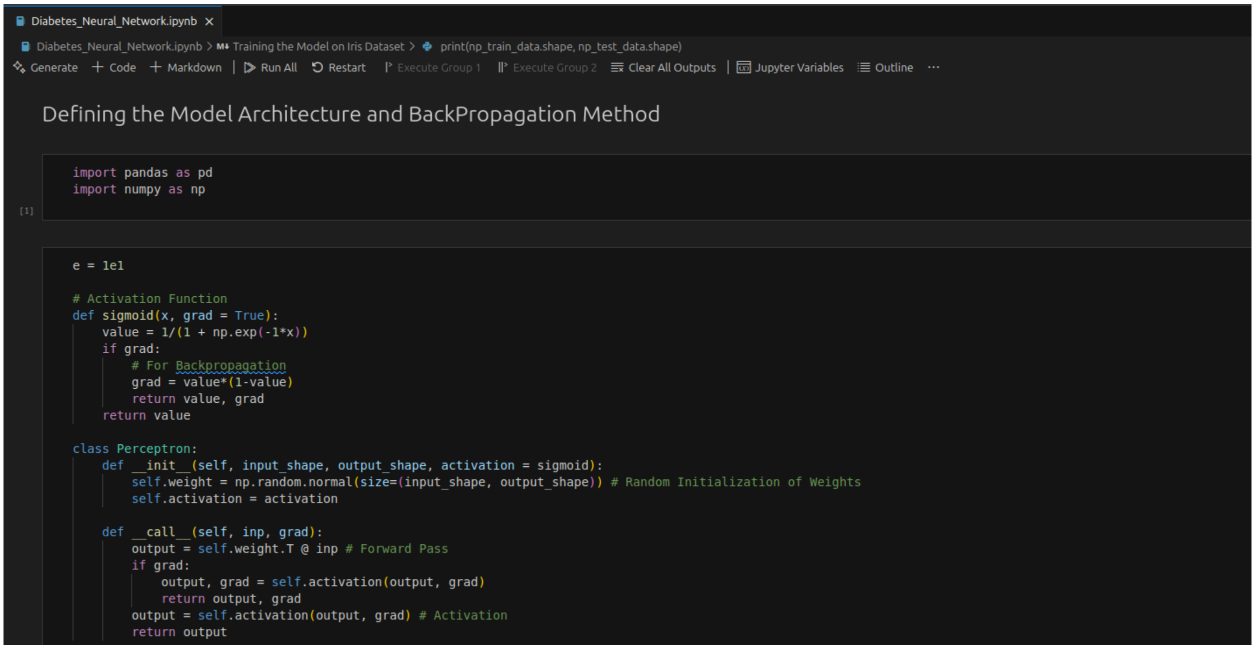
CS 354 - Assignment 6

Arnav Jain - 220002018

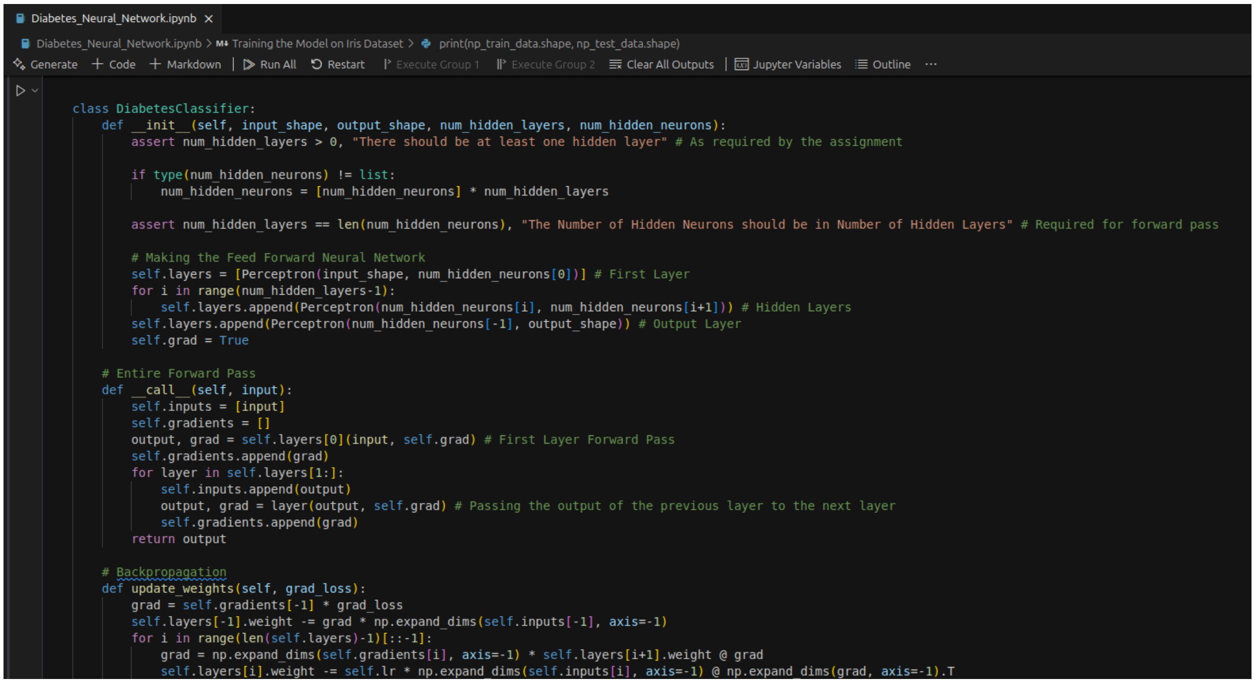
Question 1


Diabetes_Neural_Network.ipynb ×
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
···
[1]

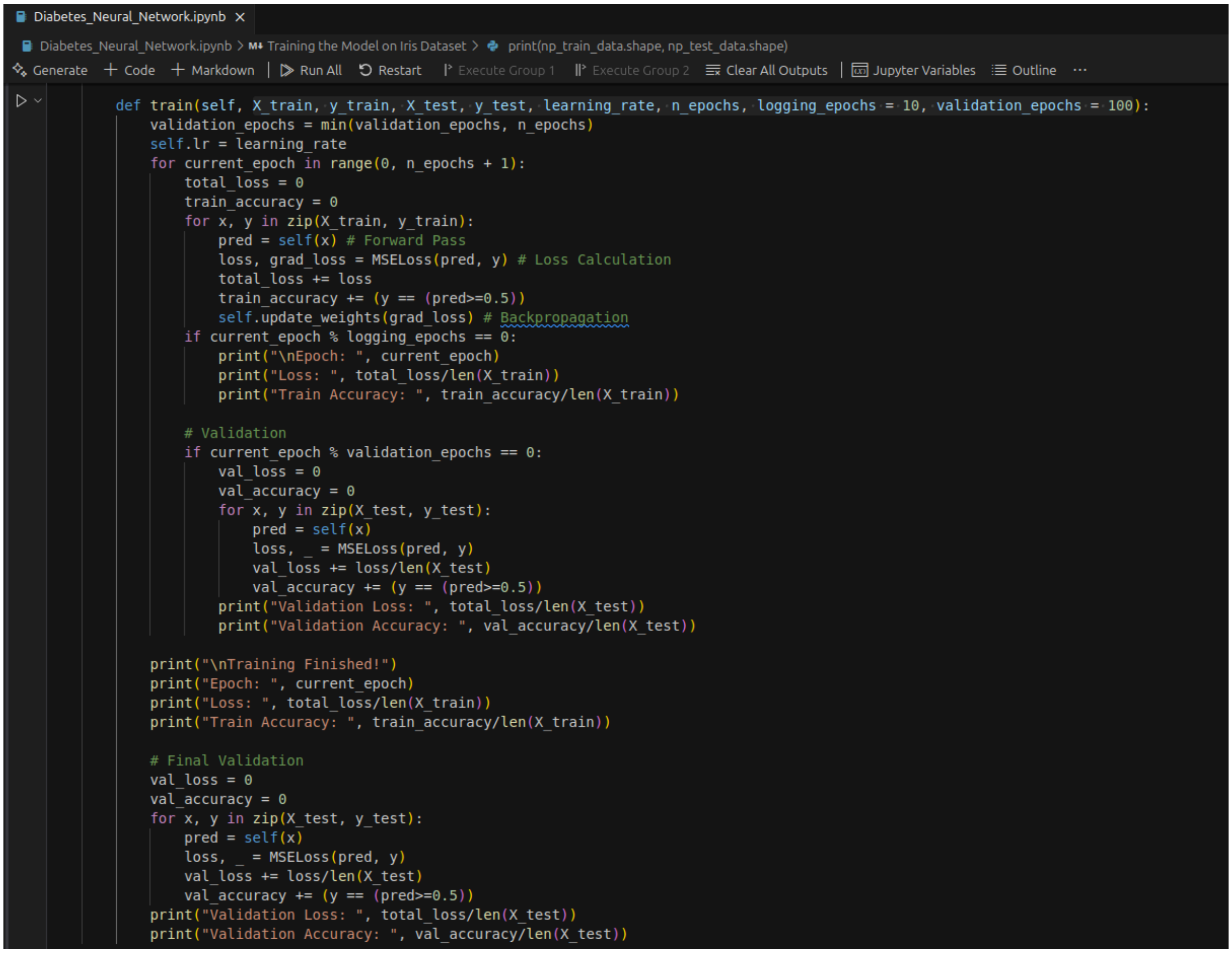
Defining the Model Architecture and BackPropagation Method

```
import pandas as pd
import numpy as np
```

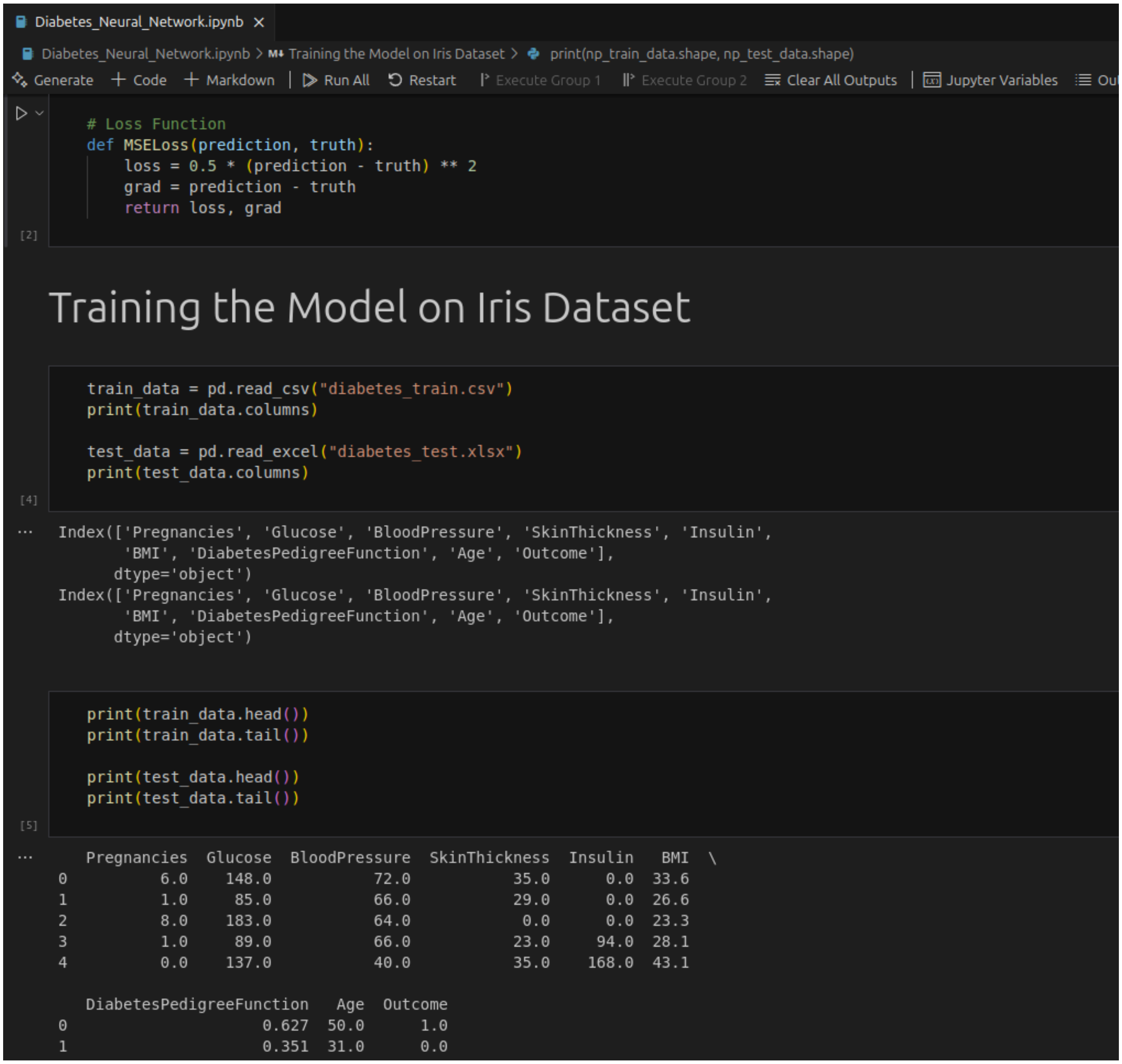
```
e = 1e1

# Activation Function
def sigmoid(x, grad = True):
    value = 1/(1 + np.exp(-1*x))
    if grad:
        # For Backpropagation
        grad = value*(1-value)
        return value, grad
    return value

class Perceptron:
    def __init__(self, input_shape, output_shape, activation = sigmoid):
        self.weight = np.random.normal(size=(input_shape, output_shape)) # Random Initialization of Weights
        self.activation = activation

    def __call__(self, inp, grad):
        output = self.weight.T @ inp # Forward Pass
        if grad:
            output, grad = self.activation(output, grad)
            return output, grad
        output = self.activation(output, grad) # Activation
        return output
```


Diabetes_Neural_Network.ipynb
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate + Code + Markdown | Run All Restart Execute Group 1 Execute Group 2 Clear All Outputs | Jupyter Variables Outline ···

```python
class DiabetesClassifier:
    def __init__(self, input_shape, output_shape, num_hidden_layers, num_hidden_neurons):
        assert num_hidden_layers > 0, "There should be at least one hidden layer" # As required by the assignment

        if type(num_hidden_neurons) != list:
            num_hidden_neurons = [num_hidden_neurons] * num_hidden_layers

        assert num_hidden_layers == len(num_hidden_neurons), "The Number of Hidden Neurons should be in Number of Hidden Layers" # Required for forward pass

        # Making the Feed Forward Neural Network
        self.layers = [Perceptron(input_shape, num_hidden_neurons[0])] # First Layer
        for i in range(num_hidden_layers-1):
            self.layers.append(Perceptron(num_hidden_neurons[i], num_hidden_neurons[i+1])) # Hidden Layers
        self.layers.append(Perceptron(num_hidden_neurons[-1], output_shape)) # Output Layer
        self.grad = True

    # Entire Forward Pass
    def __call__(self, input):
        self.inputs = [input]
        self.gradients = []
        output, grad = self.layers[0](input, self.grad) # First Layer Forward Pass
        self.gradients.append(grad)
        for layer in self.layers[1:]:
            self.inputs.append(output)
            output, grad = layer(output, self.grad) # Passing the output of the previous layer to the next layer
            self.gradients.append(grad)
        return output

    # Backpropagation
    def update_weights(self, grad_loss):
        grad = self.gradients[-1] * grad_loss
        self.layers[-1].weight -= grad * np.expand_dims(self.inputs[-1], axis=-1)
        for i in range(len(self.layers)-1)[::-1]:
            grad = np.expand_dims(self.gradients[i], axis=-1) * self.layers[i+1].weight @ grad
            self.layers[i].weight -= self.lr * np.expand_dims(self.inputs[i], axis=-1) @ np.expand_dims(grad, axis=-1).T
```


Diabetes_Neural_Network.ipynb
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
···

```python
    def train(self, X_train, y_train, X_test, y_test, learning_rate, n_epochs, logging_epochs = 10, validation_epochs = 100):
        validation_epochs = min(validation_epochs, n_epochs)
        self.lr = learning_rate
        for current_epoch in range(0, n_epochs + 1):
            total_loss = 0
            train_accuracy = 0
            for x, y in zip(X_train, y_train):
                pred = self(x) # Forward Pass
                loss, grad_loss = MSELoss(pred, y) # Loss Calculation
                total_loss += loss
                train_accuracy += (y == (pred>=0.5))
                self.update_weights(grad_loss) # Backpropagation
            if current_epoch % logging_epochs == 0:
                print("\nEpoch: ", current_epoch)
                print("Loss: ", total_loss/len(X_train))
                print("Train Accuracy: ", train_accuracy/len(X_train))

            # Validation
            if current_epoch % validation_epochs == 0:
                val_loss = 0
                val_accuracy = 0
                for x, y in zip(X_test, y_test):
                    pred = self(x)
                    loss, _ = MSELoss(pred, y)
                    val_loss += loss/len(X_test)
                    val_accuracy += (y == (pred>=0.5))
                print("Validation Loss: ", total_loss/len(X_test))
                print("Validation Accuracy: ", val_accuracy/len(X_test))

        print("\nTraining Finished!")
        print("Epoch: ", current_epoch)
        print("Loss: ", total_loss/len(X_train))
        print("Train Accuracy: ", train_accuracy/len(X_train))

        # Final Validation
        val_loss = 0
        val_accuracy = 0
        for x, y in zip(X_test, y_test):
            pred = self(x)
            loss, _ = MSELoss(pred, y)
            val_loss += loss/len(X_test)
            val_accuracy += (y == (pred>=0.5))
        print("Validation Loss: ", total_loss/len(X_test))
        print("Validation Accuracy: ", val_accuracy/len(X_test))
```


Diabetes_Neural_Network.ipynb ×
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Ou
[2]
[4]
[5]

```
# Loss Function
def MSELoss(prediction, truth):
    loss = 0.5 * (prediction - truth) ** 2
    grad = prediction - truth
    return loss, grad
```

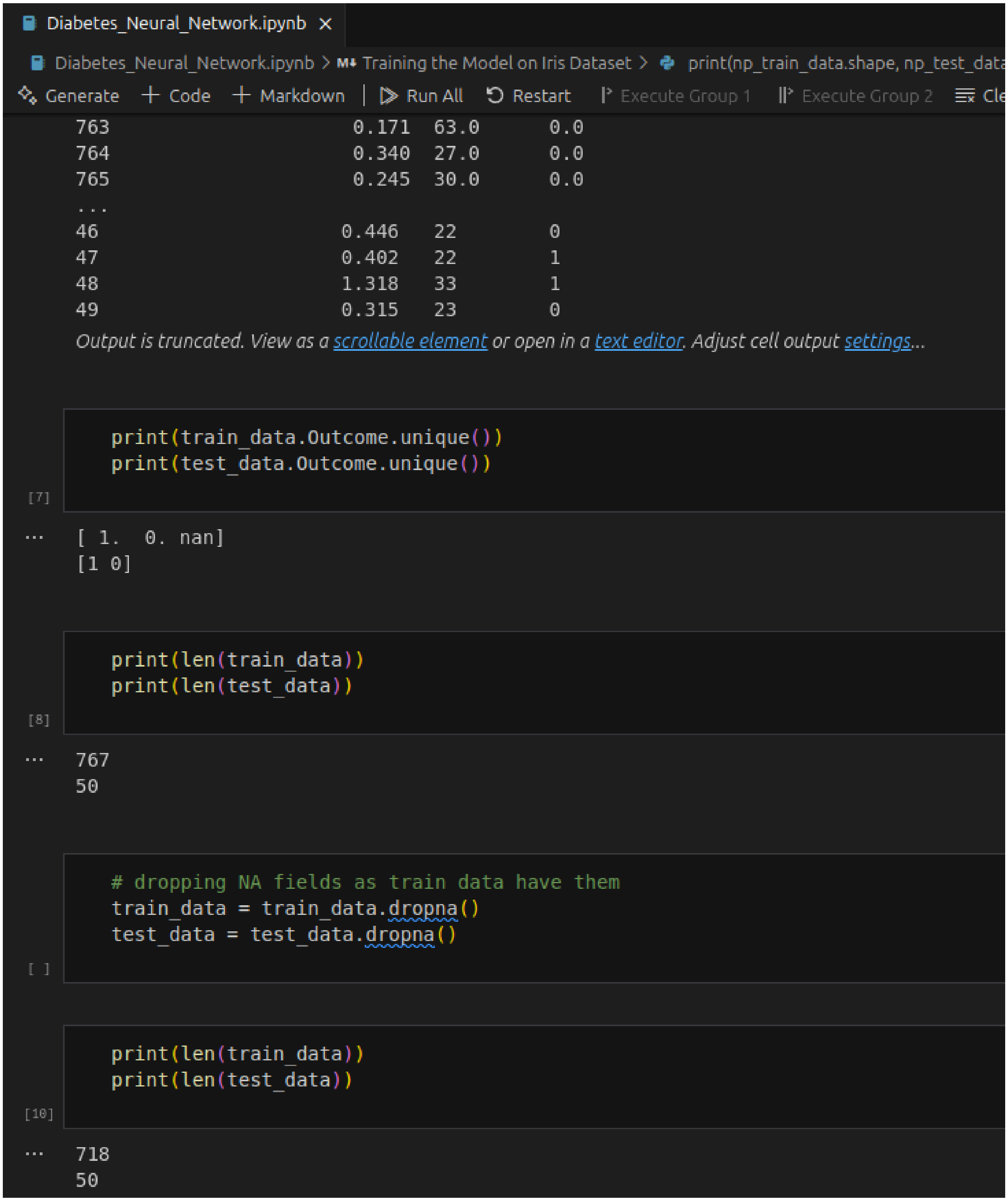
Training the Model on Iris Dataset

```
train_data = pd.read_csv("diabetes_train.csv")
print(train_data.columns)

test_data = pd.read_excel("diabetes_test.xlsx")
print(test_data.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
print(train_data.head())
print(train_data.tail())

print(test_data.head())
print(test_data.tail())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0          6.0    148.0           72.0           35.0      0.0  33.6
1          1.0     85.0           66.0           29.0      0.0  26.6
2          8.0    183.0           64.0            0.0      0.0  23.3
3          1.0     89.0           66.0           23.0     94.0  28.1
4          0.0    137.0           40.0           35.0    168.0  43.1

   DiabetesPedigreeFunction   Age  Outcome
0                     0.627  50.0      1.0
1                     0.351  31.0      0.0
```


Diabetes_Neural_Network.ipynb ×
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Cle

```
763                     0.171  63.0       0.0
764                     0.340  27.0       0.0
765                     0.245  30.0       0.0
...
46                     0.446   22         0
47                     0.402   22         1
48                     1.318   33         1
49                     0.315   23         0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

[7]

```
print(train_data.Outcome.unique())
print(test_data.Outcome.unique())
```

```
[ 1.  0. nan]
[1 0]
```

[8]

```
print(len(train_data))
print(len(test_data))
```

```
767
50
```

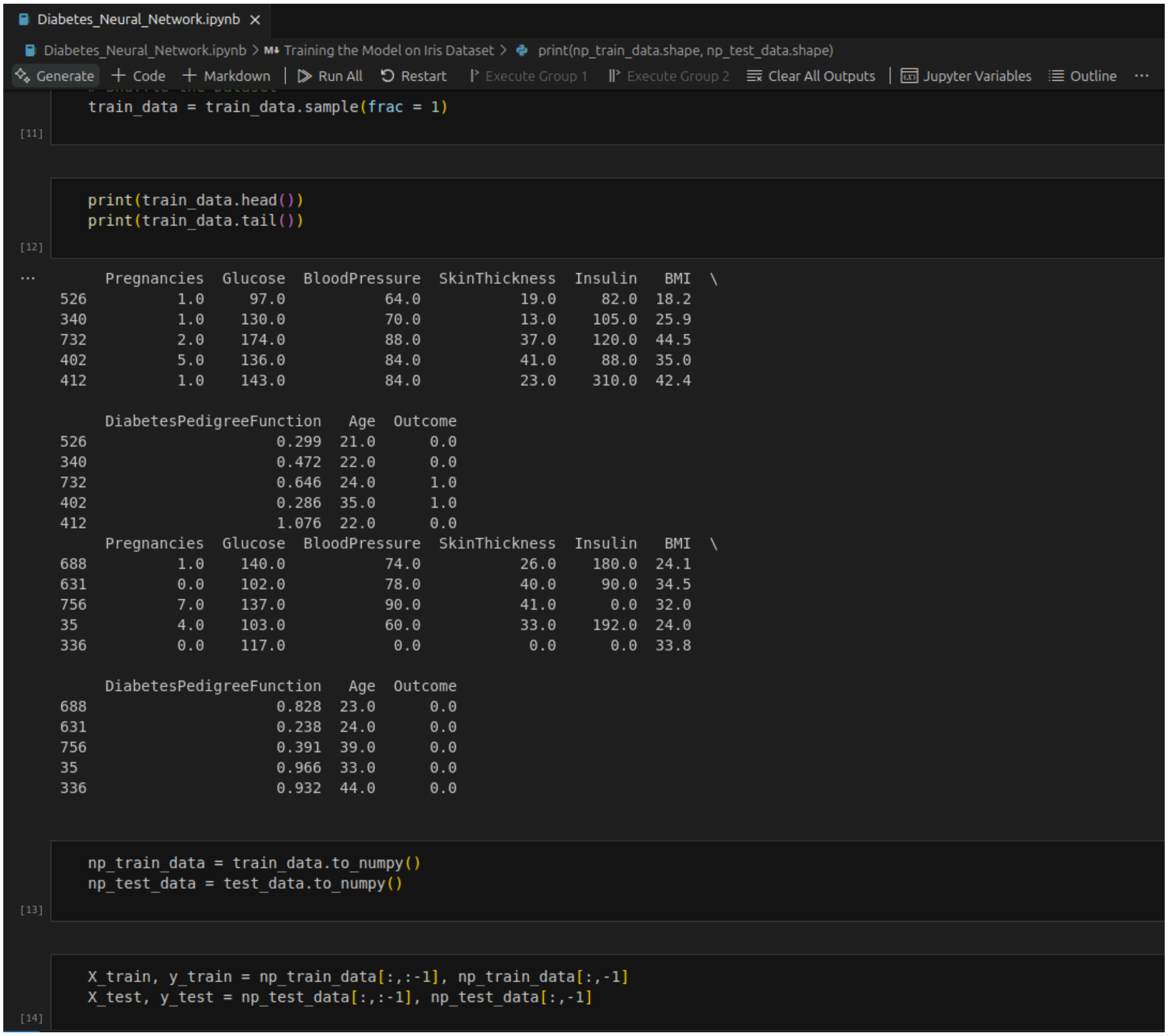
[ ]

```
# dropping NA fields as train data have them
train_data = train_data.dropna()
test_data = test_data.dropna()
```

[10]

```
print(len(train_data))
print(len(test_data))
```

```
718
50
```


Diabetes_Neural_Network.ipynb ×
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
…

```
train_data = train_data.sample(frac = 1)
```

[11]

```
print(train_data.head())
print(train_data.tail())
```

[12]

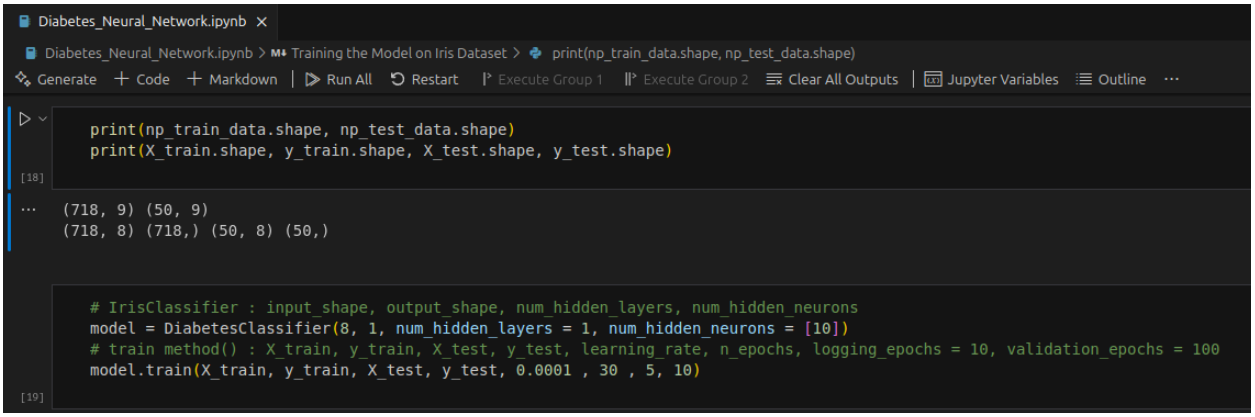
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
526          1.0     97.0           64.0           19.0     82.0  18.2
340          1.0    130.0           70.0           13.0    105.0  25.9
732          2.0    174.0           88.0           37.0    120.0  44.5
402          5.0    136.0           84.0           41.0     88.0  35.0
412          1.0    143.0           84.0           23.0    310.0  42.4

     DiabetesPedigreeFunction   Age  Outcome
526                     0.299  21.0      0.0
340                     0.472  22.0      0.0
732                     0.646  24.0      1.0
402                     0.286  35.0      1.0
412                     1.076  22.0      0.0
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
688          1.0    140.0           74.0           26.0    180.0  24.1
631          0.0    102.0           78.0           40.0     90.0  34.5
756          7.0    137.0           90.0           41.0      0.0  32.0
35           4.0    103.0           60.0           33.0    192.0  24.0
336          0.0    117.0            0.0            0.0      0.0  33.8

     DiabetesPedigreeFunction   Age  Outcome
688                     0.828  23.0      0.0
631                     0.238  24.0      0.0
756                     0.391  39.0      0.0
35                      0.966  33.0      0.0
336                     0.932  44.0      0.0
```

```
np_train_data = train_data.to_numpy()
np_test_data = test_data.to_numpy()
```

[13]

```
X_train, y_train = np_train_data[:,:-1], np_train_data[:,-1]
X_test, y_test = np_test_data[:,:-1], np_test_data[:,-1]
```

[14]


Diabetes_Neural_Network.ipynb ×
Diabetes_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
…

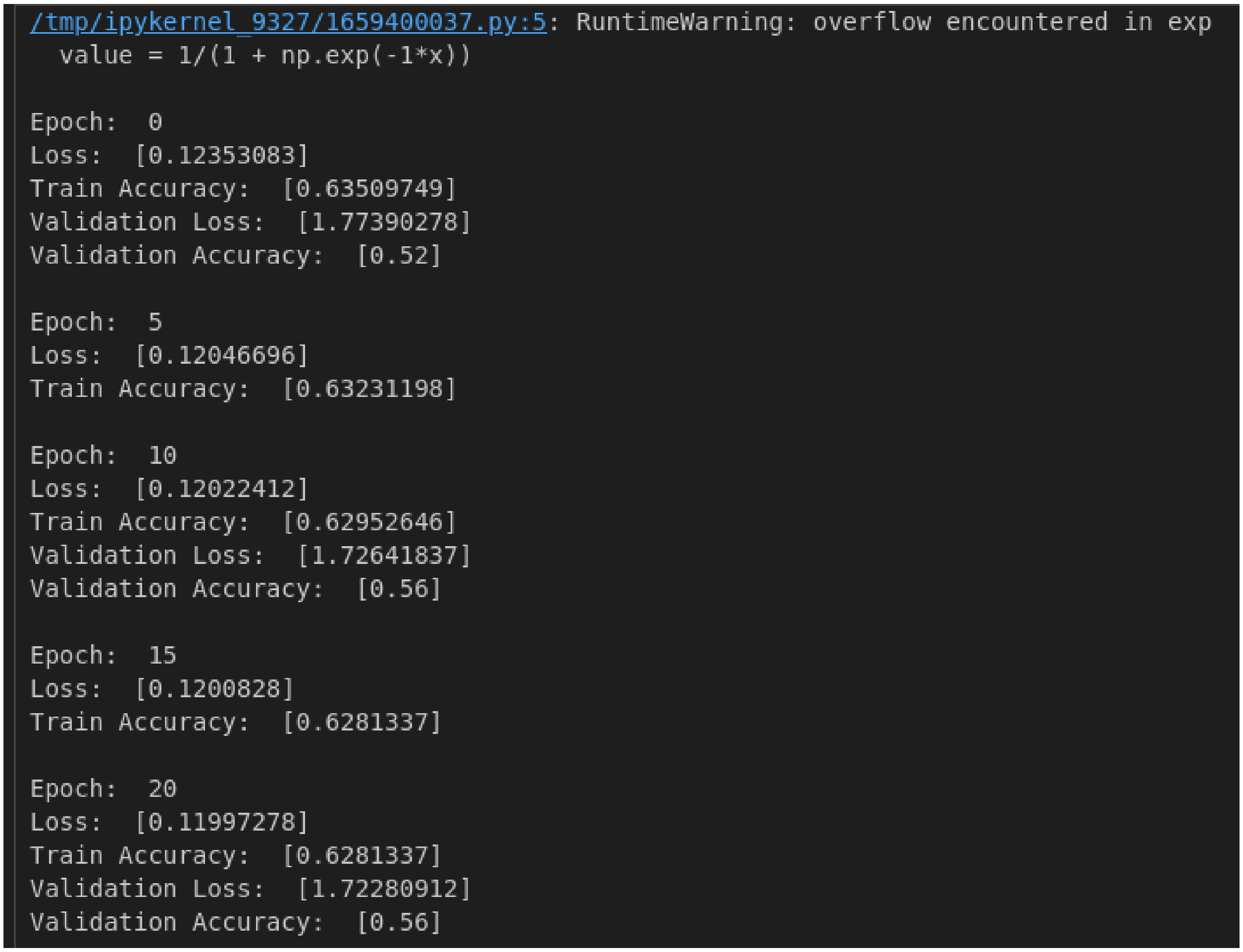
```
print(np_train_data.shape, np_test_data.shape)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

[18]

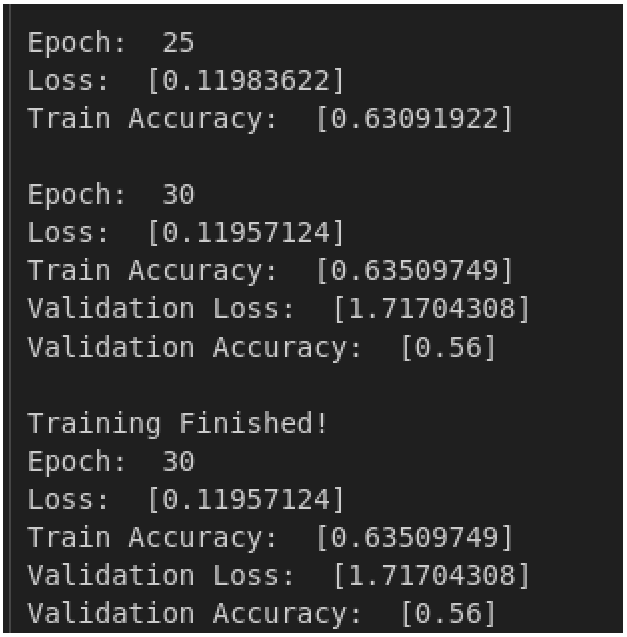
```
(718, 9) (50, 9)
(718, 8) (718,) (50, 8) (50,)
```

```
# IrisClassifier : input_shape, output_shape, num_hidden_layers, num_hidden_neurons
model = DiabetesClassifier(8, 1, num_hidden_layers = 1, num_hidden_neurons = [10])
# train method() : X_train, y_train, X_test, y_test, learning_rate, n_epochs, logging_epochs = 10, validation_epochs = 100
model.train(X_train, y_train, X_test, y_test, 0.0001 , 30 , 5, 10)
```

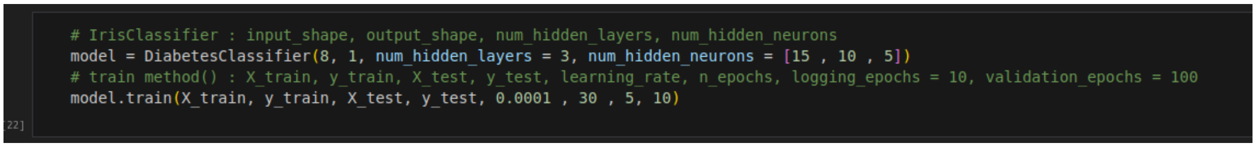
[19]



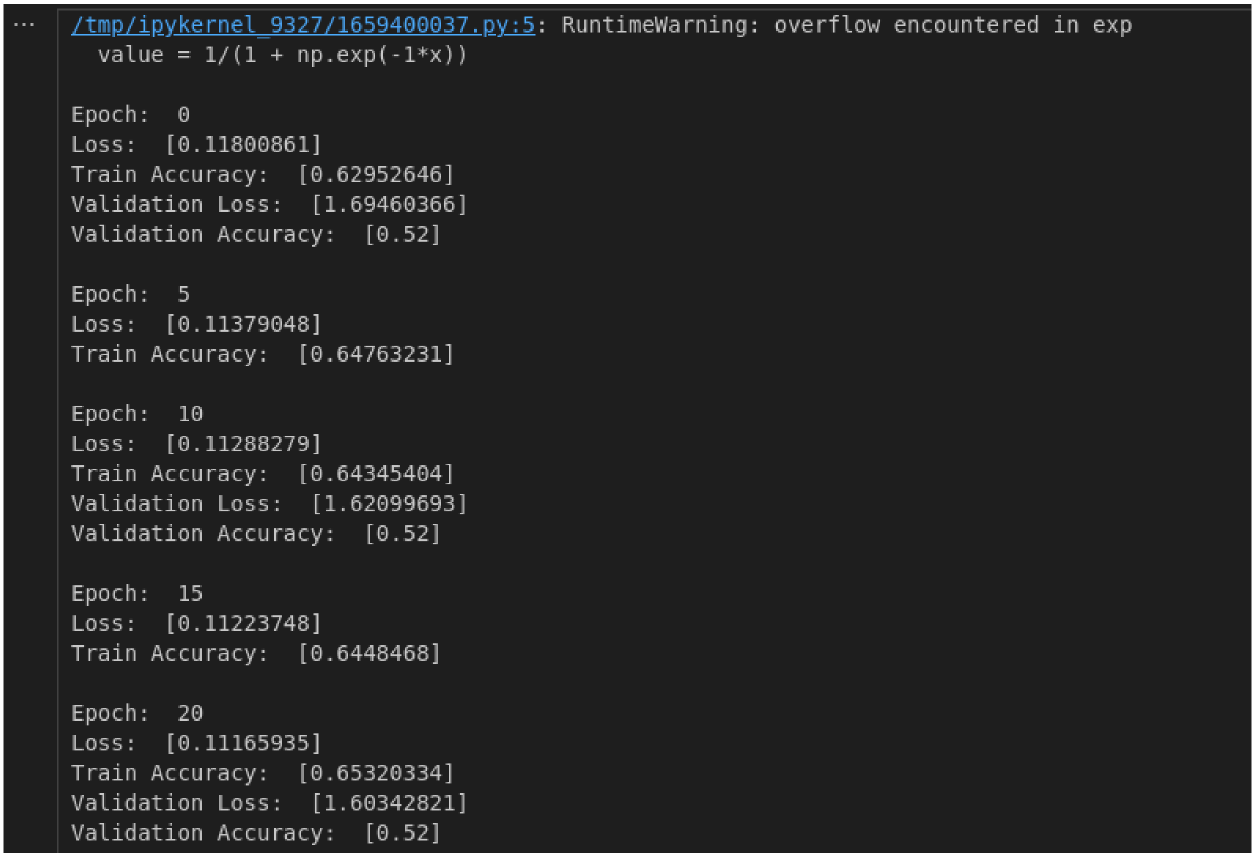
```
/tmp/ipykernel_9327/1659400037.py:5: RuntimeWarning: overflow encountered in exp
  value = 1/(1 + np.exp(-1*x))

Epoch:  0
Loss:  [0.12353083]
Train Accuracy:  [0.63509749]
Validation Loss:  [1.77390278]
Validation Accuracy:  [0.52]

Epoch:  5
Loss:  [0.12046696]
Train Accuracy:  [0.63231198]

Epoch:  10
Loss:  [0.12022412]
Train Accuracy:  [0.62952646]
Validation Loss:  [1.72641837]
Validation Accuracy:  [0.56]

Epoch:  15
Loss:  [0.1200828]
Train Accuracy:  [0.6281337]

Epoch:  20
Loss:  [0.11997278]
Train Accuracy:  [0.6281337]
Validation Loss:  [1.72280912]
Validation Accuracy:  [0.56]
```



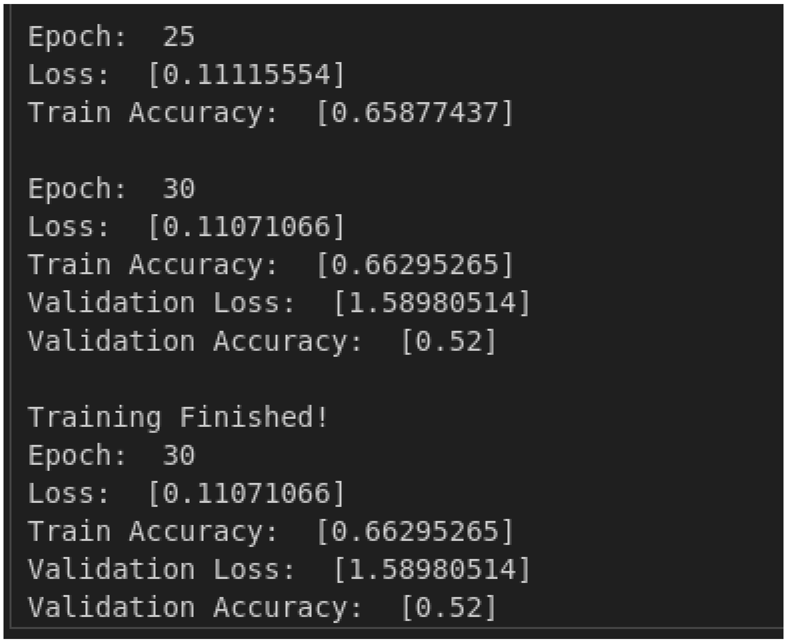
```
Epoch:  25
Loss:  [0.11983622]
Train Accuracy:  [0.63091922]

Epoch:  30
Loss:  [0.11957124]
Train Accuracy:  [0.63509749]
Validation Loss:  [1.71704308]
Validation Accuracy:  [0.56]

Training Finished!
Epoch:  30
Loss:  [0.11957124]
Train Accuracy:  [0.63509749]
Validation Loss:  [1.71704308]
Validation Accuracy:  [0.56]
```



```
# IrisClassifier : input_shape, output_shape, num_hidden_layers, num_hidden_neurons
model = DiabetesClassifier(8, 1, num_hidden_layers = 3, num_hidden_neurons = [15 , 10 , 5])
# train method() : X_train, y_train, X_test, y_test, learning_rate, n_epochs, logging_epochs = 10, validation_epochs = 100
model.train(X_train, y_train, X_test, y_test, 0.0001 , 30 , 5, 10)
```

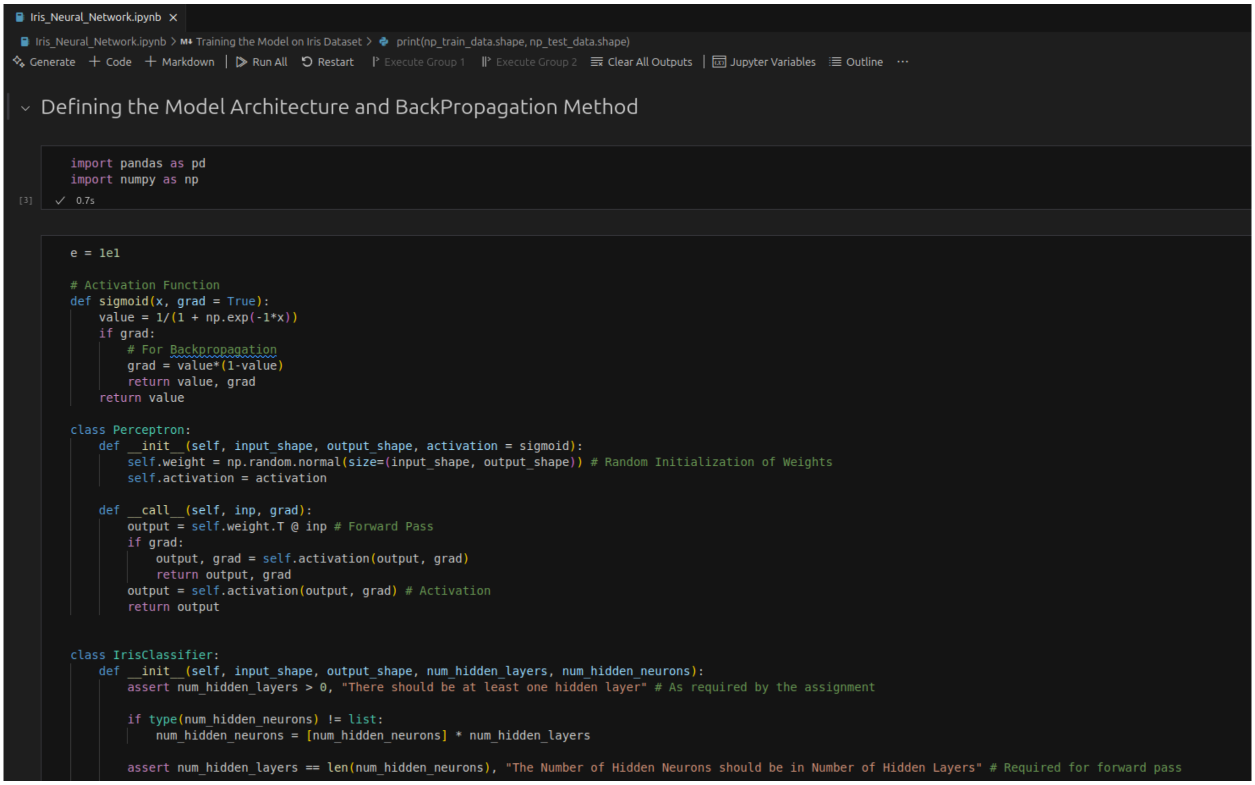


```
/tmp/ipykernel_9327/1659400037.py:5: RuntimeWarning: overflow encountered in exp
  value = 1/(1 + np.exp(-1*x))

Epoch:  0
Loss:  [0.11800861]
Train Accuracy:  [0.62952646]
Validation Loss:  [1.69460366]
Validation Accuracy:  [0.52]

Epoch:  5
Loss:  [0.11379048]
Train Accuracy:  [0.64763231]

Epoch:  10
Loss:  [0.11288279]
Train Accuracy:  [0.64345404]
Validation Loss:  [1.62099693]
Validation Accuracy:  [0.52]

Epoch:  15
Loss:  [0.11223748]
Train Accuracy:  [0.6448468]

Epoch:  20
Loss:  [0.11165935]
Train Accuracy:  [0.65320334]
Validation Loss:  [1.60342821]
Validation Accuracy:  [0.52]
```



```
Epoch:  25
Loss:  [0.11115554]
Train Accuracy:  [0.65877437]

Epoch:  30
Loss:  [0.11071066]
Train Accuracy:  [0.66295265]
Validation Loss:  [1.58980514]
Validation Accuracy:  [0.52]

Training Finished!
Epoch:  30
Loss:  [0.11071066]
Train Accuracy:  [0.66295265]
Validation Loss:  [1.58980514]
Validation Accuracy:  [0.52]
```

Question 2

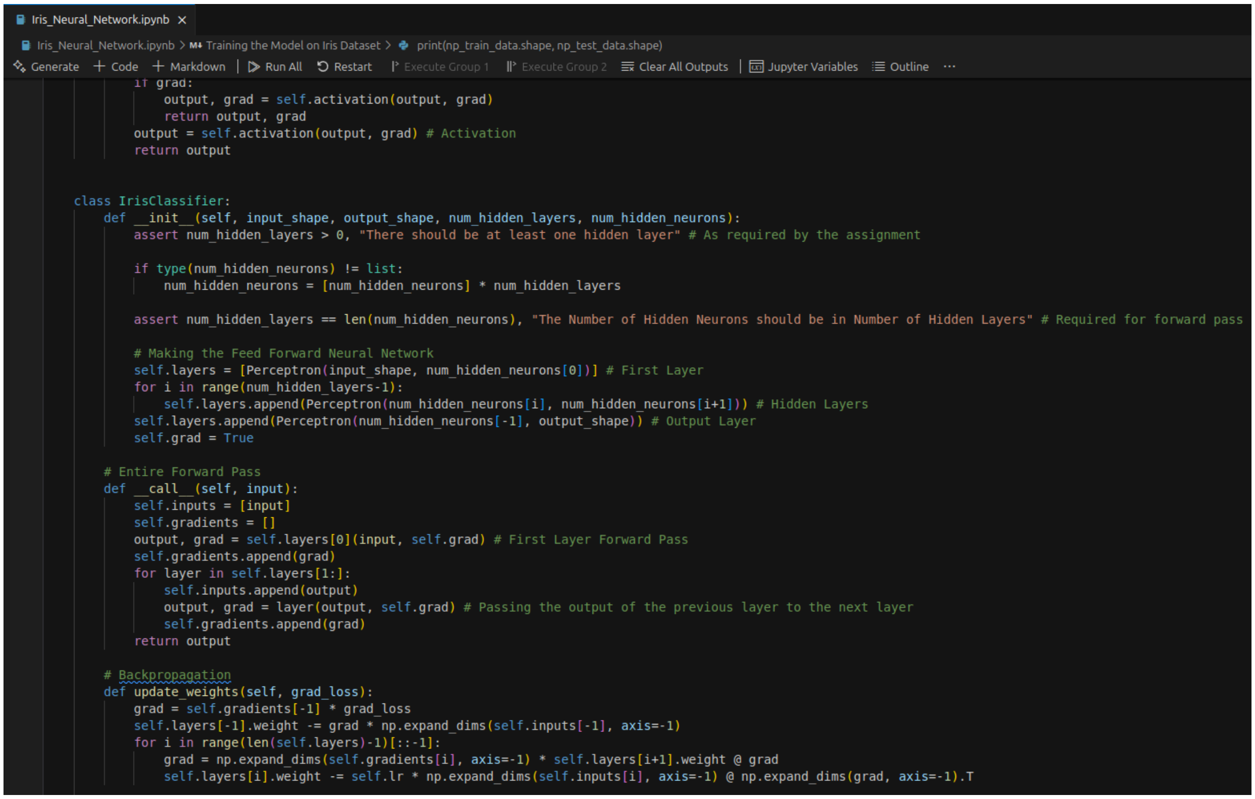

Iris_Neural_Network.ipynb
Iris_Neural_Network.ipynb > Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
Code
Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline

Defining the Model Architecture and BackPropagation Method

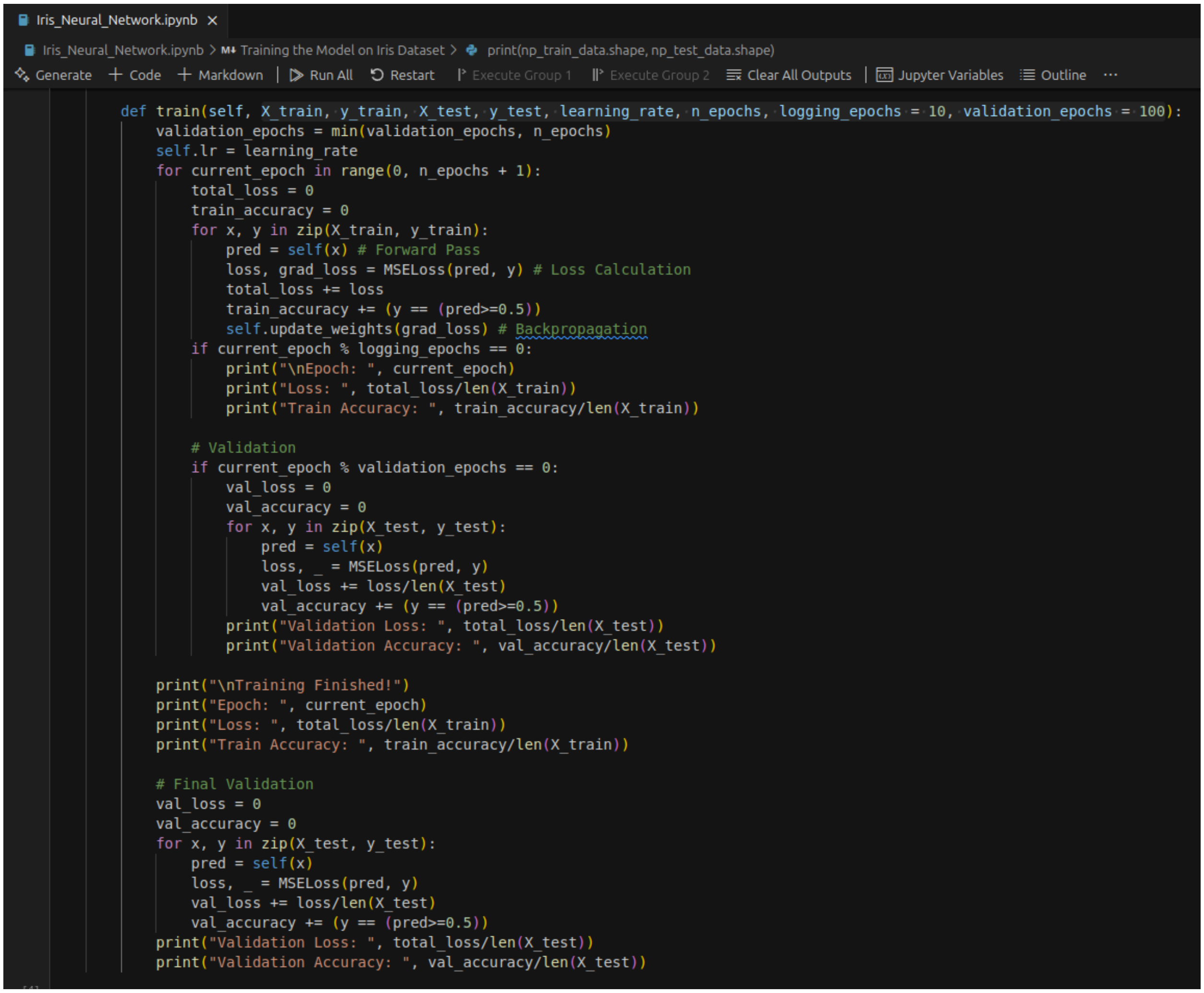
```
import pandas as pd
import numpy as np
```

[3] ✓ 0.7s

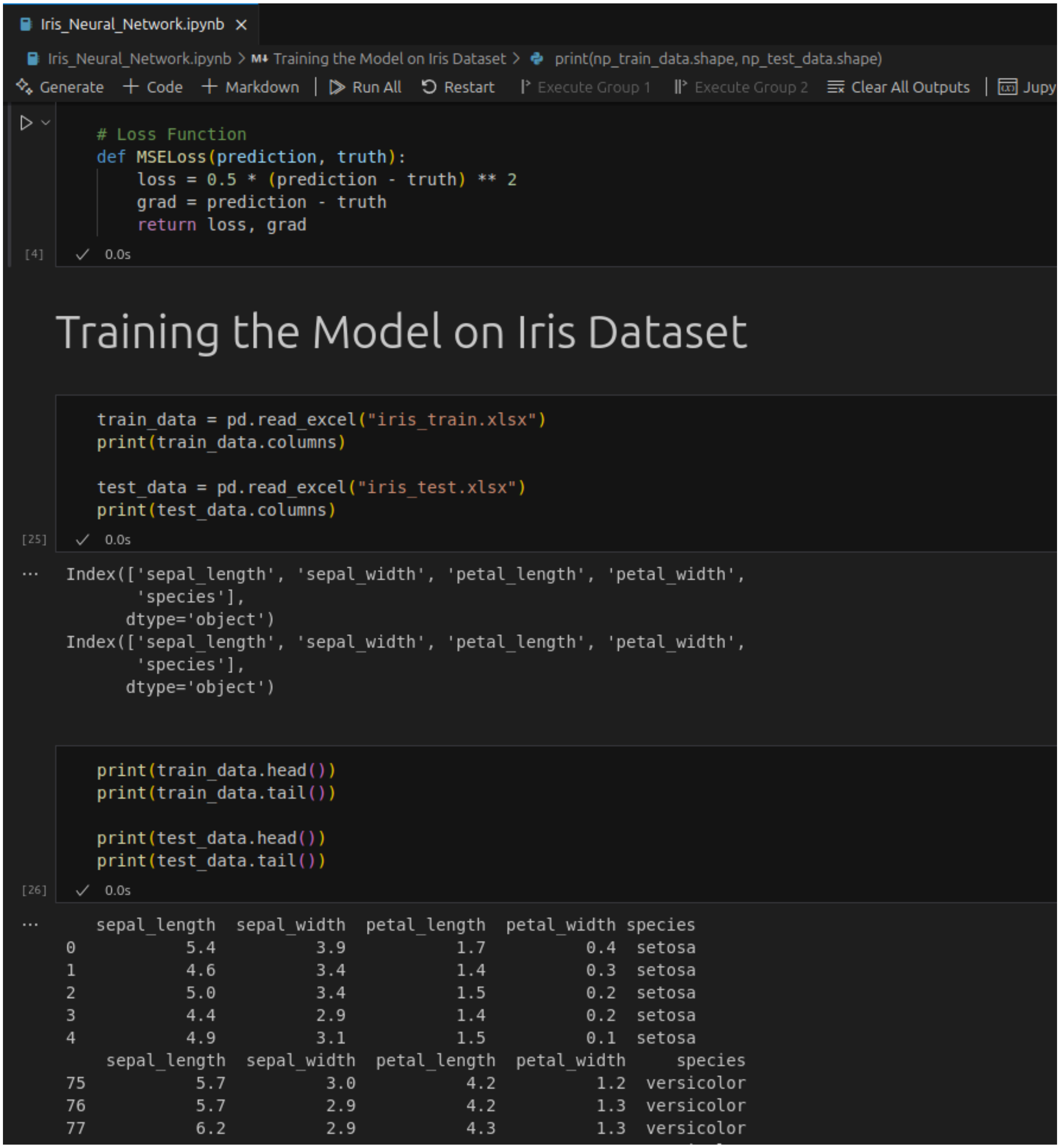
```
e = 1e1

# Activation Function
def sigmoid(x, grad = True):
    value = 1/(1 + np.exp(-1*x))
    if grad:
        # For Backpropagation
        grad = value*(1-value)
        return value, grad
    return value

class Perceptron:
    def __init__(self, input_shape, output_shape, activation = sigmoid):
        self.weight = np.random.normal(size=(input_shape, output_shape)) # Random Initialization of Weights
        self.activation = activation

    def __call__(self, inp, grad):
        output = self.weight.T @ inp # Forward Pass
        if grad:
            output, grad = self.activation(output, grad)
            return output, grad
        output = self.activation(output, grad) # Activation
        return output


class IrisClassifier:
    def __init__(self, input_shape, output_shape, num_hidden_layers, num_hidden_neurons):
        assert num_hidden_layers > 0, "There should be at least one hidden layer" # As required by the assignment

        if type(num_hidden_neurons) != list:
            num_hidden_neurons = [num_hidden_neurons] * num_hidden_layers

        assert num_hidden_layers == len(num_hidden_neurons), "The Number of Hidden Neurons should be in Number of Hidden Layers" # Required for forward pass
```


Iris_Neural_Network.ipynb
Iris_Neural_Network.ipynb > Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
Code
Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline

```python
        if grad:
            output, grad = self.activation(output, grad)
            return output, grad
        output = self.activation(output, grad) # Activation
        return output


class IrisClassifier:
    def __init__(self, input_shape, output_shape, num_hidden_layers, num_hidden_neurons):
        assert num_hidden_layers > 0, "There should be at least one hidden layer" # As required by the assignment

        if type(num_hidden_neurons) != list:
            num_hidden_neurons = [num_hidden_neurons] * num_hidden_layers

        assert num_hidden_layers == len(num_hidden_neurons), "The Number of Hidden Neurons should be in Number of Hidden Layers" # Required for forward pass

        # Making the Feed Forward Neural Network
        self.layers = [Perceptron(input_shape, num_hidden_neurons[0])] # First Layer
        for i in range(num_hidden_layers-1):
            self.layers.append(Perceptron(num_hidden_neurons[i], num_hidden_neurons[i+1])) # Hidden Layers
        self.layers.append(Perceptron(num_hidden_neurons[-1], output_shape)) # Output Layer
        self.grad = True

    # Entire Forward Pass
    def __call__(self, input):
        self.inputs = [input]
        self.gradients = []
        output, grad = self.layers[0](input, self.grad) # First Layer Forward Pass
        self.gradients.append(grad)
        for layer in self.layers[1:]:
            self.inputs.append(output)
            output, grad = layer(output, self.grad) # Passing the output of the previous layer to the next layer
            self.gradients.append(grad)
        return output

    # Backpropagation
    def update_weights(self, grad_loss):
        grad = self.gradients[-1] * grad_loss
        self.layers[-1].weight -= grad * np.expand_dims(self.inputs[-1], axis=-1)
        for i in range(len(self.layers)-1)[::-1]:
            grad = np.expand_dims(self.gradients[i], axis=-1) * self.layers[i+1].weight @ grad
            self.layers[i].weight -= self.lr * np.expand_dims(self.inputs[i], axis=-1) @ np.expand_dims(grad, axis=-1).T
```


Iris_Neural_Network.ipynb ×
Iris_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
···

```python
    def train(self, X_train, y_train, X_test, y_test, learning_rate, n_epochs, logging_epochs = 10, validation_epochs = 100):
        validation_epochs = min(validation_epochs, n_epochs)
        self.lr = learning_rate
        for current_epoch in range(0, n_epochs + 1):
            total_loss = 0
            train_accuracy = 0
            for x, y in zip(X_train, y_train):
                pred = self(x) # Forward Pass
                loss, grad_loss = MSELoss(pred, y) # Loss Calculation
                total_loss += loss
                train_accuracy += (y == (pred>=0.5))
                self.update_weights(grad_loss) # Backpropagation
            if current_epoch % logging_epochs == 0:
                print("\nEpoch: ", current_epoch)
                print("Loss: ", total_loss/len(X_train))
                print("Train Accuracy: ", train_accuracy/len(X_train))

            # Validation
            if current_epoch % validation_epochs == 0:
                val_loss = 0
                val_accuracy = 0
                for x, y in zip(X_test, y_test):
                    pred = self(x)
                    loss, _ = MSELoss(pred, y)
                    val_loss += loss/len(X_test)
                    val_accuracy += (y == (pred>=0.5))
                print("Validation Loss: ", total_loss/len(X_test))
                print("Validation Accuracy: ", val_accuracy/len(X_test))

        print("\nTraining Finished!")
        print("Epoch: ", current_epoch)
        print("Loss: ", total_loss/len(X_train))
        print("Train Accuracy: ", train_accuracy/len(X_train))

        # Final Validation
        val_loss = 0
        val_accuracy = 0
        for x, y in zip(X_test, y_test):
            pred = self(x)
            loss, _ = MSELoss(pred, y)
            val_loss += loss/len(X_test)
            val_accuracy += (y == (pred>=0.5))
        print("Validation Loss: ", total_loss/len(X_test))
        print("Validation Accuracy: ", val_accuracy/len(X_test))
```


Iris_Neural_Network.ipynb
Iris_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupy
[4]
✓ 0.0s

```
# Loss Function
def MSELoss(prediction, truth):
    loss = 0.5 * (prediction - truth) ** 2
    grad = prediction - truth
    return loss, grad
```

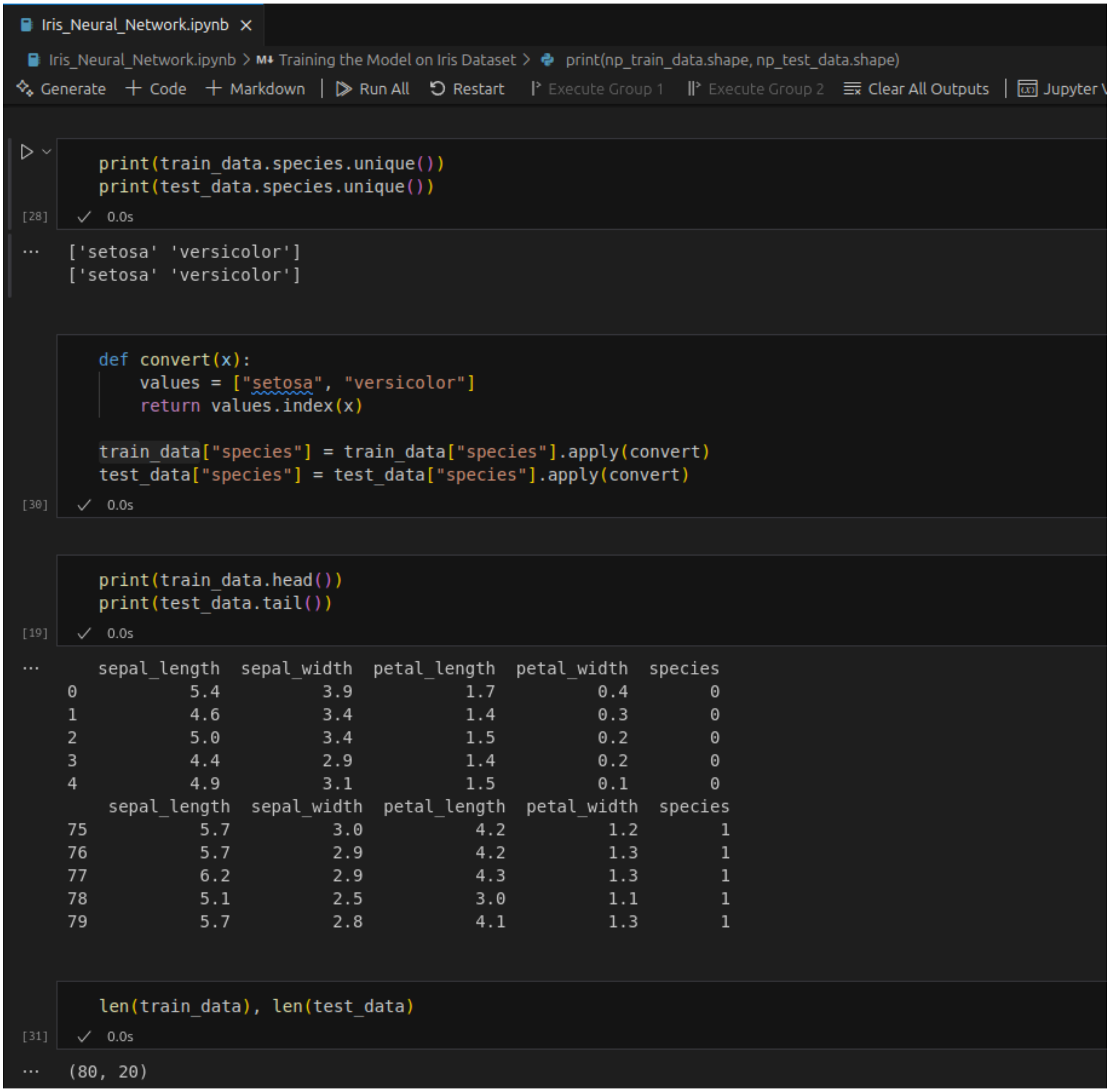
Training the Model on Iris Dataset

```
train_data = pd.read_excel("iris_train.xlsx")
print(train_data.columns)

test_data = pd.read_excel("iris_test.xlsx")
print(test_data.columns)
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

```
print(train_data.head())
print(train_data.tail())

print(test_data.head())
print(test_data.tail())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.4          3.9           1.7          0.4  setosa
1           4.6          3.4           1.4          0.3  setosa
2           5.0          3.4           1.5          0.2  setosa
3           4.4          2.9           1.4          0.2  setosa
4           4.9          3.1           1.5          0.1  setosa
    sepal_length  sepal_width  petal_length  petal_width     species
75           5.7          3.0           4.2          1.2  versicolor
76           5.7          2.9           4.2          1.3  versicolor
77           6.2          2.9           4.3          1.3  versicolor
```


Iris_Neural_Network.ipynb ×
Iris_Neural_Network.ipynb > M↓ Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
+ Code
+ Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter V
[28]
✓ 0.0s

```
print(train_data.species.unique())
print(test_data.species.unique())
```

```
['setosa' 'versicolor']
['setosa' 'versicolor']
```

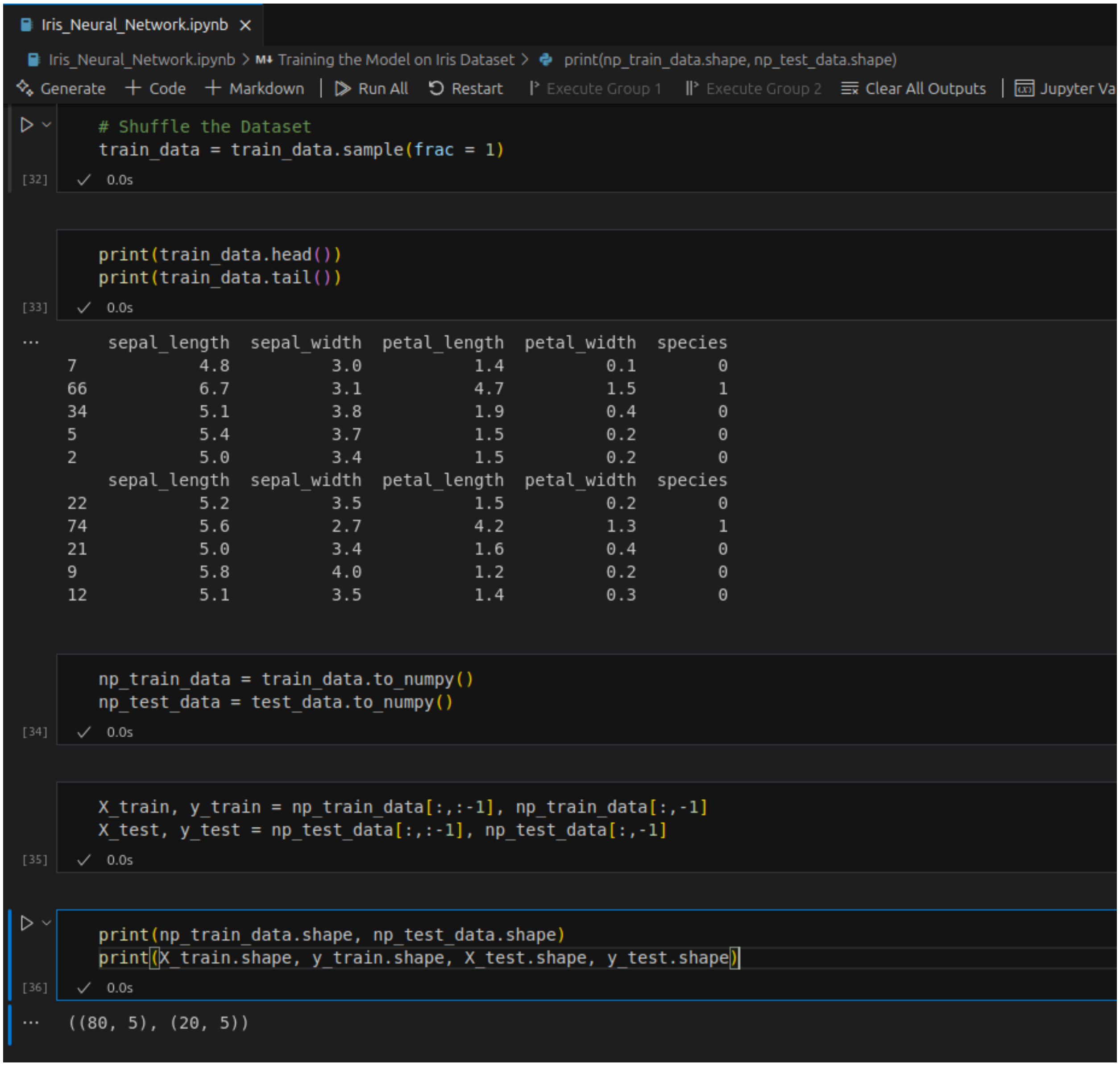
```
def convert(x):
    values = ["setosa", "versicolor"]
    return values.index(x)

train_data["species"] = train_data["species"].apply(convert)
test_data["species"] = test_data["species"].apply(convert)
```

```
print(train_data.head())
print(test_data.tail())
```

```
   sepal_length  sepal_width  petal_length  petal_width  species
0           5.4          3.9           1.7          0.4        0
1           4.6          3.4           1.4          0.3        0
2           5.0          3.4           1.5          0.2        0
3           4.4          2.9           1.4          0.2        0
4           4.9          3.1           1.5          0.1        0
    sepal_length  sepal_width  petal_length  petal_width  species
75           5.7          3.0           4.2          1.2        1
76           5.7          2.9           4.2          1.3        1
77           6.2          2.9           4.3          1.3        1
78           5.1          2.5           3.0          1.1        1
79           5.7          2.8           4.1          1.3        1
```

```
len(train_data), len(test_data)
```

```
(80, 20)
```


Iris_Neural_Network.ipynb
Iris_Neural_Network.ipynb > Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
Code
Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Va
[32]
0.0s
[33]
0.0s
[34]
0.0s
[35]
0.0s
[36]
0.0s

```python
# Shuffle the Dataset
train_data = train_data.sample(frac = 1)
```

```python
print(train_data.head())
print(train_data.tail())
```

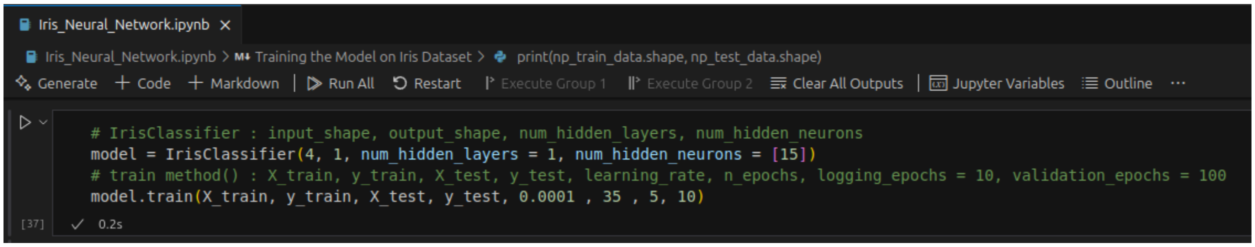
```
    sepal_length  sepal_width  petal_length  petal_width  species
7            4.8          3.0           1.4          0.1        0
66           6.7          3.1           4.7          1.5        1
34           5.1          3.8           1.9          0.4        0
5            5.4          3.7           1.5          0.2        0
2            5.0          3.4           1.5          0.2        0
    sepal_length  sepal_width  petal_length  petal_width  species
22           5.2          3.5           1.5          0.2        0
74           5.6          2.7           4.2          1.3        1
21           5.0          3.4           1.6          0.4        0
9            5.8          4.0           1.2          0.2        0
12           5.1          3.5           1.4          0.3        0
```

```python
np_train_data = train_data.to_numpy()
np_test_data = test_data.to_numpy()
```

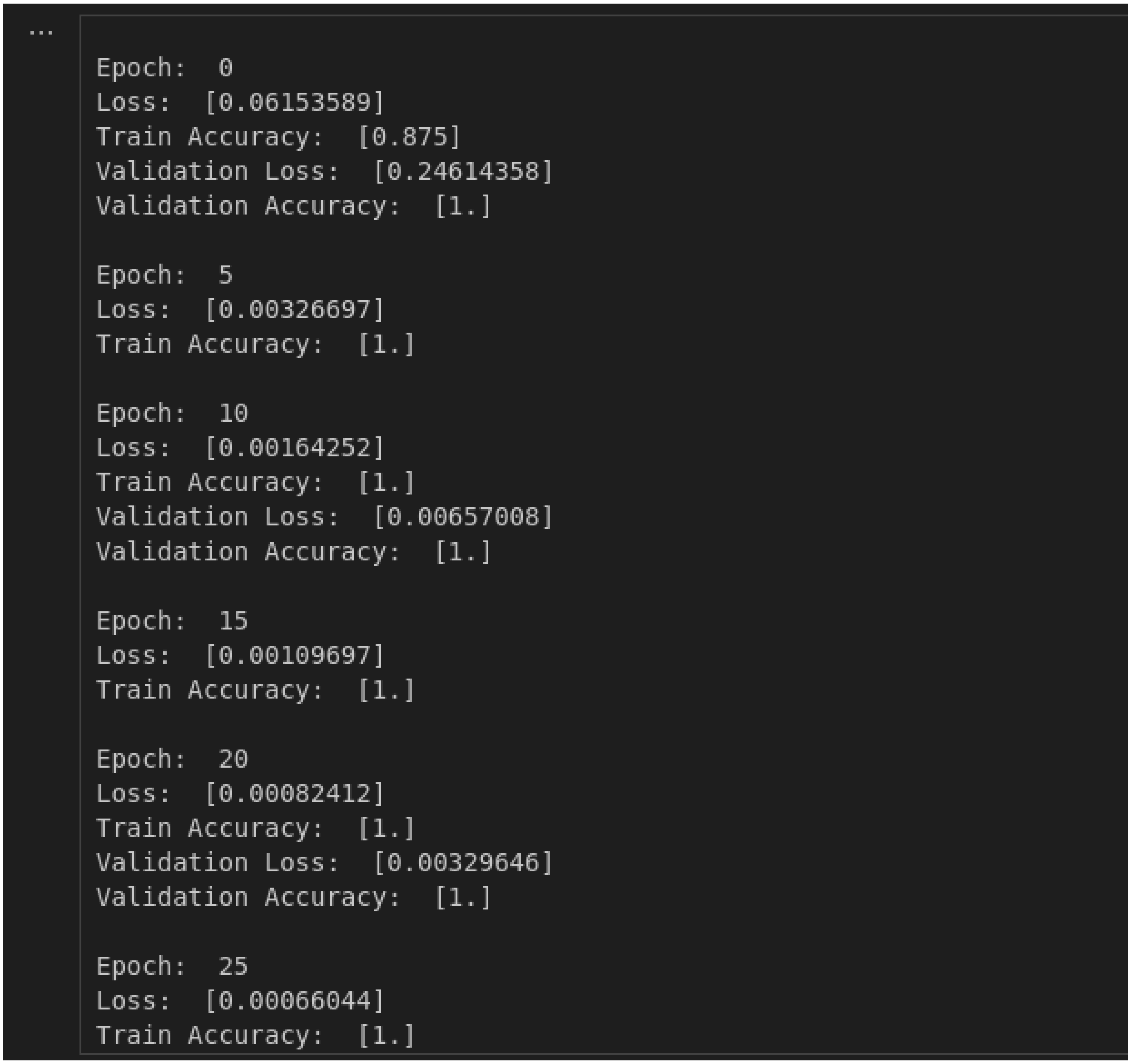
```python
X_train, y_train = np_train_data[:,:-1], np_train_data[:,-1]
X_test, y_test = np_test_data[:,:-1], np_test_data[:,-1]
```

```python
print(np_train_data.shape, np_test_data.shape)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

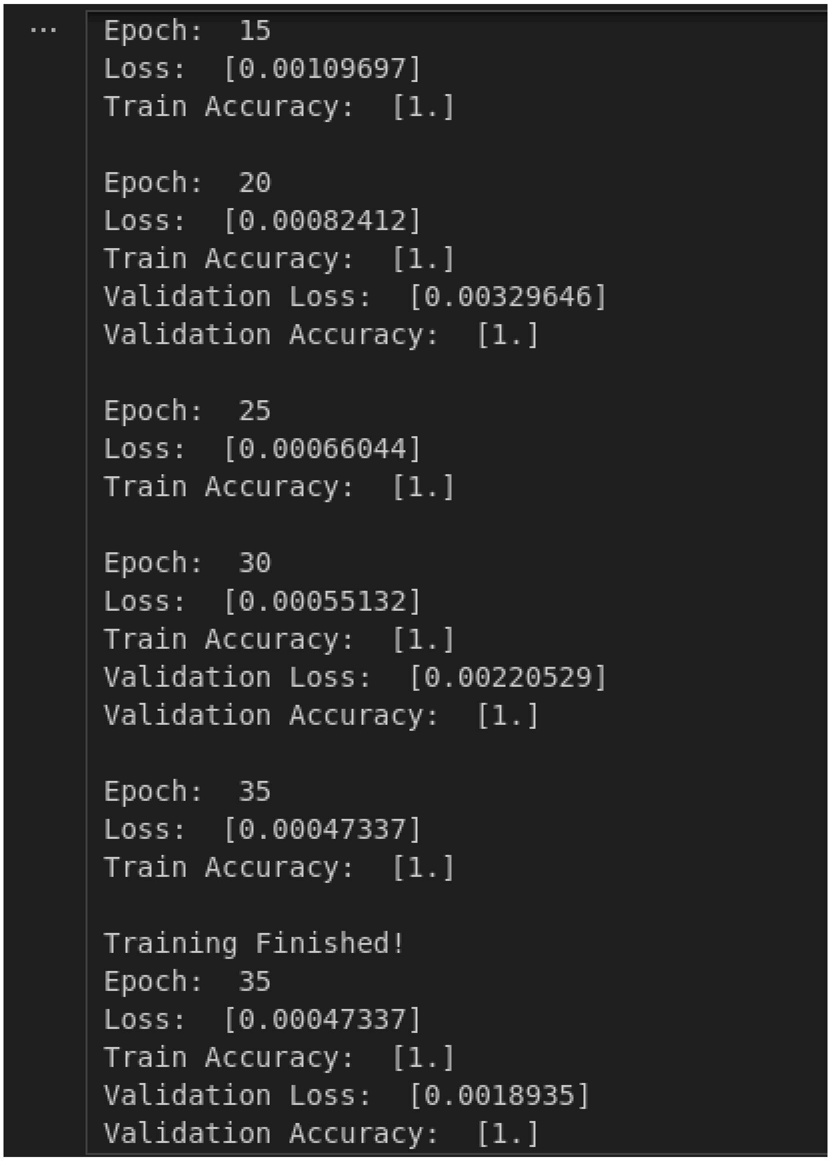
```
((80, 5), (20, 5))
```


Iris_Neural_Network.ipynb
Iris_Neural_Network.ipynb > Training the Model on Iris Dataset > print(np_train_data.shape, np_test_data.shape)
Generate
Code
Markdown
Run All
Restart
Execute Group 1
Execute Group 2
Clear All Outputs
Jupyter Variables
Outline
[37]
0.2s

```python
# IrisClassifier : input_shape, output_shape, num_hidden_layers, num_hidden_neurons
model = IrisClassifier(4, 1, num_hidden_layers = 1, num_hidden_neurons = [15])
# train method() : X_train, y_train, X_test, y_test, learning_rate, n_epochs, logging_epochs = 10, validation_epochs = 100
model.train(X_train, y_train, X_test, y_test, 0.0001 , 35 , 5, 10)
```



```
Epoch:  0
Loss:  [0.06153589]
Train Accuracy:  [0.875]
Validation Loss:  [0.24614358]
Validation Accuracy:  [1.]

Epoch:  5
Loss:  [0.00326697]
Train Accuracy:  [1.]

Epoch:  10
Loss:  [0.00164252]
Train Accuracy:  [1.]
Validation Loss:  [0.00657008]
Validation Accuracy:  [1.]

Epoch:  15
Loss:  [0.00109697]
Train Accuracy:  [1.]

Epoch:  20
Loss:  [0.00082412]
Train Accuracy:  [1.]
Validation Loss:  [0.00329646]
Validation Accuracy:  [1.]

Epoch:  25
Loss:  [0.00066044]
Train Accuracy:  [1.]
```



```
...  Epoch:  15
     Loss:  [0.00109697]
     Train Accuracy:  [1.]

     Epoch:  20
     Loss:  [0.00082412]
     Train Accuracy:  [1.]
     Validation Loss:  [0.00329646]
     Validation Accuracy:  [1.]

     Epoch:  25
     Loss:  [0.00066044]
     Train Accuracy:  [1.]

     Epoch:  30
     Loss:  [0.00055132]
     Train Accuracy:  [1.]
     Validation Loss:  [0.00220529]
     Validation Accuracy:  [1.]

     Epoch:  35
     Loss:  [0.00047337]
     Train Accuracy:  [1.]

     Training Finished!
     Epoch:  35
     Loss:  [0.00047337]
     Train Accuracy:  [1.]
     Validation Loss:  [0.0018935]
     Validation Accuracy:  [1.]
```

For code , refer GitHub

https://github.com/arnavjain2710/Computational-Intelligence-Lab-CS354N/tree/main/LAB%206