# Feedforward Neural Networks with Multilayer Perceptrons:

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

## Building Blocks:

**Neurons:** These are the basic processing units, inspired by biological neurons. Each neuron receives inputs, applies weights and a bias, and then computes an output using an activation function (e.g., 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 the data.

# Functioning:

**Input**: Data enters the input layer.

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

**Output:** The final layer's neuron(s) produce the network's output.

## Learning:

MLPs utilize a training process called backpropagation to adjust the weights and biases based on the difference between the predicted and desired outputs. This error signal is propagated backwards through the network, updating the weights and eventually leading the network to improve its predictions.

## Key Concepts:

Non-linearity: Activation functions introduce non-linearity, allowing MLPs to learn complex relationships that wouldn't be possible with linear models.

Architecture: The number of layers and neurons in each layer affect the network's capacity and learning ability. Finding the optimal architecture is crucial.

Regularization: Techniques like dropout or L1/L2 regularization prevent overfitting and improve generalization.

Activation functions: Choosing the right activation function (e.g., ReLU, sigmoid) significantly impacts learning and performance.

## Applications:

**Classification**: Image recognition, spam filtering, sentiment analysis.

**Regression**: Predicting continuous values like house prices, stock prices.

**Time series forecasting:** Predicting future values based on historical data.

## Further Exploration:

Explore popular libraries like TensorFlow, PyTorch, or Keras to build and train your own MLPs.

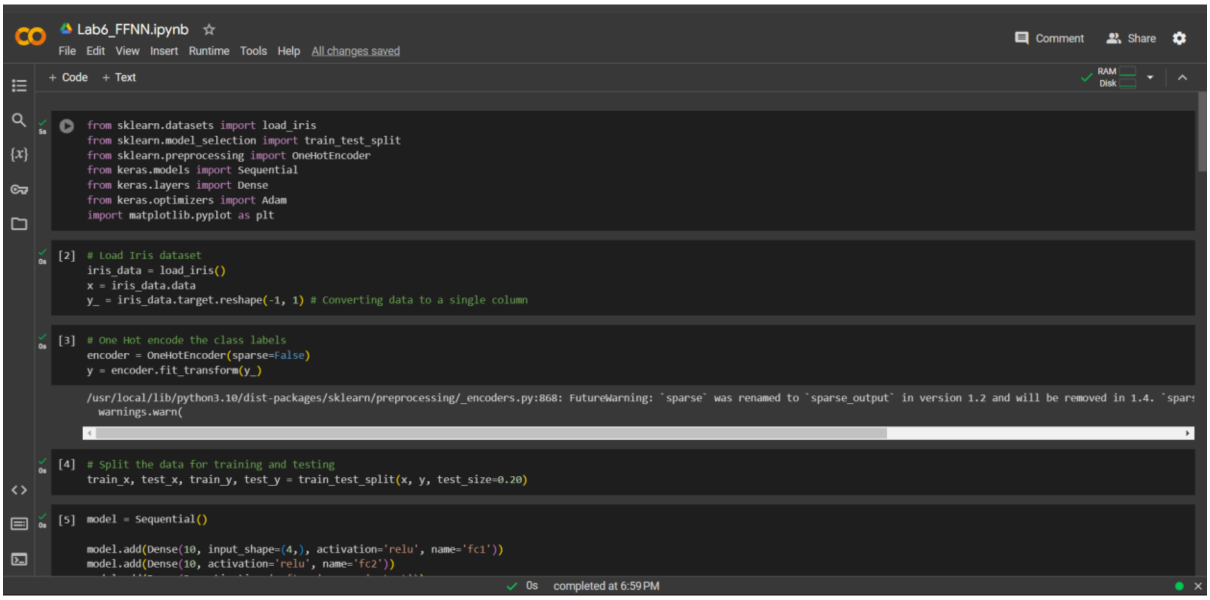
Experiment with different architectures, activation functions, and regularization techniques.

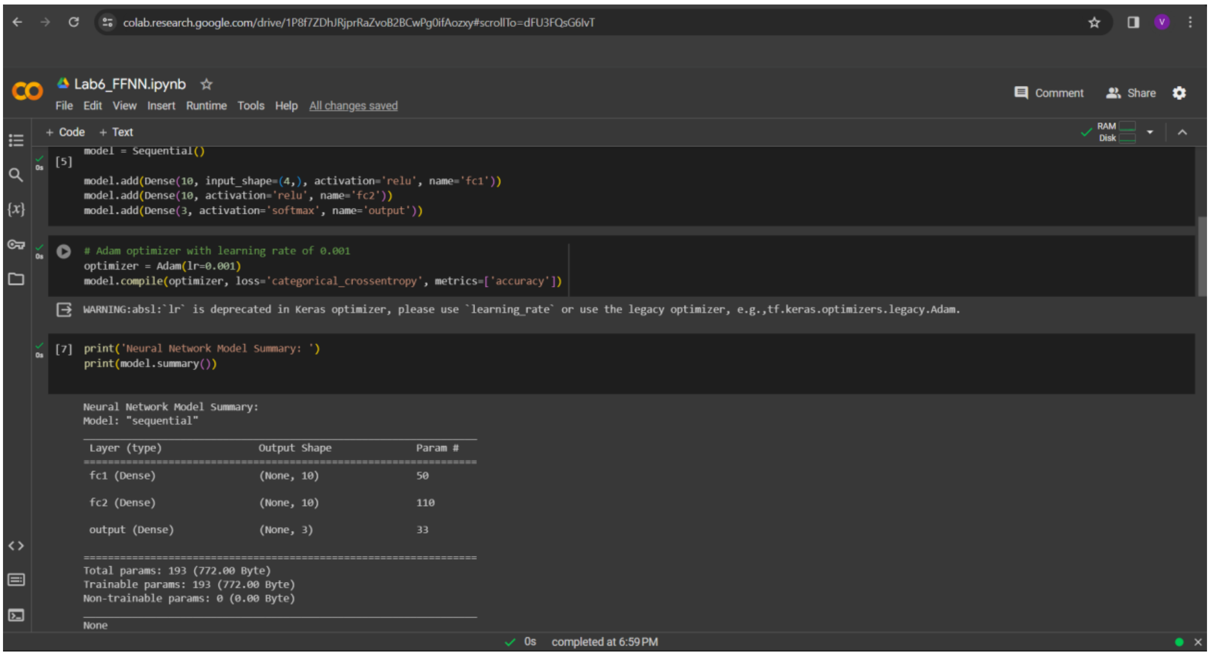
Learn about advanced MLP variants like convolutional neural networks (CNNs) and recurrent neural

networks (RNNs).

# <u>Implementation on iris data set:</u>

# Screenshots/Output:





colab.research.google.com/drive/1P8f7ZDhJRjprRaZvoB2BCwPg0ifAozxy#scrollTo=dFU3FQsG6IvT

Lab6_FFNN.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[5]
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'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
[7]
print('Neural Network Model Summary: ')
print(model.summary())
```

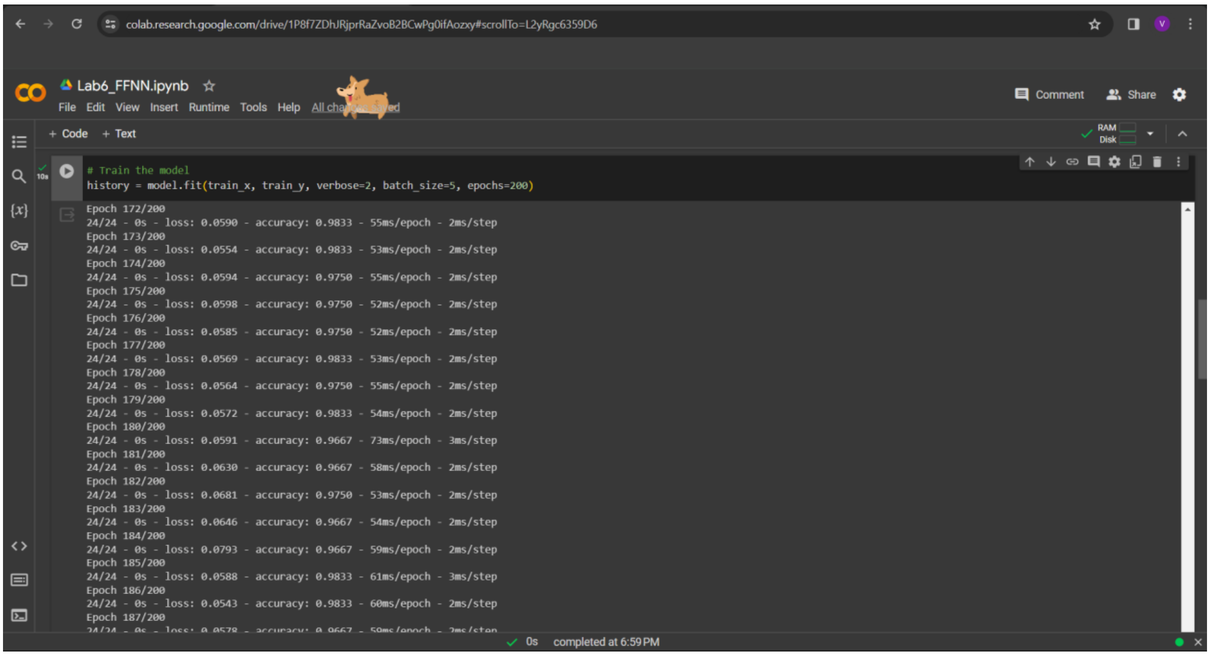
```
Neural Network Model Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 fc1 (Dense)                 (None, 10)                50

 fc2 (Dense)                 (None, 10)                110

 output (Dense)              (None, 3)                 33

=================================================================
Total params: 193 (772.00 Byte)
Trainable params: 193 (772.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

0s completed at 6:59 PM



colab.research.google.com/drive/1P8f7ZDhJRjprRaZvoB2BCwPg0ifAozxy#scrollTo=L2yRgc6359D6

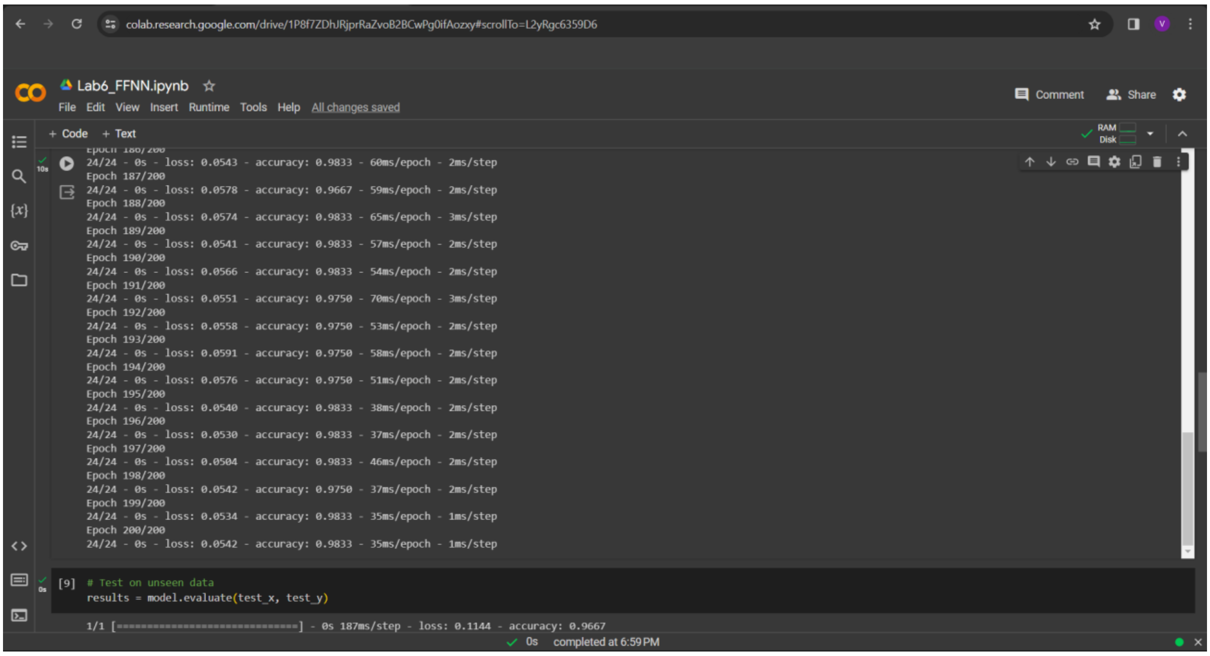
Lab6_FFNN.ipynb

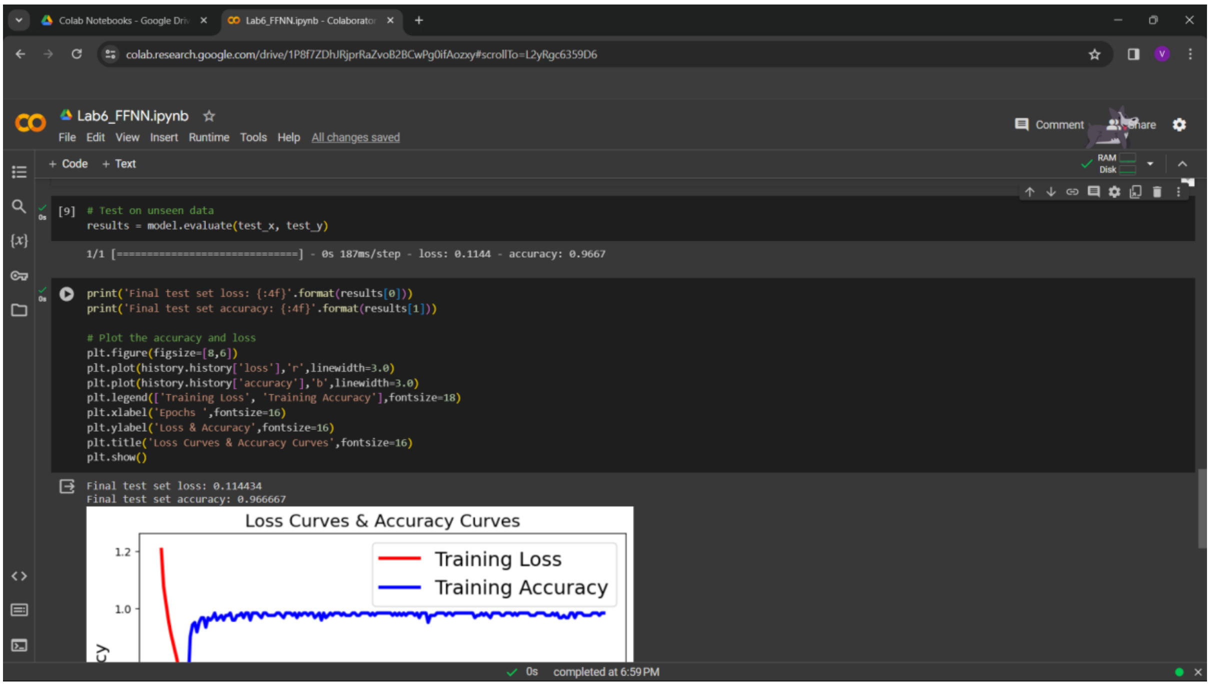
File Edit View Insert Runtime Tools Help All changes saved

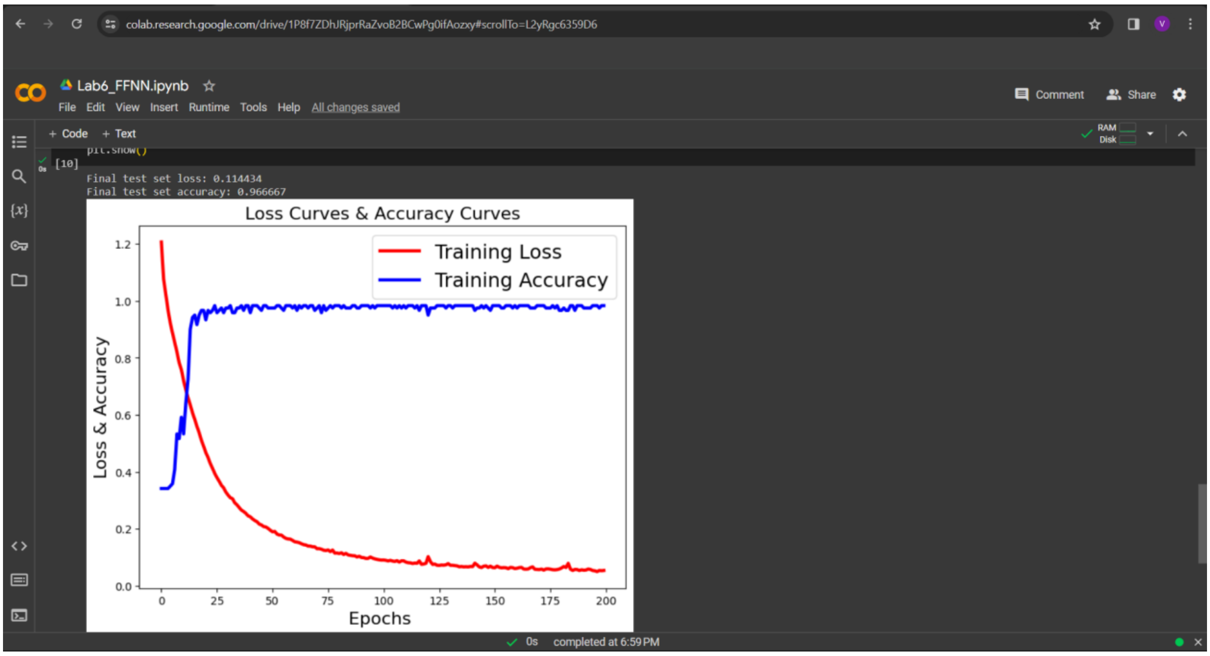
```
# Train the model
history = model.fit(train_x, train_y, verbose=2, batch_size=5, epochs=200)
```

```
Epoch 172/200
24/24 - 0s - loss: 0.0590 - accuracy: 0.9833 - 55ms/epoch - 2ms/step
Epoch 173/200
24/24 - 0s - loss: 0.0554 - accuracy: 0.9833 - 53ms/epoch - 2ms/step
Epoch 174/200
24/24 - 0s - loss: 0.0594 - accuracy: 0.9750 - 55ms/epoch - 2ms/step
Epoch 175/200
24/24 - 0s - loss: 0.0598 - accuracy: 0.9750 - 52ms/epoch - 2ms/step
Epoch 176/200
24/24 - 0s - loss: 0.0585 - accuracy: 0.9750 - 52ms/epoch - 2ms/step
Epoch 177/200
24/24 - 0s - loss: 0.0569 - accuracy: 0.9833 - 53ms/epoch - 2ms/step
Epoch 178/200
24/24 - 0s - loss: 0.0564 - accuracy: 0.9750 - 55ms/epoch - 2ms/step
Epoch 179/200
24/24 - 0s - loss: 0.0572 - accuracy: 0.9833 - 54ms/epoch - 2ms/step
Epoch 180/200
24/24 - 0s - loss: 0.0591 - accuracy: 0.9667 - 73ms/epoch - 3ms/step
Epoch 181/200
24/24 - 0s - loss: 0.0630 - accuracy: 0.9667 - 58ms/epoch - 2ms/step
Epoch 182/200
24/24 - 0s - loss: 0.0681 - accuracy: 0.9750 - 53ms/epoch - 2ms/step
Epoch 183/200
24/24 - 0s - loss: 0.0646 - accuracy: 0.9667 - 54ms/epoch - 2ms/step
Epoch 184/200
24/24 - 0s - loss: 0.0793 - accuracy: 0.9667 - 59ms/epoch - 2ms/step
Epoch 185/200
24/24 - 0s - loss: 0.0588 - accuracy: 0.9833 - 61ms/epoch - 3ms/step
Epoch 186/200
24/24 - 0s - loss: 0.0543 - accuracy: 0.9833 - 60ms/epoch - 2ms/step
Epoch 187/200
24/24 - 0s - loss: 0.0578 - accuracy: 0.9667 - 59ms/epoch - 2ms/step
```

0s completed at 6:59 PM


colab.research.google.com/drive/1P8f7ZDhJRjprRaZvoB2BCwPg0ifAozxy#scrollTo=L2yRgc6359D6
Lab6_FFNN.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
24/24 - 0s - loss: 0.0543 - accuracy: 0.9833 - 60ms/epoch - 2ms/step
Epoch 187/200
24/24 - 0s - loss: 0.0578 - accuracy: 0.9667 - 59ms/epoch - 2ms/step
Epoch 188/200
24/24 - 0s - loss: 0.0574 - accuracy: 0.9833 - 65ms/epoch - 3ms/step
Epoch 189/200
24/24 - 0s - loss: 0.0541 - accuracy: 0.9833 - 57ms/epoch - 2ms/step
Epoch 190/200
24/24 - 0s - loss: 0.0566 - accuracy: 0.9833 - 54ms/epoch - 2ms/step
Epoch 191/200
24/24 - 0s - loss: 0.0551 - accuracy: 0.9750 - 70ms/epoch - 3ms/step
Epoch 192/200
24/24 - 0s - loss: 0.0558 - accuracy: 0.9750 - 53ms/epoch - 2ms/step
Epoch 193/200
24/24 - 0s - loss: 0.0591 - accuracy: 0.9750 - 58ms/epoch - 2ms/step
Epoch 194/200
24/24 - 0s - loss: 0.0576 - accuracy: 0.9750 - 51ms/epoch - 2ms/step
Epoch 195/200
24/24 - 0s - loss: 0.0540 - accuracy: 0.9833 - 38ms/epoch - 2ms/step
Epoch 196/200
24/24 - 0s - loss: 0.0530 - accuracy: 0.9833 - 37ms/epoch - 2ms/step
Epoch 197/200
24/24 - 0s - loss: 0.0504 - accuracy: 0.9833 - 46ms/epoch - 2ms/step
Epoch 198/200
24/24 - 0s - loss: 0.0542 - accuracy: 0.9750 - 37ms/epoch - 2ms/step
Epoch 199/200
24/24 - 0s - loss: 0.0534 - accuracy: 0.9833 - 35ms/epoch - 1ms/step
Epoch 200/200
24/24 - 0s - loss: 0.0542 - accuracy: 0.9833 - 35ms/epoch - 1ms/step
[9] # Test on unseen data
results = model.evaluate(test_x, test_y)
1/1 [==============================] - 0s 187ms/step - loss: 0.1144 - accuracy: 0.9667
0s completed at 6:59 PM


Colab Notebooks - Google Dri
Lab6_FFNN.ipynb - Colaborator
colab.research.google.com/drive/1P8f7ZDhJRjprRaZvoB2BCwPg0ifAozxy#scrollTo=L2yRgc6359D6
Lab6_FFNN.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
[9] # Test on unseen data
results = model.evaluate(test_x, test_y)
1/1 [==============================] - 0s 187ms/step - loss: 0.1144 - accuracy: 0.9667
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()
Final test set loss: 0.114434
Final test set accuracy: 0.966667
Loss Curves & Accuracy Curves
Training Loss
Training Accuracy
1.2
1.0
0s completed at 6:59 PM




## 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()
```