Ethan Tobey

Introduction

In this paper, I will conduct a detailed investigation into the performance of different machine learning algorithms and models on the problem of categorizing the Iris Dataset. Throughout this investigation, I will commonly reference models developed in the Jupyter Notebook irisNetwork.ipynb. This notebook is a valuable complement to this investigation, and contains all materials used in order to create the analyses presented here. The notebook, as well as instructions for its usage and setup, can be found within the file package attached to this report.

AI Models

During this investigation, I created a total of eight different AI models in order to compare their performance in categorizing the iris dataset into different species based on its four data components (sepal length, sepal width, petal length, petal width). The first seven models were Neural Networks. From model to model, I experimented with adjustments in the nonlinearity and learning function in order to try to maximize performance and training speed. The eighth model was built without a Neural Network approach by using the K-Means Clustering algorithm. For reference, a more specific breakdown of the models is listed below.

1. Sigmoid Nonlinearity, Gradient Descent Learning
2. ReLU Nonlinearity, Gradient Descent Learning
3. Leaky ReLU Nonlinearity, Gradient Descent Learning
4. Swish Nonlinearity, Gradient Descent Learning
5. ReLU Nonlinearity, RMSpop Learning
6. ReLU Nonlinearity, Adam Learning
7. Relu Nonlinearity, Adam Learning, Dropout Layer
8. K-Means Clustering

Neural Network Architecture

With the exception of the seventh model, all the Neural Networks were built with similar architectures, varying from one model to the other in terms of their nonlinearities and learning functions. This was done intentionally in order to allow for direct comparison between the performance of these different nonlinearities and learning functions across similar model architectures.

The standard model architecture was built with a Keras Sequential modeling system. There was one dense input layer, one dense hidden layer, and one dense output layer. The input and hidden layers both consisted of 32 neurons, enabling the model to represent sufficiently

complex relationships between the data. The output layer had three neurons, one to represent each potential output category (Setosa, Versicolor, Virginica).

The input and hidden layers both always used the same nonlinearity, which across testing varied between Sigmoid, ReLU, Leaky ReLU, and Swish. The output layer consistently used the Softmax nonlinearity, due to its specialty for dividing into more than two categories.

For the seventh model, an additional layer was added between the hidden and output layers. This was a dropout layer, which was set to 'drop' 50% of the neurons in the model each epoch during training. When a neuron is 'dropped' it is set to always output 0 during that training iteration, with the aim of forcing the model to become more flexible and decreasing the likelihood of overfitting.

Implementation Steps

A detailed tutorial of steps to replicate my experiment can be found within the irisNetwork.ipynb Jupyter Notebook. While I will also record similar information here, I believe that the context added withing the Jupyter Notebook adds significantly to the clarity of the explanations, and would recommend reviewing this information as a complement to these steps.

1. Materials

   For the construction of these models, necessary materials include the following:

   - Pandas
   - Numpy
   - Seaborn
   - Keras from Tensorflow
   - Sci-kit Learn
   - Matplotlib

2. Data Setup

   The data must be loaded from its csv file, and then separated into the inputs and the classes. The inputs consist of the first four columns of the data, which include the sepal width, sepal length, petal width, and petal length. The final colum, the species name, defines the different categories that our models will attempt to sort data into.

   When separating output data, the string type species names must be encoded numerically in order to allow for their processing by the models. A simple way of doing this is to just encode each species as a number: 0 for setosa, 1 for versicolor, and 2 for virginica.

   Once the inputs and outputs have been properly set up, the data must be divided into training and test datasets. In my experiment, this was done by randomly selecting 30%

of the data to be separated out as test sets.

3. Neural Network Model Definition and Training

   The Neural Networks are defined using Keras. The library makes it relatively simple to specify the architecture. First, a sequential model must be initialized. Then, the input, hidden, and output layers can all be added with their specified nonlinearities. In order to train the model, a learning function, or optimizer, must be specified, as well as a loss metric. Across all networks in this experiment, Mean Squared Error was used as the loss function.

   When training, an early stopping system was used to automatically end training if there was no improvement in the loss function over 10 epochs in a row. This, song with some adjustments in the number of epochs specified for the training time, was used to determine an amount of training time sufficient to train each model to an accuracy of 95% or higher.

4. Neural Network Analysis

   Analysis on the Neural Network models is done by checking the performance of the models over the test data. The outputs of this testing are displayed as heat maps, which allows for the visualization of the categorization performance of the models. The speed of training offered by the adjustments in the nonlinearities and learning functions is also reflected by a learning progress graph, which shows the progress of the model throughout training, and highlights the point at which the model achieved a 50% reduction in the mean squared error.

   The performance of different models was compared with one another based on the metrics of their accuracy scores over the test data and their training speeds.

5. K-Means Clustering Definition and Training

   The K-Means Clustering model is defined using Sci-Kit Learn's KMeans package, which allows for the easy definition of the model given the data that was already separated in step 2.

   In order for direct comparison with the Neural Network Models, the clusters created by K-Means were assigned labels to represent the different category of iris they were sorting. A category was assigned a label based on which iris type was in the majority for a given cluster.

   The results of the K-Means Clustering were then compared to the same test dataset as the Neural Networks, and a heatmap was generated to demonstrate the model's performance.

Nonlinearity Exploration

Four different nonlinearities were tested across the different Neural Network models in this experiment. These were the Sigmoid, ReLU, Leaky ReLU, and Swish functions.

The Sigmoid nonlinearity is a commonly used activation function which maps input values to outputs ranging between 0 and 1. It is mathematically defined as follows:

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

A sigmoid function appears on a graph as an s-shaped curve. The advantage of the sigmoid nonlinearity is its ability to provide decimal values between the binary outputs of 0 and 1, providing more information for learning compared to ta threshold nonlinearity. While a sigmoid nonlinearity is useful, it has common issues such as vanishing gradients, which cause weights to become stuck during training of deep neural networks. The sigmoid is commonly used for binary classification tasks.

The Rectified Linear Unit (ReLU) nonlinearity is an activation function which outputs the input if it is positive, and 0 otherwise. This is mathematically defined as follows:

$$f(x) = max(0, x)$$

A ReLU nonlinearity is commonly used due to its computational efficiency, as well as its ability to mitigate the vanishing gradient problem which plagues the sigmoid nonlinearity. However, ReLU is not without its own challenges. In particular, it can suffer from the “dying ReLU” problem, in which neurons can become inactive if they are consistently fed zero inputs. This can slow down or halt the learning of a ReLU network, and is more common in deeper networks.

The Leaky ReLU nonlinearity is an activation function that is based on a modification of ReLU. The aim of Leaky ReLU is to address the “dying ReLU” problem described above. This is done by slightly changing the way in which zero outputs are passed. Rather than outputting zero for all negative inputs, Leaky ReLU uses a small gradient for negative inputs, redefining the function as:

$f(x) = x \; if \; x > 0 \; and \; f(x) = \alpha x \; if \; x \leq 0$, were α is a small positive constant

The Leaky ReLU nonlinearity retains many of the advantages of standard ReLU while minimizing the drawbacks. For the purposes of this experiment, it was used to compare with a ReLU network to see if the network was suffering from the “dying ReLU” problem.

The Swish nonlinearity is an activation function developed by researchers at Google which has been shown to frequently outperform ReLU in certain scenarios, such as deep neural networks. It is defined as follows:

$f(x) = x * \sigma(x)$, where σ(x) is the sigmoid function defined above.

Unlike most other activation functions, Swish allows for small negative values, which can improve gradient flow. It is this property that allows it to outperform ReLU in some deep learning tasks. While the network in this experiment is shallow, Swish was tested to see how its performance would compare to the other nonlinearities.

Learning Function Exploration

Three different learning functions were tested across the different Neural Network models in this experiment. These were Gradient Descent, RMSprop, and Adam learning functions.

The Gradient Descent learning function is an optimization algorithm which minimizes the loss function by updating model parameters in the direction of the steepest descent. This steepest descent is assessed by an evaluation of the gradient, hence the name of the algorithm. At each iteration of learning, parameters are adjusted based on the gradient scaled by a small amount, called the learning rate. Gradient descent is commonly used in many applications, but must be carefully tuned in order to maximize performance.

The Root Mean Square Propagation learning function is an optimization algorithm used to minimize the loss function. It builds upon the framework of gradient descent by adjusting the learning rates for each parameter dynamically. These adjustments are calculated based on the average of the squared gradients, which gives the algorithm its name. RMSprop is a powerful algorithm, excelling at normalizing the magnitude of updates to the parameters, and is particularly good at reaching convergence in noisy settings.

The Adaptive Movement Estimation (Adam) learning function is an optimization algorithm that combines RMSprop with another learning function, called AdaGrad (not tested in this experiment). Like RMSprop, Adam uses adaptive learning rates for each parameter, but computes them based on both the mean and variance of the gradients. This helps to improve the performance of the adjustment of learning rates when dealing with noisy or hard to learn from data. These adjustments allow Adam to converge much quicker than most traditional gradient descent based learning functions.

Results

As with previous sections, it is recommended to use the irisNetwork.ipynb Jupyter notebook to view sample results for the models. The data below demonstrates results based on one test, but due to the randomization involved in separating the test and training datasets, as well as the variation in training performance across different model training attempts, results when running the notebook may vary slightly. From my sample testing, here are the results for each of the eight models.

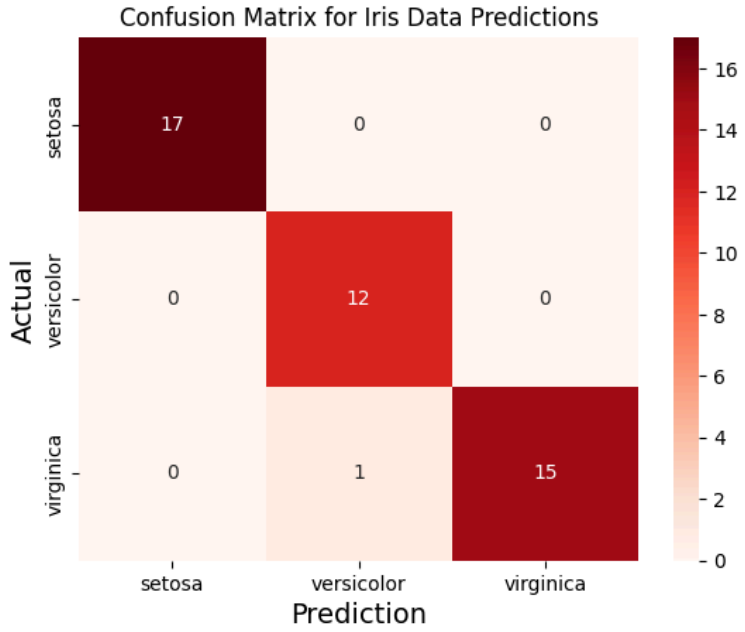
Model 1: Sigmoid Nonlinearity, Gradient Descent Learning

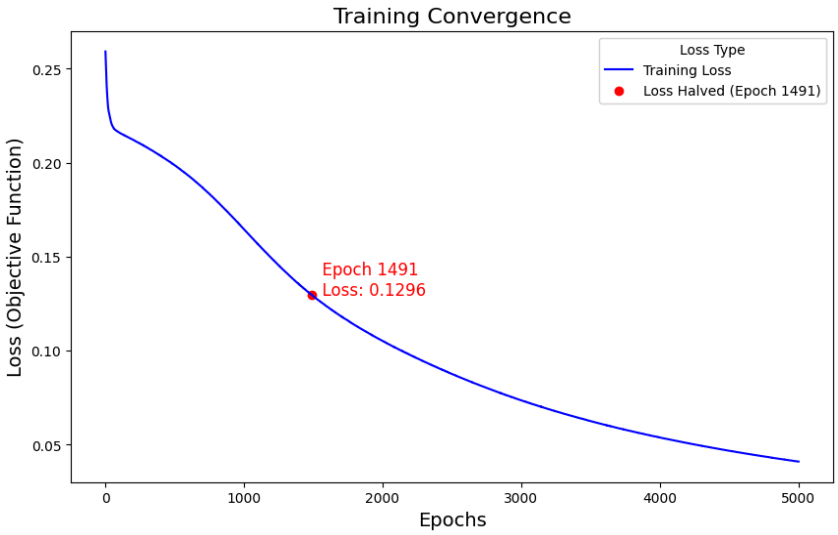
Training Accuracy: 96.17%

Test Accuracy: 97.77%

Epochs to 50% Loss Reduction: 1491 Epochs

Epochs in Training: 5000 Epochs





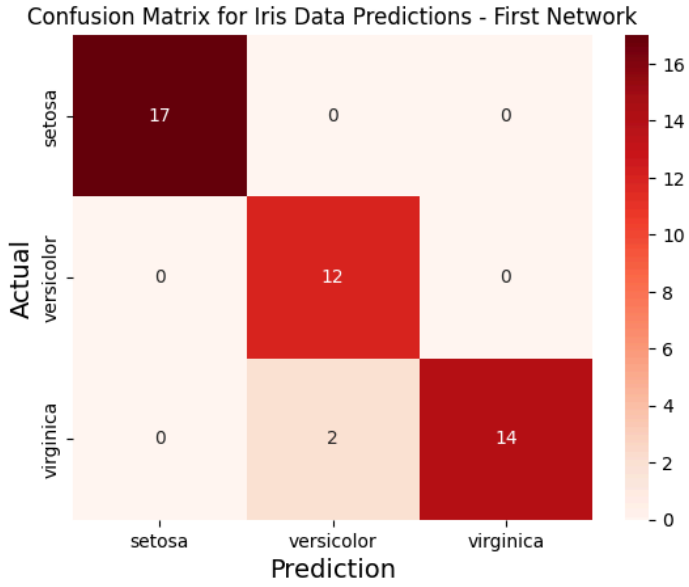
Model 2: ReLU Nonlinearity, Gradient Descent Learning

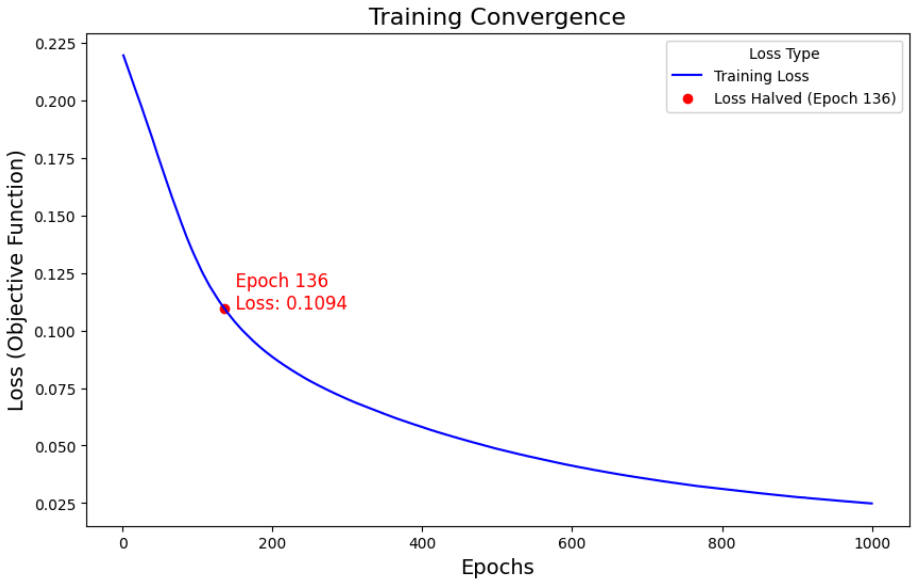
Training Accuracy: 95.24%

Test Accuracy: 95.55%

Epochs to 50% Loss Reduction: 136 Epochs

Epochs in Training: 1000 Epochs





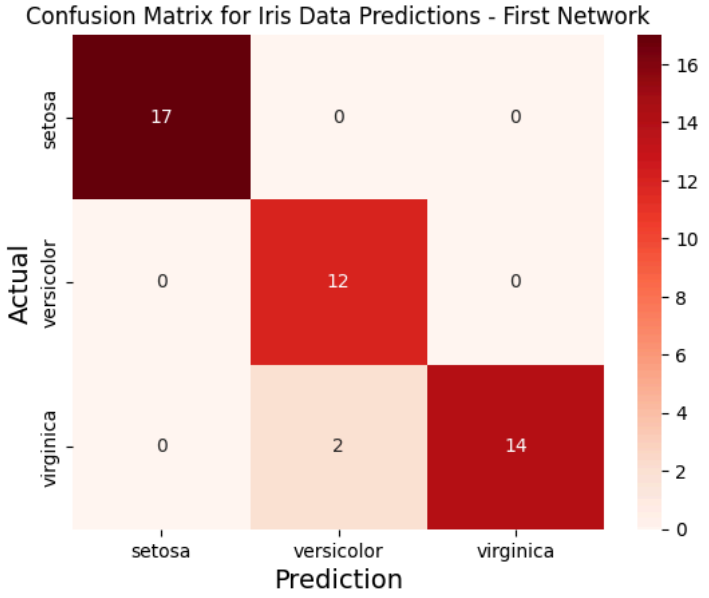
Model 3: Leaky ReLU Nonlinearity, Gradient Descent Learning

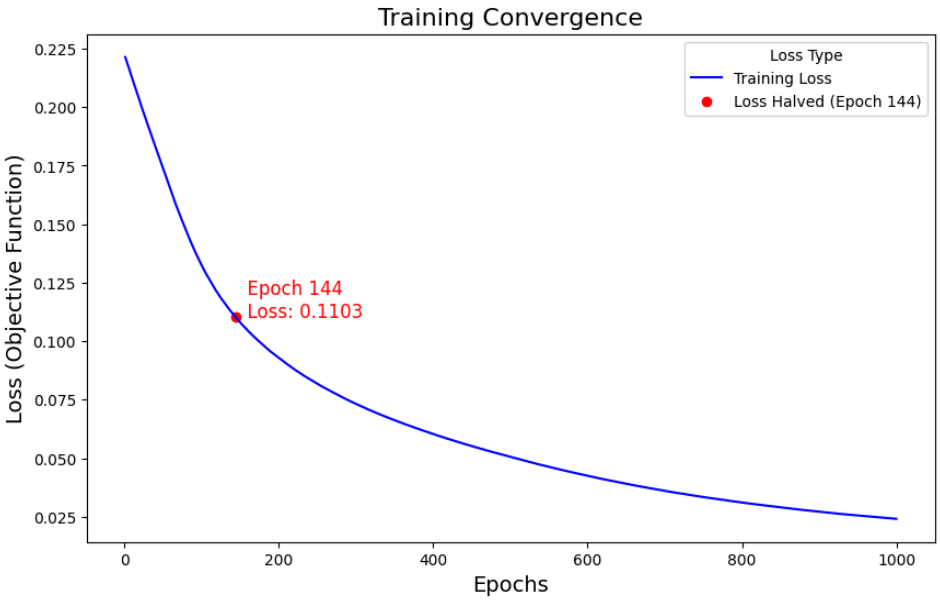
Training Accuracy: 96.19%

Test Accuracy: 95.55%

Epochs to 50% Loss Reduction: 144 Epochs

Epochs in Training: 1000 Epochs





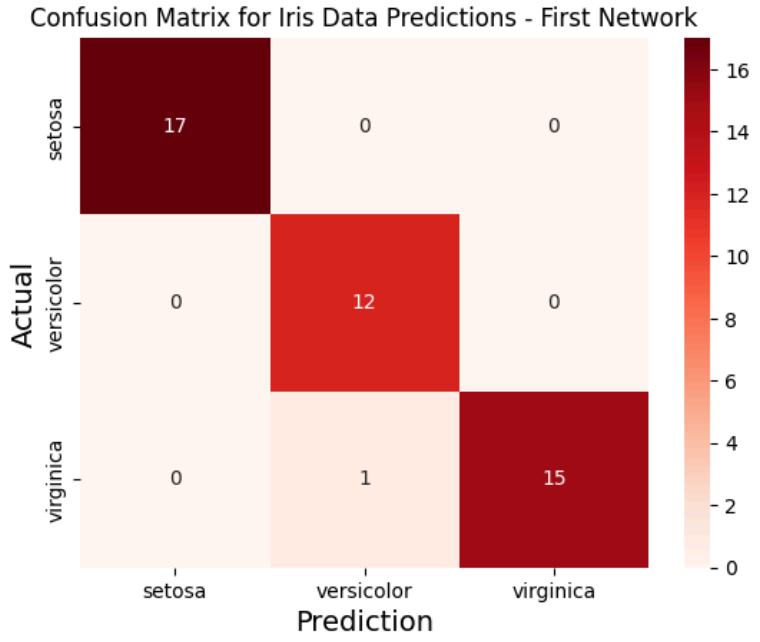
Model 4: Swish Nonlinearity, Gradient Descent Learning

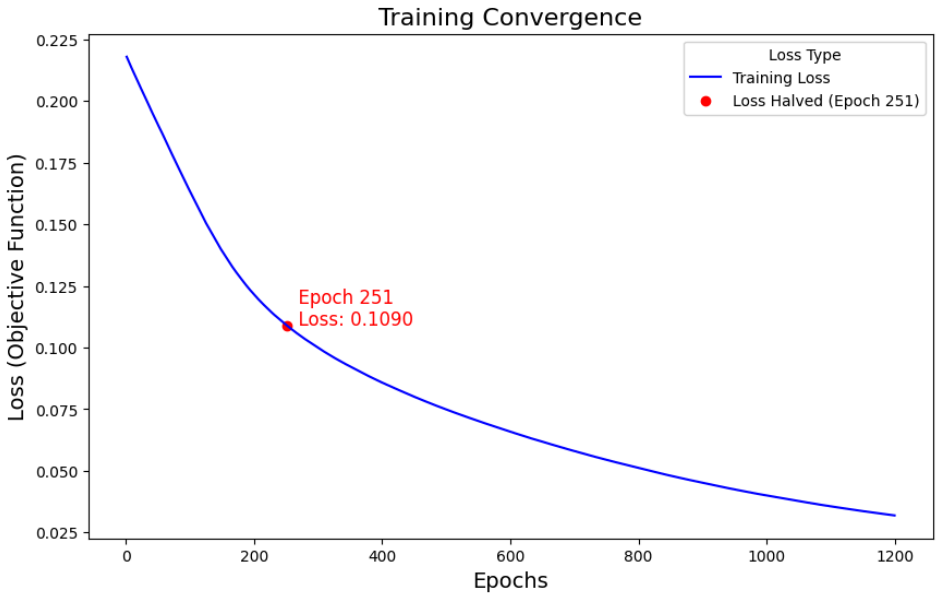
Training Accuracy: 97.14%

Test Accuracy: 97.77%

Epochs to 50% Loss Reduction: 251 Epochs

Epochs in Training: 1200 Epochs





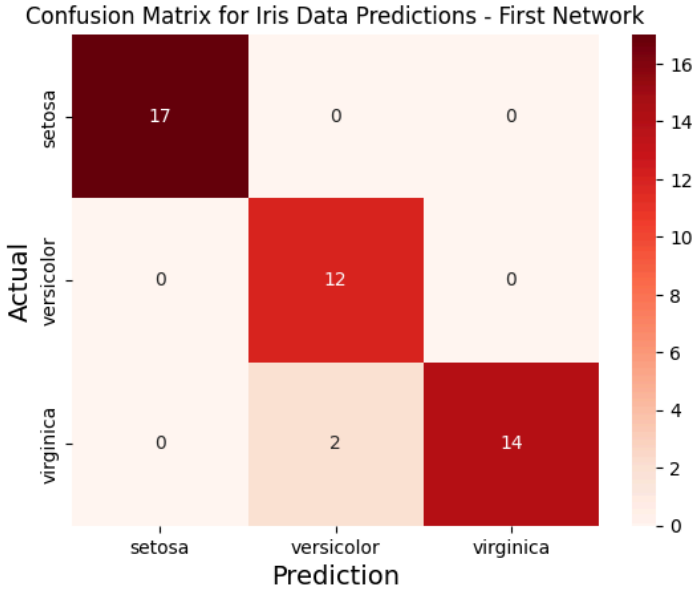
Model 5: ReLU Nonlinearity, RMSpop Learning

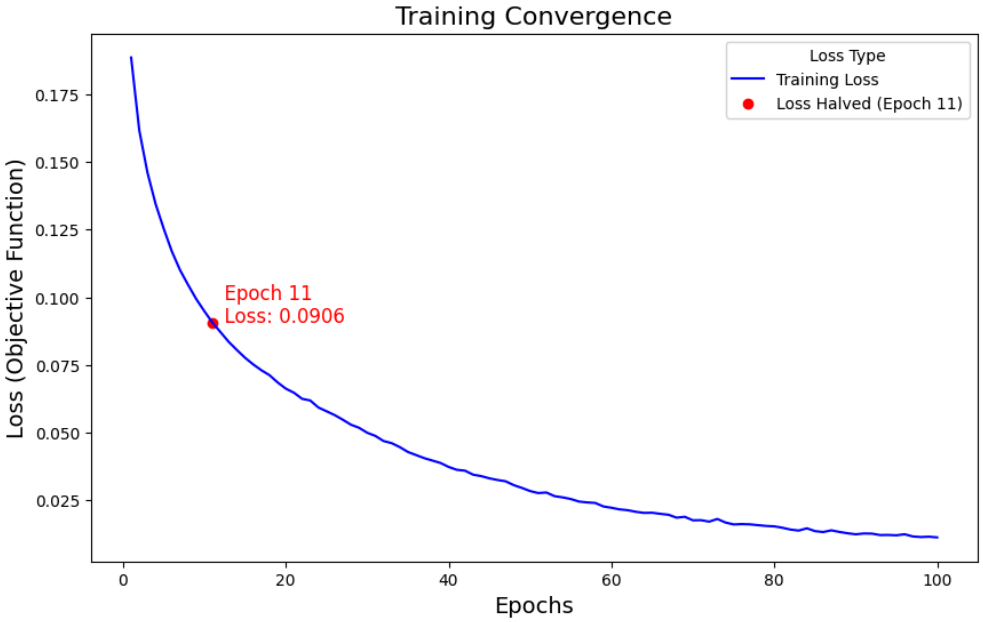
Training Accuracy: 98.10%

Test Accuracy: 95.55%

Epochs to 50% Loss Reduction: 11 Epochs

Epochs in Training: 100 Epochs





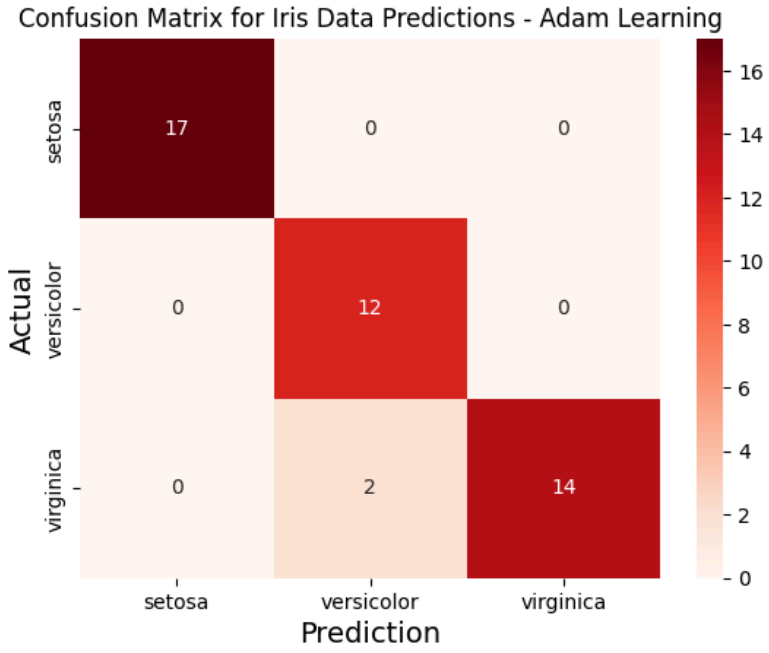
Model 6: ReLU Nonlinearity, Adam Learning

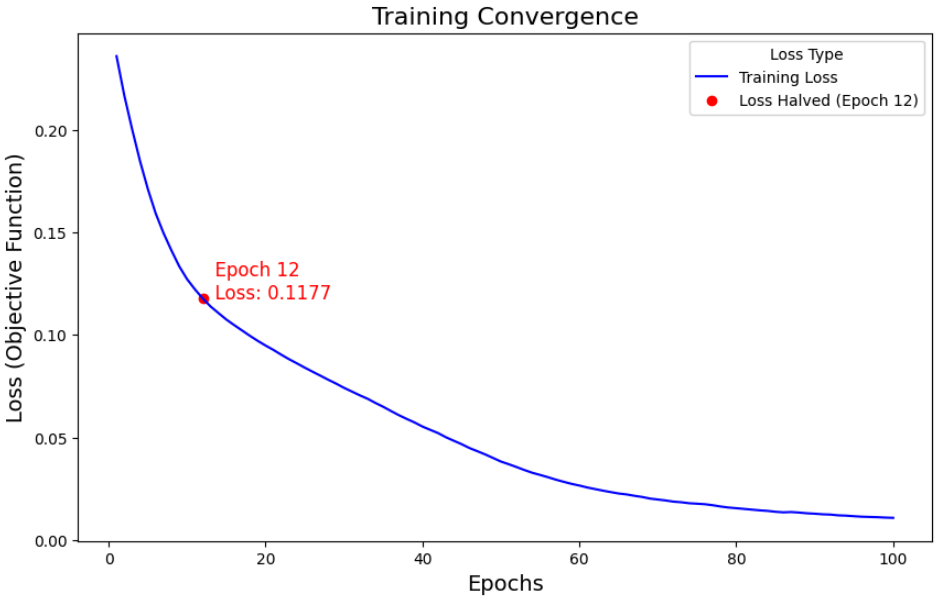
Training Accuracy: 98.10%

Test Accuracy: 95.55%

Epochs to 50% Loss Reduction: 12 Epochs

Epochs in Training: 100 Epochs





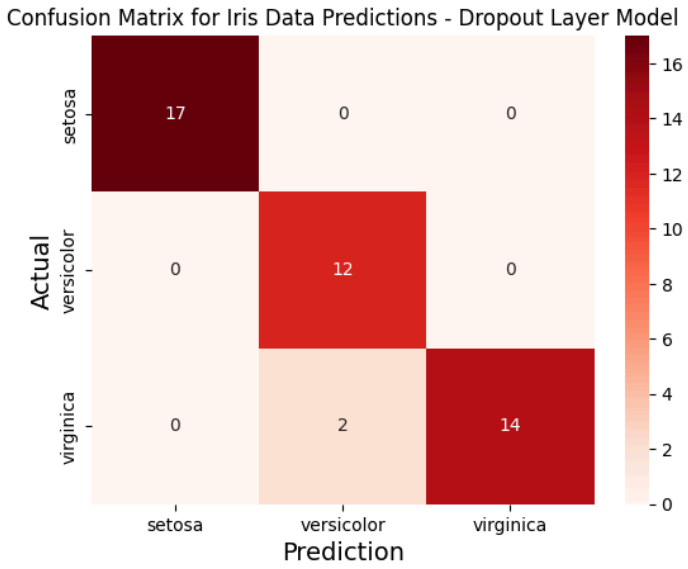
Model 7: Relu Nonlinearity, Adam Learning, Dropout Layer

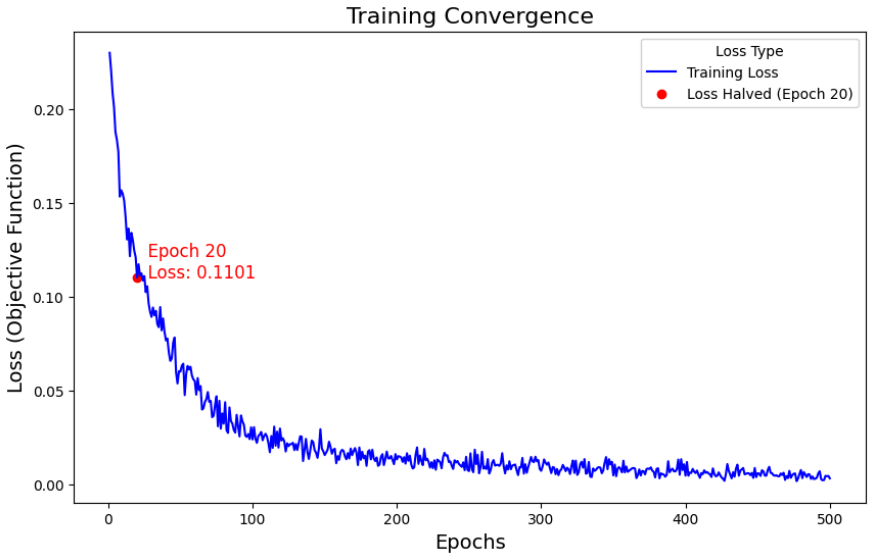
Training Accuracy: 100%

Test Accuracy: 95.55%

Epochs to 50% Loss Reduction: 20 Epochs

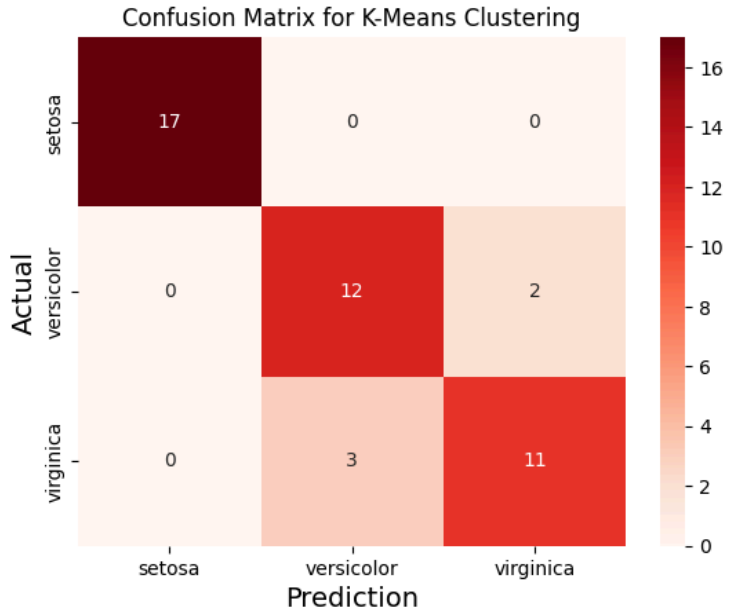
Epochs in Training: 500 Epochs





Model 8: K-Means Clustering

Test Accuracy: 88.89%



Analysis

To analyze the various models and their performance on the Iris dataset, a comparison of the results above from the Neural Network models and K-Means Clustering model is detailed below.

Sigmoid Model (Model 1):

The Sigmoid model, built with one hidden layer and using a Sigmoid nonlinearity, Gradient Descent learning function, and Mean Squared Error objective function, achieved 96.17% accuracy on the training data and 97.77% accuracy on the test data. The training took 5000 epochs, with a 50% decrease in loss after 1491 epochs. This model showed good performance, though the training time was relatively long. This slow training time paved the way for the improvements of later models.

ReLU Model (Model 2):

The ReLU model, which used a ReLU nonlinearity with the same Gradient Descent learning function and Mean Squared Error objective function, reached 95.24% accuracy on the training data and 95.55% accuracy on the test data. The training took only 1000 epochs, with loss decreasing by 50% after 136 epochs. This model showed significant improvements in training time compared to the Sigmoid model, taking less than a quarter of the epochs to halve the loss. Though it achieved slightly lower accuracy, the performance suggests that if trained longer, it could rival the Sigmoid model's accuracy.

Leaky ReLU Model (Model 3):

The Leaky ReLU model, using the same setup as the ReLU model but with a Leaky ReLU nonlinearity, achieved 96.19% accuracy on the training data and 95.55% accuracy on the test data. The training took 1000 epochs, with a 50% decrease in loss after 144 epochs. Compared to the ReLU model, there were no observable improvements, indicating that the "Dead Neurons" issue did not significantly affect the ReLU model's performance. Consequently, I decided to focus on the standard ReLU model for further investigation.

Swish Model (Model 4):

The Swish model, which used a Swish nonlinearity, achieved 97.14% accuracy on the training data and 97.77% accuracy on the test data. The training took 1200 epochs, with loss halving after 251 epochs. While it performed better than the Sigmoid model in terms of accuracy, the Swish model took nearly twice as many epochs to reduce loss by 50% compared to the ReLU model. The Swish model, developed by Google for deep networks, may not have been the best choice for this shallow network, as it did not outperform the ReLU model.

RMSProp Model (Model 5):

The RMSProp model, using ReLU nonlinearity and a Mean Squared Error objective function, achieved 98.10% accuracy on the training data and 95.55% accuracy on the test data. The model trained in 100 epochs, with loss halving after just 11 epochs. Compared to the standard ReLU model with Gradient Descent, the RMSProp model was much faster and more efficient, reducing loss much more quickly and requiring far fewer epochs for similar accuracy. Despite a slight overtraining indication (with a small gap between training and test accuracy), the RMSProp model was highly efficient.

Adam Model (Model 6):

The Adam model, using the same ReLU nonlinearity and Mean Squared Error objective function, achieved the same 98.10% accuracy on the training data and 95.55% accuracy on the test data as the RMSProp model. The training took 100 epochs, with loss reducing by 50% after 12 epochs. Performance between the RMSProp and Adam models was nearly identical, though the Adam model reduced loss by 50% slightly faster, in 12 epochs compared to 11. Similar to RMSProp, there was a small overtraining effect, but the model performed very well overall.

Dropout Layer Model (Model 7):

The Dropout Layer model, added to prevent overfitting, showed no significant improvement over the previous models and may have slightly harmed performance. It trained for 500 epochs, reaching 100% accuracy on the training set, but only 95.55% on the test set. The Dropout Layer slowed training considerably, with the loss halving in 20 epochs compared to the faster 11 and 12 epochs in the RMSProp and Adam models, respectively. The addition of the dropout layer did not prove effective in this case and seemed to exacerbate the overfitting issue.

K-Means Clustering Model (Model 8):

The K-Means model, which was trained using the same dataset, achieved 88.89% accuracy on the test data. While it correctly classified all Setosa flowers, it struggled to differentiate between Versicolor and Virginica, misclassifying five samples. In comparison, the most successful Neural Network model, using ReLU and RMSProp, achieved 95.55% accuracy on the test set. The K-Means model showed variability, with accuracies ranging from 80% to 90% depending on the training session, and performed significantly worse than the neural network models.

Conclusions

This analysis highlights the significant variations in model performance across different neural network architectures and the K-Means Clustering approach. The models examined differ in terms of accuracy, training efficiency, and their ability to generalize well to unseen data.

Neural Network Models:

- The ReLU, RMSProp, and Adam models demonstrated the best performance, delivering high accuracy with fast convergence times. These models were able to achieve significantly better results compared to the other models tested.
- By adjusting the nonlinearity and learning function, training times were drastically reduced, with the time to reduce the loss by 50% decreasing by 13,554.55% and the total model training time reducing by 5000%.
- The Swish model, despite its potential in deeper networks, performed poorly in this shallow network setup. It failed to outperform the ReLU-based models in terms of both training speed and accuracy.
- The Dropout layer, introduced to combat overfitting, did not provide the expected improvement. In fact, it slowed down training and showed no meaningful improvement, and in some cases worsened the overfitting problem.

K-Means Clustering Model:

- The K-Means Clustering model performed reasonably well at differentiating linearly separable categories like Setosa and Versicolor, but struggled with distinguishing Versicolor from Virginica. This model's performance was limited in its ability to handle more complex categorization tasks, particularly when there were subtle differences between classes.
- While K-Means was a useful clustering approach, it was significantly outperformed by the neural network models, with the neural networks achieving at least 6% higher accuracy, and in some cases, up to 16% better accuracy.

Final Conclusion:

While K-Means Clustering can be a valuable algorithm for simpler categorization tasks, the Neural Network models, particularly those using ReLU, RMSProp, and Adam, outperformed the K-Means approach in this specific dataset. The neural networks demonstrated a better ability to handle complex classification problems, particularly when distinguishing between similar classes like Versicolor and Virginica. Therefore, for this problem and dataset, the conclusion is that Neural Networks are the superior choice over K-Means Clustering.

In summary, improvements in training efficiency through the use of ReLU nonlinearity and advanced optimization methods like RMSProp and Adam have led to substantial gains in both speed and accuracy. Although dropout layers may have their place in certain scenarios, they did not improve performance in this context, underscoring the importance of carefully selecting model modifications.