DNN - Experiment 04

- **SIA VASHIST**
- PRN: 20190802107

In [8]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the iris dataset
iris = load_iris()
```

In [9]:
```
# Convert the dataset to a pandas DataFrame
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
```

In [10]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [11]:
```
# Define the DNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=[4]),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(3)
])

# Compile the model
model.compile(loss='mse', optimizer='adam')
```

In [12]:
```
# Train the model
model.fit(X_train, y_train, epochs=20)

# Evaluate the model on the testing set
loss = model.evaluate(X_test, y_test)
print('Test loss:', loss)
```

```
Epoch 1/20
4/4 [==============================] - 2s 6ms/step - loss: 1.4979
Epoch 2/20
4/4 [==============================] - 0s 7ms/step - loss: 1.2613
Epoch 3/20
4/4 [==============================] - 0s 6ms/step - loss: 1.0532
Epoch 4/20
4/4 [==============================] - 0s 6ms/step - loss: 0.8619
Epoch 5/20
4/4 [==============================] - 0s 7ms/step - loss: 0.6768
Epoch 6/20
4/4 [==============================] - 0s 8ms/step - loss: 0.5306
Epoch 7/20
4/4 [==============================] - 0s 7ms/step - loss: 0.3959
Epoch 8/20
4/4 [==============================] - 0s 8ms/step - loss: 0.2910
Epoch 9/20
4/4 [==============================] - 0s 6ms/step - loss: 0.2177
Epoch 10/20
4/4 [==============================] - 0s 6ms/step - loss: 0.1806
Epoch 11/20
4/4 [==============================] - 0s 6ms/step - loss: 0.1629
Epoch 12/20
4/4 [==============================] - 0s 5ms/step - loss: 0.1552
Epoch 13/20
4/4 [==============================] - 0s 6ms/step - loss: 0.1486
Epoch 14/20
4/4 [==============================] - 0s 7ms/step - loss: 0.1377
Epoch 15/20
4/4 [==============================] - 0s 7ms/step - loss: 0.1246
Epoch 16/20
4/4 [==============================] - 0s 8ms/step - loss: 0.1142
Epoch 17/20
4/4 [==============================] - 0s 7ms/step - loss: 0.1043
Epoch 18/20
4/4 [==============================] - 0s 7ms/step - loss: 0.0981
Epoch 19/20
4/4 [==============================] - 0s 6ms/step - loss: 0.0916
Epoch 20/20
4/4 [==============================] - 0s 6ms/step - loss: 0.0845
2/2 [==============================] - 0s 13ms/step - loss: 0.0801
Test loss: 0.08012888580560684
```

In [13]:
```
# Predict the target values using the trained model
y_pred = model.predict(X_test)

# Print the predicted and actual target values for the first 10 samples
print('Predicted targets:', np.argmax(y_pred[:10], axis=1))
print('Actual targets:', y_test[:10])
```

```
2/2 [==============================] - 0s 3ms/step
Predicted targets: [1 2 2 2 2 1 2 0 2 2]
Actual targets: [1 0 2 1 1 0 1 2 1 1]
```

> # Observation:
>
> The code then defines a DNN model using TensorFlow's Keras API, with two hidden layers of 64 neurons each and a final output layer with 3 neurons corresponding to the 3 target classes in the iris dataset. The model is compiled using mean squared error (MSE) as the loss function and Adam as the optimizer.

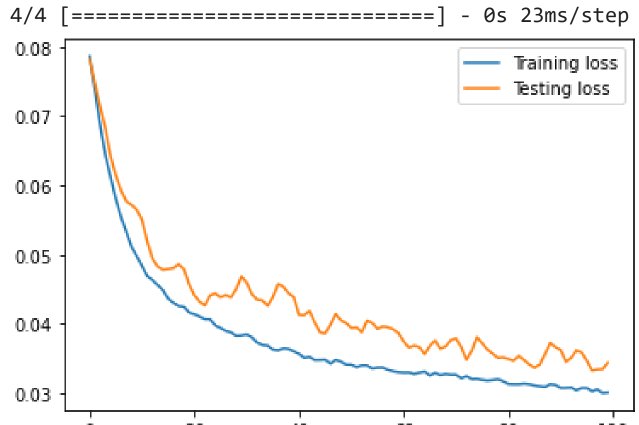
In [14]:

```
#Visualisation:
import matplotlib.pyplot as plt

history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test))
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Testing loss')
plt.legend()
plt.show()
```

```
Epoch 1/100
4/4 [==============================] - 0s 61ms/step - loss: 0.0787 - val_loss: 0.0782
Epoch 2/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0738 - val_loss: 0.0747
Epoch 3/100
4/4 [==============================] - 0s 28ms/step - loss: 0.0688 - val_loss: 0.0715
Epoch 4/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0644 - val_loss: 0.0683
Epoch 5/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0612 - val_loss: 0.0642
Epoch 6/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0580 - val_loss: 0.0614
Epoch 7/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0554 - val_loss: 0.0593
Epoch 8/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0533 - val_loss: 0.0577
Epoch 9/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0511 - val_loss: 0.0572
Epoch 10/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0498 - val_loss: 0.0565
Epoch 11/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0483 - val_loss: 0.0550
Epoch 12/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0469 - val_loss: 0.0519
Epoch 13/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0463 - val_loss: 0.0494
Epoch 14/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0456 - val_loss: 0.0483
Epoch 15/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0449 - val_loss: 0.0478
Epoch 16/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0436 - val_loss: 0.0479
Epoch 17/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0430 - val_loss: 0.0480
Epoch 18/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0426 - val_loss: 0.0486
Epoch 19/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0425 - val_loss: 0.0479
Epoch 20/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0416 - val_loss: 0.0457
Epoch 21/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0414 - val_loss: 0.0441
Epoch 22/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0410 - val_loss: 0.0432
Epoch 23/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0407 - val_loss: 0.0427
Epoch 24/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0407 - val_loss: 0.0441
Epoch 25/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0397 - val_loss: 0.0444
Epoch 26/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0394 - val_loss: 0.0438
Epoch 27/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0390 - val_loss: 0.0442
Epoch 28/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0387 - val_loss: 0.0438
Epoch 29/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0383 - val_loss: 0.0450
Epoch 30/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0383 - val_loss: 0.0468
Epoch 31/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0385 - val_loss: 0.0460
Epoch 32/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0381 - val_loss: 0.0443
Epoch 33/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0373 - val_loss: 0.0435
Epoch 34/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0369 - val_loss: 0.0434
Epoch 35/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0368 - val_loss: 0.0426
Epoch 36/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0363 - val_loss: 0.0438
Epoch 37/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0362 - val_loss: 0.0458
Epoch 38/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0364 - val_loss: 0.0453
Epoch 39/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0364 - val_loss: 0.0444
Epoch 40/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0360 - val_loss: 0.0439
Epoch 41/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0357 - val_loss: 0.0413
Epoch 42/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0351 - val_loss: 0.0412
Epoch 43/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0352 - val_loss: 0.0419
Epoch 44/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0347 - val_loss: 0.0402
Epoch 45/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0348 - val_loss: 0.0388
Epoch 46/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0348 - val_loss: 0.0386
Epoch 47/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0342 - val_loss: 0.0399
Epoch 48/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0348 - val_loss: 0.0414
Epoch 49/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0345 - val_loss: 0.0405
Epoch 50/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0341 - val_loss: 0.0401
Epoch 51/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0341 - val_loss: 0.0394
Epoch 52/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0337 - val_loss: 0.0395
Epoch 53/100
4/4 [==============================] - 0s 28ms/step - loss: 0.0340 - val_loss: 0.0388
Epoch 54/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0340 - val_loss: 0.0405
Epoch 55/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0335 - val_loss: 0.0401
Epoch 56/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0337 - val_loss: 0.0393
```

```
Epoch 57/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0337 - val_loss: 0.0396
Epoch 58/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0333 - val_loss: 0.0395
Epoch 59/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0331 - val_loss: 0.0392
Epoch 60/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0330 - val_loss: 0.0387
Epoch 61/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0329 - val_loss: 0.0374
Epoch 62/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0329 - val_loss: 0.0366
Epoch 63/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0327 - val_loss: 0.0369
Epoch 64/100
4/4 [==============================] - 0s 30ms/step - loss: 0.0329 - val_loss: 0.0366
Epoch 65/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0331 - val_loss: 0.0356
Epoch 66/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0325 - val_loss: 0.0368
Epoch 67/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0329 - val_loss: 0.0376
Epoch 68/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0326 - val_loss: 0.0364
Epoch 69/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0327 - val_loss: 0.0367
Epoch 70/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0326 - val_loss: 0.0376
Epoch 71/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0326 - val_loss: 0.0379
Epoch 72/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0321 - val_loss: 0.0366
Epoch 73/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0324 - val_loss: 0.0348
Epoch 74/100
4/4 [==============================] - 0s 27ms/step - loss: 0.0320 - val_loss: 0.0361
Epoch 75/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0320 - val_loss: 0.0381
Epoch 76/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0319 - val_loss: 0.0372
Epoch 77/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0318 - val_loss: 0.0363
Epoch 78/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0319 - val_loss: 0.0357
Epoch 79/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0320 - val_loss: 0.0351
Epoch 80/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0317 - val_loss: 0.0351
Epoch 81/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0313 - val_loss: 0.0349
Epoch 82/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0313 - val_loss: 0.0347
Epoch 83/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0313 - val_loss: 0.0354
Epoch 84/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0313 - val_loss: 0.0347
Epoch 85/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0313 - val_loss: 0.0341
Epoch 86/100
4/4 [==============================] - 0s 26ms/step - loss: 0.0311 - val_loss: 0.0336
Epoch 87/100
4/4 [==============================] - 0s 25ms/step - loss: 0.0310 - val_loss: 0.0341
Epoch 88/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0309 - val_loss: 0.0356
Epoch 89/100
4/4 [==============================] - 0s 24ms/step - loss: 0.0313 - val_loss: 0.0372
Epoch 90/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0312 - val_loss: 0.0366
Epoch 91/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0307 - val_loss: 0.0362
Epoch 92/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0307 - val_loss: 0.0345
Epoch 93/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0308 - val_loss: 0.0351
Epoch 94/100
4/4 [==============================] - 0s 21ms/step - loss: 0.0304 - val_loss: 0.0361
Epoch 95/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0307 - val_loss: 0.0359
Epoch 96/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0307 - val_loss: 0.0346
Epoch 97/100
4/4 [==============================] - 0s 21ms/step - loss: 0.0303 - val_loss: 0.0332
Epoch 98/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0305 - val_loss: 0.0334
Epoch 99/100
4/4 [==============================] - 0s 22ms/step - loss: 0.0300 - val_loss: 0.0334
Epoch 100/100
4/4 [==============================] - 0s 23ms/step - loss: 0.0301 - val_loss: 0.0344
```



Observation:

The model is trained on the training set using the fit method and evaluated on the testing set using the evaluate method. The predicted target values are then computed using the trained model and printed alongside the actual target values for the first 10 samples.

Conclusion:

The given code is an example of how to build and train a deep neural network (DNN) model for classification on the iris dataset using TensorFlow's Keras API. The code demonstrates how to prepare the dataset using scikit-learn, define a DNN model with an input layer, two hidden layers, and an output layer, and train and evaluate the model on the dataset. The code also emphasizes the importance of splitting the dataset into training and testing sets to evaluate the model's performance on unseen data.