HW9 Extra Credit

I used scikit-learn to perform classification on the full-iris dataset. I will show my results along with my exploration of different non-linearities and different network architectures. I followed this tutorial playlist to set up.
“Machine learning in Python with scikit-learn” by Data School.
https://youtube.com/playlist?list=PL5-da3qGB5ICeMbQuqbbCOQWcS6OYBr5A&si=kuXjcIeFNEbh4aRp

Code Explanation

I began with loading the data and scaling for better convergence. Before doing this, it often hit the maximum number of iterations. The iris dataset is available in scikit-learn

```python
iris = load_iris()
X = iris.data
y = iris.target

iris = load_iris()
X = iris.data
y = iris.target
```

I then split the data into training and testing sets and initialized the neural network. The hidden_layer_sizes determines the structure of the hidden layers. This is set as a variable to compare different network architectures. The activation was set as a variable to compare different non-linearities including ReLU, Tanh, and Sigmoid (logistic). I set early stopping to true to stop training if performance on a set stops improving.

```python
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=221)

mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                    activation=activation, solver='adam', warm_start=True,
                    early_stopping=True, validation_fraction=0.1)
```

I then trained the model iteratively. The errors are tracked for the training and testing sets to plot. The plot shows the error rate over iterations.

```python
train_errors = []
test_errors = []

for i in range(1, max_iter + 1):
       mlp.max_iter = i
       mlp.fit(x_train, y_train)
       train_errors.append(1 - mlp.score(x_train, y_train))
```

```
        test_errors.append(1 - mlp.score(x_test, y_test))

    # plot training and testing error over iterations
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, max_iter + 1), train_errors, label="Training Error",
linestyle='-')
    plt.plot(range(1, max_iter + 1), test_errors, label="Testing Error",
linestyle='-')
    plt.title(f"Nonlinearity: {activation}, hidden layers:
{hidden_layer_sizes}")
    plt.xlabel("Number of Iterations (Epochs)")
    plt.ylabel("Error Rate")
    plt.legend()
    plt.grid()
    plt.show()
```

Then I generated and plotted the confusion matrix. This compares the actual labels versus the labels predicted by the algorithm.
I also call the test_accuracies function to print the model's accuracy.

Python

```
pred = mlp.predict(X)
class_names = iris.target_names
cm = confusion_matrix(y, pred, labels=[0, 1, 2])
df_cm = df(cm, index=class_names, columns=class_names)

set(style="whitegrid", palette="muted")
set(font_scale=1.2)
plt.figure(figsize=(6, 5))
heatmap(df_cm, annot=True, annot_kws={"size": 18}, cmap="PiYG")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix", fontsize=20)
plt.show()

test_accuracies(X, y, mlp)
```

The test_accuracies function finds the accuracies of the training and testing set for a given neural network. It averages the results from 10 different times.

Python

```
def test_accuracies(X, y, mlp):
    train_accuracies = []
```

```
    test_accuracies = []
    for i in range(10):
        x_train, x_test, y_train, y_test = train_test_split(X, y,
random_state=i)
        mlp.fit(x_train, y_train)
        train_accuracies.append(mlp.score(x_train, y_train))
        test_accuracies.append(mlp.score(x_test, y_test))

    print(f"Average training accuracy: ", np.mean(train_accuracies))
    print(f"Average test accuracy: ", np.mean(test_accuracies))
```

Lastly, I tested with different configurations including varying hidden layer sizes and different activation logistics. I chose these values because (50,) represents the simplest single-layer network. (100,) represents a more complex single-layer network. (50,50) shows a deeper two-layer network but with the same neuron count per layer. (100,50) is also deep but increases neurons in one layer.

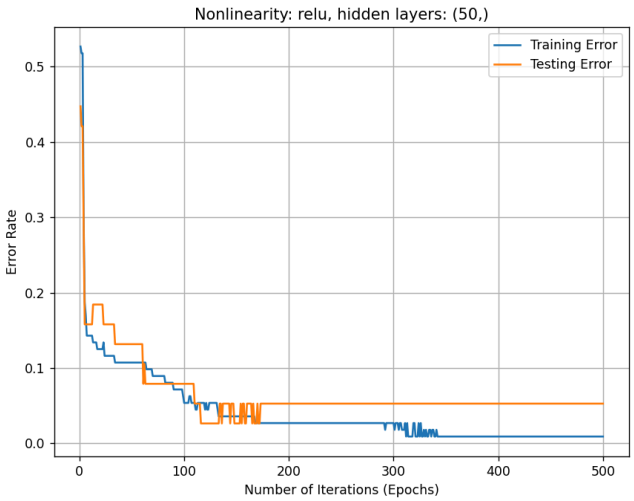
Python

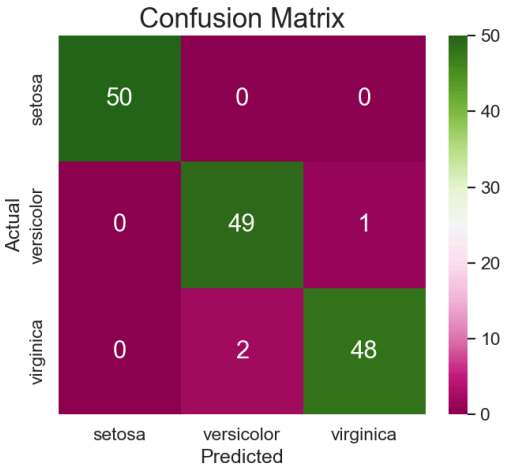
```
classify_iris((50,), 'relu')
classify_iris((100,), 'relu')
classify_iris((50, 50), 'relu')
classify_iris((100, 50), 'relu')

classify_iris((50,), 'tanh')
classify_iris((100,), 'tanh')
classify_iris((50, 50), 'tanh')
classify_iris((100, 50), 'tanh')

classify_iris((50,), 'logistic')
classify_iris((100,), 'logistic')
classify_iris((50, 50), 'logistic')
classify_iris((100, 50), 'logistic')
```

Exploring Different Non-Linearities and Different Network Architectures.
We will try to design and train a neural network classifier for the iris dataset. We will explore different nonlinearities and different network architectures and contrast these performances. The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity ReLU and one hidden layer with 50 neurons.



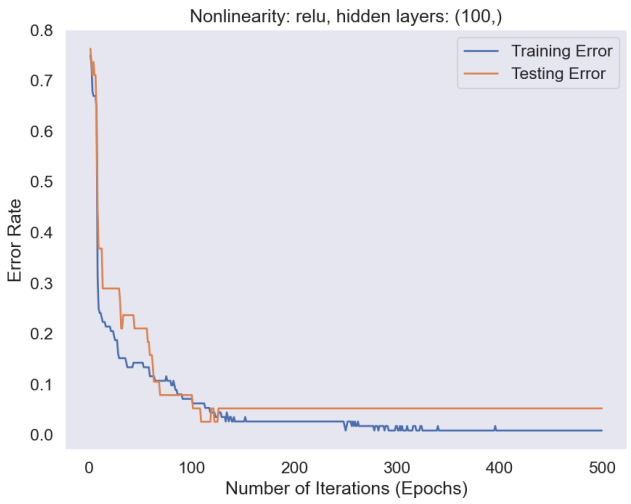


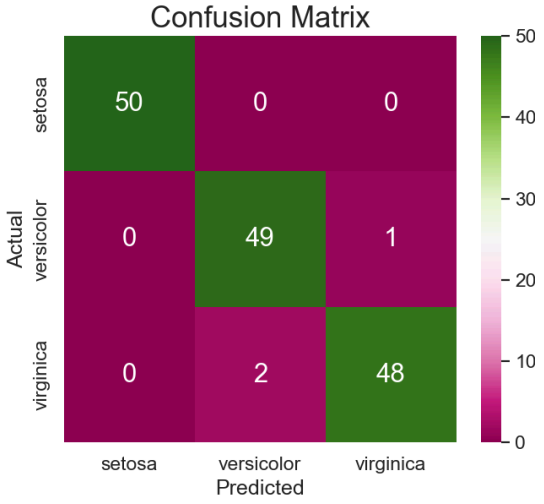
Average training accuracy: 0.9794642857142858

Average test accuracy: 0.9763157894736842

Both plots decrease sharply and stabilize around 200 iterations. The values on the diagonal show a high accuracy.

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity ReLU and one hidden layer with 100 neurons.



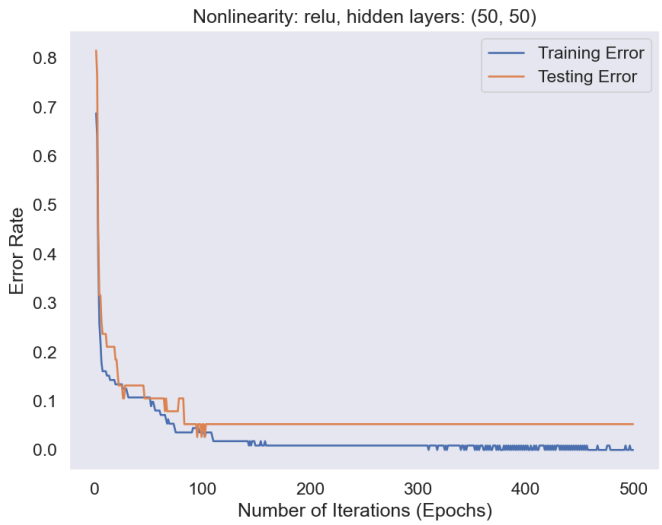


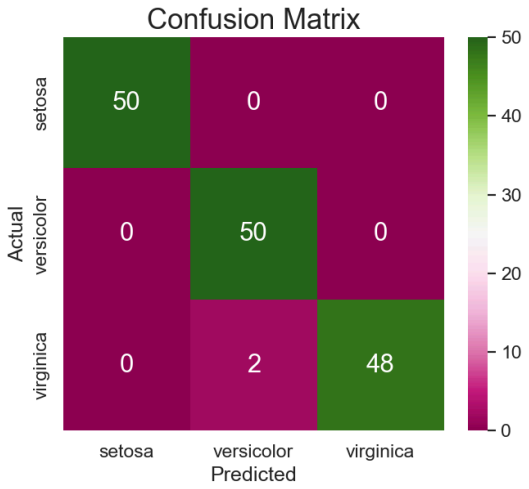
Average training accuracy: 0.9794642857142856

Average test accuracy: 0.9710526315789474

The larger hidden layer has very similar training accuracy, but slightly lower testing accuracy. This indicates slight overfitting.

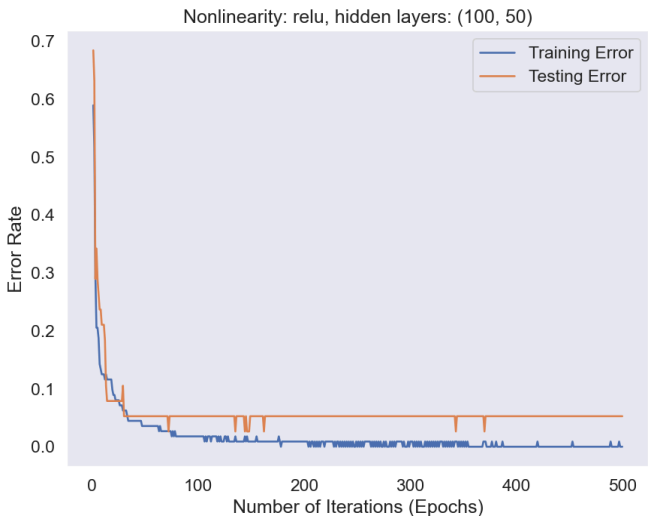
The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity ReLU and two hidden layers with 50 neurons.

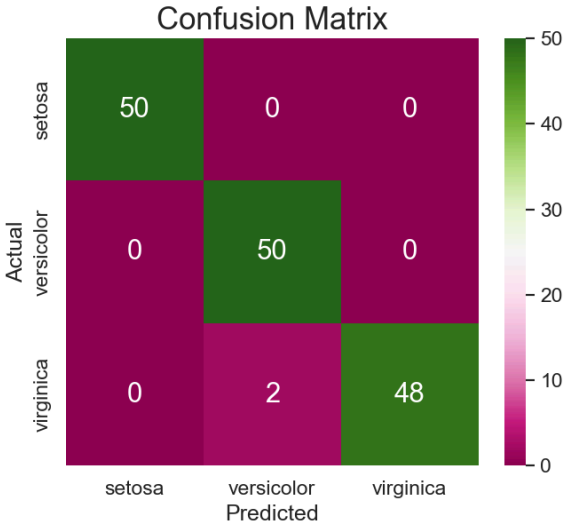




Average training accuracy: 0.9848214285714286
Average test accuracy: 0.9815789473684211
The average accuracies are higher for both sets, indicating the model fits the training data very well. This indicates low overfitting. The confusion matrix indicates that both setosa and versicolor were classified completely correctly. However, this has higher computational complexity.

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity ReLU and two hidden layers with 100 and 50 neurons respectively.

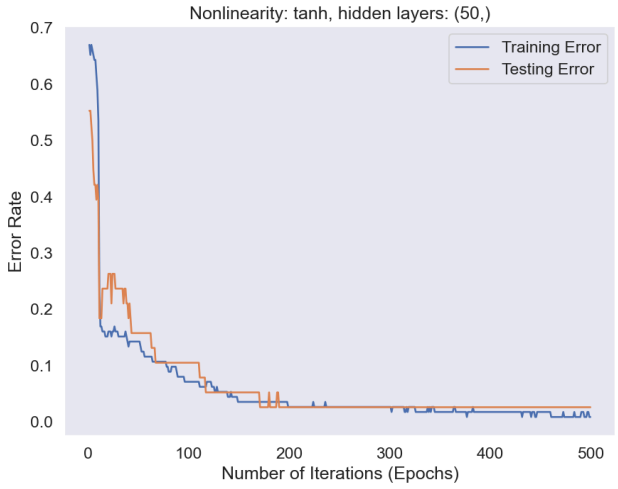


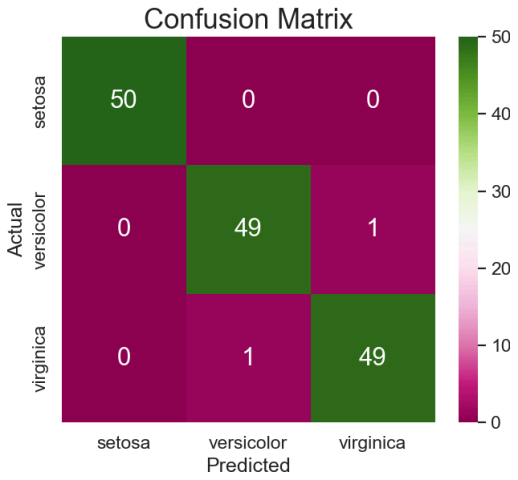


Average training accuracy: 0.9857142857142858
Average test accuracy: 0.9815789473684211

The average training accuracy has increased slightly, while the test accuracy remains the same. This shows an even stronger performance, as the model is better able to learn patterns in the data with a larger and deeper architecture.

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity Tanh and one hidden layer with 50 neurons.



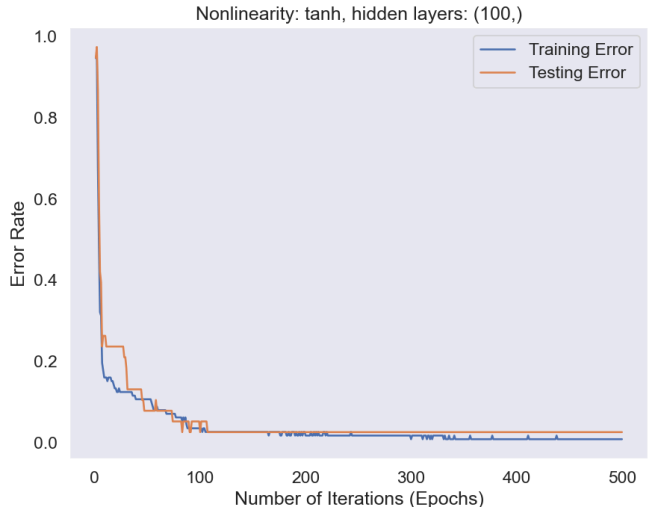


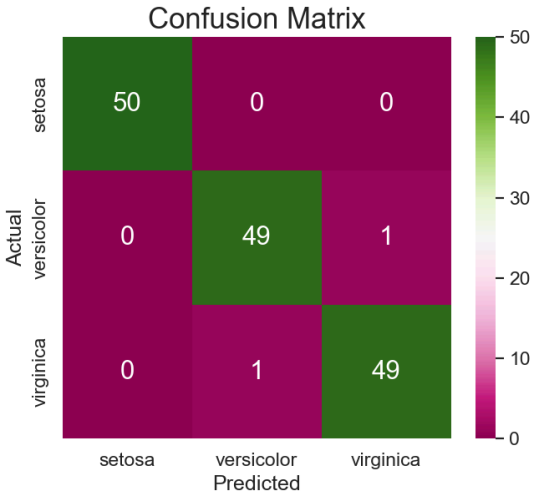
Average training accuracy: 0.98125

Average test accuracy: 0.9789473684210526

These accuracies are higher than those of the ReLU with one hidden layer with 50 neurons.

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity Tanh and one hidden layer with 100 neurons.

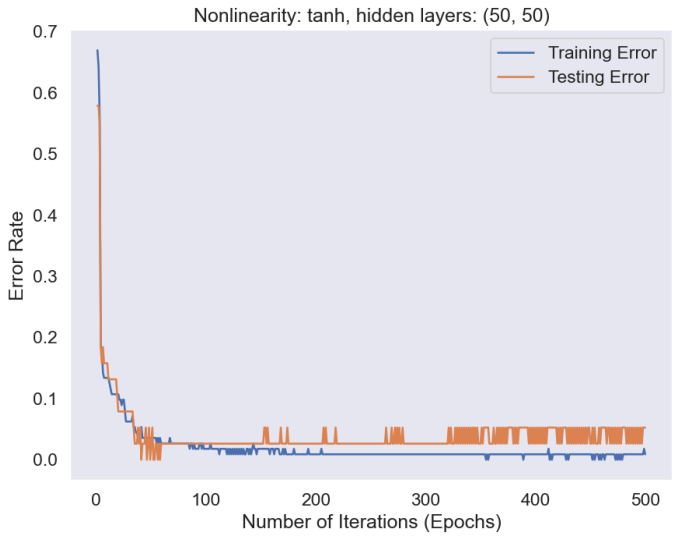


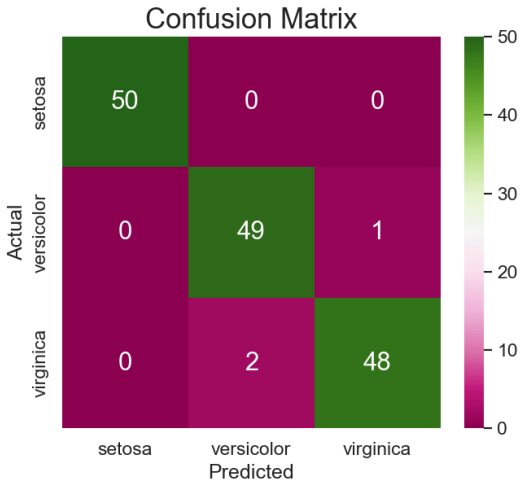


Average training accuracy: 0.98660714285714

Average test accuracy: 0.98684210526316

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity Tanh and two hidden layers with 50 neurons.

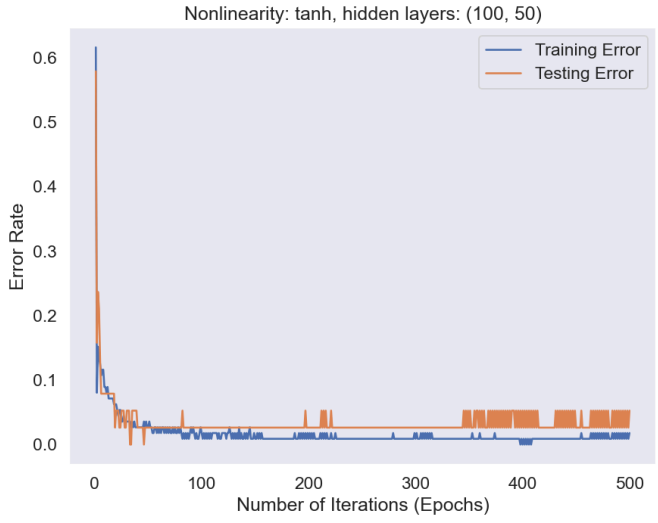


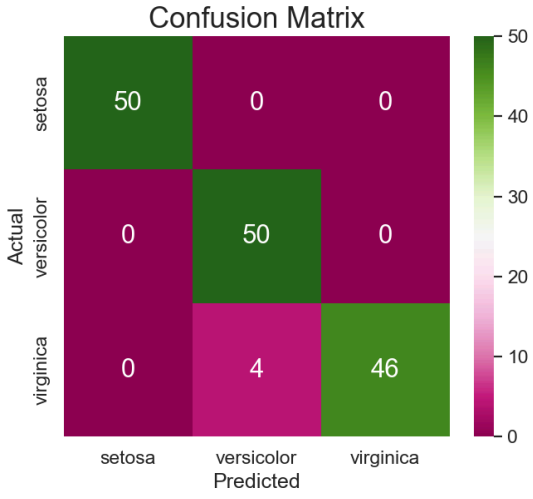


Average training accuracy: 0.9866071428571429

Average test accuracy: 0.9868421052631579

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity Tanh and two hidden layers with 100 and 50 neurons respectively.

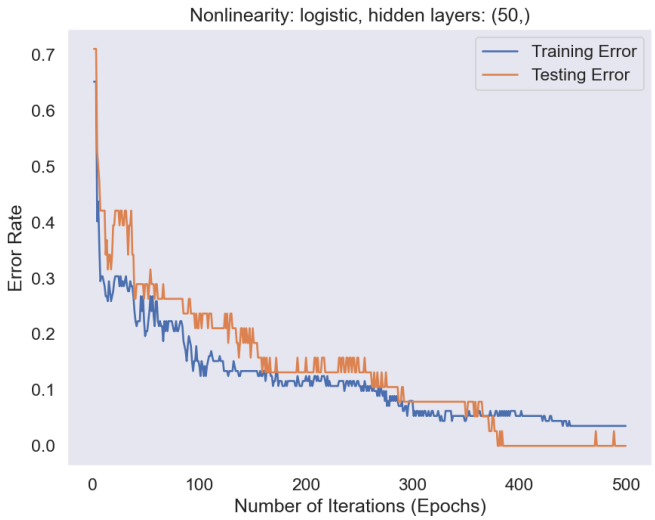


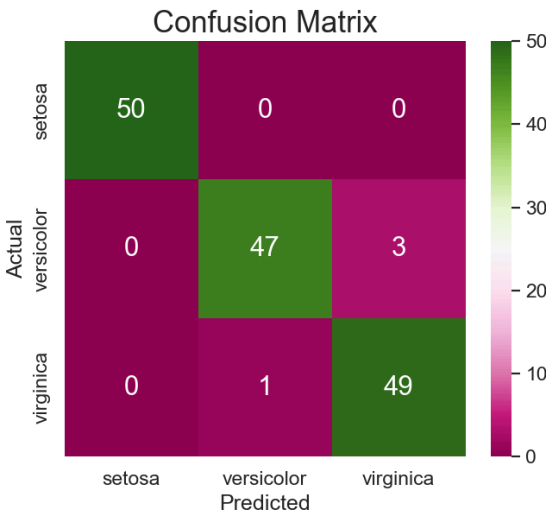


Average training accuracy: 0.9857142857142858

Average test accuracy: 0.9789473684210528

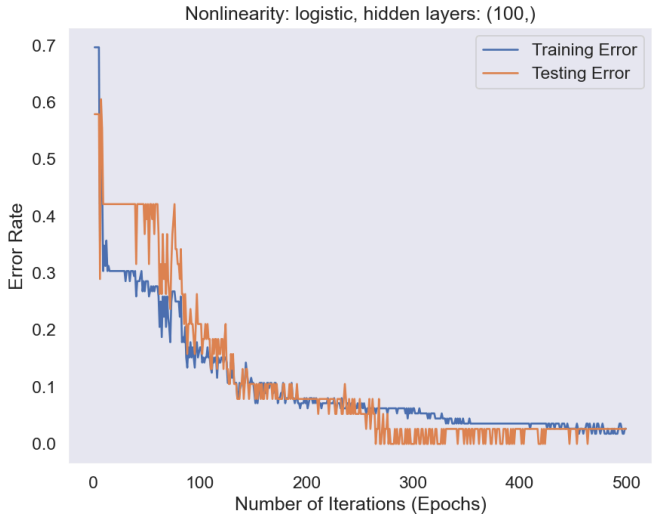
The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity sigmoid (logistic) and one hidden layer with 50 neurons.

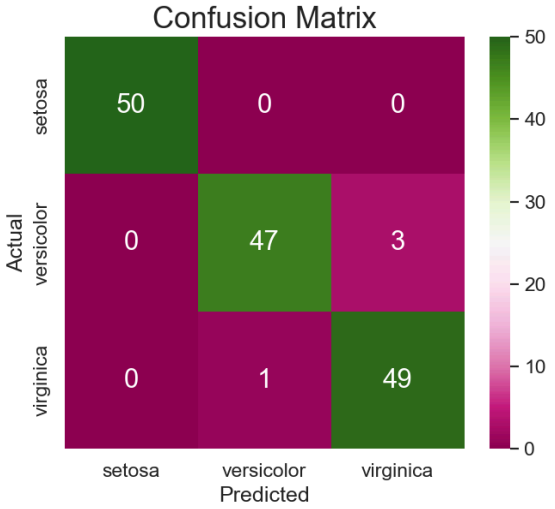




Average training accuracy: 0.9714285714285713
Average test accuracy: 0.9710526315789473

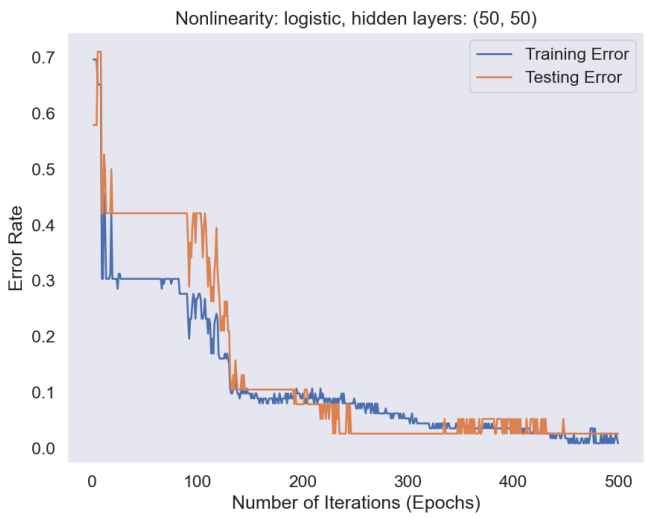
The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity sigmoid (logistic) and one hidden layer with 100 neurons.

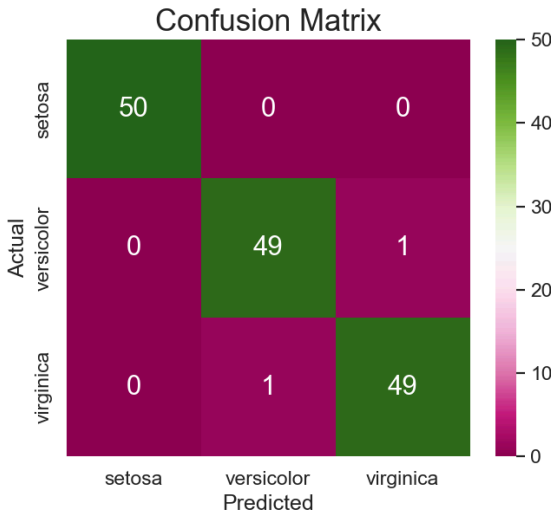




Average training accuracy: 0.9776785714285715
Average test accuracy: 0.9763157894736842

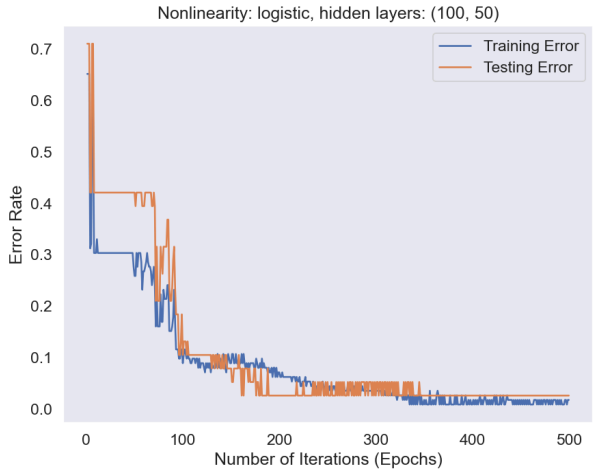
The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity sigmoid (logistic) and two hidden layers with 50 neurons.

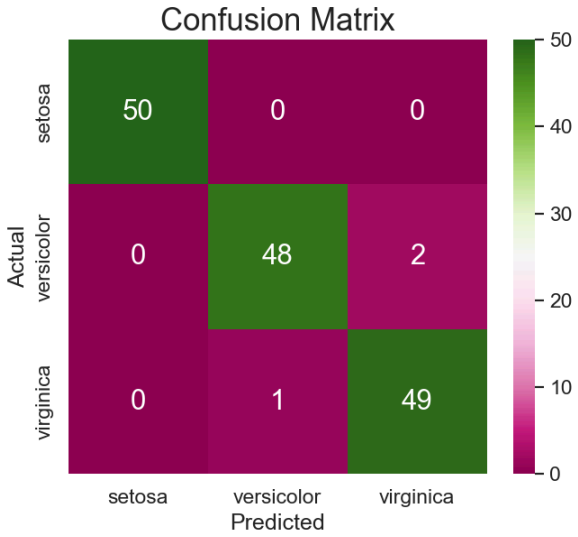




Average training accuracy: 0.9794642857142858
Average test accuracy: 0.9736842105263157

The plot shows the training error and testing error as functions of the number of iterations (epochs) for a neural network with nonlinearity sigmoid (logistic) and two hidden layers with 100 and 50 neurons respectively.





Average training accuracy: 0.9803571428571429
Average test accuracy: 0.9815789473684211

Average training accuracy
Average testing accuracy

	(50,)	(100,)	(50,50)	(100,50)
ReLU	0.9794642857143 0.9763157894737	0.97946428571429 0.97105263157895	0.9848214285714 0.9815789473684	0.9857142857143 0.9815789473684

Tanh	0.98125 0.9789473684211	0.98660714285714 0.98684210526316	0.9866071428571 0.9868421052632	0.9857142857143 0.9789473684211
Sigmoid	0.9714285714286 0.9710526315789	0.97767857142857 0.97631578947368	0.9794642857143 0.9736842105263	0.9803571428571 0.9815789473684

Overall, the best test accuracy was tanh(100,) and (50,50) with 98.68%, slightly better than ReLU. Based on these results, I found that Tanh performs well with deeper architectures. ReLU also performed well, particularly for the two-layer configurations (50,50) and (100,50) at 98.16%. Lastly, sigmoid (logistic) results were generally lower than both the others, except for (100,50) which matched ReLU's accuracy.

The deeper architectures such as (50,50) and (100,) were consistently more accurate than shallower architectures such as (50,). Tanh handled both architectures well, but as mentioned previously, deep architectures maximized accuracy. ReLU was effective for deep architectures and lower in shallow. Sigmoid (logistic) underperformed overall but improved with deeper layers.

With deep architectures, the training accuracy always increases. The testing accuracy decreases slightly, indicating slight overfitting of the data.