Tutorial #1: Building a Neural Network for Iris Classification Using TensorFlow and Keras

In this tutorial, we will build a simple neural network using TensorFlow and Keras to classify the Iris dataset. The Iris dataset is a famous dataset in machine learning, consisting of 150 samples of three species of Iris flowers: Setosa, Versicolour, and Virginica. Each sample has four features: sepal length, sepal width, petal length, and petal width.

1. Prerequisites

Before starting, ensure you have the required libraries installed:

```
- TensorFlow
- scikit-learn
- NumPy
```

You can install them using pip:

```
pip install tensorflow scikit-learn numpy
```

Then, import the library below:

```
import tensorflow as tf
from tf_keras.models import Sequential
from tf_keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import numpy as np
```

2. Load and Explore the Iris Dataset

The Iris dataset can be loaded directly using scikit-learn. Here's how we load and explore the dataset:

```
iris_data = load_iris() # load the iris dataset

print('Example data: ')
print(iris_data.data[:5])
print('Example labels: ')
print(iris_data.target[:5])
```

3. Preprocess the Data

Neural networks work better with one-hot encoded labels, so we'll convert the labels into a one-hot representation.

```
x = iris_data.data
y_ = iris_data.target.reshape(-1, 1) # Convert data to a single column
encoder = OneHotEncoder(sparse_output=False)
y = encoder.fit_transform(y_)
```

Split the data into training and testing sets (80% training and 20% testing):

```
train_x, test_x, train_y, test_y = train_test_split(x, y,
test_size=0.20)
```

4. Build the Neural Network

We use the Keras Sequential API to define the model architecture.

Model Architecture:

1. **Input Layer**: Accepts 4 features.
2. **Hidden Layer 1**: 10 neurons with ReLU activation.
3. **Hidden Layer 2**: 10 neurons with ReLU activation.
4. **Output Layer**: 3 neurons (one for each class) with softmax activation.

```
model = Sequential()

model.add(Dense(10, input_shape=(4,), activation='relu', name='HL_1'))
model.add(Dense(10, activation='relu', name='HL2'))
model.add(Dense(3, activation='softmax', name='output'))
```

5. Compile the Model

The model uses:
- Adam Optimizer: For efficient training.
- Categorical Crossentropy Loss: Suitable for multi-class classification.
- Accuracy Metric: To measure model performance.

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

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

6. Train the Model

Train the model using the training data for 200 epochs with a batch size of 5. Set verbose=2 for detailed output.

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

7. Evaluate the Model

Evaluate the trained model on the test data to calculate the loss and accuracy.

```
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]))
```

8. Inspect Model Weights

Print the weights and biases of each layer for interpretability.

```python
for layer in model.layers:
    weights, biases = layer.get_weights()
    print(f"Layer: {layer.name}")
    print(f"Weights:\n{weights}")
    print(f"Biases:\n{biases}")
```

9. Conclusion

In this tutorial, we successfully built, trained, and evaluated a neural network to classify the Iris dataset using TensorFlow and Keras. By following these steps, you have learned to:

1. Preprocess datasets effectively for machine learning tasks.
2. Design and implement a simple yet effective neural network.
3. Train models to achieve high accuracy on a multi-class classification problem.
4. Evaluate model performance and analyze weights for better interpretability.