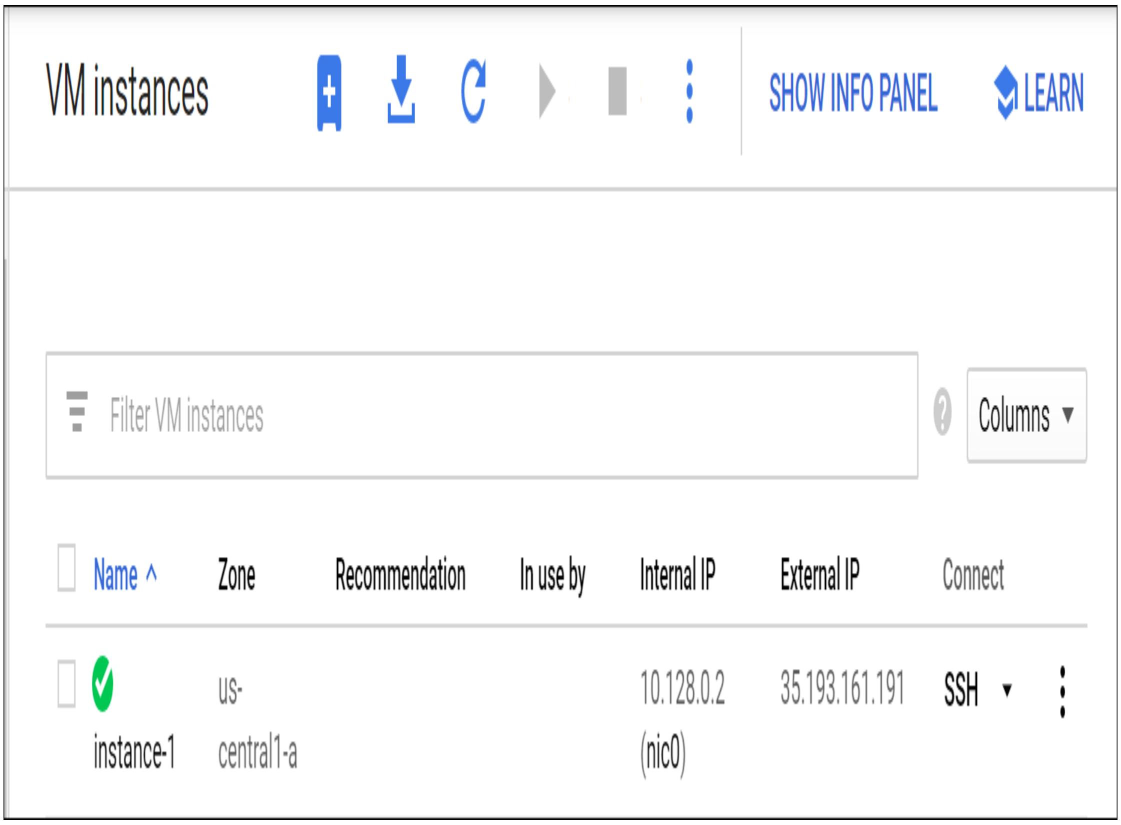
Create a VM

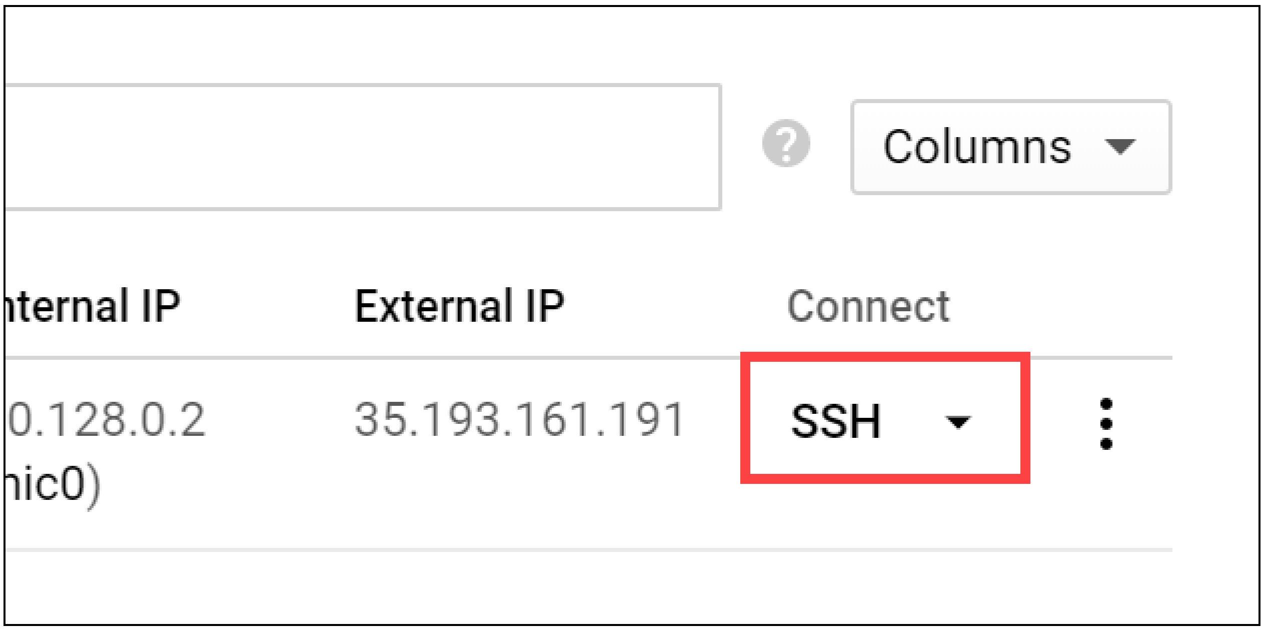
Set up VM to carry out the lab tasks.

1.Select **Navigation menu** > **Compute Engine**, and then click **Create**.
2.In the **Create an instance** dialog, in the right pane, set the Machine type to **n1-standard1**. Leave all other fields at the default values.
3.Click **Create**.
You new VM is listed in the VM instance list, you see a green check your VM has been successfully created.



	Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect
✓	instance-1	us-central1-a			10.128.0.2 (nic0)	35.193.161.191	SSH ▾

4.Click **SSH** to the right of your new VM instance to connect to the console of the VM via SSH.



Click Check my progress to verify the objective.

Install the Python development environment on your system

1.Check if your Python environment is already configured:

```
python3 --version
pip3 --version
virtualenv --version
content_copy
```

Example output:

```
gcpstaging82617_student@instance-1:~$ python3 --version
Python 3.5.3
gcpstaging82617_student@instance-1:~$ pip3 --version
-bash: pip3: command not found
gcpstaging82617_student@instance-1:~$ virtualenv --version
-bash: virtualenv: command not found
gcpstaging82617_student@instance-1:~$
content_copy
```

The output shows that Python 3.5.3 is already in the environment, but not the pip package manager, and Virtualenv.

2.Install the pip package manager, and Virtualenv:

```
sudo apt update
content_copy
sudo apt install python3-pip
content_copy
```

When prompted, click **Y** to continue.

```
sudo pip3 install -U virtualenv  # system-wide install
content_copy
```

3.Go back and confirm the pip package manager and Virtualenv is installed:

```
pip3 --version
virtualenv --version
content_copy
```

A version for the pip package manager and Virtualenv is output.

Example output:

```
gcpstaging82439_student@instance-1:~$ pip3 --version
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.5)
gcpstaging82439_student@instance-1:~$ virtualenv --version
16.7.2
content_copy
```

Create a virtual environment

1.Create a new virtual environment. Create a directory (./venv) and then choose a Python interpreter to put in the directory:

```
virtualenv --system-site-packages -p python3 ./venv
content_copy
```

2.Activate the virtual environment:

```
source ./venv/bin/activate  # sh, bash, ksh, or zsh
```

```
content_copy
```

When virtualenv is active, your shell prompt is prefixed with (venv), check it out!

3.Upgrade pip to install packages within a virtual environment without affecting the host system setup.

```
pip install --upgrade pip
content_copy
```

4.View the packages installed within the virtual environment

```
pip list  # show packages installed within the virtual environment
content_copy
```

Install the TensorFlow pip package

TensorFlow is Google's open source library for machine learning. In this lab you install TensorFlow as the library for your machine learning model.

1.Install the TensorFlow package:

```
pip install --upgrade tensorflow
content_copy
```

2.Verify the install:

```
python -c "import warnings;warnings.simplefilter(action='ignore',
category=FutureWarning);import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000,
1000])))"
content_copy
```

Expected output:

```
Tensor("Sum:0", shape=(), dtype=float32)
content_copy
```

TensorFlow is now installed!

Create your first machine learning models

Consider the following sets of numbers. Can you see the relationship between them?

X:	-1				3	4
Y:	-2				10	13

As you look at them you might notice that the X value is increasing by 1 as you read left to right, and the corresponding Y value is increasing by 3. So you probably think Y=3X plus or minus something. Then you'd probably look at the zero on X and see that Y = 1, and you'd come up with the relationship Y=3X+1.

That's almost exactly how you would use code to train a model, known as a neural network, to spot the patterns between these items of data!

Looking at the code used to do it, you see you use data to train the neural network! By feeding it with a set of Xs and a set of Ys, it should be able to figure out the relationships.

Step through creating the script piece by piece.

Create and open the file model.py.

```
nano model.py
content_copy
```

Keep this file open in nano as you'll add code in the next sections.

Imports

Add the following code to your imports:

•TensorFlow and call it tf for ease of use
•A library called numpy, which helps represent the data as lists
•keras, the framework for defining a neural network as a set of sequential layers
Add the import code into the model.py:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False
content_copy
```

Leave model.py open in nano for the next section.

Define and compile the neural network

Next, create the simplest possible neural network. It has 1 layer, and that layer has 1 neuron, and the input shape is 1 value.

1.Add the following code to model.py:

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
content_copy
```

Next, you write the code to compile your neural network. When you do, you must specify 2 functions, a loss and an optimizer.

If you've seen lots of math for machine learning, here's where you would usually use it, but in this case it's nicely encapsulated in functions for you. Step through what's happening:

•You know that in the function, the relationship between the numbers is y=2x-1.

•When the computer tries to 'learn' that, it makes a guess...maybe y=10x+10. The loss function measures the guessed answers against the known correct answers and measures how well or how badly it did.

•Next, the model uses the optimizer function to make another guess. Based on the loss function's result, it will try to minimize the loss. At this point maybe it will come up with something like y=5x+5. While this is still pretty bad, it's closer to the correct result (i.e. the loss is lower).

•The model repeats this for the number of epochs you specify.

•But first, you add code to model.py to tell it to use mean squared error for the loss and stochastic gradient descent (sgd) for the optimizer. You don't need to understand the math for these yet, but you can see that they work! :)

2.Add the following code to model.py:

```
model.compile(optimizer='sgd', loss='mean_squared_error')
content_copy
```

Leave model.py open in nano for the next section.

Providing the data

Next up, feed in some data. In this lab, you're using the 6 Xs and 6 Ys used earlier:

X:	-1				3	4
Y:	-2				10	13

As you can see, the relationship between these is that Y=2x-1, so where X = -1, Y=-3 etc. etc.

A python library called numpy provides lots of array type data structures that are a defacto standard way of feeding in data. To use these, specify the values as an array in numpy using np.array\[\]

Add the following code to model.py:

```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
content_copy
```

Your script, model.py, contains all of the code needed to define the neural network. In the next section, you add code to train the neural network to infer the patterns between these numbers and use those to create a model.

Training the neural network

To train the neural network to 'learn' the relationship between the Xs and Ys, you step it through a loop: make a guess, measure how good or bad it is (aka the loss), use the optimizer to make another guess, etc. It will do it for the number of epochs you specify, which in this lab is 500.

1.Add the following code to model.py:

```
model.fit(xs, ys, epochs=500)
content_copy
```

Your script, model.py, should look like this:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.python.util import deprecation
deprecation._PRINT_DEPRECATION_WARNINGS = False
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
model.fit(xs, ys, epochs=500)
content_copy
```

2.Press **Ctrl+x** to close nano, then press **Y** to save the model.py script, then **Enter** to confirm the script name.
Click Check my progress to verify the objective.

Run your script

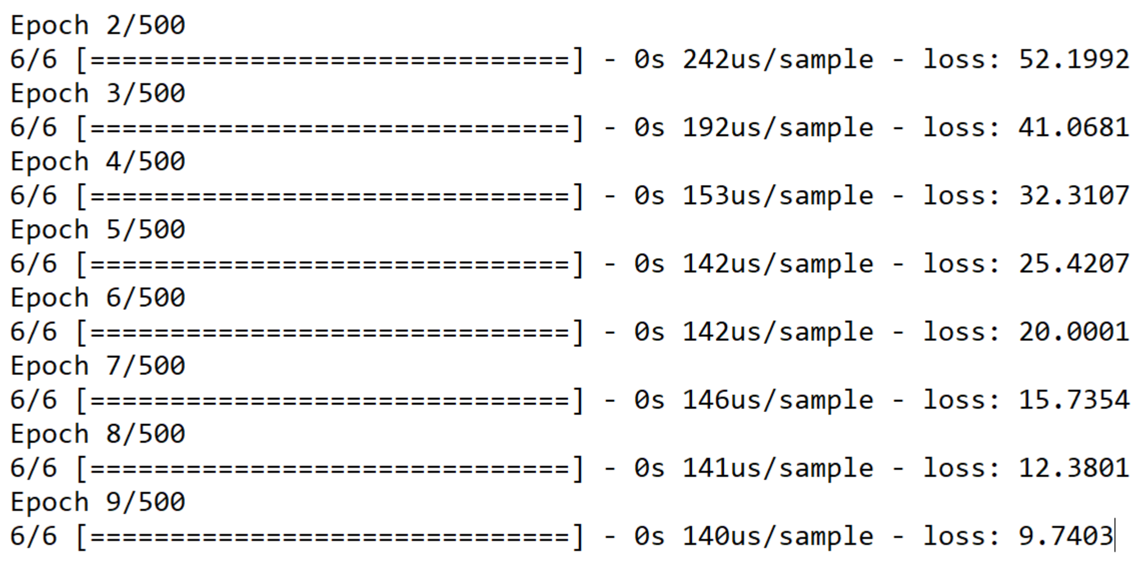
Your script is ready! Run it to see what happens:

```
python model.py
```

content_copy

Look at the output, yours may be slightly different. Notice that the script prints out the loss for each epoch.

If you scroll through the epochs, you see that the loss value is quite large for the first few epochs, but gets smaller with each step. For example:



```
Epoch 2/500
6/6 [==============================] - 0s 242us/sample - loss: 52.1992
Epoch 3/500
6/6 [==============================] - 0s 192us/sample - loss: 41.0681
Epoch 4/500
6/6 [==============================] - 0s 153us/sample - loss: 32.3107
Epoch 5/500
6/6 [==============================] - 0s 142us/sample - loss: 25.4207
Epoch 6/500
6/6 [==============================] - 0s 142us/sample - loss: 20.0001
Epoch 7/500
6/6 [==============================] - 0s 146us/sample - loss: 15.7354
Epoch 8/500
6/6 [==============================] - 0s 141us/sample - loss: 12.3801
Epoch 9/500
6/6 [==============================] - 0s 140us/sample - loss: 9.7403
```

As the training progresses, the loss soon gets very small:

```
Epoch 45/500
6/6 [==============================] - 0s 146us/sample - loss: 0.0023
Epoch 46/500
6/6 [==============================] - 0s 154us/sample - loss: 0.0020
Epoch 47/500
6/6 [==============================] - 0s 145us/sample - loss: 0.0017
Epoch 48/500
6/6 [==============================] - 0s 139us/sample - loss: 0.0014
Epoch 49/500
6/6 [==============================] - 0s 149us/sample - loss: 0.0012
Epoch 50/500
6/6 [==============================] - 0s 178us/sample - loss: 0.0011
Epoch 51/500
6/6 [==============================] - 0s 142us/sample - loss: 9.5416e-04
Epoch 52/500
6/6 [==============================] - 0s 149us/sample - loss: 8.5538e-04
Epoch 53/500
6/6 [==============================] - 0s 145us/sample - loss: 7.7552e-04
Epoch 54/500
6/6 [==============================] - 0s 145us/sample - loss: 7.1057e-04
```

And by the time the training is done, the loss is extremely small, showing that our model is doing a great job of inferring the relationship between the numbers:

```
Epoch 495/500
6/6 [==============================] - 0s 150us/sample - loss: 5.4194e-08
Epoch 496/500
6/6 [==============================] - 0s 149us/sample - loss: 5.3076e-08
Epoch 497/500
6/6 [==============================] - 0s 154us/sample - loss: 5.1974e-08
Epoch 498/500
6/6 [==============================] - 0s 145us/sample - loss: 5.0963e-08
Epoch 499/500
6/6 [==============================] - 0s 152us/sample - loss: 4.9884e-08
Epoch 500/500
6/6 [==============================] - 0s 139us/sample - loss: 4.8896e-08
```

You probably don't need all 500 epochs, try experimenting with different amounts. Looking at this example, the loss is really small after only 50 epochs, so that might be enough!

Using the model

You now have a model that has been trained to learn the relationship between X and Y. You can use the model.predict method to figure out the Y for a previously unknown X. So, for example, if X = 10, what do you think Y will be?

1.Open model.py:

```
nano model.py
```

content_copy

2.Add the following code to the end of the script:

```
print(model.predict([10.0]))
```

content_copy

3.Press **Ctrl+x**, **Y**, then **Enter** to save and close model.py.

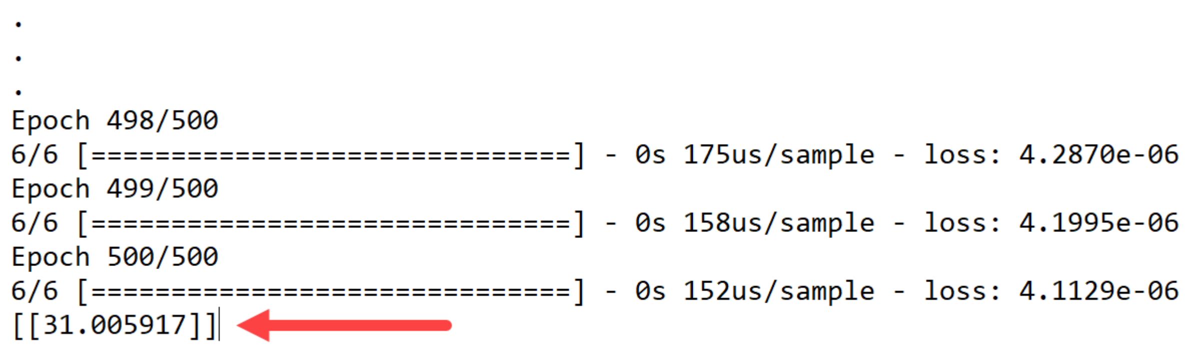
4.Take a guess about the Y value, then run your script:

```
python model.py
```

content_copy

The Y value is listed after the epochs.

Example output:



```
.
.
.
Epoch 498/500
6/6 [==============================] - 0s 175us/sample - loss: 4.2870e-06
Epoch 499/500
6/6 [==============================] - 0s 158us/sample - loss: 4.1995e-06
Epoch 500/500
6/6 [==============================] - 0s 152us/sample - loss: 4.1129e-06
[[31.005917]]
```

You might have thought Y=31, right? But in the example output above, it ended up being a little over (31.005917). Why do you think that is?

Answer: Neural networks deal with probabilities, so given the data that you fed in to it, the neural network calculated a very high probability that the relationship between X and Y is Y=3X+1, but with only 6 data points you can't know for sure. As a result, the result for 10 is very close to 31, but not necessarily 31.

As you work with neural networks, you'll see this pattern recurring. You will almost always deal with probabilities, not certainties, and will do a little bit of coding to figure out what the result is based on the probabilities, particularly when it comes to classification.

Click Check my progress to verify the objective.