Introduction to Regression with neural networks in TensorFlow

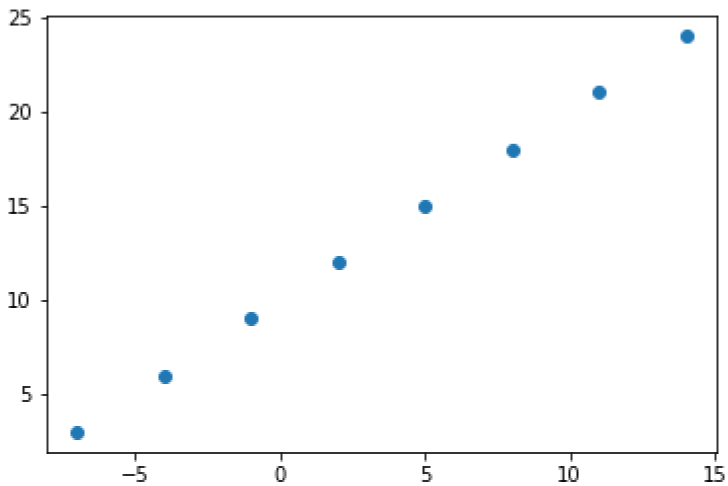
A simple way to define regression is: predicting a numerical variable based on some other combination of variables. In short, predicting a number

```
# Import TensorFlow
import tensorflow as tf
print(tf.__version__)
```

```
2.5.0
```

Creating data to view and fit

```
import numpy as np
import matplotlib.pyplot as plt

# Create features
X = np.array([-7.0, -4.0, -1.0, 2.0, 5.0, 8.0, 11.0, 14.0])

# Create label
y = np.array([3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0])

#Visualize the data
plt.scatter(X, y);
```




```
y == X +10
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

Input and output shapes

```
# Create a demo data for our housing price problem
```

```
house_info = tf.constant(["bedroom", "bathroom", "garage"])
house_price = tf.constant([937000])

house_info, house_price
```

```
(<tf.Tensor: shape=(3,), dtype=string, numpy=array([b'bedroom', b'bathroom', b'garage'], dtype=ob
 <tf.Tensor: shape=(1,), dtype=int32, numpy=array([937000], dtype=int32)>)
```

```
input_shape = X.shape
output_shape = y.shape

input_shape, output_shape
```

```
((8,), (8,))
```

```
# Turn our numpy array into tensors
X = tf.constant(X)
y = tf.constant(y)

X, y
```

```
(<tf.Tensor: shape=(8,), dtype=float64, numpy=array([-7., -4., -1.,  2.,  5.,  8., 11., 14.])>,
 <tf.Tensor: shape=(8,), dtype=float64, numpy=array([ 3.,  6.,  9., 12., 15., 18., 21., 24.])>)
```

```
input_shape = X[0].shape
output_shape = y[0].shape
input_shape, output_shape
```

```
(TensorShape([]), TensorShape([]))
```

Steps in modelling with TensorFlow

1. **Creating a model** - define the input and output layers, as well as hidden layers of a deep learning model.
2. **Compiling the model** - define loss function (in other words, the function which tells how wrong our model is) and optimizer (tells our model how to improve the pattern its learning) and evaluation metrics (what we can use to interpret the performance of our model).
3. **Fitting a model** - letting the model to find patterns between X and y (feaures and labels).

```
# Set the random seed
tf.random.set_seed = 42

# 1. Creating a model using the Sequential API
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# Compiling our model
model.compile(loss=tf.keras.losses.mae,
              optimizer = tf.keras.optimizers.SGD(),
              metrics=["mae"])
```

```
              metrics=["mae"])

# Fit the model
model.fit(X, y, epochs=5)
```

```
Epoch 1/5
1/1 [==============================] - 1s 561ms/step - loss: 21.5702 - mae: 21.5702
Epoch 2/5
1/1 [==============================] - 0s 7ms/step - loss: 21.1771 - mae: 21.1771
Epoch 3/5
1/1 [==============================] - 0s 6ms/step - loss: 20.7840 - mae: 20.7840
Epoch 4/5
1/1 [==============================] - 0s 7ms/step - loss: 20.4234 - mae: 20.4234
Epoch 5/5
1/1 [==============================] - 0s 6ms/step - loss: 20.1421 - mae: 20.1421
<tensorflow.python.keras.callbacks.History at 0x7fdc81658610>
```

```
#  Try and make a prediction with our trained model
y_pred = model.predict([17.0])
y_pred
```

```
array([[-23.068537]], dtype=float32)
```

Improving our model

We can improve a model by altering the steps we took in creating the model.

1. **Creating model** - Here we can add more layers, increase the number of hidden units(neurons), also change the activation function in each layer.
2. **Compiling model** - Here we might change the optimizer function or change the learning rate of the optimization function.
3. **Fitting model** - Here we might fit the model with more epochs or give the model more data

```
# Let's rebuild our model

# 1. Create the model
model = tf.keras.Sequential([
        tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])

# 3. Fit the model
model.fit(X, y, epochs=100)
```

```
1/1 [==============================] - 0s 5ms/step - loss: 6.9956 - mae: 6.9956
Epoch 72/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9900 - mae: 6.9900
Epoch 73/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9844 - mae: 6.9844
Epoch 74/100
```

```
1/1 [==============================] - 0s 6ms/step - loss: 6.9788 - mae: 6.9788
Epoch 75/100
1/1 [==============================] - 0s 4ms/step - loss: 6.9731 - mae: 6.9731
Epoch 76/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9675 - mae: 6.9675
Epoch 77/100
1/1 [==============================] - 0s 16ms/step - loss: 6.9619 - mae: 6.9619
Epoch 78/100
1/1 [==============================] - 0s 11ms/step - loss: 6.9563 - mae: 6.9563
Epoch 79/100
1/1 [==============================] - 0s 4ms/step - loss: 6.9506 - mae: 6.9506
Epoch 80/100
1/1 [==============================] - 0s 5ms/step - loss: 6.9450 - mae: 6.9450
Epoch 81/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9394 - mae: 6.9394
Epoch 82/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9338 - mae: 6.9338
Epoch 83/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9281 - mae: 6.9281
Epoch 84/100
1/1 [==============================] - 0s 5ms/step - loss: 6.9225 - mae: 6.9225
Epoch 85/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9169 - mae: 6.9169
Epoch 86/100
1/1 [==============================] - 0s 5ms/step - loss: 6.9113 - mae: 6.9113
Epoch 87/100
1/1 [==============================] - 0s 6ms/step - loss: 6.9056 - mae: 6.9056
Epoch 88/100
1/1 [==============================] - 0s 7ms/step - loss: 6.9000 - mae: 6.9000
Epoch 89/100
1/1 [==============================] - 0s 5ms/step - loss: 6.8944 - mae: 6.8944
Epoch 90/100
1/1 [==============================] - 0s 4ms/step - loss: 6.8888 - mae: 6.8888
Epoch 91/100
1/1 [==============================] - 0s 6ms/step - loss: 6.8831 - mae: 6.8831
Epoch 92/100
1/1 [==============================] - 0s 8ms/step - loss: 6.8775 - mae: 6.8775
Epoch 93/100
1/1 [==============================] - 0s 6ms/step - loss: 6.8719 - mae: 6.8719
Epoch 94/100
1/1 [==============================] - 0s 7ms/step - loss: 6.8663 - mae: 6.8663
Epoch 95/100
1/1 [==============================] - 0s 4ms/step - loss: 6.8606 - mae: 6.8606
Epoch 96/100
1/1 [==============================] - 0s 9ms/step - loss: 6.8550 - mae: 6.8550
Epoch 97/100
1/1 [==============================] - 0s 6ms/step - loss: 6.8494 - mae: 6.8494
Epoch 98/100
1/1 [==============================] - 0s 5ms/step - loss: 6.8438 - mae: 6.8438
Epoch 99/100
1/1 [==============================] - 0s 4ms/step - loss: 6.8381 - mae: 6.8381
Epoch 100/100
1/1 [==============================] - 0s 13ms/step - loss: 6.8325 - mae: 6.8325
<tensorflow.python.keras.callbacks.History at 0x7fdc7d956d10>
```

```
# Let's see the prediction of new model
model.predict([17.0])
```

```
array([[29.427305]], dtype=float32)
```

```
# Let's try to change one more parameter
```

```
# Let's try to change one more parameter

# 1. Create a model
model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation=None),
        tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.Adam(lr=.01),
              metrics=["mae"])

# 3. Fit the model
model.fit(X, y, epochs=100)
```

```
Epoch 72/100
1/1 [==============================] - 0s 26ms/step - loss: 0.1582 - mae: 0.1582
Epoch 73/100
1/1 [==============================] - 0s 13ms/step - loss: 0.5127 - mae: 0.5127
Epoch 74/100
1/1 [==============================] - 0s 5ms/step - loss: 0.7366 - mae: 0.7366
Epoch 75/100
1/1 [==============================] - 0s 8ms/step - loss: 0.6627 - mae: 0.6627
Epoch 76/100
1/1 [==============================] - 0s 26ms/step - loss: 0.4188 - mae: 0.4188
Epoch 77/100
1/1 [==============================] - 0s 4ms/step - loss: 0.6900 - mae: 0.6900
Epoch 78/100
1/1 [==============================] - 0s 4ms/step - loss: 0.8357 - mae: 0.8357
Epoch 79/100
1/1 [==============================] - 0s 5ms/step - loss: 0.8087 - mae: 0.8087
Epoch 80/100
1/1 [==============================] - 0s 5ms/step - loss: 0.6280 - mae: 0.6280
Epoch 81/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3078 - mae: 0.3078
Epoch 82/100
1/1 [==============================] - 0s 5ms/step - loss: 0.5312 - mae: 0.5312
Epoch 83/100
1/1 [==============================] - 0s 4ms/step - loss: 0.6668 - mae: 0.6668
Epoch 84/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4799 - mae: 0.4799
Epoch 85/100
1/1 [==============================] - 0s 5ms/step - loss: 0.1433 - mae: 0.1433
Epoch 86/100
1/1 [==============================] - 0s 6ms/step - loss: 0.6336 - mae: 0.6336
Epoch 87/100
1/1 [==============================] - 0s 6ms/step - loss: 0.8773 - mae: 0.8773
Epoch 88/100
1/1 [==============================] - 0s 4ms/step - loss: 0.7935 - mae: 0.7935
Epoch 89/100
1/1 [==============================] - 0s 6ms/step - loss: 0.4541 - mae: 0.4541
Epoch 90/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3729 - mae: 0.3729
Epoch 91/100
1/1 [==============================] - 0s 6ms/step - loss: 0.5931 - mae: 0.5931
Epoch 92/100
1/1 [==============================] - 0s 8ms/step - loss: 0.6333 - mae: 0.6333
Epoch 93/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3962 - mae: 0.3962
Epoch 94/100
```

```
1/1 [==============================] - 0s 7ms/step - loss: 0.0906 - mae: 0.0906
Epoch 95/100
1/1 [==============================] - 0s 6ms/step - loss: 0.2437 - mae: 0.2437
Epoch 96/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1645 - mae: 0.1645
Epoch 97/100
1/1 [==============================] - 0s 6ms/step - loss: 0.2005 - mae: 0.2005
Epoch 98/100
1/1 [==============================] - 0s 8ms/step - loss: 0.2122 - mae: 0.2122
Epoch 99/100
1/1 [==============================] - 0s 5ms/step - loss: 0.0941 - mae: 0.0941
Epoch 100/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0654 - mae: 0.0654
```

```
# Let's see the prediction again
model.predict([17.0])
```

```
array([[27.357454]], dtype=float32)
```

Evaluating a model

A typical workflow in building a neural network model is:

```
Build a model -> fit it -> evaluate it -> tweak a model ->
evaluate it ->  ...
```

```
# Make a bigger dataset
X = tf.range(-100, 100, 4)
X
```

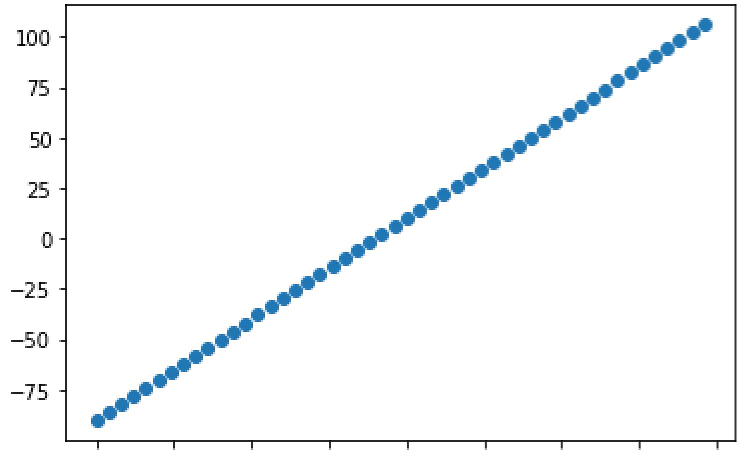
```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([-100,  -96,  -92,  -88,  -84,  -80,  -76,  -72,  -68,  -64,  -60,
        -56,  -52,  -48,  -44,  -40,  -36,  -32,  -28,  -24,  -20,  -16,
        -12,   -8,   -4,    0,    4,    8,   12,   16,   20,   24,   28,
         32,   36,   40,   44,   48,   52,   56,   60,   64,   68,   72,
         76,   80,   84,   88,   92,   96], dtype=int32)>
```

```
# Make labels for the data
y = X + 10
y
```

```
<tf.Tensor: shape=(50,), dtype=int32, numpy=
array([-90, -86, -82, -78, -74, -70, -66, -62, -58, -54, -50, -46, -42,
       -38, -34, -30, -26, -22, -18, -14, -10,  -6,  -2,   2,   6,  10,
        14,  18,  22,  26,  30,  34,  38,  42,  46,  50,  54,  58,  62,
        66,  70,  74,  78,  82,  86,  90,  94,  98, 102, 106], dtype=int32)>
```

```
# Visualize the data
plt.scatter(X, y)
```

```
<matplotlib.collections.PathCollection at 0x7fdc79887550>
```



The 3 sets...

- Training set - This is the set from which the model learns, typically 70-80% of the data.
- Validation set - In this set the model gets tuned, typically 10-15% of the data.
- Test set - In this set the model gets evaluated, typically 10-15% of the data.

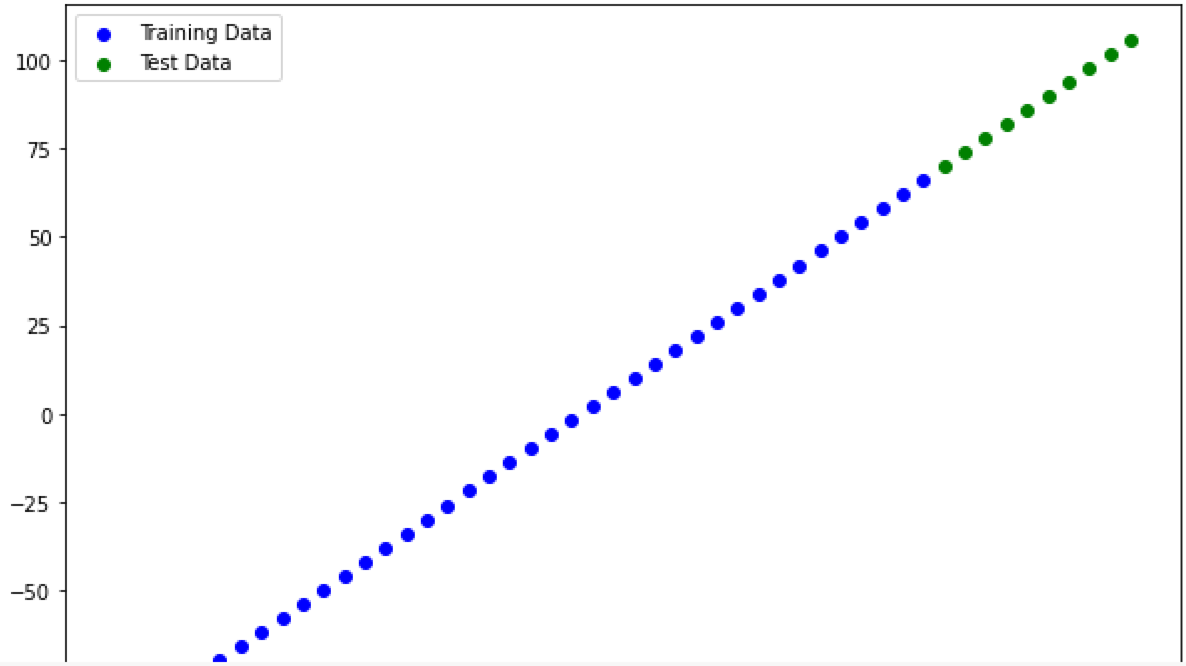
```
# Check the number of samples
len(X)
```

```
50
```

```
# Split the data into train and test sets
X_train = X[:40]
X_test = X[40:]

y_train = y[:40]
y_test = y[40:]

len(X_train), len(X_test), len(y_train), len(y_test)
```

```
(40, 10, 40, 10)
```

```
### Visualizing the data
plt.figure(figsize=(10, 7))

# Plot trainig data in blue
plt.scatter(X_train, y_train, c="b", label="Training Data")

# Plot test data in green
plt.scatter(X_test, y_test, c="g", label="Test Data")

# Show legend
plt.legend();
```




```
# Let's build another neural network for our new data

# 1. Create a model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics = ["mae"])

# # 3. Fit the model
# model.fit(X_train, y_train, epochs=100)
```

```
# model.summary()
```

```
# Let's create a model which builds automatically by defining the input_shape argument
tf.random.set_seed = 42

# Build a model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, input_shape=[1], name="input_layer"),
    tf.keras.layers.Dense(10, name="hidden_layer_1"),
    tf.keras.layers.Dense(1, name="output_layer"),
], name="first_model")

# Compile model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.Adam(lr=.01),
              metrics=["mae"])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375:
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

```
model.summary()
```

```
Model: "first_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (Dense)          (None, 10)                20
_________________________________________________________________
hidden_layer_1 (Dense)       (None, 10)                110
_________________________________________________________________
output_layer (Dense)         (None, 1)                 11
=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
_________________________________________________________________
```

- Total params - total number of parameters in the model.
- Trainable params - these are the parameters (patterns) a model can update as it trains.
- Non-trainable params - these parameters aren't updated during training.

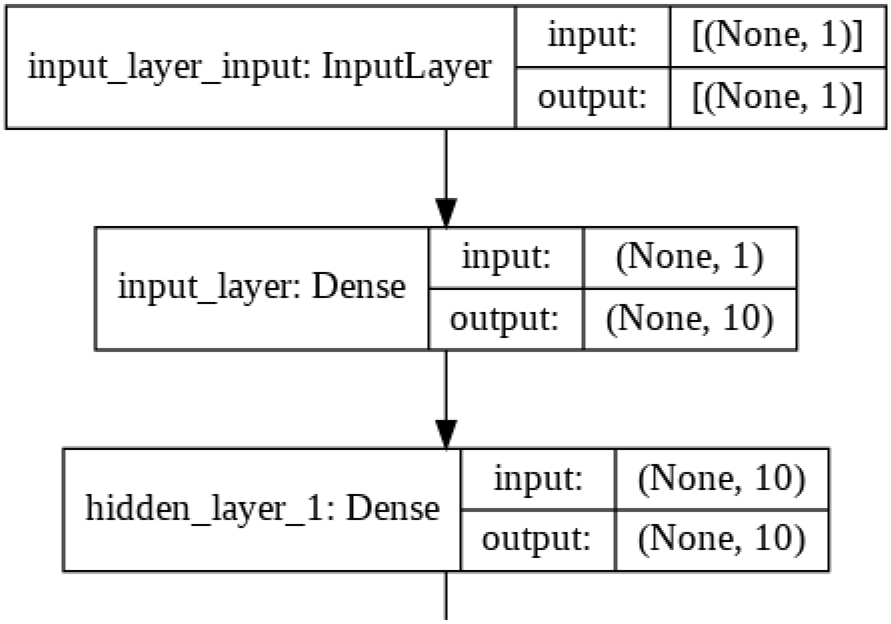
```
# Let's fit our model with the training data
model.fit(X_train, y_train, epochs=100, verbose=0)
```

```
<tensorflow.python.keras.callbacks.History at 0x7fdc78f3d9d0>
```

```
# Get our model summary
model.summary()
```

```
Model: "first_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (Dense)          (None, 10)                20
_________________________________________________________________
hidden_layer_1 (Dense)       (None, 10)                110
_________________________________________________________________
output_layer (Dense)         (None, 1)                 11
=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model

plot_model(model, show_shapes=True)
```



Visualizing models prediction

```
# Make predictions
y_pred = model.predict(X_test)
y_pred
```

```
array([[ 69.18848 ],
       [ 73.13339 ],
       [ 77.0783  ],
       [ 81.023224],
       [ 84.96814 ],
       [ 88.913055],
       [ 92.857956],
       [ 96.80288 ],
       [100.7478  ],
       [104.69271 ]], dtype=float32)
```

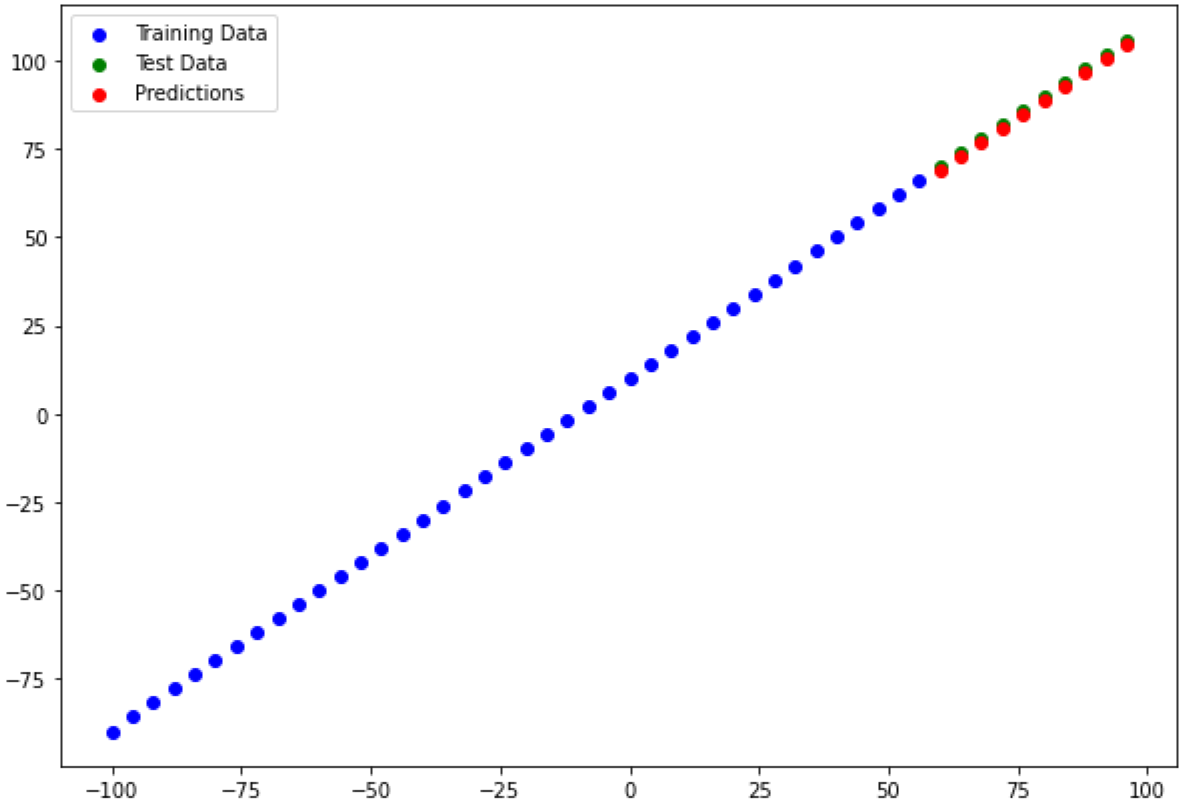
```
y_test
```

```
<tf.Tensor: shape=(10,), dtype=int32, numpy=array([ 70,  74,  78,  82,  86,  90,  94,  98, 102, 1
```

```
# Let's create a plotting function

def plot_predictions(train_data=X_train, train_label=y_train,
                     test_data=X_test, test_label=y_test,
                     predictions=y_pred):
  """
  Plots training data, test data and compares predictions to ground truth
  """
  plt.figure(figsize=(10,7))
  # Plot training data in blue
  plt.scatter(train_data, train_label, c="b", label="Training Data")

  # Plot test data in green
  plt.scatter(test_data, test_label, c="g", label="Test Data")
```

```
  # Plot model's prediction in red
  plt.scatter(test_data, predictions, c="r", label="Predictions")

  # Show legend
  plt.legend()
```

```
plot_predictions(X_train, y_train, X_test, y_test, y_pred)
```



Evaluating our model's prediction with evaluation metrics

```
# Evaluate the model on test data
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 133ms/step - loss: 1.0594 - mae: 1.0594
[1.0594024658203125, 1.0594024658203125]
```

```
# Calculate the mean absolute error
mae = tf.keras.losses.MAE(y_test, tf.constant(tf.squeeze(y_pred)))
mae
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=1.0594063>
```

```
# Calculate the mean squared error
mse = tf.metrics.mean_squared_error(y_test, tf.constant(tf.squeeze(y_pred)))
mse
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=1.1473751>
```

Running experiments to improve our model

Build a model with 1 layer and 100 epochs

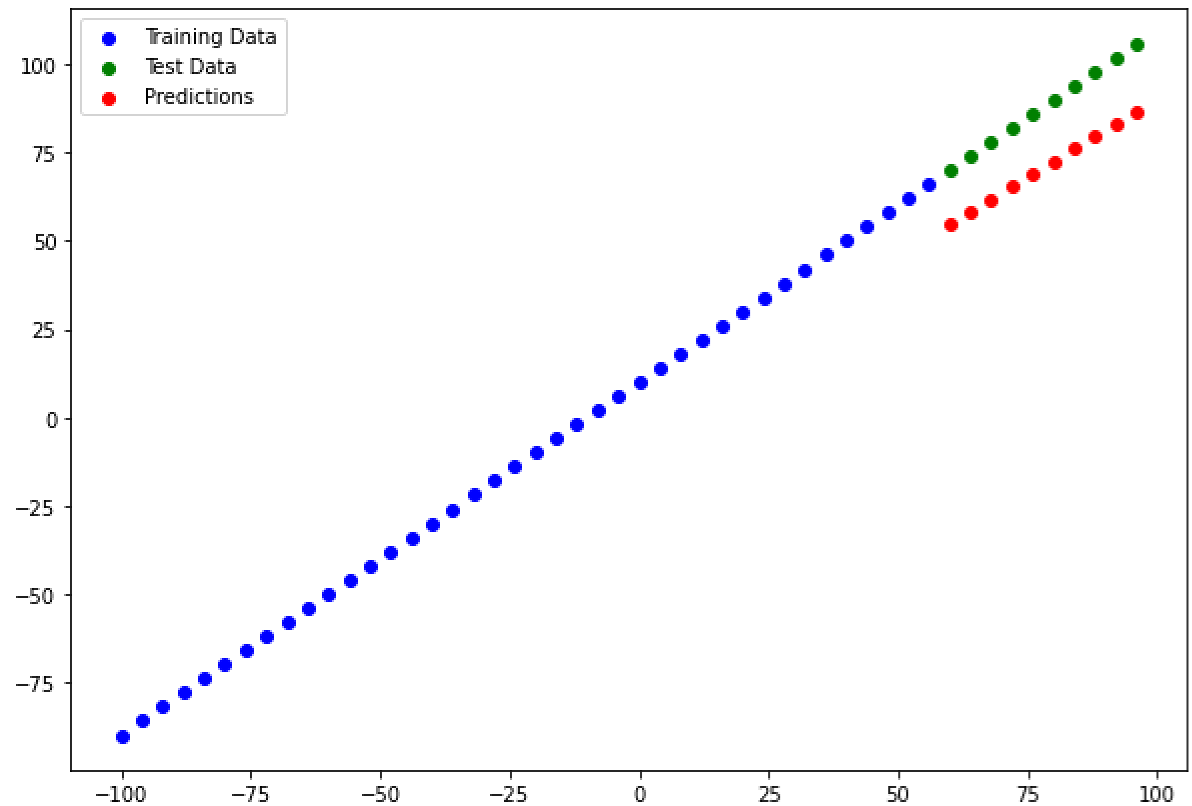
```
# Set random seed
tf.random.set_seed=42

# 1. Create a model
model_1 = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model_1.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])

# 3. Fit the model
model_1.fit(X_train, y_train, epochs=100)
```

```
2/2 [==============================] - 0s 3ms/step - loss: 7.9682 - mae: 7.9682
Epoch 72/100
2/2 [==============================] - 0s 5ms/step - loss: 7.7240 - mae: 7.7240
Epoch 73/100
2/2 [==============================] - 0s 6ms/step - loss: 13.3774 - mae: 13.3774
Epoch 74/100
2/2 [==============================] - 0s 5ms/step - loss: 8.5017 - mae: 8.5017
Epoch 75/100
2/2 [==============================] - 0s 8ms/step - loss: 10.4065 - mae: 10.4065
Epoch 76/100
2/2 [==============================] - 0s 6ms/step - loss: 9.5440 - mae: 9.5440
Epoch 77/100
2/2 [==============================] - 0s 4ms/step - loss: 10.5926 - mae: 10.5926
Epoch 78/100
2/2 [==============================] - 0s 6ms/step - loss: 13.7356 - mae: 13.7356
Epoch 79/100
2/2 [==============================] - 0s 3ms/step - loss: 13.7776 - mae: 13.7776
Epoch 80/100
2/2 [==============================] - 0s 4ms/step - loss: 8.1534 - mae: 8.1534
Epoch 81/100
2/2 [==============================] - 0s 4ms/step - loss: 14.1404 - mae: 14.1404
Epoch 82/100
2/2 [==============================] - 0s 5ms/step - loss: 7.9231 - mae: 7.9231
Epoch 83/100
2/2 [==============================] - 0s 3ms/step - loss: 13.2788 - mae: 13.2788
Epoch 84/100
2/2 [==============================] - 0s 6ms/step - loss: 11.3882 - mae: 11.3882
Epoch 85/100
2/2 [==============================] - 0s 7ms/step - loss: 9.2880 - mae: 9.2880
Epoch 86/100
2/2 [==============================] - 0s 9ms/step - loss: 15.0003 - mae: 15.0003
Epoch 87/100
2/2 [==============================] - 0s 5ms/step - loss: 9.4284 - mae: 9.4284
Epoch 88/100
2/2 [==============================] - 0s 6ms/step - loss: 8.5861 - mae: 8.5861
Epoch 89/100
```

```
2/2 [==============================] - 0s 9ms/step - loss: 9.4448 - mae: 9.4448
Epoch 90/100
2/2 [==============================] - 0s 4ms/step - loss: 24.1670 - mae: 24.1670
Epoch 91/100
2/2 [==============================] - 0s 6ms/step - loss: 12.6051 - mae: 12.6051
Epoch 92/100
2/2 [==============================] - 0s 6ms/step - loss: 8.6111 - mae: 8.6111
Epoch 93/100
2/2 [==============================] - 0s 4ms/step - loss: 10.2668 - mae: 10.2668
Epoch 94/100
2/2 [==============================] - 0s 6ms/step - loss: 6.6726 - mae: 6.6726
Epoch 95/100
2/2 [==============================] - 0s 3ms/step - loss: 11.4520 - mae: 11.4520
Epoch 96/100
2/2 [==============================] - 0s 5ms/step - loss: 15.1483 - mae: 15.1483
Epoch 97/100
2/2 [==============================] - 0s 6ms/step - loss: 9.6731 - mae: 9.6731
Epoch 98/100
2/2 [==============================] - 0s 5ms/step - loss: 9.8415 - mae: 9.8415
Epoch 99/100
2/2 [==============================] - 0s 6ms/step - loss: 11.7518 - mae: 11.7518
Epoch 100/100
2/2 [==============================] - 0s 5ms/step - loss: 8.4396 - mae: 8.4396
<tensorflow.python.keras.callbacks.History at 0x7fdc740e0ed0>
```

```python
# Make predictions and plot predictions
y_pred_1 = model_1.predict(X_test)
plot_predictions(predictions=y_pred_1)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.pr
```




```python
# Calculate model_1 evaluation
mae_1 = tf.metrics.mean_absolute_error(y_test, tf.squeeze(y_pred_1))
mse_1 = tf.metrics.mean_squared_error(y_test, tf.squeeze(y_pred_1))
```

```
mae_1, mse_1
```

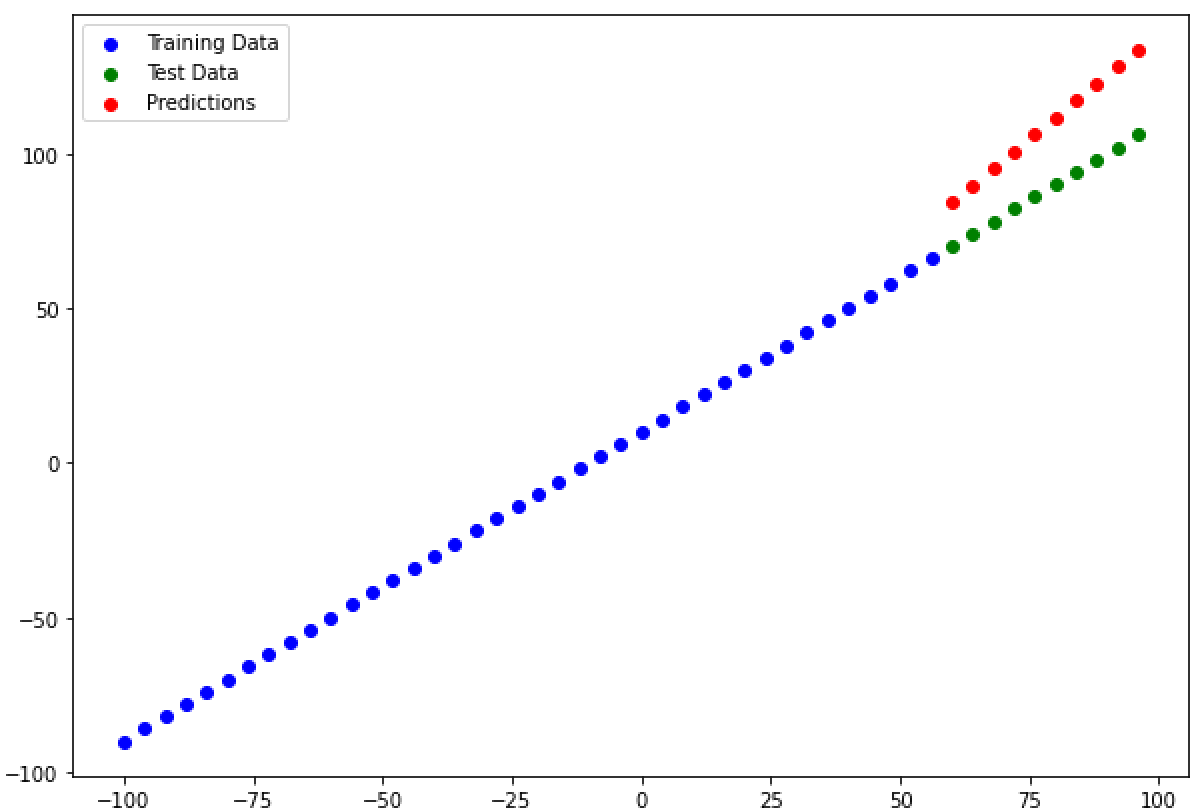
```
(<tf.Tensor: shape=(), dtype=float32, numpy=17.27472>,
 <tf.Tensor: shape=(), dtype=float32, numpy=300.02484>)
```

Build model with 2 layers and 100 epochs

```
# Set random seed
tf.random.set_seed = 42

# 1. Create a model
model_2 = tf.keras.Sequential([
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model_2.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=["mae"])

# 3. Fit the model
model_2.fit(X_train, y_train, epochs=100, verbose=0)
```

```
<tensorflow.python.keras.callbacks.History at 0x7fdc71520950>
```

```
# Make predictions and plot predictions
y_pred_2 = model_2.predict(X_test)
plot_predictions(predictions=y_pred_2)
```



```
# Calculate model_2 evaluation
mae_2 = tf.metrics.mean_absolute_error(y_test, tf.squeeze(y_pred_2))
mse_2 = tf.metrics.mean_squared_error(y_test, tf.squeeze(y_pred_2))

mae_2, mse_2
```

```
(<tf.Tensor: shape=(), dtype=float32, numpy=20.94144>,
 <tf.Tensor: shape=(), dtype=float32, numpy=456.9501>)
```

Comparing results of our experiments

```
# Let's compare our model results with Pandas DataFrame
import pandas as pd

model_results = [["model_1", mae_1.numpy(), mse_1.numpy()],
                 ["model_2", mae_2.numpy(), mse_2.numpy()]]
all_results = pd.DataFrame(model_results, columns=["model", "mae", "mse"])

all_results
```

	model	mae	mse
0	model_1	17.274719	300.024841
1	model_2	20.941441	456.950104

Saving our model

Saving models allows us to use them in future use.

There are two formats in which we can save our model.

1. The SavedModel format
2. HDF5 format.

```
# Save model using SavedModel format
model.save("best_model_saved_model")
```

```
INFO:tensorflow:Assets written to: best_model_saved_model/assets
```

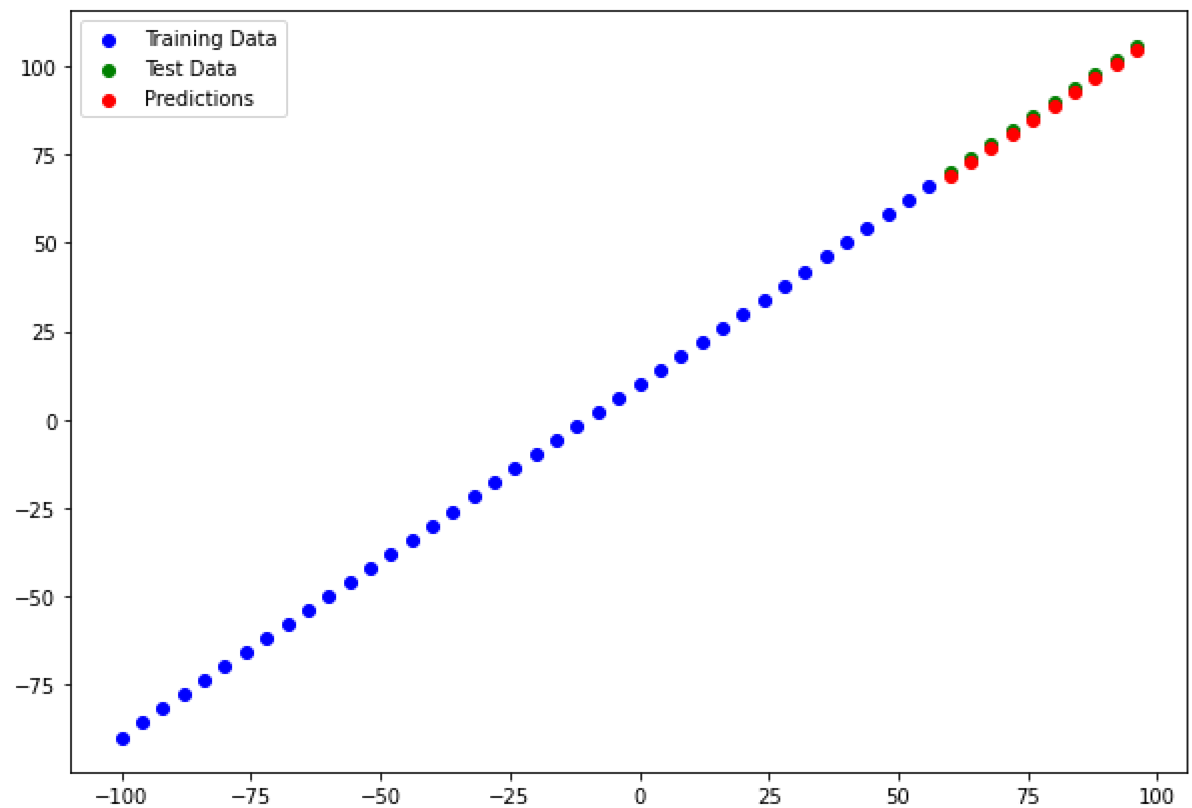
```
# Save model using HDF5 format
model.save('best_model_hdf5.h5')
```

Loading in a saved model

```
# Load in saved model format
loaded_saved_model = tf.keras.models.load_model("best_model_saved_model")
loaded_saved_model.summary()
```

```
Model: "first_model"

_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_layer (Dense)         (None, 10)                20

_________________________________________________________________
hidden_layer_1 (Dense)      (None, 10)                110

_________________________________________________________________
output_layer (Dense)        (None, 1)                 11
=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0

_________________________________________________________________
```

```
y_pred_saved_model = loaded_saved_model.predict(X_test)
plot_predictions(predictions=y_pred_saved_model)
```




```
y_pred == y_pred_saved_model
```

```
array([[ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True]])
```

```
# Load in hdf5 format model
loaded_h5_model = tf.keras.models.load_model('best_model_hdf5.h5')
```

```python
loaded_h5_model = tf.keras.models.load_model("best_model_hdf5.h5")
loaded_saved_model.summary()
```

```
Model: "first_model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_layer (Dense)         (None, 10)                20
_________________________________________________________________
hidden_layer_1 (Dense)      (None, 10)                110
_________________________________________________________________
output_layer (Dense)        (None, 1)                 11
=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
_________________________________________________________________
```

```python
# Download a file from google colab
from google.colab import files
files.download("/content/best_model_hdf5.h5")
```

✓ 0s completed at 6:36 PM