Classifying Table Wines

Tre' R. Jeter

October 5, 2023

Abstract

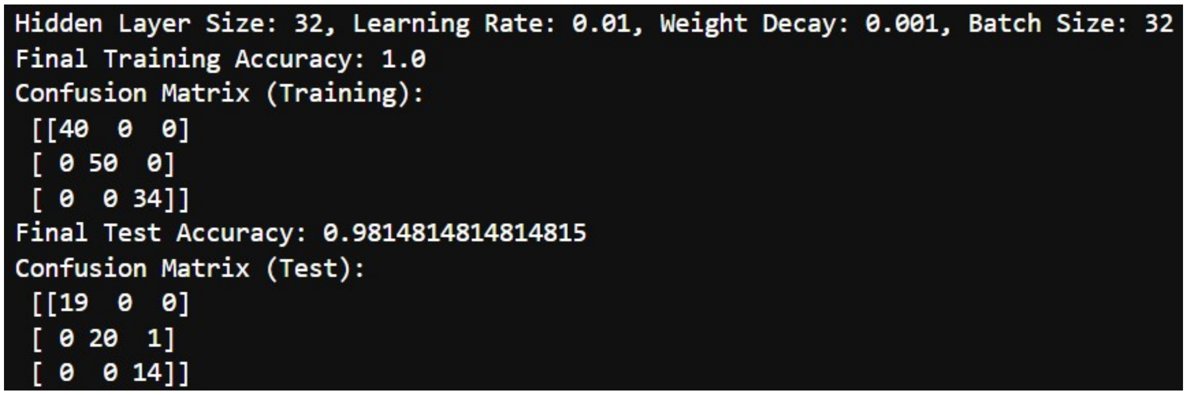
This study explores the application of Multilayer Perceptrons (MLPs) and Back-Propagation for classifying the Wine Recognition dataset. MLPs, a type of artificial neural network, provide a flexible framework for complex pattern recognition tasks. The research investigates the performance of MLPs with varying architectures and hyperparameters to classify wines into distinct categories based on 13 chemical attributes. The experiments showcase the impact of hyperparameters on classification accuracy and convergence behavior, emphasizing the significance of parameter tuning for optimal performance. This investigation demonstrates the potential of MLPs and Back-Propagation as effective tools for multiclass classification tasks like Wine Recognition, offering valuable insights for practitioners in utilizing neural networks for similar applications.

1 Setup

The Wine Recognition dataset contains information on various attributes of three different classes of wine, making it a multi-class classification problem. I use the *.data* and *.target* variables that are built-in to the dataset to denote the feature data and target labels, respectively. To establish training and testing sets, I implement a 70/30 split for training and testing. I then scale the data using the *StandardScaler* library.

2 Experiment 1

The initial experiment involves using a single layer MLP to classify the Wine Recognition Dataset. The input size of the model is the shape of the training data, 13. The output size of the model is the number of unique labels within the target variable, 3. Besides the input and output size, the initial hyperparameters include a hidden layer size of 32, learning rate of 0.01, weight decay of 0.001, batch size of 32, ReLU/Softmax activations, and 1000 epochs. In training, we see that the training set was classified 100% accurately and is viewed in the training confusion matrix in Figure 1. In testing, we see that the testing set was classified at ≈98.15% and is also viewed in the testing confusion matrix in Figure 1. The related plot showing the drop in loss per iteration is viewed in Figure 2.



```
Hidden Layer Size: 32, Learning Rate: 0.01, Weight Decay: 0.001, Batch Size: 32
Final Training Accuracy: 1.0
Confusion Matrix (Training):
 [[40  0  0]
 [ 0 50  0]
 [ 0  0 34]]
Final Test Accuracy: 0.9814814814814815
Confusion Matrix (Test):
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]
```

Figure 1: Initial classification.

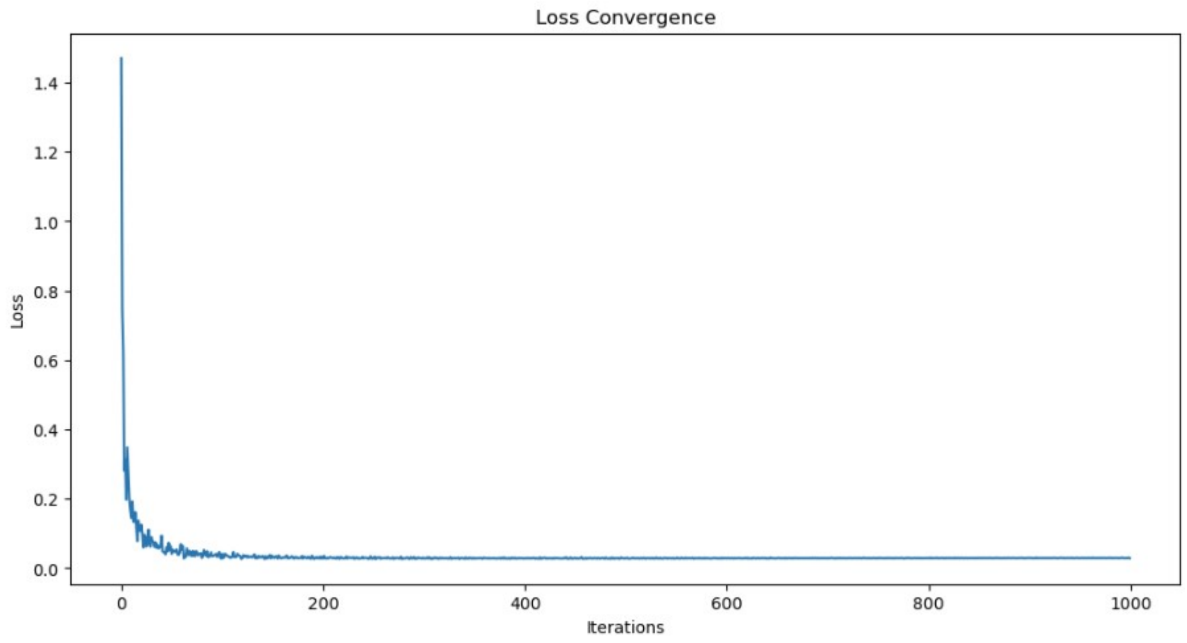


Figure 2: Initial experiment plot.

3 Experiment 2

Table 1 describes the second experiment where multiple hyperparameters are tested. Of each combination, 28 of them resulted in 100% accuracy in the training set and ≈96% accuracy or higher on the test set. The specific combinations are below in Table 2.

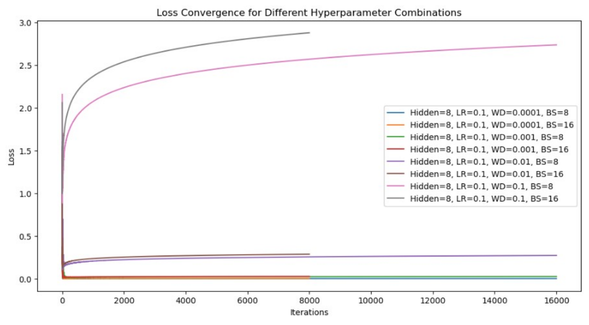
Hidden Size	Learning Rate	Weight Decay	Batch Size	Epochs
$[8, 16, 32, 64, 128]$	$[0.0001, 0.001, 0.01, 0.1]$	$[0.0001, 0.001, 0.01, 0.1]$	$[8, 16, 32, 64]$	$[100, 1000, 1000]$

Table 1: Tested hyperparameters with a single hidden layer.

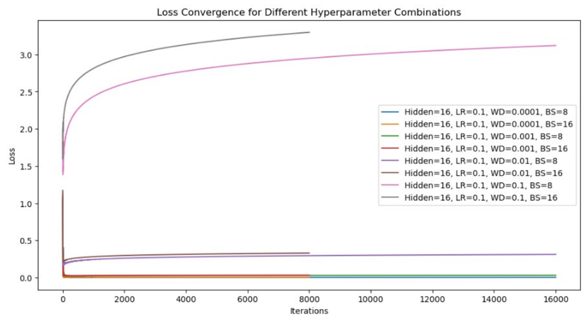
Hidden Size	8	16	32	64	128
	$[0.1, 0.0001, 8]$	$[0.1, 0.0001, 8]$	$[0.1, 0.0001, 8]$	$[0.1, 0.0001, 8]$	$[0.1, 0.0001, 16]$
	$[0.1, 0.0001, 16]$	$[0.1, 0.0001, 16]$	$[0.1, 0.001, 8]$	$[0.1, 0.001, 8]$	$[0.1, 0.001, 16]$
	$[0.1, 0.001, 8]$	$[0.1, 0.001, 8]$	$[0.1, 0.01, 8]$	$[0.1, 0.01, 8]$	$[0.1, 0.01, 16]$
	$[0.1, 0.001, 16]$	$[0.1, 0.001, 16]$	$[0.1, 0.1, 8]$	$[0.1, 0.1, 8]$	$[0.1, 0.1, 16]$
	$[0.1, 0.01, 8]$	$[0.1, 0.01, 8]$			
	$[0.1, 0.01, 16]$	$[0.1, 0.01, 16]$			
	$[0.1, 0.1, 8]$	$[0.1, 0.1, 8]$			
	$[0.1, 0.1, 16]$	$[0.1, 0.1, 16]$			

Table 2: Combinations of hyperparameters that yielded 100% accuracy on the training set and ≈96% or higher on the test set. From left to right within the brackets is learning rate, weight decay, and batch size, respectively.

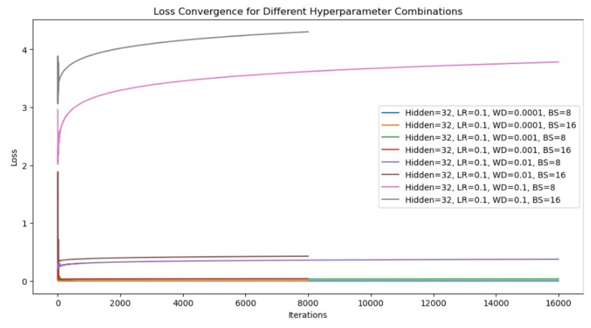
To view the behavior of these hyperparameters, I plotted the loss vs. number of iterations. Below in Figure 3, we see that the weight decay has a huge impact on the loss value. Although some combinations reported 100% classification accuracy on the training, the loss value grows in proportion to the weight decay.



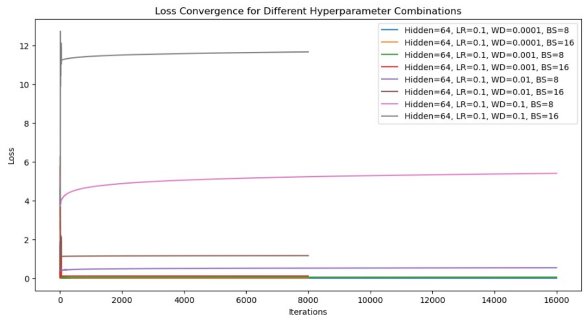
(a) Hidden Layer Size = 8.



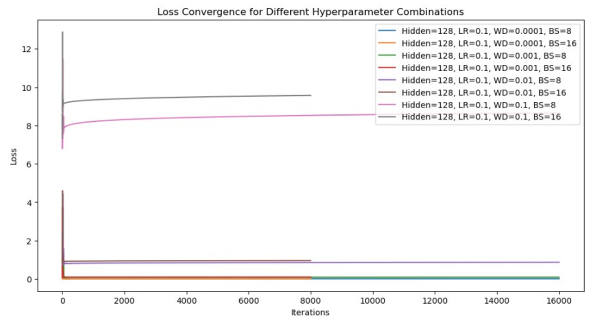
(b) Hidden Layer Size = 16.



(c) Hidden Layer Size = 32.



(d) Hidden Layer Size = 64.

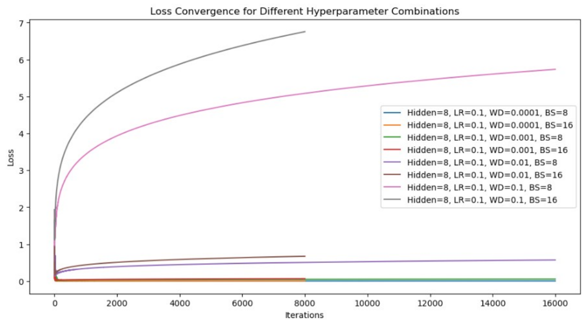


(e) Hidden Layer Size = 128.

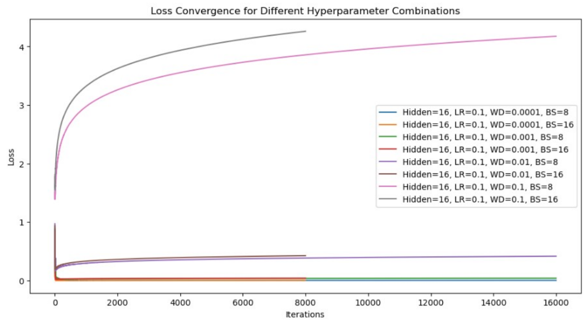
Figure 3: Multiple hyperparameters with varying hidden layer size.

4 Experiment 3

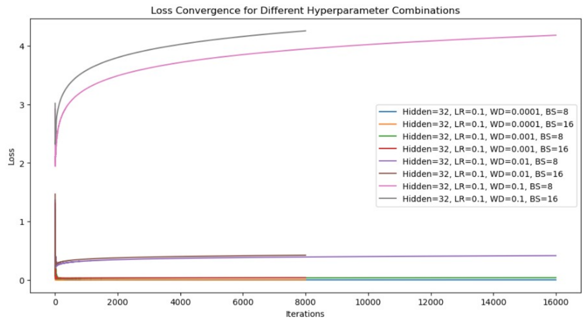
In this experiment, I compare the difference between the ReLU activation function with the Tanh activation function. Because the ReLU is used in the previous experiments, plots for this section are only for the Tanh activation function in Figure 4. From earlier experiments, I adopt the hyperparameters that are known to yield 100% with the ReLU activation function. As we can see, the results including the Tanh activation function are *very* similar to the results with the ReLU activation function. The loss behavior acts in a similar manner over iterations. The ranges between loss curves are slightly bigger with the Tanh activation function, but they still follow the same trends.



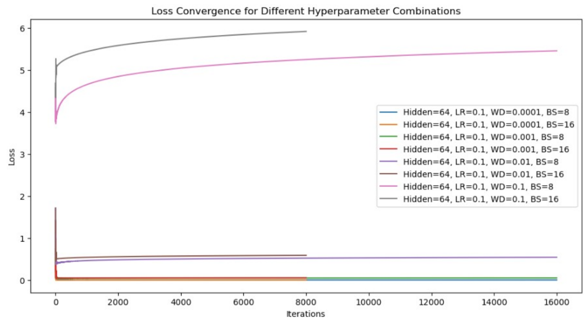
(a) Hidden layer = 8.



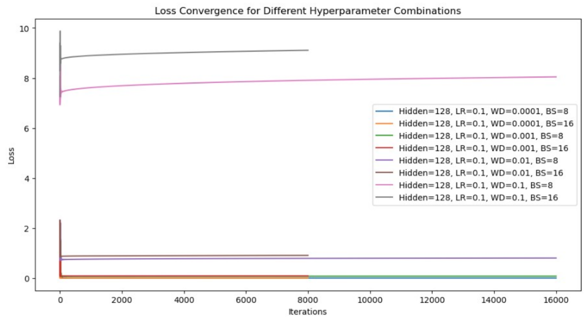
(b) Hidden layer = 16.



(c) Hidden layer = 32.



(d) Hidden layer = 64.

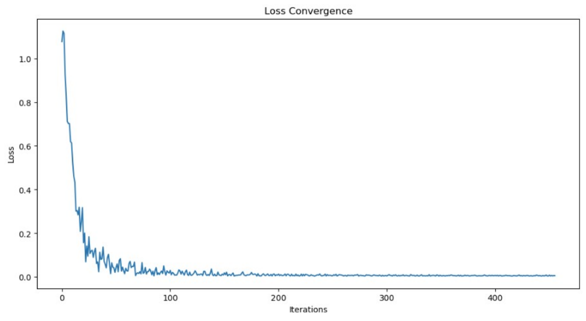


(e) Hidden layer = 128.

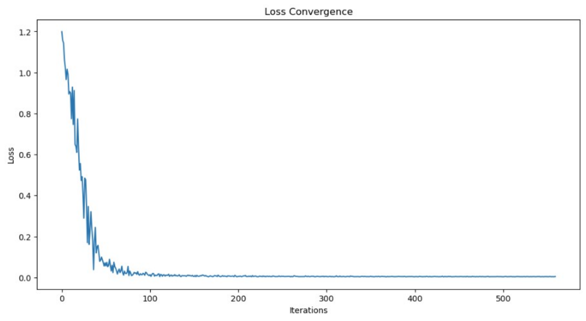
Figure 4: Multiple hyperparameters with varying hidden layer size and the Tanh activation function.

5 Experiment 4

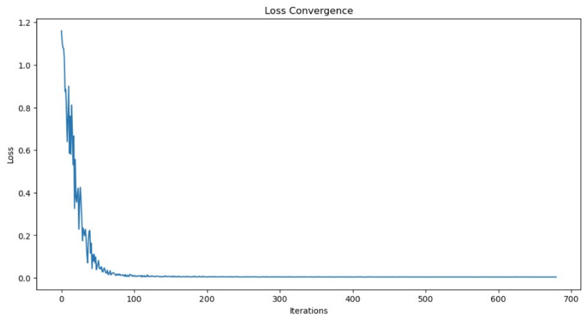
Before the announcement was released correcting the assignment description, I had already experimented classifying the Wine Recognition Dataset with varying hidden layers. Results below in Figure 5 show the loss curve with 2, 3, 4, and 5 layers, respectively with an early stopping rate (patience) of 20 epochs.



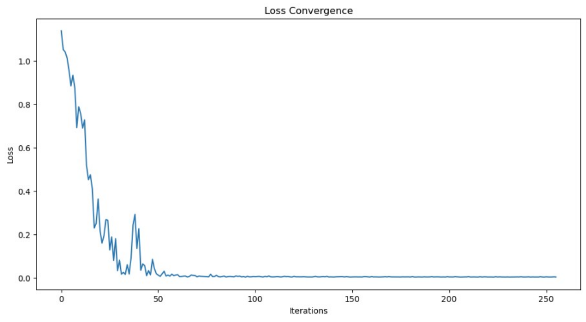
(a) Loss curve with two hidden layers.



(b) Loss curve with three hidden layers.



(c) Loss curve with four hidden layers.



(d) Loss curve with five hidden layers.

Figure 5: Loss curve results for multiple hidden layers.

6 References

******Note:*** The references below are the same from Homework 2. Because we were instructed to adapt/modify our previous code, the references from Homework 2 are still valid for citation here.***

Below is a list of references used to assist with this assignment:
https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
https://pyimagesearch.com/2021/05/06/backpropagation-from-scratch-with-python/
https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide
https://github.com/ahmedfgad/IntroDLPython/tree/master
https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
https://medium.com/swlh/how-to-build-a-neural-network-from-scratch-b712d59ae641
https://medium.com/towards-data-science/how-to-program-a-neural-network-f28e3f38e811
https://github.com/c-bruce/artificial_neural_network