```
# Check whether a simple NN having one neuron can learn a linear
function, e.g., y = ax + b.
# Sangeeta Biswas
# 31.12.2021
#
# How to run:
# $ Python_Path ProgramFile_Path
# $ Tensorflow/bin/python DeepLearning/1D_LinearFunction_Learner.py

from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import Model
import numpy as np
from tensorflow.keras.callbacks import EarlyStopping, History
import matplotlib.pyplot as plt

DIR = '/home/bibrity/DeepLearning/'

def main():
        # Build a model
        model = build_model()

        # Prepare data sets
        testX, testY, trainX, trainY = prepare_datasets()

        # Train the model
        callbackList = [EarlyStopping(monitor = 'val_loss', patience =
20), History()]
        history = model.fit(trainX, trainY, epochs = 300, batch_size =
32, callbacks = callbackList, validation_split = 0.2)
        plot_loss(history)

        # Test the performance
        predictedY = model.predict(testX)
        print(testY)
        print(predictedY)

        predictedA = model.layers[1].get_weights()[0][0][0]
        predictedB = model.layers[1].get_weights()[1][0]
        print('a: {}, b: {}'.format(predictedA, predictedB))

def prepare_datasets():
        testX = np.arange(100)
        testY = hidden_function(testX)
        trainX = np.arange(100, 65000)
        trainY = hidden_function(trainX)

        return testX, testY, trainX, trainY

def plot_loss(history):
        loss = history.history['loss']
        valLoss = history.history['val_loss']
        epochs = range(1, len(loss) + 1)

        plt.figure(figsize = (20, 20))
        plt.rcParams['font.size'] = '20'
        plt.plot(epochs, loss, 'bo-', label = 'Training loss')
        plt.plot(epochs, valLoss, 'k*-', label = 'Validation loss')
        plt.title('Training Loss Vs. Validation Loss')
        plt.legend()

        figPath = DIR + 'TrainvsVal_Loss.png'
        plt.savefig(figPath)
        plt.close()

def hidden_function(x):
        a = 5; b = 3
        y = a * x + b

        return y

def build_model():
        # Layers of the model.
        inputs = Input(1,)
        outputs = Dense(1)(inputs)

        # Build the model.
        model = Model(inputs, outputs)

        # Configures the model for training.
        model.compile(loss = 'mse', optimizer = 'rmsprop')

        # Display the model architecture.
        model.summary()

        return model

if __name__ == '__main__':
        main()
```