:

$$L = -\sum_i y_i \cdot log(\hat{y_i})$$

For regression, we just use mean squared error:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Now, we can calculate the gradient of the loss with respect to each weight and bias to backpropogate through the network.

2.3 Backpropogation

Backpropogation is the learning component of our neural network. The goal is to adjust the network's weights and biases based on the loss computation, layer by layer. To start we compute the gradient of the loss, with respect to each weight, using the derivative of the logistic function:

$$f'(x) = f(x) \cdot (1 - f(x))$$

As for classification the gradient calculation is the difference between the predicted and true values, adjusted by the softmax derivative. For each hidden layer, the error is propagated backward from the layer ahead. The weights are updated through gradient descent, scaled by a learning rate and adjusted with momentum:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

$$\triangle w = \alpha \cdot \triangle w_{prior} + \eta \cdot \frac{\partial L}{\partial w}$$

Where w is the weight, η is the learning rate, $\frac{\partial L}{\partial w}$ is the gradient of the loss with respect to the weight and α is the momentum term. Gradient descent adjusts each weight by moving it in the direction that reduces the error. Similarly, biases are updated by calculating the gradient of the loss with respect to each bias. They shift the activation function's input and allow the network to better fit to the data. The updated weights and biases are used in the next forward pass, where this process repeats iteratively until the network converges.

2.4 Hyperparameter Tuning

With the goal of producing the best possible output for each task, a range of hyperparameters were tested to minimize loss. To combat the problems of overfitting and runtime, early stopping was implemented. The goal is to stop training the network by finding the convergence point. We stored the validation loss for each epoch, and if the validation loss for a given epoch exceeds the lowest validation loss for n epochs the network stops training. We tested this using 5 and 10 for values of n, specifically called patience.

Grid search suggested the number of hidden layers, the number of nodes per hidden layer, learning rate, momentum, batch count, patience and maximum number of epochs. Each combination was tested with one another, and once the 'best' combination is found 10-fold stratified cross-validation uses the found hyperparameters and generalizes results using different subsets of the data. At this point the network is completely trained and can be given out of sample data to make new predictions for.

3 Results

We will present a summary of the model after grid search along with the model architecture for each dataset. Additionally, the average loss from cross validation will be discussed. All models will test data on zero, one and two hidden layers. Starting with classification, below are insights into how each network fit to the data.

Parameter	**Model 1**	**Model 2**	**Model 3**
Input Layer	9 Nodes	9 Nodes	9 Nodes
Hidden Layers	0	1	2
Hidden Nodes	None	10	10 and 5
Output Layer	2 Nodes	2 Nodes	2 Nodes
Learning Rate	0.001	0.001	0.0001
Momentum	0.1	0.5	0.9
Batch Size	16	32	16
Max Epochs	50	200	200
Patience	10 Epochs	5 Epochs	5 Epochs
Validation Loss	2.3832	2.3950	2.3950
Avg. Cross-val Loss	2.3897	2.3978	2.3976

Table 1: Breast Cancer Wisconsin (Classification)

Breast Cancer Wisconsin performed best with zero hidden layers. Subsequent hidden layers marginally decreased performance, but found different hyperparameters. Though generally, this suggests that the data does not necessarily require a neural network to make predictions - a linear approach may be just as effective.

Parameter	**Model 1**	**Model 2**	**Model 3**
Input Layer	9 Nodes	9 Nodes	9 Nodes
Hidden Layers	0	1	2
Hidden Nodes	None	5	5, 10
Output Layer	6 Nodes	6 Nodes	6 Nodes
Learning Rate	0.01	0.01	0.01
Momentum	0.9	0.9	0.9
Batch Size	16	16	32
Max Epochs	200	50	100
Patience	5 Epochs	10 Epochs	5 Epochs
Validation Loss	0.7429	0.7562	0.7566
Avg. Cross-val Loss	0.8215	0.8329	0.8335

Table 2: Glass (Classification)

Glass also performed best with zero hidden layers, and subsequent layers did not find a different learning rate or momentum value during grid search. Similar to Breast Cancer Wisconsin, a linear approach may be just as effective to make predictions.

Parameter	Model 1	Model 2	Model 3
Input Layer	34 Nodes	34 Nodes	34 Nodes
Hidden Layers	0	1	2
Hidden Nodes	None	5	5, 5
Output Layer	4 Nodes	4 Nodes	4 Nodes
Learning Rate	0.01	0.01	0.01
Momentum	0.9	0.9	0.9
Batch Size	16	16	16
Max Epochs	100	200	200
Patience	5 Epochs	5 Epochs	5 Epochs
Validation Loss	0.0	0.0	0.0
Avg. Cross-val Loss	0.1799	0.2234	0.4468

Table 3: Soybean Small (Classification)

For soybean small, the validation loss is zero across the board suggesting perfect accuracy on the validation set. However this is unlikely, and the model may be erroneously built for this dataset. The hyperparameters generally stayed the same across the models, while cross-validation loss consistently increased. Like the other classification datasets, the best model appears to be the one with zero hidden layers - meaning a linear approach may be just as effective to make predictions.

Parameter	Model 1	Model 2	Model 3
Input Layer	29 Nodes	29 Nodes	29 Nodes
Hidden Layers	0	1	2
Hidden Nodes		5	10 per layer
Output Layer	1 Node	1 Node	1 Node
Learning Rate	0.01	0.001	0.001
Momentum	0.9	0.9	0.9
Batch Size	32	16	32
Max Epochs	100	50	200
Patience	10 Epochs	10 Epochs	5 Epochs
Validation Loss	895.5249	924.0517	923.0452
Avg. Cross-val Loss	4240.0446	4287.5858	4332.1625

Table 4: Forest Fires (Regression)

Moving on to regression, forestfires has 29 nodes in the input layer. On the cross-validation dataset, the data shows a slight preference to a 0 hidden layer model. The results are intriguing: grid search found a substantially lower validation loss than for cross-validation subsets. This could be expected since forestfires is a large, complex dataset. This model also stopped earlier than classification models, indicating faster convergence. Improvements can be made, such as increasing the number of epochs and hidden layers. Additionally, the gap between validation loss and average cross-validation loss appears to be the same, meaning

overfitting is clearly an issue but the scale of overfitting does not depend on the number of layers.

Parameter	**Model 1**	**Model 2**	**Model 3**
Input Layer	246 Nodes	246 Nodes	246 Nodes
Hidden Layers	0	1	2
Hidden Nodes		5	10, 5
Output Layer	1 Node	1 Node	1 Node
Learning Rate	0.01	0.0001	0.0001
Momentum	0.5	0.9	0.5
Batch Size	32	16	32
Max Epochs	100	50	50
Patience	5 Epochs	10 Epochs	10 Epochs
Validation Loss	410.3156	311.8553	520.2390
Avg. Cross-val Loss	3584.4021	6275.7707	14095.4836

Table 5: Machine (Regression)

The machines dataset has 246 nodes in the input layer due to there being a massive amount of individual levels that the two categorical variables can take up. The validation dataset showed a rather low loss, with the best model being a model with 1 hidden node with 5 nodes in it. However, the cross-validation dataset's mean square error exploded with larger numbers of hidden layers. This indicates that overfitting became a much worse issue with additional layers. The lower number of max epochs also shows that the model ran much quicker than expected, meaning the model on 200 epochs likely was far more overfit. A potential alternate way of calculating convergence could be beneficial, as well as possibly cutting out the categorical variables entirely.

Parameter	**Model 1**	**Model 2**	**Model 3**
Input Layer	10 Nodes	10 Nodes	10 Nodes
Hidden Layers	0	1	2
Hidden Nodes		5	10, 5
Output Layer	1 Node	1 Node	1 Node
Learning Rate	0.01	0.01	0.001
Momentum	0.9	0.9	0.9
Batch Size	16	16	32
Max Epochs	100	200	200
Patience	5 Epochs	5 Epochs	10 Epochs
Validation Loss	4.3978	3.7950	3.7738
Avg. Cross-val Loss	5.0108	4.8425	4.5222

Table 6: Abalone (Regression)

The abalone dataset has 10 nodes in the input layer. The validation dataset shows a very low loss, and the cross-validation average loss performed was slightly larger, but performed around the same. Overall, the neural network performed very well on the abalone dataset.

This likely comes from the smaller number of columns and much larger amount of data resulting in very little overfitting occurring. The increased number of max epochs indicates that the model was much more willing to be fine tuned with larger numbers of hidden layers.

4 Discussion

Our hypothesis about overfitting ended up not being true; the level of overfitting seemed to depend on the dataset. The level of overfitting sometimes changed depending on the number of hidden layers drastically, but it also sometimes improved with better datasets. Additionally, neural networks with larger numbers of layers did not necessairly perform better with predictions. For example, the forestfires dataset showed better predictions on networks with no hidden layers on both the validation and average cross-validation loss. The one hypothesis that did check out was that neural networks with less nodes were faster to run than networks with multiple layers. This was the most noticeable on the abalone dataset, which takes much longer to run than other models.

Neural nets proved to be challenging as while some of the math may be straightforward, the math behind backpropagation was difficult to grasp and far more difficult to code. Much of this project was spent fixing bugs because of how many different ways things can go wrong inside of the backpropagation algorithm's linear algebra calculations. This neural net seemed to perform very different, as while its performance on the abalone and soybean datasets was decent, the performance on the breast cancer and machine datasets was rather poor. Further adjustments may be needed to make this neural network perform better.

5 Appendix

Jackson wrote the following:

- Feedforward model
- Full codebase overhaul of model to fix backpropogation
- Grid search method
- Results and design section of paper

Turner wrote the following:

- Min-max normalization
- Final code cleanup + cross validation
- Initial backpropagation version
- Introduction, abstract, results and discussion section of paper
- Video