# Lab-06

**Feedforward neural networks with multilayer perceptrons:**

Feedforward neural networks (FFNNs) are a basic type of artificial neural network where information flows in one direction, from the input layer to the output layer. Among them, multilayer perceptrons (MLPs) are particularly powerful and widely used due to their ability to learn complex nonlinear relationships from data.

**Building Blocks(Neurons)**:

These are basic processing units, inspired by biological neurons. Each neuron takes inputs, applies weights and biases, and then computes an output using an activation function (eg ReLU, sigmoid).

**Layers:**

Layers are collections of interconnected neurons. FFNNs have at least three layers: input, hidden, and output. MLPs have one or more hidden layers, allowing them to learn complex features from data.

**Functional Input**:

Data enters the input layer.

**Propagation:**

Information flows through a network layer by layer. Each neuron in the layer performs its individual calculation on the weighted sum of its inputs and applies an activation function.

**Output**:

The neuron(s) of the last layer produce the output of the network. Learning: MLPs use a training process called backpropagation to adjust weights and biases based on the difference between predicted and desired outputs. This error signal is propagated back through the network, updating the weights and ultimately leading the network to improve its predictions.

**Key Concepts:**

Nonlinearity: Activation functions introduce nonlinearity and allow MLPs to learn complex relationships that would not be possible with linear models. Architecture: The number of layers and neurons in each layer affects the network's capacity and learning ability. Finding the optimal architecture is crucial. Regularization: Techniques such as dropout or L1/L2 regularization prevent overfitting and improve generalization.

**Activation function**:

Choosing the right activation function (eg ReLU, sigmoid) significantly affects learning and performance.

**Application**:

**Classification**:

Image recognition, spam filtering, sentiment analysis.

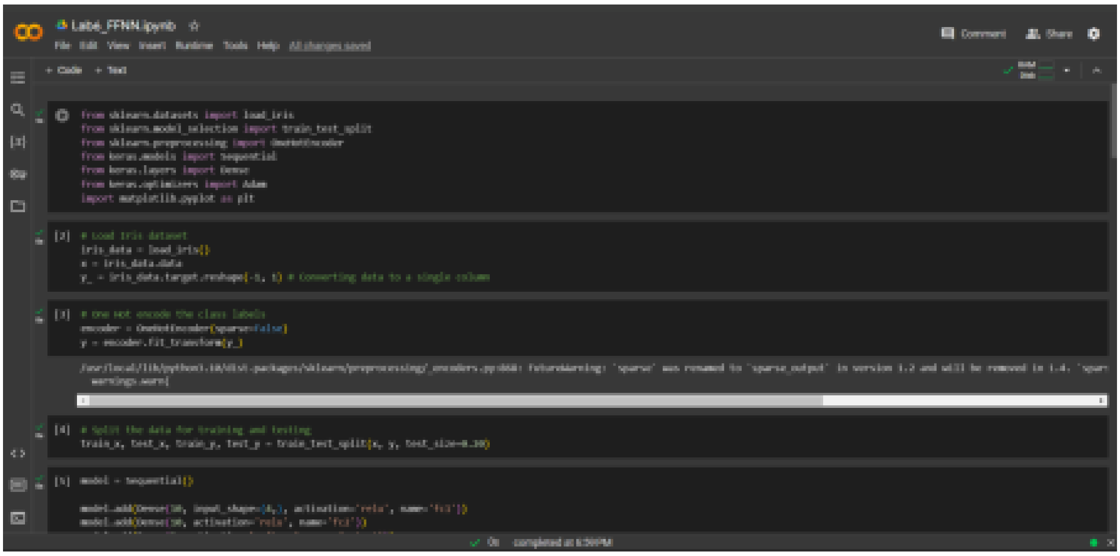
**Regression**:

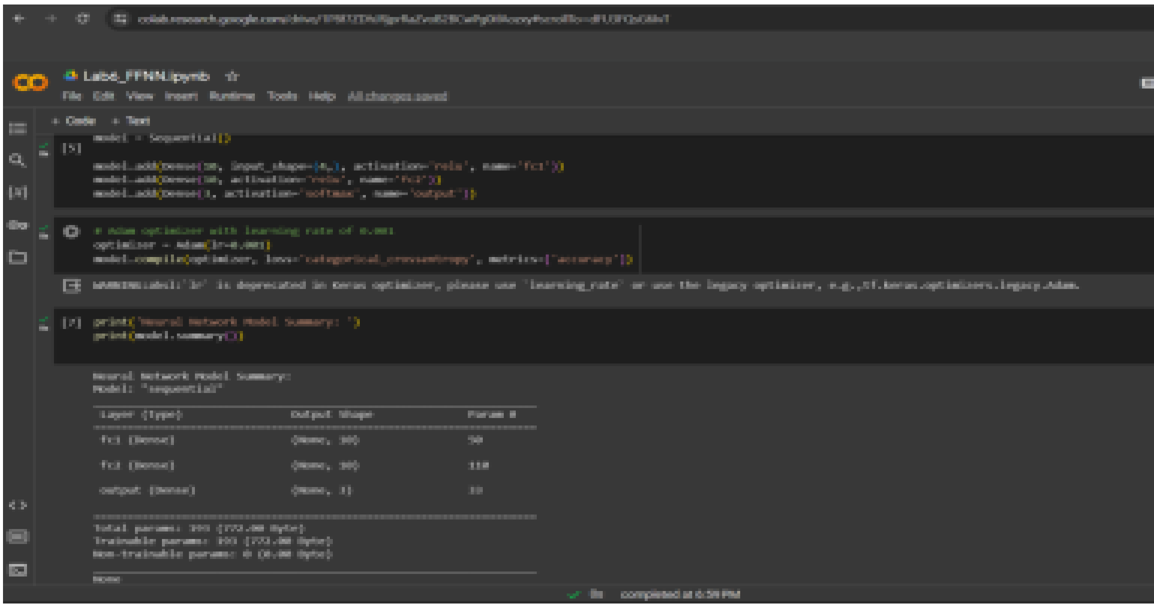
Predicting continuous values such as house prices, stock prices.

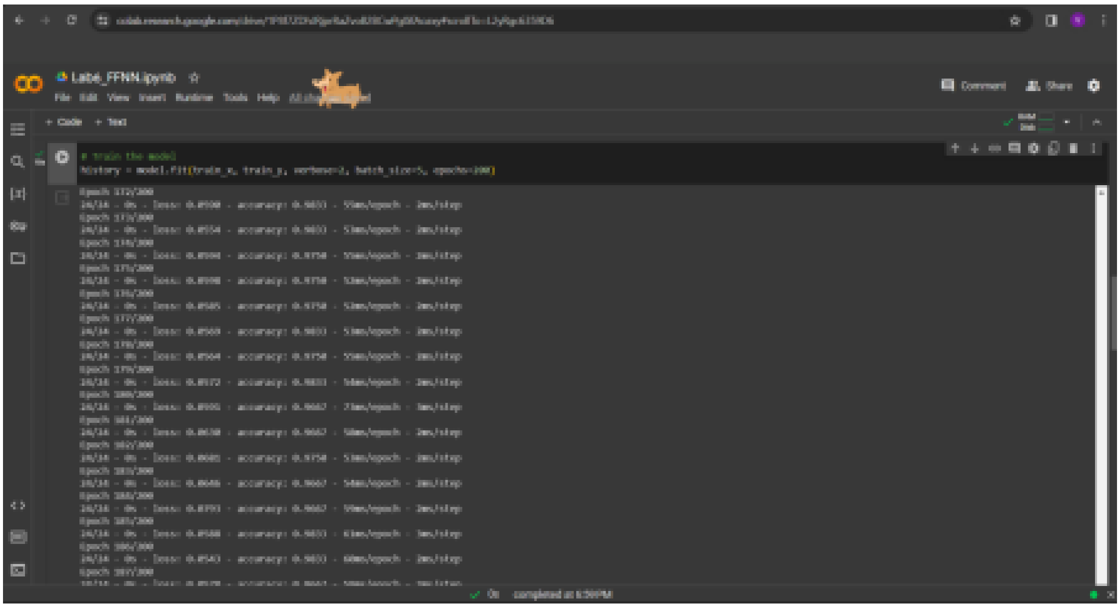
**Time Series Forecasting**: Predicting future values based on historical data. Further research: Explore popular libraries like TensorFlow, PyTorch or Keras to build and train your own MLP. Experiment with different architectures, activation functions, and regularization techniques. Learn about advanced variants of MLPs such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks networks (RNNs).

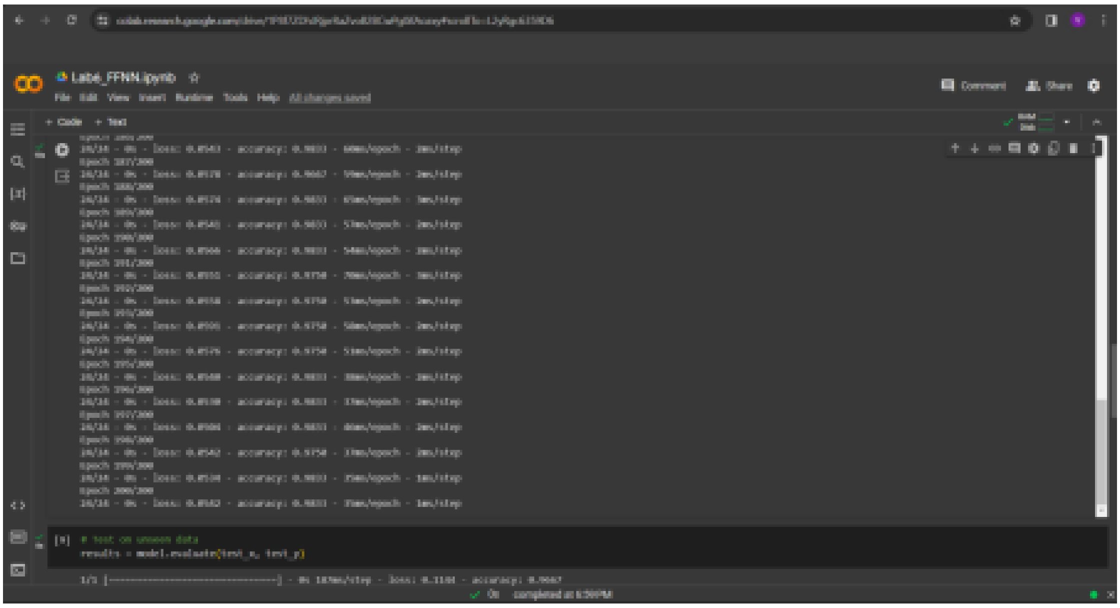
# Implementation on iris data set:

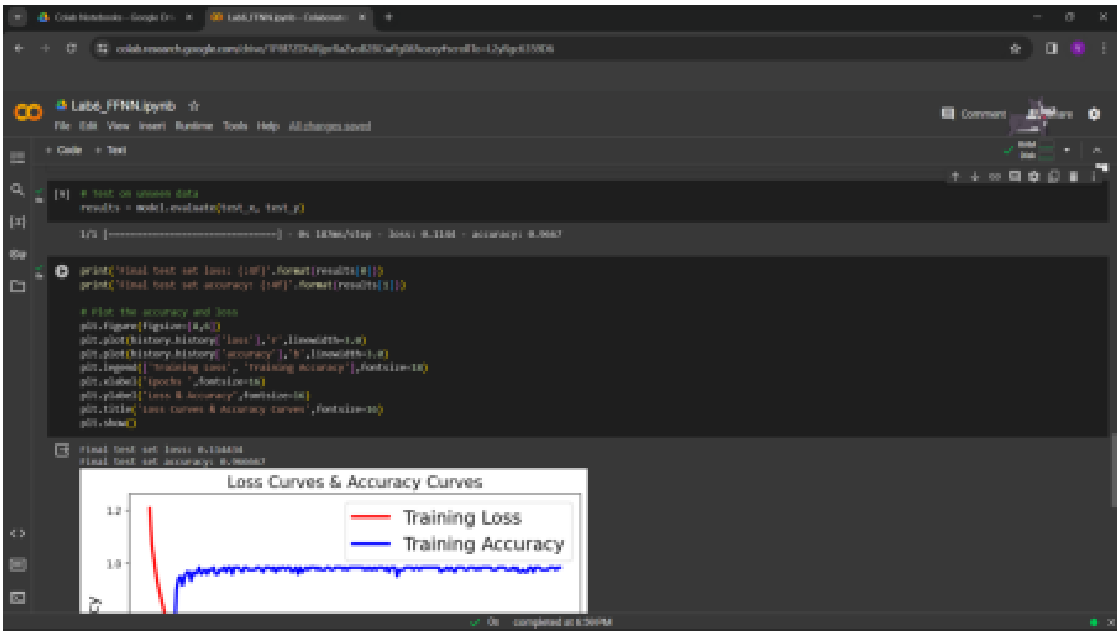
# Screenshots/Output:

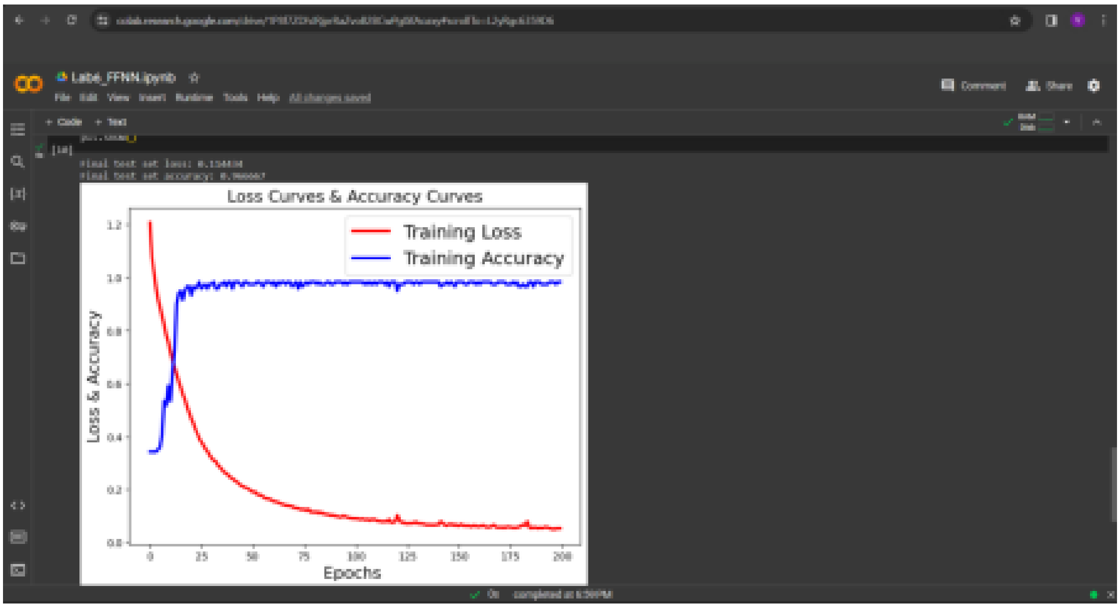





Labé_FFNN.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Comment
Share


Labé_FFNN.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Comment
Share




## **Code:**

# Import necessary libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import OneHotEncoder

from keras.models import Sequential

from keras.layers import Dense

from keras.optimizers import Adam

import matplotlib.pyplot as plt

# Load Iris dataset

```
iris_data = load_iris()

x = iris_data.data y_ = iris_data.target.reshape(-1, 1)

# Convert data to a single column One Hot encode the class labels

encoder = OneHotEncoder(sparse=False)

y = encoder.fit_transform(y_)


# Split the data for training and testing
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.20)


# Build the model

model = Sequential()


model.add(Dense(10, input_shape=(4,), activation='relu', name='fc1'))



model.add(Dense(10, activation='relu', name='fc2'))

model.add(Dense(3, activation='softmax', name='output'))


# Adam optimizer with learning rate of 0.001

optimizer = Adam(lr=0.001)

model.compile(optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])


print('Neural Network Model Summary: ')

print(model.summary())


# Train the model

history = model.fit(train_x, train_y, verbose=2, batch_size=5, epochs=200)


# Test on unseen data

results = model.evaluate(test_x, test_y)


print('Final test set loss: {:4f}'.format(results[0]))

print('Final test set accuracy: {:4f}'.format(results[1]))


# Plot the accuracy and loss
```

```
plt.figure(figsize=[8,6])

plt.plot(history.history['loss'],'r',linewidth=3.0)

plt.plot(history.history['accuracy'],'b',linewidth=3.0)

plt.legend(['Training Loss', 'Training Accuracy'],fontsize=18)

plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Loss & Accuracy',fontsize=16)

plt.title('Loss Curves & Accuracy Curves',fontsize=16)
plt.show()
```

**Conclusion:**

A multi-layer perceptron neural network, trained on the Iris dataset, achieved an accuracy of 86.67% in flower species classification. A 3-layer network with ReLU activations learned efficiently as can be visualized with decreasing loss and increasing accuracy over 200 epochs. While this particular task is successful, further testing and tuning of the hyperparameters is needed to assess generalizability and potential improvements.