Open in Colab

In this exercise you'll try to build a neural network that predicts the price of a house according to a simple formula.

So, imagine if house pricing was as easy as a house costs 50k + 50k per bedroom, so that a 1 bedroom house costs 100k, a 2 bedroom house costs 150k etc.

How would you create a neural network that learns this relationship so that it would predict a 7 bedroom house as costing close to 400k etc.

Hint: Your network might work better if you scale the house price down. You don't have to give the answer 400...it might be better to create something that predicts the number 4, and then your answer is in the 'hundreds of thousands' etc.

(**Note:** You can run the notebook using TensorFlow 2.5.0)

```
import tensorflow as tf
import numpy as np
from tensorflow import keras


# GRADED FUNCTION: house_model
def house_model(y_new):
    xs = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], dtype=float) # No. of Bed(s) per House
    ys = np.array([1.0, 1.5, 2.0, 2.5, 3.0, 3.5], dtype=float) # Price of House as per No. of Bed(s)

    model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
    model.compile(optimizer='sgd', loss='mean_squared_error')
    model.fit(xs, ys, epochs=1000)
    return model.predict(y_new)[0]


prediction = house_model([7.0])
print(prediction)
```

```
Epoch 972/1000
1/1 [==============================] - 0s 6ms/step - loss: 4.4092e-06
Epoch 973/1000
1/1 [==============================] - 0s 4ms/step - loss: 4.3770e-06
Epoch 974/1000
1/1 [==============================] - 0s 5ms/step - loss: 4.3452e-06
Epoch 975/1000
1/1 [==============================] - 0s 8ms/step - loss: 4.3135e-06
Epoch 976/1000
```

```
1/1 [==============================] - 0s 12ms/step - loss: 4.2821e-06
Epoch 977/1000
1/1 [==============================] - 0s 11ms/step - loss: 4.2507e-06
Epoch 978/1000
1/1 [==============================] - 0s 14ms/step - loss: 4.2198e-06
Epoch 979/1000
1/1 [==============================] - 0s 8ms/step - loss: 4.1891e-06
Epoch 980/1000
1/1 [==============================] - 0s 14ms/step - loss: 4.1585e-06
Epoch 981/1000
1/1 [==============================] - 0s 5ms/step - loss: 4.1284e-06
Epoch 982/1000
1/1 [==============================] - 0s 6ms/step - loss: 4.0984e-06
Epoch 983/1000
1/1 [==============================] - 0s 8ms/step - loss: 4.0684e-06
Epoch 984/1000
1/1 [==============================] - 0s 3ms/step - loss: 4.0388e-06
Epoch 985/1000
1/1 [==============================] - 0s 6ms/step - loss: 4.0095e-06
Epoch 986/1000
1/1 [==============================] - 0s 6ms/step - loss: 3.9801e-06
Epoch 987/1000
1/1 [==============================] - 0s 8ms/step - loss: 3.9512e-06
Epoch 988/1000
1/1 [==============================] - 0s 4ms/step - loss: 3.9225e-06
Epoch 989/1000
1/1 [==============================] - 0s 7ms/step - loss: 3.8939e-06
Epoch 990/1000
1/1 [==============================] - 0s 3ms/step - loss: 3.8654e-06
Epoch 991/1000
1/1 [==============================] - 0s 13ms/step - loss: 3.8373e-06
Epoch 992/1000
1/1 [==============================] - 0s 15ms/step - loss: 3.8092e-06
Epoch 993/1000
1/1 [==============================] - 0s 6ms/step - loss: 3.7816e-06
Epoch 994/1000
1/1 [==============================] - 0s 4ms/step - loss: 3.7539e-06
Epoch 995/1000
1/1 [==============================] - 0s 3ms/step - loss: 3.7267e-06
Epoch 996/1000
1/1 [==============================] - 0s 3ms/step - loss: 3.6995e-06
Epoch 997/1000
1/1 [==============================] - 0s 4ms/step - loss: 3.6726e-06
Epoch 998/1000
1/1 [==============================] - 0s 4ms/step - loss: 3.6458e-06
Epoch 999/1000
1/1 [==============================] - 0s 4ms/step - loss: 3.6191e-06
Epoch 1000/1000
```

```
1/1 [==============================] - 0s 3ms/step - loss: 3.5927e-06
[4.0027337]
```